Diss. ETH No. 18270

Multi–Level Change Awareness for Collaborative Authoring Applications

A dissertation submitted to the
ETH ZURICH

for the degree of
Doctor of Sciences

presented by

Stavroula Papadopoulou

Dipl. Ing. Aristotle University of Thessaloniki, Greece
born November 22, 1980
citizen of Greece

accepted on the recommendation of

Prof. Dr. Moira C. Norrie, examiner
Prof. Dr. Wolfgang Prinz, co-examiner
Prof. Dr. Chengzheng Sun, co-examiner

2009

Copyright © 2009 by Stavroula Papadopoulou

To my husband,
Ioannis

Abstract

Awareness has been identified as one of the key issues of collaborative work. Maintaining knowledge about the status and activities of their collaborators can help people to achieve better coordination and efficient collaboration. Many types of awareness have been defined for a range of collaborative activities and change awareness has been identified as an important element of collaborative authoring.

Identifying the importance of change awareness, many applications have been designed and developed which aim to help users maintain change awareness by enriching the knowledge that they have about the editing activity of collaborators. User studies conducted throughout the years show that users appreciate the awareness features provided by current applications. However, they also report that they are not provided with all of the change awareness information that they require.

A brief look at the currently implemented collaborative tools is sufficient to realise that there is no common approach followed by developers, with respect to the change awareness mechanisms that the tools offer. Systems from different domains offer different awareness information and fulfil different requirements. Very few approaches are reusable for other document types or all three collaboration modes – synchronous, asynchronous and semi-synchronous. Finally, the most important drawback of current collaborative tools is that different document models are used for different document types. Even though hierarchically structured documents are widely used in various systems, there is no commonly accepted document model in the CSCW community and only a few systems exploit the document structure.

We propose a generic mechanism for the computation and visualisation of multi-level awareness in applications that enable the co-authoring of documents. This mechanism addresses all the above issues. It uses the structured model of the co-authored documents to provide awareness at different granularity levels. The modifications made to a document are recorded and their "severity" is computed based on various metrics defined in the framework. This information is collected for all document parts at the different document levels and then presented to the users through appropriate visualisation tools, for instance, an edit profile. The definitions of the basic concepts included in our awareness mechanism and the framework resulting from it are kept general to enable the framework's extensibility and applicability to existing collaborative authoring tools, independently of the type of the co-authored documents or the working code of the collaborators.

The framework's applicability has been tested through its integration into an asynchronous text application. We used the implemented prototype of the application enriched with multi-level change awareness, to conduct user studies. User reports show that the awareness-enabled application successfully provides multi-level awareness to the users and that users are able to correctly interpret the information provided to them.

The framework's generality and extensibility is shown through its integration into a set of collaborative editors that support the co-authoring of different document types in different working modes. In detail, we used the framework to extend an asynchronous text editor with change awareness in real time and to develop a shared workspace that provides, in real time, change awareness information for various documents. The use of shadow document sets additionally enables the monitoring and computation of awareness information about concurrent committed and uncommitted changes. Finally, we show how our awareness mechanism could be used to enrich existing applications that support the co-authoring of graphical documents and webpages or the co-authoring of documents in cases where privacy issues arise. The integration of the framework into the above applications required a set of various metrics and visualisation tools that could be defined based on the applications' functionalities and the users' needs and collaborating tasks.

We believe that our awareness mechanism provides a platform for experimentation to be used by developers of awareness-enabled collaborative authoring tools. The mechanism's main components and the metamodel describing it are expected to assist developers in experimenting with various metrics and visualisation tools for the computation and visualisation of the appropriate change awareness information for each application. The functionalities offered by each application, as well as the needs of the users should be considered for the above procedure.

Zusammenfassung

Bewusstsein oder „Awareness” hat sich als eines der wichtigsten Themen der Zusammenarbeit entwickelt. Das Wissen über den Status und die Aktivitäten ihrer Mitarbeiter kann den Menschen helfen, Koordinierter und effizienter Zusammenarbeit zu schaffen. Mehrere Typen von Awareness wurden definiert, um die unterschiedlichen kollaborativen Aktivitäten zu unterstützen. So hat sich Change Awareness als ein wichtiges Element des Co-Authoring entwickelt.

Nachdem die Wichtigkeit der Change Awareness identifiziert worden war, sind viele Anwendungen entwickelt worden. Sie haben als gemeinsames Ziel, Benutzern dabei zu helfen, ihre Change Awareness zu schärfen, d.h. die Kenntnisse zu erweitern, die sie über das Verhalten ihrer Mitarbeiter beim Editieren von Dokumenten haben. User-Studien der letzten Jahre zeigen, dass Benutzer die durch die gegenwärtigen Anwendungen zur Verfügung gestellten Awareness-Eigenschaften schätzen. Jedoch berichten sie, dass sie nicht mit der ganzen erforderlichen Information über die Change Awareness versorgt werden.

Ein kurzer Blick auf die zurzeit aktuellen Co-Authoring-Werkzeuge genügt, um zu erkennen, dass es keine allgemein gültige Methode in Bezug auf Change Awareness-Mechanismen gibt. Systeme aus verschiedenen Forschungsgebiete bieten verschiedene Awareness-Informationen an und erfüllen verschiedene Anforderungen. Sehr wenige Methoden sind zum Beispiel für andere Dokumenten-Typen oder alle drei Kollaborationsarten – gleichzeitig, asynchron und semi-synchron – wiederverwendbar. Schließlich ist der wichtigste Nachteil der derzeitigen kollaborativen Anwendungen, dass verschiedene Dokumenten-Modelle für die verschiedenen Dokumenten-Typen verwendet werden. Obwohl die hierarchisch strukturierten Dokumente häufig in verschiedenen Anwendungen verwendet werden, gibt es kein allgemein akzeptiertes Dokumenten-Modell im CSCW Forschungsgebiet, und nur wenige Systeme nutzen die Struktur des Dokuments aus.

Wir zeigen einen allgemeinen Mechanismus auf für die Berechnung und Visualisierung der Multi-Level Awareness in Anwendungen, die das Co-Authoring von Dokumenten ermöglichen. Dieser Mechanismus verwendet das strukturierte Modell der Co-Authored Dokumente, um Awareness mit verschiedenen Granularitäten zu ermöglichen. Die Änderungen an einem Dokument werden erfasst, und ihre „Severity” wird beruhend auf den verschiedenen, im Framework definierten Metriken geschätzt. Diese Information wird für alle Dokumententeile an den verschiedenen Strukturebenen der Dokumente gesammelt und dann den Benutzern durch passende Visualisierungs-Werkzeuge, wie zum Beispiel ein „Editing Profile” präsentiert. Die Definition der grundlegenden Konzepte zusammen mit unserem Awareness-Mechanismus und dem Framework, das sich daraus ergibt, wird allgemein präsentiert, um die Erweiterbarkeit und Anwendbarkeit des Frameworks auf vorhandene Co-Authoring Werkzeuge unabhängig vom Typ der Co-Authored Dokumente oder des Arbeitsmodus der Mitarbeiter zu ermöglichen.

Die Anwendbarkeit des Frameworks wird durch dessen Integration in eine asynchrone Textanwendung geprüft. Wir verwenden den implementierten Prototypen einer Anwendung, die um Multi-Level Change Awareness erweitert wurde, um Benutzerstudien durchzuführen. Unsere Studien zeigen, dass die um Awareness erweiterte Anwendung den Benutzern erfolgreich Multi-Level Change Awareness bietet, und dass die Benutzer in der Lage sind, die ihnen gegebenen Informationen richtig zu interpretieren.

Außerdem wird die Allgemeingültigkeit und Erweiterbarkeit unseres Frameworks durch dessen Integration in eine Reihe von kollaborativen Anwendungen gezeigt, wobei das Co-Authoring von verschiedenen Dokumententypen in verschiedenen Arbeitsmodus unterstützt wird. Insbesondere verwenden wir das Framework, um eine asynchrone Textanwendung um Change Awareness in Echtzeit zu erweitern, und um einen Shared Workspace zu entwickeln, der Change-Awareness Informationen für verschiedene Dokumente in Echtzeit zur Verfügung stellt.

Der Gebrauch von „Shadow-Dokumenten” ermöglicht zusätzlich die Überwachung und Verarbeitung der Awareness-Information über „concurrent committed” und „uncommitted” Änderungen. Schließlich zeigen wir, wie unser Awareness-Mechanismus verwendet werden könnte, um vorhandene Anwendungen zu erweitern, damit sie das Co-Authoring von graphischen Dokumenten und Webseiten oder das Co-Authoring von solchen Dokumenten unterstützen, bei denen die Privatsphäre gewahrt werden muss. Um die Integration des Frameworks in die obengenannten

Anwendungen durchzuführen, benötigen wir eine Reihe verschiedener Metriken und Visualisierungs-Werkzeuge. Diese konnten wir auf Basis der Anwendungs-Features und der Bedürfnisse der Benutzer und ihrer kollaborativen Aufgaben definieren.

Wir glauben, dass unser Awareness-Mechanismus eine Plattform zur Verfügung stellt, die von Entwicklern von Awareness-fähigen Co-Authoring Werkzeuge zum Experimentieren verwendet werden kann. Wir erwarten, dass die grundlegenden Konzepte des Mechanismus den Entwicklern helfen werden, mit verschiedenen Metriken und Visualisierungs-Werkzeuge zu experimentieren, um die Berechnung und Visualisierung der optimalen Change-Awareness Information für jede Anwendung unterstützen zu können. In jedem Fall sollten dann die Funktionalitäten, die durch die individuelle Anwendung angeboten werden, und die Bedürfnisse der Benutzer für das obengenannte Verfahren berücksichtigt werden.

Acknowledgements

I would like to thank everybody who directly or indirectly contributed to the outcome of this thesis.

First and foremost, I would like to thank Prof. Dr. Moira C. Norrie for giving me the opportunity of pursuing my doctoral studies in her group and the freedom of choosing the content of my work. I am grateful for her constant support and the fact that she was always there, to encourage but also challenge new ideas. I believe that through our discussions I learnt a lot about research but also life in general and I am grateful to her about this.

I would like to specially thank Prof. Dr. Wolfgang Prinz and Prof. Dr. Chengzheng Sun for accepting the co-supervision of my work. Their constant interest in my work and the discussions we had at various stages of the thesis have helped better shape the direction of my research. Finally, their comments during the last stages of this work have been most helpful and constructive. Dear Professors, it has been an honour and my pleasure working with you.

Special thanks go to my collaborators and friends Claudia-Lavinia Ignat and Gérald Oster. Through their unconventional point of view, they have often challenged my ideas and helped to polish and strengthen the fundamental concepts of my work. I would like to thank them very much for sharing with me their experience and knowledge of the CSCW research area. I feel very lucky to have collaborated with them at both ETH Zurich and INRIA Nancy.

I would also like to thank all my colleagues from the Global Information Systems research group for their direct and indirect support. Their interest in my work helped me retain my motivation for quality work through both the pleasant as well as the difficult stages of my research. I would like to specially thank Alexandre de Spindler for his help in the implementation of the BeAware shared workspace, Fabrice Matulic for thoroughly proof-reading this thesis, Michael Nebeling for his help in the

German translation of the abstract of this thesis and Beat Signer for the fruitful discussions on the metamodel of the awareness framework. Finally, I thank everyone for making the GlobIS group such a great place to be.

What I have achieved until now in my life I largely owe to my family. I consider myself very privileged to have grown up in such a caring and loving family. My parents Dimitris and Eleni and my brother Kostas have constantly supported me throughout all my life stages. I especially would like to thank my parents for helping me build self-confidence and teaching me that dreams are meant to be fulfilled. Mom, Dad, Kostas, I deeply thank you for everything.

However, the person who mostly supported me regarding this thesis and gives meaning to it is my cherished husband Ioannis. I thank him for being always there, to share the happy moments but also to offer support and understanding when difficulties arose. In the journey of our life, Ioannis has always acted as a catalyst, making my dreams come true. Ioannis, it is to you that I would like to dedicate this thesis and I deeply thank you for everything.

Table of Contents

List of Figures

List of Tables

1 Introduction

Many of the activities individuals do in their everyday life are collaborative. Either at work, at home, in education, or for entertainment, people collaborate all the time. When collaborating, people form groups and work on a common task. They may collaborate during different phases of a collaborative task, sometimes more tightly than others. To support people in collaborative tasks, a wide variety of tools have been developed. Examples are video conferencing facilities, decision rooms, group calendars, shared screens, email clients and version control systems. In this chapter, we discuss about Computer-Supported Cooperative Work (CSCW), a field of research that aims at supporting people when involved in collaborative tasks. We present some of the dimensions, used to categorise collaborative systems as well as some of the key issues of CSCW, including awareness. We then go on to provide the context and motivation of our work by presenting some of the most common drawbacks of current collaborative authoring systems before presenting the goals of the thesis.

1.1 Computer-supported cooperative work

Collaborative work is not a recent phenomenon. Already in 1985, user studies showed that most of the work in business and academia was performed by groups of people [24]. It is no coincidence that around the same time, the term *Computer-supported collaborative work (CSCW)* first appeared. It was introduced by Irene Greif and Paul M. Cashman at a workshop organised by "people from various disciplines who shared an interest in how people work, with an eye to understanding how technology could support them" [55]. Carstensen and Schmidt later reported that CSCW addresses "how collaborative activities and their coordination can be supported by means of computer systems" [29]. Around the same time, Ellis et al. defined *groupware* as "computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment" [41]. Many authors consider the terms CSCW and groupware as synonyms, while in reality, CSCW is a broader term than groupware. While groupware refers to real computer-based systems, CSCW focuses on the study of tools and techniques of groupware as well as their psychological, social and organisational effects. The definition of CSCW provided by Wilson [118] clarifies the difference between the two terms.

Definition 1. *CSCW* (is) a generic term which combines the understanding of the way people work in groups with the enabling technologies of computer networking, and associated hardware, software, services and techniques.

The classification of collaborative activities and systems into different categories is made based on two characteristics; the *time* when the activity occurs and the *place* where the collaborators are situated. Collaborators can be at the same place (*collocated*) or at different places (*remote*) and their activity may take place at the same time (*synchronous*), or at different times (*asynchronous*). The time/space matrix [23, 70], known also as the CSCW matrix, has often been used to categorise systems in CSCW as shown in Figure 1.1. Examples of collaborative situations for each category are shown in the same figure.

Since 1988, when the CSCW matrix was first introduced, an increasing amount of research has been devoted to systems that support collaborative work. The modes of collaboration as well as the system requirements and characteristics have changed, making the matrix out-dated with respect to the time distinction of systems and the mapping of system properties to specific user interactions. As systems evolve, they are often used for tasks other than the ones for which they were designed. The most common examples of such interactions is the use of instant messaging between groups of co-located people, or the use of emails for synchronous communication. To better understand the time distinction

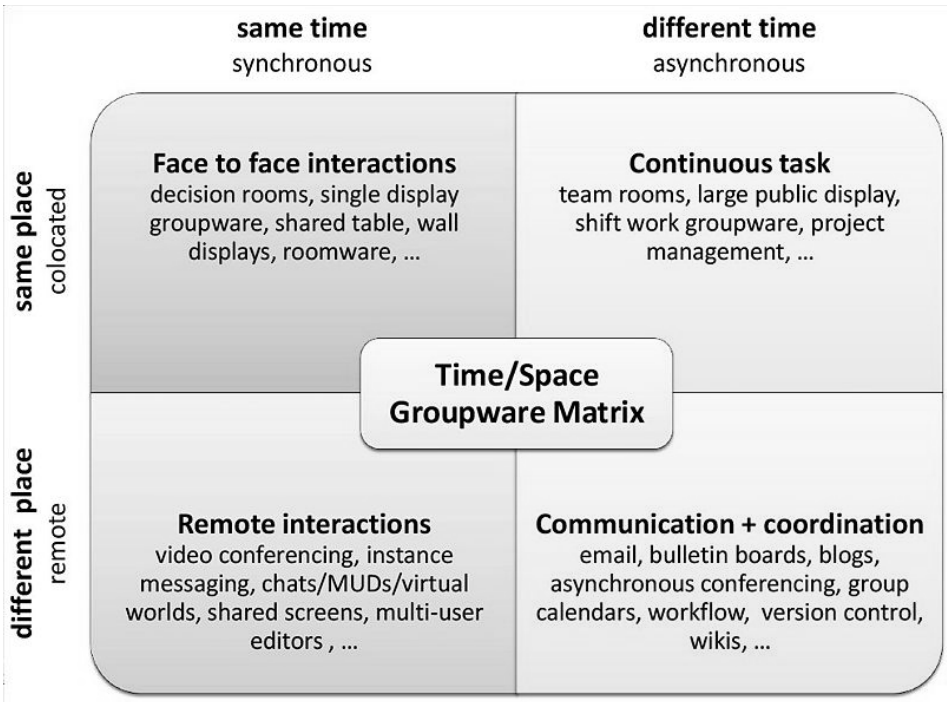

same time
synchronous
different time
asynchronous
same place
colocated
different place
remote
Face to face interactions
decision rooms, single display groupware, shared table, wall displays, roomware, ...
Continuous task
team rooms, large public display, shift work groupware, project management, ...
Time/Space Groupware Matrix
Remote interactions
video conferencing, instance messaging, chats/MUDs/virtual worlds, shared screens, multi-user editors , ...
Communication + coordination
email, bulletin boards, blogs, asynchronous conferencing, group calendars, workflow, version control, wikis, ...

Figure 1.1: The CSCW Matrix

of current collaborative systems, empirical studies have been conducted to identify how people collaborate.

Posner and Baecker [96], for instance, looked into the way people collaborate in writing and identified a number of different writing strategies used in different phases when authoring a document. They conclude that both synchronous and asynchronous strategies are used and that a system should provide both of them as well as a smooth transition between them. *Semi-synchronous* collaboration was then introduced, as an attempt to fill the gap between asynchronous and synchronous modes [84].

The fact that different situations demand different working styles has been acknowledged by other authors as well. Dourish and Bellotti call for an editing model which supports both synchronous and asynchronous work and a smooth transition between them [35]. Haake and Wilson introduced SEPIA, a hypertext authoring system, that supports both *individual, loosely coupled* and *tightly coupled* modes and also switching between them [60]. Finally, Molli et al. introduced the notion of *multi-synchronous environments* where users freely modify documents which continuously go through divergence and convergence phases. While all of these newly introduced working modes have a slightly different definition, they all serve one common objective, namely, being the description of a working mode that lies between synchronous and asynchronous modes.

We will refer to this working mode using the term *semi-synchronous collaboration.*

Finally, some of the key issues of CSCW identified by [30] are "awareness, multi-user interfaces, concurrency control, communication and coordination within the group, shared information space and the support of a heterogeneous, open environment which integrates existing single-user applications". In the next section we discuss awareness, which is the central issue of this thesis.

1.2 Awareness

Awareness is a natural phenomenon that people rarely need to think about. It is generally defined in terms of two notions: knowledge and consciousness. Awareness as knowledge means that there is some piece of information in a person's memory, while awareness as consciousness means that the person is conscious of this information. In the context of this thesis, we concentrate on the first notion and think of awareness as the outcome of an agent's interaction with its environment. In this sense, awareness can be simply defined as "knowing what is going on" [42].

Some of the basic characteristics of awareness identified by Gutwin [59] are:

- Awareness is knowledge about the state of some environment, a setting bounded in time and space.
- Environments change over time, so awareness is knowledge that must be maintained and kept up-to-date.
- People interact with the environment and the maintenance of awareness is accomplished through this interaction.
- Awareness is almost always part of another activity. This means that awareness is usually a secondary activity that accompanies a primary activity which involves the completion of a task in an environment.

Awareness as presented above, has been identified as *situation awareness* [50]. Situation awareness research usually involves complex environments such as aircrafts and power plants and the interaction of an expert with such systems. *Spatial awareness* and *mode awareness* are two of the specialisations of situation awareness. Spatial awareness is, for instance, a pilot's understanding of their location in an aircraft [44], while mode awareness is “the ability of a supervisor to track and anticipate the behaviour of (mode-based) automated systems” [106].

However, awareness can be maintained about the activity of other persons or groups as well. When people work together they need to maintain awareness about their collaborators to facilitate the coordination of users and effective collaboration. In CSCW research, many awareness types have been identified and formally defined. As shown in Figure 1.2, all of them can be seen as subtypes of situation awareness.

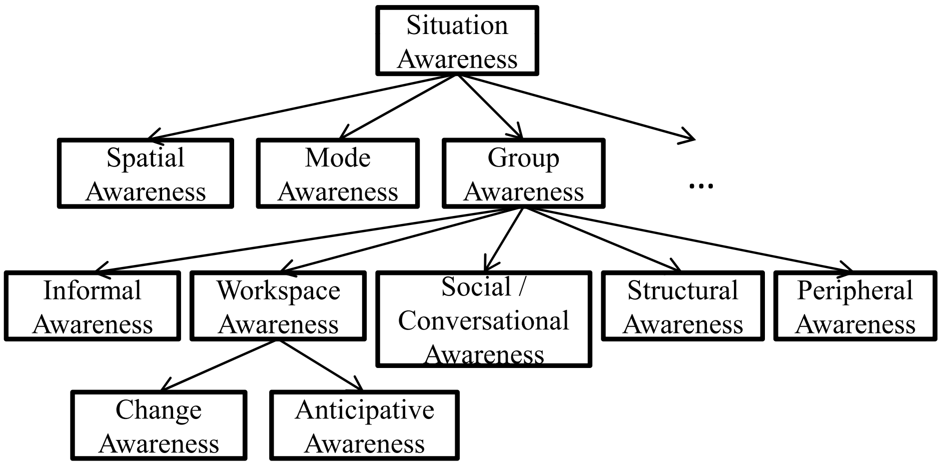


Figure 1.2: Categorisation of awareness types

The main awareness type defined in CSCW is *group awareness*. It was first introduced by Dourish and Bellotti [35] as the “understanding of the activities of others, which provides a context for your own activity”. Group awareness is useful for coordinating actions, managing coupling, discussing tasks, anticipating others actions, and finding help [56]. Group awareness is made up of several kinds of knowledge about what is happening in a collaborative environment, each of them forming a new type of awareness.

Informal awareness in a collaborative community is the knowledge of who is around, what they are doing and whether they are available [36]. This knowledge is usually acquired as a side effect of other activities, for instance, a person walking down a corridor to the water cooler could be informally informed about the status or the current activity of other people.

Social or conversational awareness is the implicit knowledge someone acquires when involved in a conversation. This involves facial expression of conversational partners, eye contact, gestures, intonation or the use of particular words. The mechanisms of social awareness have been extensively studied by linguistics researchers(e.g. [33, 104]). CSCW research in this area focuses on how technology can enable conversations made over distance by using, for instance, teleconferencing systems that take into account social rules and protocols.

Structural awareness is the knowledge of a group's organisation, including user roles, responsibilities and working relationships between group members. Various groupware systems are built in an effort to support structural awareness. Such systems use formalisations such as floor control [51] where the user that has the floor is allowed to speak. Other systems assign user roles to people to define the views of a document available to individuals and the degree of access individuals have to the document as in the Quilt editor [79].

Peripheral awareness is the ability of a person to capture snapshots of the activities of other people while focusing on a different task. A study conducted at the London Underground Line Control Rooms [62, 63] showed the important role of peripheral awareness in a collaborative environment.

Workspace awareness (WA) is the knowledge about how others interact with a shared workspace. It has been defined as the "up-to-the minute knowledge a person requires about another group member's interaction with a shared workspace if they are to collaborate effectively" [53, 59]. Such a general definition results in a great amount of information that could be delivered to users as WA. To ease the design and development of collaborative applications, Gutwin constructed a conceptual framework that "operationalises different aspects of workspace awareness". The framework has three parts: the elements of knowledge that make up WA, the process and mechanisms by which it is maintained, and the uses of workspace awareness in collaboration. These parts correspond to the designer's tasks of determining "what information to present in the interface, how to present the information, and in what situations the information will be useful" [59]. The above framework was first introduced to describe workspace awareness in synchronous collaborative situations. A new version of it appeared later to describe workspace awareness in asynchronous collaborative situations as well.

Tam later introduced *change awareness* as "the ability of a person to track the changes that other collaborators have made to a group project" referring to past activities, thus asynchronous collaboration [114, 115, 116]. He also developed a framework to describe the information elements that construct change awareness. These elements are: "knowing *who* changed the artefact, *what* those changes involve, *where* changes occur, *when* changes were made, *how* things have changed and *why* people made the changes".

We believe that describing the changes made by collaborators on artefacts that belong to a shared workspace, which is what change awareness does, is only part of the knowledge that workspace awareness describes. For example, it also includes information about which users are currently logged in the workspace and what they are currently doing in terms of where they are looking at as well as the changes they are making. Therefore we decided to insert change awareness as a specialisation of workspace awareness in the awareness hierarchy shown in Figure 1.2.

Workspace awareness, as described above, is the knowledge about collaborators interacting with a shared artefact in the past, or in the present. *Anticipative (or expectation) awareness* complements that knowledge by offering information about activities that will or will not happen in the future. Expectation awareness offers the above information by enabling users to "specify their anticipation of future activities on a certain artefact or a group of artefacts in a shared workspace. The users are informed at a chosen point in time if the expected activities happened or not" [97].

Although there already seems to be an immense number of awareness types and definitions, they surprisingly do not fully describe all collaborative activities. *Multi-level change awareness*, defined later in Section 1.3 as a specialisation of change awareness, is an example of an awareness type that has not yet been addressed. It describes the need of users to be informed on different *granularity levels* about the modifications of their collaborators. A further example, is the partial definition of change awareness that describes only asynchronous activities. However, collaborative activities typically involve synchronous, asynchronous and semi-synchronous collaboration and users need to be informed about the changes made by their collaborators to the shared elements independently of the collaboration mode. Therefore, we redefine change awareness to include all of the above collaborative situations.

Definition 2. *Change awareness* is the ability of a person to track the changes that other collaborators make to a group project at any time while interacting with it.

Finally, for each of the awareness types, we define *awareness information* as below.

Definition 3. *Awareness information* is the value of information elements that need to be collected and presented to users, to preserve their awareness.

In the next section we give the motivation and the exact context of this thesis.

1.3 Motivation and context of our work

Since awareness has been identified from the CSCW community as an important feature of collaborative applications, *change awareness* has been investigated by many research teams dealing with collaborative applications for text [64, 91], graphical [116], or software engineering applications [38]. However, user studies [58, 95] have shown that, although users appreciate the new features offered by the increasing number of awareness-enabled collaborative applications, many features users would like to have are still not supported.

These studies highlight a number of issues with respect to change awareness provided in current tools. For instance, Pankoke-Babatz et al. reported that “the time people are willing to spend for consumption of awareness information before starting their action is limited” [95]. Users reported “they missed the opportunity to get an overview first, before taking a closer look” and that they would have preferred to move to locations of interest, directly from an overview. Gutwin et al. reported that participants of their user study embraced the idea of overviews, and preferred to use features, namely the “miniature view” and the “radar view” that informed them about changes made in the whole document as well as about the part where each user was working [58].

The main *goal* of the research conducted in the frame of this thesis is to *increase the level of change awareness delivered by collaborative authoring systems where the collaborators are distributed in time and space* while taking into consideration user needs and preferences emerging from studies like the ones described above. We focus on collaborative applications that manage the authoring of various types of documents, examples of which are text, graphical, software and web documents. The working mode of the collaborators can be synchronous, asynchronous or semi-synchronous collaboration. To achieve such a goal, we need to investigate the problems that the awareness mechanisms of current collaborative authoring tools have and build a list of finer goals that address each of the problems that appear. In the rest of this section, we report on the problems of current tools by presenting some of these tools and briefly discussing their main disadvantages. More details on a complete set of related work on collaborative authoring tools can be found in Chapter 2.

Most of the currently available systems display the changes made to a document either as explanatory notes pointing to the changed parts of the documents ("annotations or markups" technique) or in the document itself, by directly changing the appearance of the document contents ("highlighting" technique). Both techniques impose restrictions on the information returned to users about the changes made to the document, as they do not provide any awareness information associated with the structure of the document, such as an overview of the document parts that changed significantly. For instance, in both Microsoft Word [4] and PREP Editor [91], users need to scroll through the document to detect the parts that have changed. Therefore, there is a need for a less detailed representation alongside the main document, showing the changes made to a document in such a way that users are aware of the parts that have been heavily changed as well as the level of user activity throughout the whole document.

The need for awareness information associated with the structure of the authored document goes beyond the use of an *overview* which is related to the visualisation of awareness information. It influences the computation of awareness information as well. Knowing the *document part* where a change was made has been rated by users as the most important information element of workspace awareness [100]. Therefore, awareness information computed about a whole document that would inform the users about whether the document has changed or not is not sufficient. Rather, the computation of awareness information about changes made to the various parts of a document is required. The advantage of computing awareness information about a document part, is that it can later be provided separately from information computed about other document parts and be visualised upon request.

What is a document though and how is a document part defined? The document engineering community defines documents as well-structured entities that consist of document parts of various syntactic document levels. A text document, for instance a book, consists of chapters, chapters consist of sections, sections of paragraphs etc. down to the character level. In a similar manner, graphical documents can be described as a set of pages, each of them being a set of layers. Layers include graphical objects and groups of graphical objects. Similarly, XML documents consist of elements which build up a hierarchy.

However, research conducted in the CSCW community has mainly represented documents as arrays of characters and document parts as blocks with a specific number of characters [64, 65]. We believe that adopting such an approach is highly restrictive for the semantics of the computed awareness information as well as for its use by users with different roles, involved in different tasks. People are used to assigning tasks or ownerships in the range of syntactic document parts. Therefore, information about user activity on document parts with a syntactic meaning holds much more information for the owners of these parts, i.e. the users mainly responsible for these parts, or other collaborators with tasks related to the corresponding parts.

To satisfy this requirement, the document parts we address above should be defined on different granularity levels. For instance, it should be possible for a collaborative application to return awareness information about all the sections of a document, or all of its paragraphs, or only the paragraphs of a specific section, or the sentences of a paragraph etc. Consider, for instance, a collaborative situation where a group of authors and an editor are responsible for the authoring of a book. While the editor might need to have an overview of the document in terms of its chapters and how much each one has changed, or how it evolves, the authors might also need to see each chapter separately, possibly with an overview of each chapter in terms of its sections. This detailed information would help them compare different parts of the document on a level required by their current task and easily spot document parts that have undergone many changes, or parts where several authors have worked on, etc. Certainly, defining document parts of only one pre-specified document level is not enough. A mechanism that can address document parts of various syntactic levels is needed.

The awareness information people require when collaboratively authoring a document is highly related to their *roles* as well. Some users may need to be informed about every last detail of changes made to a document while others may be satisfied with information showing only major revisions. Examples of the first category of user roles are lawyers collaboratively doing contract work, or authors in a publishing company that have their articles corrected by proofreaders. Users in collaborative situations like these might need information about each single word edited in a document or each spelling mistake corrected. Examples of the second category are publishing editors who need to know the current stage of an article, or professors who want to see the progress of scientific papers authored by students.

There exist approaches [78, 102, 105] that structure program documents in terms of software packages and provide awareness information for classes of a package and how changes on one class may affect “related” classes. Although this effort is a move towards adopting a structured document model and computing awareness information on different structural levels, they currently only concentrate on awareness at the level of a document (class). In a similar way, the collaborative workspaces Groove [13] and BSCW [20] provide information about folders containing collaboratively authored documents, but do not give any awareness information about intra-document changes.

Building a system that computes awareness information satisfying the above requirements can already be a very demanding task. However, tracking the changes made to a document or a document part may not be enough. Consider, for instance, three users working on the same section of a document and making one change each. The first user notices a spelling mistake and corrects it by adding a character. The second clarifies an existing concept by adding a new sentence and the third realises that a concept is missing from the section and corrects this by adding a new paragraph. Although the modification of each user was different, they can all be described as an insertion of a single document node to the modified section. Therefore, the information that each user added a document node is not enough to infer the "importance" of the modifications. We believe that information about the level of the affected (deleted, inserted, modified, etc.) document node should be provided as well.

However, even modifications of the same level do not always have the same importance. Consider an example of two students, each inserting a section to a scientific publication they are co-authoring. The first student adds a section of two paragraphs while the second student adds a section of 10 paragraphs. For the professor supervising them, this could be an indication of the importance of their modifications and their contribution to the publication. It could be argued that the length of each paragraph could also then play a role. And even more importantly, the conceptual value of the modifications should not be undermined. Although the definition of the "semantic" value of a modification would be a very interesting issue, it is out of the scope of this thesis. However, we do consider the "syntactic" value of a modification. For the rest of this thesis, we will refer to this value as the "severity" of a modification and we define it as follows.

Definition 4. The *severity* of a modification is a value describing how much a document is affected when the modification is made to it.

As explained in the examples above, the severity of a modification varies based on the criteria, i.e. *metrics* used to compute it. These criteria depend on the collaborative task, the user roles involved and the current focus of interest for each user.

The concept of a modification's severity is not new in the CSCW research. In a similar way, Lanza and Ducasse [78], Robbes and Lanza [102] and Sarma et al. [105] provide mechanisms for computing a severity measure for changes made through software engineering environments. However, the computation of awareness information in these approaches is based on changes made to lines rather than structural elements. Our definition of the severity concept is more general and it can be computed based on various metrics. The work of "computational wear" [64] is very close to our proposal as well. Actions of reading and authoring lines of a text document are graphically depicted. This is achieved by counting how many times a line is read or updated, and presenting this information to the user as bar charts drawn in the editor scrollbars. The lack of a structured document model restricts considerably the accuracy of the information provided. No flexible way is provided for filtering the above information according to user preferences and presenting it on different granularity levels. Finally, Molli et al. [86] proposed a metric to measure divergence between copies of the same document. Informing users how their copies diverge from each other, and presenting a measure of the conflicts that the changes will cause when published, is expected to generate auto-coordination in a group working collaboratively. Although this approach seems to be promising, the unit for computing the divergence is the document, rather than structural elements of the document. Therefore, it is not possible to provide users with a detailed view of the modifications performed on document parts.

The problem of *reusability* of existing awareness mechanisms also motivated the research conducted in the frame of this thesis. An examination of the CSCW literature, shows that research relative to the computation and presentation of awareness information in currently available collaborative authoring tools follows no common strategy. Various approaches exist for the different document types and the different collaboration modes. There is a need to provide an awareness mechanism applicable to any existing collaborative authoring application, independently of the document type authored or the collaborative working mode chosen by the users.

A plethora of *visualisation tools* have been designed to show changes made to documents. Document comparison tools have been created for text files [2, 3, 9], software [5] or webpages [1, 6, 7]. Unfortunately, all of them can only be used for asynchronous collaboration. They usually implement the copy-modify-merge paradigm and a state-based diff algorithm which can result in a loss of information since some changes made between two versions of a document may not be taken into account either in the computation of the difference, or during the merging process. Additionally, the difference is computed on a line-based approach, where lines with no syntactic meaning in a document are compared. We believe that the use of an operation-based-comparison mechanism and a structured document model can ensure that no information loss will occur and awareness information can be computed for syntactic document parts.

Visualisation tools have also been designed for applications that support synchronous collaboration, with miniatures and radar views [57] being the most commonly used. While they both succeed in providing an overview of the changes made to text or graphical documents [115], they fail to provide awareness on syntactic levels other than the document level. Microsoft Word [4], PREP Editor [91] and TeNDaX [65] manage to show the changes superimposed on the document but do not provide any overview of them. Users need to scroll through the whole document to visualise the parts that have been modified. This requires explicit actions on behalf of the user and will be tiresome in the case of large documents. Finally, "history flow visualisations" [117] provide an overview of a document's evolution, but unfortunately they also rely on a state based approach and the document's evolution, is computed only on the document level.

Summarising, we conclude that the awareness mechanisms provided by current collaborative systems do not sufficiently assist users in maintaining change awareness. The framework introduced by Tam successfully addresses some of the information elements included in change awareness, however, we believe that some information elements are missing. Below we present the complete list of elements, with the newly introduced elements in bold font.

- *who* changed the artefact,
- *what* those changes involve,
- *where* changes occur,
- *when* changes were made,
- *how* things have changed,

- *why* people made the changes,
- ***how much* an artefact has changed**,
- ***which part* of the artefact changed** and
- ***if* the change of one artefact *influences* another artefact**.

Finally, the discussion in this section indicated the need for change awareness at different granularity levels. We believe, therefore, that new flexible awareness mechanisms are needed that would provide multi-level awareness. We define multi-level awareness as follows.

Definition 5. *Multi-level (change) awareness* is the ability of a person to have a view, at different levels of detail, of all changes made by other collaborators to a shared artefact.

Our approach aims at providing a mechanism to compute awareness information about document changes at all available syntactic document levels as well as at levels higher than the document level, such as documents in a workspace. That is to provide a mechanism that preserves the multi-level change awareness of users in collaborative authoring applications. In the next section we discuss in detail our goals.

1.4 Thesis goals and contribution

In the previous sections, we identified the main drawbacks of existing approaches for the computation and visualisation of awareness information in collaborative authoring applications and set the main goal of our research which is to *increase the level of multi-level change awareness provided by current collaborative authoring tools.* Here we present a detailed list of the goals of this thesis and a summary of its contributions.

Prior to developing a new awareness mechanism, we believe that it is important to know what the current mechanisms provide, what their drawbacks are and whether they meet user requirements. For this reason, extensive research is conducted to investigate whether it is beneficial for users to be provided with awareness information at different granularity levels and what kind of information do users need. This is achieved by collecting feedback from user studies testing current collaborative authoring tools.

Additionally, we investigate the granularity of the levels in which users need to be informed and the possible relation of these levels to the users' roles or collaborative tasks. For instance, users can be assigned tasks based on the existing syntactic levels of a document, i.e. user Mary is responsible for the second section. We investigate whether the structure of a document can also be used in the computation of awareness.

As discussed in the previous section, the severity of a modification can be essential for a user to distinguish the interesting changes. For that, appropriate metrics are defined, that compute a modification's severity based on the needs of users. However, the way that the computed information is visualised is equally important. As described in the previous section, users favour the idea of overviews, therefore, we investigate ways of appropriately informing users about document modifications.

Finally, we investigate the possibility of a generic solution that provides the above-mentioned awareness information and is applied to existing collaborative authoring applications independently of the document type or the mode of collaboration. The definition of a general model is investigated, to successfully describe and implement a general awareness framework.

Summarising all of the above, the work in the frame of this thesis aims to provide answers to the above mentioned research questions and has the following list of goals:

- provide a list of requirements about the multi-level awareness required by users,
- investigate different metrics and visualisation tools that compute the required detail of information and present it in an appropriate way depending on users' roles and tasks,
- investigate the creation of a generic framework that provides the required information independently of the types of the co-authored documents and the working modes.

In this thesis, we derive a set of requirements about the multi-level awareness that users need when co-authoring documents. We then design and develop an awareness mechanism that satisfies the requirements. The mechanism uses the document's structure and the concept of operations applied to documents to model the changes made to them. The mechanism includes the concepts of users, document parts, metrics, operations and visualisation tools and can compute the required awareness information and present it to the users at different granularity levels. We show that the mechanism is general enough to be easily applied to various collaborative applications. A framework is also developed based on the above mechanism.

We also show the framework's generality by using it to enhance existing collaborative authoring tools that handle different types of documents and support different working modes. We apply the above framework in an asynchronous collaborative text editor by materialising the model's general concepts to describe the editor's specific characteristics. In a similar manner, we apply the framework in a semi-synchronous shared workspace. To efficiently collect and compute the required awareness information in semi-synchronous environments, as well as present it to the users without distracting them, we define the notion of shadow document sets. This approach aims at keeping users that work in asynchronous or semi-synchronous environments, aware of document modifications made by their collaborators in real time. This thesis also provides a prototype of a shared workspace that implements both the notion of the shadow documents and the previously mentioned awareness framework.

The results of our research on appropriate metrics and visualisation tools, based on the roles of users and the characteristics of applications, are presented through a set of metrics and variations of a visualisation tool for graphical authoring applications and applications for the authoring of websites.

Finally, the results of our user study, testing the prototype of the asynchronous text editor are presented in this thesis. We show how multi-level awareness, the available metrics and a visualisation tool were appreciated by users, how users interpret the information they are provided with and which are their additional requirements when working in semi-synchronous collaboration.

1.5 Thesis overview

This section gives an overview of the thesis by presenting its evolution and structure. The evolution of this thesis is shown in Figure 1.3. We started by collecting requirements for multi-level change awareness. We analysed the results of user studies testing existing awareness-enhanced systems. Our analysis offered a list of requirements and an assessment of existing systems with regard to the change awareness they provide.

We then concentrated on the core of this thesis, being the definition of an awareness mechanism that provides the required change awareness information independently of the types of the co-authored documents

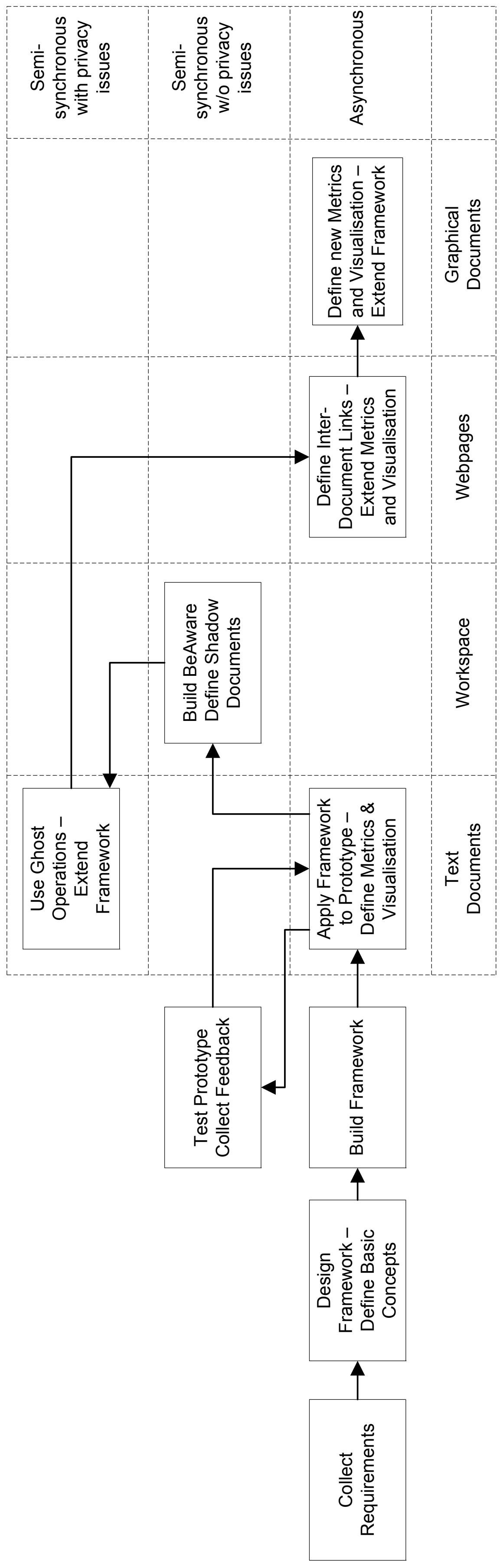

Collect Requirements
Design Framework – Define Basic Concepts
Build Framework
Test Prototype Collect Feedback
Apply Framework to Prototype – Define Metrics & Visualisation
Use Ghost Operations – Extend Framework
Build BeAware Define Shadow Documents
Define Inter-Document Links – Extend Metrics and Visualisation
Define new Metrics and Visualisation – Extend Framework
Text Documents
Workspace
Webpages
Graphical Documents
Semi-synchronous with privacy issues
Semi-synchronous w/o privacy issues
Asynchronous

Figure 1.3: Evolution of the awareness mechanism presented in the thesis

and the working modes of the collaborators. The awareness mechanism was described in terms of its key concepts and their relations, the combination of which resulted in the metamodel describing the mechanism. The definition of the main concepts was kept general to enable the applicability of the mechanism to various collaborative applications. An awareness framework, which materialised the mechanism's key concepts, was later developed.

To prove the generality of the framework, we tested its applicability to existing collaborative systems and the extensibility of its concepts to describe a large scope of applications. To test the applicability of the framework to existing collaborative authoring systems, we used it to enhance an asynchronous text editor. For this, the framework's basic concepts were extended to describe the editor's document model and its functionalities. The resulting prototype was tested through a user study and the users' feedback was collected. To test the generality of the framework's concepts we extended the framework for different collaborative situations in terms of the users' working modes and the types of co-authored documents. The collaborative applications on which we concentrated are shown in the table included in Figure 1.3. With each application we advanced in at least one of the table's two dimensions, i.e. the users' working mode and the documents' type.

We first concentrated on semi-synchronous collaboration over a set of documents. We then investigated the collaborative authoring of text documents and the applicability of our mechanism to semi-synchronous collaboration where privacy issues affect the detail of available information about a user's modifications. Finally, we studied the extensibility of our mechanism for asynchronous authoring of webpages and graphical documents.

In the rest of this section we present the structure of this thesis by giving an overview of the content of the chapters.

Chapter 2 discusses the related work of this thesis and its requirements in detail. Applications that enable people to collaboratively author documents are presented in this chapter. They are grouped into categories according to the collaborative activity that they intend to support and the document types that they handle. We look into applications with integrated awareness mechanisms and test their compliance to a set of requirements for multi-level awareness. An overview of all applications is presented at the end of the chapter, where their main drawbacks are highlighted and the need for a new solution that fulfils all requirements for multi-level awareness becomes apparent.

Chapter 3 presents our solution, i.e. the metamodel of our multi-level awareness framework. Our framework uses the underlying document structure of existing collaborative applications to enrich them with multi-level awareness information. We start the chapter with a discussion of why an underlying structured model is needed and how it can be used. Then, we give some background information about structured documents and how the basic concepts of a document node, an operation and a structured document are defined. We then introduce the concepts included in our metamodel, i.e. the "document", "node", "operation", "user", "operationValue", "nodeValue", "metric" and "visualisation tool", how they are defined and what are the relationships between them. The solution we propose, described by the metamodel, is intentionally kept general to enable the framework's application to various collaborative tools, independently of the type of the co-authored document and the working mode of the collaborators. The ease of the framework's applicability is shown in the next chapters.

In Chapter 4 we describe how we applied our awareness framework to an asynchronous collaborative text editor. We detail how the framework's concepts were materialised to describe the concepts included in the text editor. We propose a number of metrics to compute the required information and a flexible visualisation tool, called "edit profile" to present the computed information to the users. While this chapter addresses the specifics of the text editor used, it also shows the procedure that needs to be followed for the integration of any other editor.

In Chapter 5 we present the design and the results of a user study we conducted to test the functionality of the enhanced text editor we described in the previous chapter. In the first part of the study, we present the feedback collected from users testing the editor. This includes their evaluation of all the editor's functionalities and the way that users interpreted the presented information. In the second part, we present user requirements that we collected for the design of an asynchronous editor with synchronously updated awareness information.

The framework's extensibility with regard to various working modes is shown in Chapter 6 where, using the feedback from the user studies, we describe how an asynchronous editor was enhanced with real-time awareness. The notion of "shadow documents" is defined and used to ensure that users working in privacy are informed in real-time about modifications made from their collaborators. While this newly introduced concept ensures that users are aware of their collaborators' modifications, it also ensures that they continue working on their local copy undisrupted, because the remote modifications are not integrated in their local document copy. The editor implementing the concept of shadow documents is integrated in a shared workspace, the BeAware workspace, which is also presented in the same chapter.

While the BeAware workspace and its integrated editor successfully provide real-time awareness in situations where users are working privately, it does not consider any privacy issues. These issues are addressed in Chapter 7 where the notion of “ghost” operations and flexible filters that create the ghost operations from real operations are presented. The notion of ghost operations aims to enable users, working in asynchronous or semi-synchronous environments, to set their preferred level of privacy when they inform their collaborators in real-time about their modifications. The way that ghost operations are integrated in our awareness framework and a comparison of the concept of ghost operations and the notion of shadow document sets are also provided in the same chapter.

To prove the extensibility of our framework and its applicability to collaborative editors with document types other than text, Chapter 8 presents how the proposed framework could be used to enhance the awareness information provided by an editor for websites and a graphical editor. In a similar manner to that of Chapter 4, where the framework was integrated into a text editor, in this chapter, we extend the framework's basic concepts based on the specifics of the two editors. However, the structure of websites is not similar to that of conventional text documents because links between documents of the same level, i.e. between webpages, exist. Therefore, in this chapter a new extended version of the awareness framework is presented, where “html” and “transclusion” links between webpages are taken into consideration when computing and visualising multi-level awareness.

We summarise the results of the work conducted in the frame of this thesis and discuss about future work in Chapter 9.

2 Collaborative applications

We start this chapter by giving a set of requirements that a collaborative authoring application should fulfil in order for it to provide adequate change awareness information. We then present some of the best known collaborative applications grouped according to the collaborative activity they intend to support and the document types that they handle. For each application we discuss its main features and investigate whether the awareness offered by the systems satisfies the proposed requirements. The applications we present belong to one of the following domains: Document comparison tools (DCT), version control systems (VCS), collaborative authoring of text (CAT), graphical (CAG) and web documents (CAW), collaborative software development (CSD) and shared workspaces (CSW). Finally, we revisit the set of requirements and formulate our hypothesis as to how a general awareness framework for multi-level change awareness could be built.

2.1 Requirements for collaborative authoring tools

In this section we present a set of features that collaborative authoring tools should have to increase the awareness of users about the changes made by their collaborators. Following the discussion in the previous chapter, we specifically address the features needed to increase multi-level change awareness. The list of requirements presented here summarises the issues which were discussed in the user studies and the examples of collaborative situations presented in the previous chapter. This list is by no means exhaustive. However, we believe that it includes the most important elements that need to be considered.

The requirements we consider are:

- computation of awareness at different structural levels,
- computation of awareness relative to user roles and tasks, i.e. the detail of information handled by the application should be variable,
- visualisation of the computed information at different granularity levels including an overview of all changes throughout the authored document.

Finally, to allow for reusability of collaborative applications, two more requirements can be added to the list, namely,

- the computation and visualisation of awareness information independently of the mode of collaboration
- the computation and visualisation of awareness information independently of the document type.

2.2 Document comparison tools

When many people work on the same document without the use of collaborative editors, or when a person accesses a document from different computers, it often occurs that the users end up with different versions of the document. Inspecting the differences between two versions of a document is a very difficult process to perform manually. To solve that problem, many document comparison tools have been developed.

The procedure followed is that the user chooses the documents to be compared, and the system presents the documents with the differences highlighted. To collect information about the changes made to a document and compute the difference between the two document states, two different approaches have been developed – the state-based and operation-based approaches. In the first one, given two different states of a document, the application computes the difference between them. In the second approach, the actual changes that transformed a document from one state to another are tracked and later used to compute the difference. Most of the document comparison tools adopt the state-based-approach. In order to produce the most accurate "difference" between the two document states, many "diff" algorithms have been introduced for both linear [27] and hierarchical [19, 91] documents. The same approach is used also by version control systems as presented in Section 2.3.

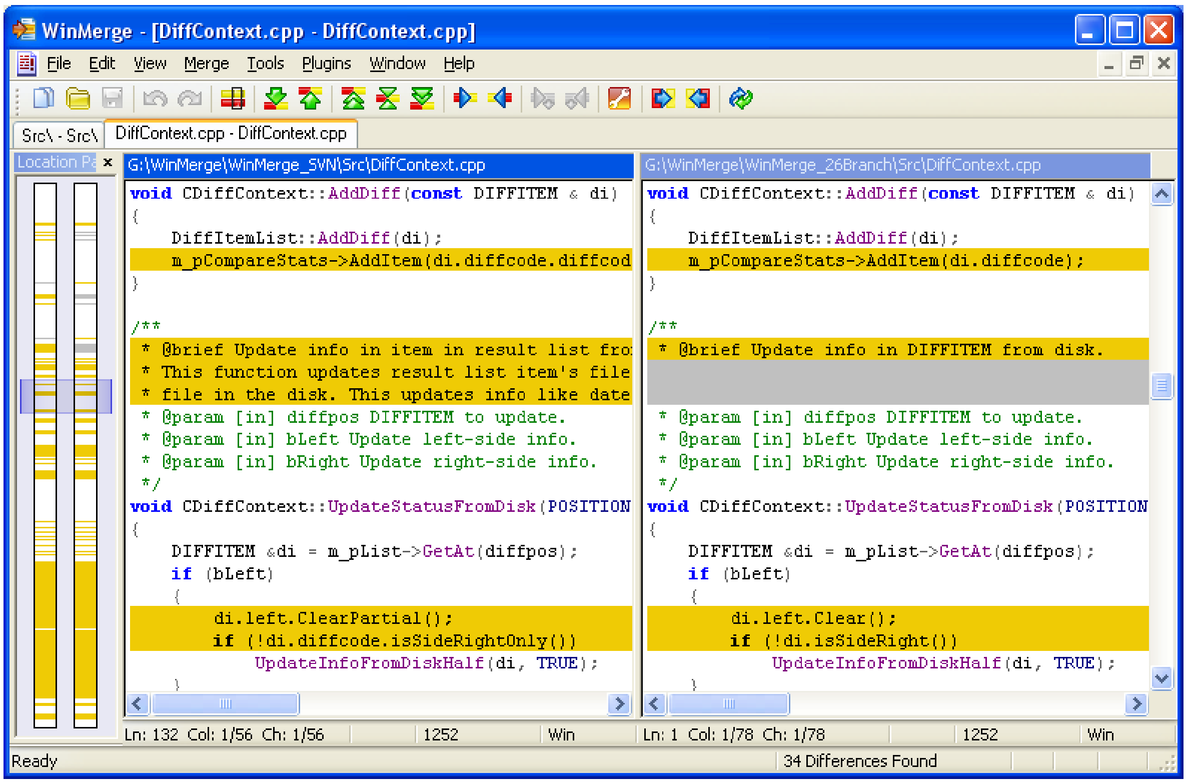
In the rest of this section we present in details three very popular comparison tools, WinMerge [9], DiffDoc [2] and DiffDog [3] .

WinMerge

WinMerge is an open source visual file differencing and merging tool for asynchronous collaboration. It handles text files and changes made to them including insertion, deletion or move of lines or parts of lines. The compared files can be visualised side-by-side with the differences highlighted as seen in Figure 2.1. Some features offered by WinMerge are the detection of the exact changes inside a line, the possibility to edit a document inside the application and a visualisation mechanism. In what follows we briefly discuss each of these in turn.

Detecting the exact part of a line that has been altered between two document versions is certainly helpful, since otherwise the user would need to manually scan the line and detect the changes. However, the amount of information delivered to the users is still very much restricted. Knowing that a line has been changed might, for instance, be useful for software developers, where each line of code has a meaning on its own. However this is not the case with other semantically richer documents, where for instance a concept or an idea may span many lines, or even paragraphs, sections, etc. In such documents, information about changes made to elements with higher syntactic value is more valuable. This applies even to lines of code. Although, syntactically, a line of code can exist by itself, most of the times it is interdependent with other lines to create elements of higher syntactic value such as methods and classes. To enable the support of multi-level awareness, i.e. to compute awareness information about document elements of different document levels, elements of a richer syntactic value should be taken into account when comparing two documents. Unfortunately this is a feature currently not supported by WinMerge.

When a user chooses to edit a document with WinMerge, the operations that transform the document from one version to another are generated by the tool. Unfortunately, the operations are discarded when


WinMerge - [DiffContext.cpp - DiffContext.cpp]
File Edit View Merge Tools Plugins Window Help
Src\ - Src\
DiffContext.cpp - DiffContext.cpp
Location Pa
Ready
34 Differences Found

G:\WinMerge\WinMerge_SVN\Src\DiffContext.cpp

```
void CDiffContext::AddDiff(const DIFFITEM & di)
{
    DiffItemList::AddDiff(di);
    m_pCompareStats->AddItem(di.diffcode.diffcod
}

/**
 * @brief Update info in item in result list fro
 * This function updates result list item's file
 * file in the disk. This updates info like date
 * @param [in] diffpos DIFFITEM to update.
 * @param [in] bLeft Update left-side info.
 * @param [in] bRight Update right-side info.
 */
void CDiffContext::UpdateStatusFromDisk(POSITION
{
    DIFFITEM &di = m_pList->GetAt(diffpos);
    if (bLeft)
    {
        di.left.ClearPartial();
        if (!di.diffcode.isSideRightOnly())
            UpdateInfoFromDiskHalf(di, TRUE);
    }
```

Ln: 132 Col: 1/56 Ch: 1/56 1252 Win

G:\WinMerge\WinMerge_26Branch\Src\DiffContext.cpp

```
void CDiffContext::AddDiff(const DIFFITEM & di)
{
    DiffItemList::AddDiff(di);
    m_pCompareStats->AddItem(di.diffcode);
}

/**
 * @brief Update info in DIFFITEM from disk.


 * @param [in] diffpos DIFFITEM to update.
 * @param [in] bLeft Update left-side info.
 * @param [in] bRight Update right-side info.
 */
void CDiffContext::UpdateStatusFromDisk(POSITION
{
    DIFFITEM &di = m_pList->GetAt(diffpos);
    if (bLeft)
    {
        di.left.Clear();
        if (!di.isSideRight())
            UpdateInfoFromDiskHalf(di, TRUE);
    }
```

Ln: 1 Col: 1/78 Ch: 1/78 1252 Win

Figure 2.1: Screenshot of WinMerge, a document comparison tool

the new version is created and are not used for the computation of the difference. Using the state-based approach instead of the operation-based-approach generates a difference that is computed based only on the two document states and can differ from the actual changes made to the document. Consider, for instance, a user that inserts some content into a document, but decides to delete it before committing his version. Computing the difference with a state-based approach will prevent their collaborators from being informed about that content. In a similar way, users comparing two non-continuous versions would not be informed about content that was inserted to the document in a version and deleted from the document at a later version. As a result, important information about the evolution of a document might be lost.

Finally, an important feature offered by WinMerge, is a bar serving as an overview of the changes applied to the document, indicating the places where the changes occurred. If a line is changed, the part of the visualisation tool representing this line is coloured accordingly. The bars can be seen on the left side of Figure 2.1. The users can click on the bar and navigate through the differences of the two documents. At the same time, they are informed about the parts of the documents where the differences occur. Although the information is presented only on the level of lines, this is a very interesting feature that allows the users to be instantly informed about all the changes made to a document and the places where they occur. The visualisation of this information on different document levels, as well as a mechanism that would compute the severity of each change are unfortunately missing. Currently, a one-character-change in a line is depicted in the same way as a change of a whole line.

DiffDoc

DiffDoc handles a richer set of documents, including MS Word and Excel, PDF, RTF, Text, HTML and XML documents. The level of the document elements checked for changes can be the word or the character. Thus, the granularity of the computation can vary according to the desired level of detail. However, any other higher document levels are missing. The reason for this is the fact that DiffDoc treats a document as an array of characters, with only the notions of word and line defined. The attempt from the developers to refer to document paragraphs, shows the need for a document representation via higher syntactic elements. However, the "paragraph" element is currently only a set of sentences written on the same "line" element that can span multiple lines in the interface. The approach of DiffDoc seems promising. An extension to include elements of various levels would be beneficial for the computation of multi-level awareness. Finally, a proper visualisation tool and a mechanism for the computation of the severity of changes based on user needs would also be beneficial.

DiffDog

DiffDog seems to be one of the most complete document comparison tools, with comparison and merging of directories and a more complete set of document types. It offers many comparison modes, including the "XML-aware" mode that compares XML documents taking into account their structure. The user can choose whether to include in the comparison comments, attributes, elements, etc. from the XML file. Simple text comparison is still offered for XML files if required. The differences can be seen in text or a grid view and there is support for XML editing and validation. Although DiffDog seems to be a fully functional editor and document comparison tool, it lacks a proper awareness mechanism. The comparison of documents seems to be based on comparison of lines where the mechanism can recognise or ignore, if needed, presentational and structural mark-up. However, no information is gathered, for instance, about the number of changes made to document elements, or the effect of these changes to elements of other document levels, etc. Finally, the existing visualisation tool neither gives an overview of changes throughout the document, nor reports changes at different document levels.

Summary of document comparison tools

A problem common to all of the above tools is that they implement the state-based-approach, only taking into account the initial and final states of the document and lose information about the process of transforming one state into another [101]. A drawback of this approach, apart from not supporting merging and conflict resolution in the way that operation-based approaches do [81], is that some of the changes made to a document might be omitted from the information offered by the tool as explained above.

Another disadvantage of document comparison tools is that there is almost no semantic comparison between the documents. Text is treated as an array of characters, without creating any semantic units in the documents. Even in cases where paragraph, sentence and word separators are specially treated so changes to them will not generate differences between the documents, the units are still based on lines and have no semantic value. This is also true in cases where words can be merged to generate a line and lines can be merged to generate a paragraph. The disadvantage arising from this approach is that no information is, for instance, returned to the user about differences detected at the paragraph level between the documents under comparison.

Another drawback of document comparison tools is the fact that most of them are used only to visualise the differences between documents and not to edit the documents. Even in the tools that offer editing features, the operations performed are not stored by the editor to be further used to compute the difference between the documents. The difference is recomputed, if needed, based only on the initial and final states.

A further disadvantage of document comparison tools is the complete absence of a mechanism that computes the effect of a change. The mechanism for handling and presenting a change of a character and a change of a line in a text document is exactly the same. Similarly, there is no distinction between a change of an element's attribute and an insertion of a new element in an XML document. Both are simply marked as changes.

Finally, most of the document comparison tools have neither a flexible visualisation tool to display information on the different document levels (which is expected since there are no syntactic document levels defined in the document models handled by the tools), nor an overview of changes made to the document to help users be instantly informed about changes throughout the whole document and identify heavily changed parts of the document.

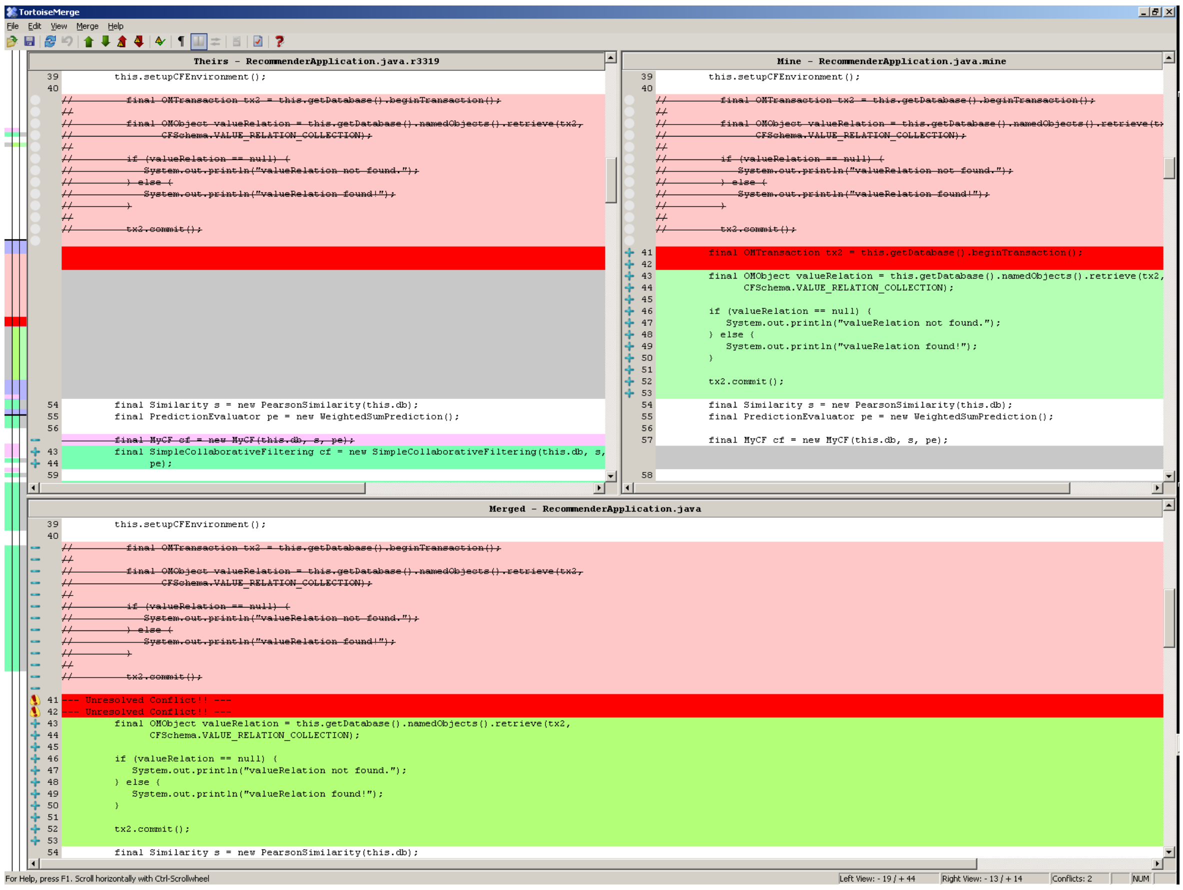
2.3 Version control systems

Version control systems are used for the management of multiple revisions of the same unit of information. They are most commonly used in engineering and software development to manage ongoing development of digital documents such as application source code and other projects that may be worked on by a team of people. Version control systems are used for asynchronous collaboration and they implement the copy-modify-merge paradigm. Users work in privacy on their local copy of, usually, the latest version of the co-authored unit. When they finish modifying it and decide to publish their work, they can commit their changes, which are identified as a new version of the unit. If multiple users modify the same unit in parallel, more than one version is created. These need to be merged to one resulting version that includes the modifications of all users.

Version control systems implement either the distributed or the client-server model. In the first approach, each user has a local repository of the co-authored units, and modifications are shared between the repositories. In the second approach, users share a common repository. They keep locally only a copy of a unit's version and changes made by all users are sent to the central repository. In the rest of this section, we will discuss the awareness mechanisms present in Concurrent Versions System (CVS) [27] and Subversion (SVN) [5], two of the best known version control systems that implement the client-server model.

CVS and SVN

CVS and SVN are version control systems that notify users about the status of a set of documents co-authored by many collaborators. SVN was built as a successor to the widely used CVS and includes a richer set of features. However, both systems provide to the users only basic information about the changes made by their collaborators. Therefore, we decided to introduce the two systems together and detail the awareness features that they provide.


TortoiseMerge
File Edit View Merge Help
Theirs - RecommenderApplication.java.r3319
Mine - RecommenderApplication.java.mine
Merged - RecommenderApplication.java
For Help, press F1. Scroll horizontally with Ctrl-Scrollwheel
Left View: - 19 / + 44
Right View: - 13 / + 14
Conflicts: 2
NUM

```
        this.setupCFEnvironment();

//          final OMTransaction tx2 = this.getDatabase().beginTransaction();
//
//          final OMObject valueRelation = this.getDatabase().namedObjects().retrieve(tx2,
//                  CFSchema.VALUE_RELATION_COLLECTION);
//
//          if (valueRelation == null) {
//             System.out.println("valueRelation not found.");
//          } else {
//             System.out.println("valueRelation found!");
//          }
//
//          tx2.commit();

        final Similarity s = new PearsonSimilarity(this.db);
        final PredictionEvaluator pe = new WeightedSumPrediction();

        final MyCF cf = new MyCF(this.db, s, pe);
        final SimpleCollaborativeFiltering cf = new SimpleCollaborativeFiltering(this.db, s,
            pe);
```

```
        this.setupCFEnvironment();

//          final OMTransaction tx2 = this.getDatabase().beginTransaction();
//
//          final OMObject valueRelation = this.getDatabase().namedObjects().retrieve(tx
//                  CFSchema.VALUE_RELATION_COLLECTION);
//
//          if (valueRelation == null) {
//             System.out.println("valueRelation not found.");
//          } else {
//             System.out.println("valueRelation found!");
//          }
//
//          tx2.commit();
        final OMTransaction tx2 = this.getDatabase().beginTransaction();

        final OMObject valueRelation = this.getDatabase().namedObjects().retrieve(tx2,
            CFSchema.VALUE_RELATION_COLLECTION);

        if (valueRelation == null) {
           System.out.println("valueRelation not found.");
        } else {
           System.out.println("valueRelation found!");
        }

        tx2.commit();

        final Similarity s = new PearsonSimilarity(this.db);
        final PredictionEvaluator pe = new WeightedSumPrediction();

        final MyCF cf = new MyCF(this.db, s, pe);
```

```
        this.setupCFEnvironment();

//          final OMTransaction tx2 = this.getDatabase().beginTransaction();
//
//          final OMObject valueRelation = this.getDatabase().namedObjects().retrieve(tx2,
//                  CFSchema.VALUE_RELATION_COLLECTION);
//
//          if (valueRelation == null) {
//             System.out.println("valueRelation not found.");
//          } else {
//             System.out.println("valueRelation found!");
//          }
//
//          tx2.commit();

--- Unresolved Conflict!! ---
--- Unresolved Conflict!! ---
        final OMObject valueRelation = this.getDatabase().namedObjects().retrieve(tx2,
            CFSchema.VALUE_RELATION_COLLECTION);

        if (valueRelation == null) {
           System.out.println("valueRelation not found.");
        } else {
           System.out.println("valueRelation found!");
        }

        tx2.commit();

        final Similarity s = new PearsonSimilarity(this.db);
```

Figure 2.2: Screenshot of the conflict editor of TortoiseSVN an interface of SVN

Both systems allow the users to choose two revisions of a file and display their changes. Conflicting modifications from more than one user on the same file are automatically detected and displayed through a visualisation tool, as shown in Figure 2.2. A colour code is used to display the different types of changes, i.e. insertions and deletions as well as the conflicts. The version from the repository, the local copy as well as the merged copy are visualised together as seen in Figure 2.2. An overview of all changes and conflicts throughout the whole file is given on the left hand side of the interface. It consists of three bars, one for each of the above mentioned versions. On the top, the local and remote versions are visualised and their differences are displayed by means of a colour code and the + / - signs in the beginning of each line of code.

Although version control systems offer many advanced functionalities in comparison to document comparison tools, they do not offer increased awareness information. The changes made to the documents concern lines rather than meaningful structural elements such as sentences in text, elements in XML, or methods in programs. Additionally, the lack of metrics and operations defined at the various syntactic document levels renders the applications incapable of computing and presenting awareness information at different levels of granularity as required by users based on their current role and situation.

2.4 Collaborative authoring of text documents

The list of collaborative applications for the authoring of text documents is by far greater than the one for document comparison tools, or version control systems. Since awareness has been identified as one of the key issues of collaborative applications, a considerable amount of research has been devoted to enriching, with awareness information, existing applications that support the collaborative authoring of text documents, or developing new applications that would help users maintain awareness of other users' activities.

Some of the early systems, concentrated mainly on providing support for workspace awareness in terms of the users that were active on a document during a certain period, where they focused and what they were doing. Telepointers [61], radar views [22, 57], fisheye views [54] and multi-user scrollbars [21, 22, 103] are examples of tools that concentrate on presenting information about where in a document users are concurrently working. Other examples of early collaborative systems are Quilt [43, 79], Grove [40, 41], ShrEdit [35, 83] and REDUCE [99, 111]. Unfortunately they offer no support for change awareness information. Therefore, we will not discuss these systems in detail and choose instead to present collaborative applications that provide richer mechanisms in terms of change awareness.

Edit Wear and Read Wear

Hill et al. introduced in 1992 the notion of "physical wear" as a metaphor for the editing activity of users in a collaborative environment [64]. The metaphor of a document that wears out as it is used was presented to describe the reading and editing activities of users. This concept was introduced for asynchronous collaborative editing of text documents. "Edit wear" recorded editing activity in document lines, while "read wear" recorded the time that a line was displayed on the user's screen. Edit wear and read wear aim to display an "edit-by-edit history of a document in progress".

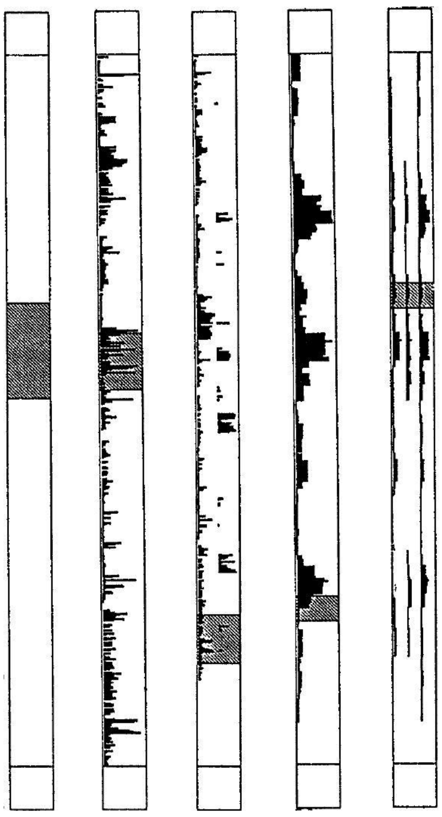


Figure 2.3: Edit Wear and Read Wear visualise editing and reading activity through multiple scroll bars with integrated wear marks

When document files are saved, the editing and reading activity recorded is also saved. Users are informed about the recorded activity through a flexible visualisation tool, which takes the form of a scroll bar with integrated "wear marks" as seen in Figure 2.3. Graphs, showing how often each line in the document has been read or edited by a user, are created and added to the editor scrollbar. From left to right, the scrollbars in Figure 2.3 show a normal scrollbar, a scrollbar with integrated edit wear activity of one user, the edit wear activity of two users, a scrollbar with the total read wear activity of all active users in a document and a scrollbar with the read wear activity of each one of the three active users. Note that the left part of the third scrollbar is a compressed version of the second scrollbar and the sum of the read activity of the three users in the last scrollbar results in the total read activity shown in the firth scrollbar. In this way, users are able to compare document sections to find the most stable ones, find the edits made during the last editing activity, who did them and where in the document.

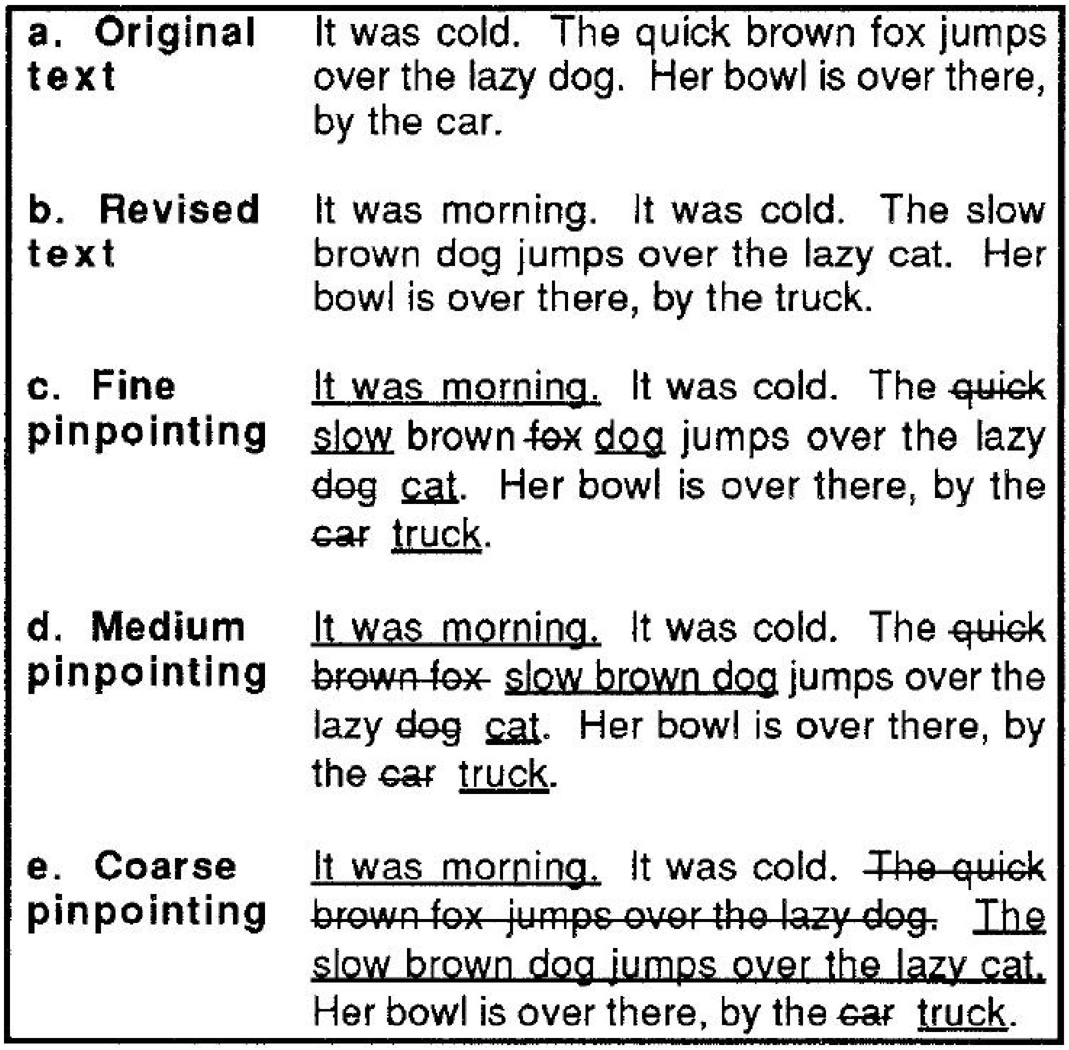
However, the above procedure is very much restricted by the assumption that a document is a set of lines. The tracking of activity as well as the information displayed through the visualisation tool is relative to document lines. The lack of a structured document model renders it impossible to deliver information relative to a syntactic document element, for instance a section or a paragraph. Finally, edits of elements of different structural level inside a line are treated in the same way. For instance, an edit of a character or an edit of a whole sentence in a line are both treated as an edit at a line. There is no mechanism that differentiates the above modifications or a mechanism that computes their severity. However, users have different needs based on their roles and the tasks assigned to them, which also determine the granularity of the modifications they want to be informed about. Therefore, the above mechanism fails to deliver adequate information to the users.

PREP Editor

The PREP Editor, or “work in preparation” editor, was introduced in 1990 as an asynchronous text editor that aims to help users during the co-authoring and commenting process. At the structural level, PREP Editor is more flexible than the wear metaphor, since it defines blocks of text, called “chunks”. Chunks roughly correspond to the notion of “ideas” used by Neuwirth et al. and contain text, grids, trees or arbitrary images [90]. The system also defines “links” between chunks so that networks of concepts can be built. Authors are allowed to build a “draft” of a document as a grid of chunks where the grid also includes annotations and guiding instructions on the content.

The system also provides an approach for communicating the changes made by users to their collaborators. The changes between two versions of the document are first computed via a “flexible diff” algorithm and then displayed through a visualisation mechanism.

The diff algorithm computes the differences on specific document levels. Various parameters such as “coarseness” and “maximum distance to look for commonalities” can be set by the user when pinpointing the changes [91]. By combining the parameters users can set the granularity of the changes that will be pinpointed by the diff algorithm. Figure 2.4c, for instance, shows the differences between the original and revised version of a document when users select to be informed about changes of words and require them to be reported as modifications on words. Figure 2.4d shows the differences, when modifications on words are tracked and reported as modifications of short phrases and Figure 2.4e shows how changes of words are reported as modifications of longer phrases. This could be a promising approach if applied on top of a structured document model. Currently, the user can set the above parameters to values that would return changes on phrases or words but no other syntactic elements are available. However, even in an extended version of this approach where more document levels are included, the flexibility of this approach seems to be heavily dependent on, and restricted by, the diff algorithm, since it is impossible to visualise information that is not computed during the differencing process. Combining this approach with a structured document and an operation-based diff algorithm would add much more flexibility.



a. Original text	It was cold. The quick brown fox jumps over the lazy dog. Her bowl is over there, by the car.
b. Revised text	It was morning. It was cold. The slow brown dog jumps over the lazy cat. Her bowl is over there, by the truck.
c. Fine pinpointing	It was morning. It was cold. The ~~quick~~ slow brown ~~fox~~ dog jumps over the lazy ~~dog~~ cat. Her bowl is over there, by the ~~car~~ truck.
d. Medium pinpointing	It was morning. It was cold. The ~~quick brown fox~~ slow brown dog jumps over the lazy ~~dog~~ cat. Her bowl is over there, by the ~~car~~ truck.
e. Coarse pinpointing	It was morning. It was cold. ~~The quick brown fox jumps over the lazy dog.~~ The slow brown dog jumps over the lazy cat. Her bowl is over there, by the ~~car~~ truck.

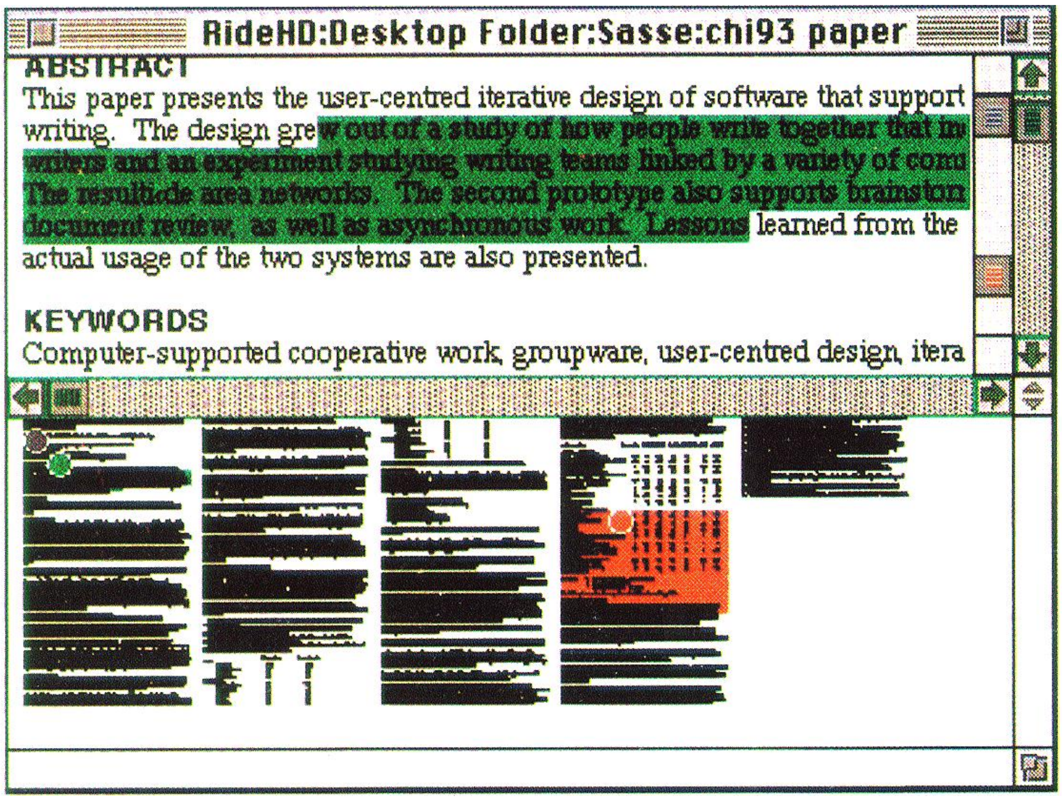
Figure 2.4: Different levels of pinpointing changes using the PREP Editor

SASE and SASSE

SASE was the first of two prototypes presented by Baecker et al. [22]. It is a synchronous text editor that provides support for "focused collaboration and independent work". Changes made by a user are synchronously integrated in the local copies of other users. Users can point to text using telepointers, while colour-coded selections indicate where people are working. Finally, multiple read-only scrollbars inform users about the focus of each user.

SASSE, an extension of SASE, provides support for additional activities, such as annotation, outlining and reviewing [22]. It also provides support for both synchronous and asynchronous collaboration. However, in the context of this thesis, we concentrate on two aspects of SASSE – the document model and the visualisation mechanism.

Apparently, SASSE treats documents as arrays of characters. The combination of this with a locking algorithm used to enable synchronous collaboration, resulted in unacceptable delays [22]. Unfortunately, the lack of a document structure restricts the reviewing and collaborating activities, since the user can only be informed about changes made in a range of characters.

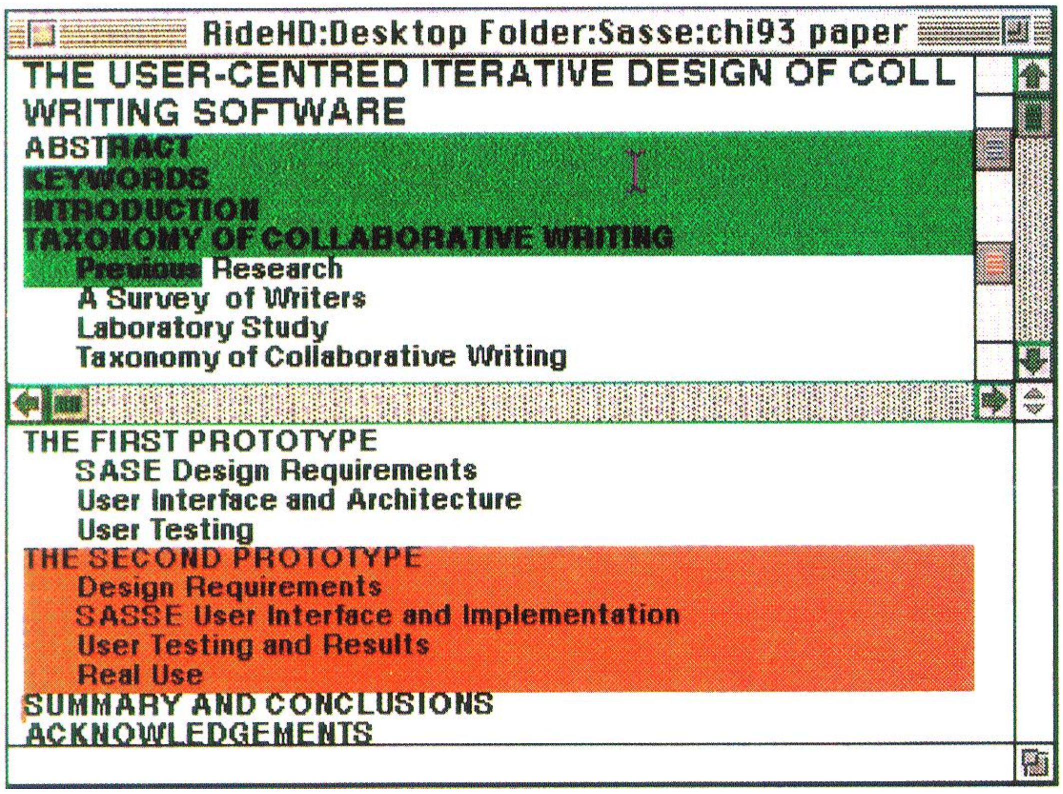


RideHD:Desktop Folder:Sasse:chi93 paper

ABSTRACT
This paper presents the user-centred iterative design of software that support writing. The design grew out of a study of how people write together that in writers and an experiment studying writing teams linked by a variety of comu The resultide area networks. The second prototype also supports brainston document review, as well as asynchronous work. Lessons learned from the actual usage of the two systems are also presented.

KEYWORDS
Computer-supported cooperative work, groupware, user-centred design, itera

Figure 2.5: The gestalt view of SASSE


RideHD:Desktop Folder:Sasse:chi93 paper

THE USER-CENTRED ITERATIVE DESIGN OF COLL
WRITING SOFTWARE
ABSTRACT
KEYWORDS
INTRODUCTION
TAXONOMY OF COLLABORATIVE WRITING
Previous Research
A Survey of Writers
Laboratory Study
Taxonomy of Collaborative Writing

THE FIRST PROTOTYPE
SASE Design Requirements
User Interface and Architecture
User Testing
THE SECOND PROTOTYPE
Design Requirements
SASSE User Interface and Implementation
User Testing and Results
Real Use
SUMMARY AND CONCLUSIONS
ACKNOWLEDGEMENTS

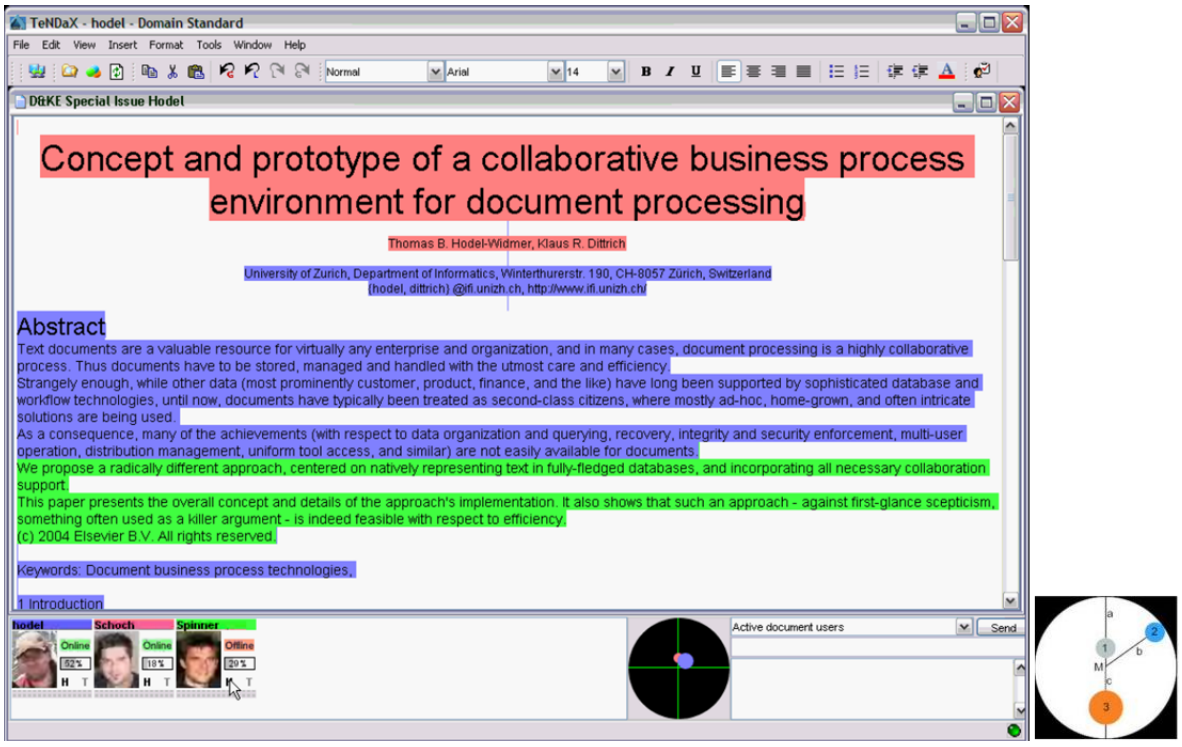
Figure 2.6: The observation view of SASSE

SASSE offers two views of user activities in a document. The first one, called “gestalt view”, presents a condensed image of the entire document as well as the positions of all collaborators and the part of the document currently selected by each of them. The position of users is mapped to small coloured circles, while the selection of text in the document also uses the colour code. In the example of Figure 2.5, there are three active users in the document, two of them active in the beginning of the document and one in the second half where they have also selected part of the text. The second view, called “observation view” allows users to “look over the shoulder” of a collaborator and see exactly what they are seeing and doing as presented in the lower part of Figure 2.6. The colour coded multi scrollbars are visible in the upper half of both figures.

The two views presented by SASSE are the two extremes of a document's spectrum. Information with regard to other levels between the existing ones would be very useful. However, the most important disadvantage of SASSE is that users can only be informed about the location of other users' changes. No information is provided about the type of changes, or their effect on the document.

TeNDaX

TeNDax, is a synchronous text editor supported by database technologies. The document is represented as a linked list of character objects that are stored in a database. Editing commands for operations of insertion and deletion of characters are mapped to database transactions. TeNDaX presents the changes made to a document superimposed on the text using a colour code that conveys information about the user who did the change. Additionally, it provides an overview of the document, the black circle at the lower half of Figure 2.7. In this tool, the edits of each user are mapped to a coloured circle. The colour conveys the identity of the user and the circle's diameter is related to the amount of edits that the specific user did. Small circles indicate few changes, while big circles indicate many changes. Finally, the positioning of the small circles in the circular document overview conveys information about the frequency of the changes (distance from centre) and the place in the document where the changes are made.



Concept and prototype of a collaborative business process environment for document processing

Thomas B. Hodel-Widmer, Klaus R. Dittrich

University of Zurich, Department of Informatics, Winterthurerstr. 190, CH-8057 Zürich, Switzerland
{hodel, dittrich} @ifi.unizh.ch, http://www.ifi.unizh.ch/

Abstract
Text documents are a valuable resource for virtually any enterprise and organization, and in many cases, document processing is a highly collaborative process. Thus documents have to be stored, managed and handled with the utmost care and efficiency.
Strangely enough, while other data (most prominently customer, product, finance, and the like) have long been supported by sophisticated database and workflow technologies, until now, documents have typically been treated as second-class citizens, where mostly ad-hoc, home-grown, and often intricate solutions are being used.
As a consequence, many of the achievements (with respect to data organization and querying, recovery, integrity and security enforcement, multi-user operation, distribution management, uniform tool access, and similar) are not easily available for documents.
We propose a radically different approach, centered on natively representing text in fully-fledged databases, and incorporating all necessary collaboration support.
This paper presents the overall concept and details of the approach's implementation. It also shows that such an approach - against first-glance scepticism, something often used as a killer argument - is indeed feasible with respect to efficiency.
(c) 2004 Elsevier B.V. All rights reserved.

Keywords: Document business process technologies,

1 Introduction

Figure 2.7: TeNDaX

The idea of blocks of text, called "zones" were added to TeNDaX to enable users to specify regions of interest in the document. They have a variable length of character sequences and can include any document part. Since this approach does not include any syntactic structural elements, the associated awareness information is also not available in different granularity levels. Finally, TeNDaX integrates a layout manager, which associates layout data with a document's content. If this approach that takes into consideration a document's structure, is also adopted by the mechanisms for computing and visualising awareness, and an overview of the changes throughout the document is added, TeNDaX could provide increased change awareness.

CoWord and CoPowerPoint

CoWord and CoPowerPoint are the collaborative versions of Microsoft Word and PowerPoint. They implement the transparent adaptation approach, by which existing single user applications can be transformed into collaborative applications[112, 121]. Both collaborative tools are well known to the CSCW community, because they successfully provide users with a rich pallet of modifications to be made to the authored documents and due to the large number of users already using the single user versions of the applications. Multi-user telepointers and radar views are used to enhance the applications by providing workspace awareness. However, no multi-level change awareness is yet provided.

Divergence metric

Molli et al. proposed a metric to measure divergence between copies of the same document [86]. By informing users about how their copies are diverging from each other, and presenting a measure of the conflicts that the changes will cause when published, it is expected to generate auto-coordination in a group working collaboratively.

This approach aims to enhance the collaboration of users in multi-synchronous collaboration [85], where people work either online or offline on shared objects. A user's local copy is continuously compared with the copies that their collaborators keep in their local workspaces and with the latest committed document version. When two versions are found with changes from two different users which, if committed, will generate conflicts, users are informed about the (potential) conflicting situation. Since changes made on a document alter its size, and conflicts are marked as a set of conflicting changes, the divergence is expressed in number of bytes.

Although this approach seems to be promising, the unit for computing the divergence is the document, rather than structural units of the document. Therefore, it is not possible to provide users with a detailed view of the modifications performed on document parts. Additionally, users may be erroneously informed about conflicts. Since changes are defined on a document level, two changes modifying different document parts, and therefore not creating any conflicts, will be translated as two edits on the same document and therefore as conflicting changes.

2.5 Collaborative authoring of graphical documents

Graphical documents are also often edited collaboratively. Although the content of the documents differs from that of text documents, the collaborative activity is the same, in terms of need for awareness. However, there are far fewer collaborative graphical authoring systems available. Additionally, the research conducted until now in this area is far less detailed than in the case of text editors. Most of the existing systems offer only basic support for change awareness and do not satisfy most of the requirements presented at the beginning of this chapter. However, we decided to present some of the existing change awareness approaches in the domain of graphical editing to offer a complete overview of collaborative systems.

MS Word

Microsoft Word [4] offers rich awareness support for text editing, while awareness about modifications of images is very limited. The model used for the representation of documents is a linear one, i.e. the document is seen as a sequence of characters and images. Although characters inserted or deleted are tracked, modifications of images are unfortunately ignored.

Rational Rose

Rational Rose [15], from IBM, provides a UML editor for software engineering design and is an example of a two dimensional scene of objects. Even though this UML editor is a two dimensional application, changes performed on the design are not marked on the graphical view. Objects are displayed within a hierarchical class browser, and items that have changed are marked by special symbols positioned next to the items. A text representation is also available showing the differences between the class hierarchies of two versions of the design.

Chimera

Another mechanism for change awareness is the replaying of changes over time by showing the actions that represent the evolution of the document between two states such as in the Chimera system [75, 76]. The system's visualisation includes a sequence of panels, each of which illustrates an important moment in a story. The result is an editable graphical history that is generated automatically as the user interacts with the graphical editor. For instance, in Figure 2.8 the evolution of a document from its creation (upper-left rectangle) to its current status (lower-right rectangle) is shown. Unfortunately, such a mechanism is time-inefficient for users, since it requires them to go through the entire sequence of panels and search for the information they need. It also lacks an overall presentation of the document modifications.

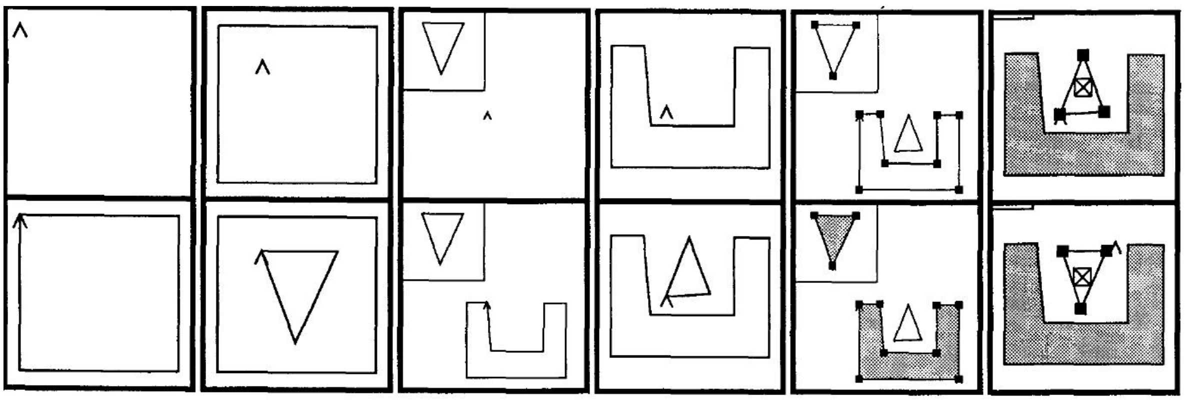


Figure 2.8: The graphical history by chimera presents a document's evolution between two different states

Bitmap-based graphics

In [80], a multi-level coding method has been proposed as an awareness mechanism. The approach is applicable to collaborative authoring of bitmap-based graphics where different collaborators can receive different data streams in different granularities based on their network connectivity and bandwidth. An "awareness strength" function is defined between each pair of users to return a value reflecting how strongly users may be aware of each other. Each editing operation is encoded into n granularity levels. For instance, the levels to encode a graphics editing operation might be the text representation of the operation, a simple outline of the operation effects in black/white colour and degrading colour intensities of the operation effects. Depending on the awareness strength between a user and their collaborators, an operation performed by that user is sent to the collaborators in the corresponding encoded form.

While the idea of awareness in multiple granularity levels in this system is appealing, it originates from a completely different problem of collaborative work, user needs in terms of network availability, rather than user needs in terms of user roles and current task. This approach neither returns information about the severity of their collaborators' changes nor provides an overview of changes throughout a document.

PastDraw

PastDraw [114, 115] is a two dimensional structured drawing application for asynchronous collaboration. It is augmented with various techniques for displaying change awareness information about a scene of objects: animated replays, storyboards, iconic displays and documentation methods. Animated replays employ playback animations of changes referring to a certain part of the scene of objects. The storyboard technique illustrates the changes that took place by capturing them with a series of still frames (similar to the Chimera system), while the iconic display technique consists of attaching various icons corresponding to different types of changes as shown in Figure 2.9. A colour code is used to denote whether (object coloured in red) and how much (intensity of red colour) an object has changed. Additional information about the type of the change and the user that initiated the change is given in a text-callout if a user clicks on the modified object as shown in Figure 2.9 for the "Client" class. Finally, the documentation technique describes textually the changes that took place. PastDraw offers an overview intended to communicate at a glance all of the changes made to a graphical document. In this approach, various colour intensities are associated with objects to illustrate the number of changes.

However, this method was applied only to documents where objects do not have a background colour and could not be used for a general approach where objects have an associated colour. Furthermore, this approach is not suitable for large-sized documents. The scene of objects would need to be mapped to an overview and objects would have a very small size. Additionally, the intensity of an object's colour would no longer be an efficient way to convey information about the number of changes made to objects. Moreover, the difference in intensity of the colour of two objects is not always easily distinguishable, especially when the objects are not close to each other in the scene of objects. Finally, Past-

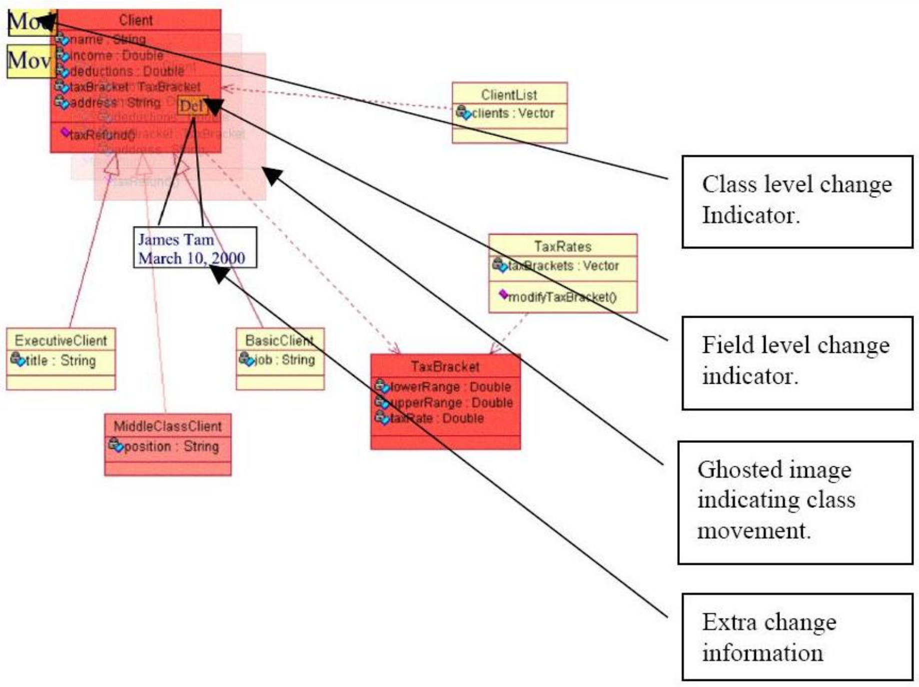


Figure 2.9: Iconic representations of changes in PastDraw

Draw does not consider object groups or any other inheritance hierarchy that may exist in a graphical document, for instance layers. Therefore, this approach can not be applied to an editor that handles documents with more complex structures and cannot capture and display information on different granularity levels.

Summary of collaborative graphical tools

Summarising the above, we see that current collaborative graphical editors offer far less functionality and detailed change awareness information than text editors. However, the need of users for detailed information regarding the changes made to a document is independent of the document type.

Undoubtedly, an overview of the changes made to a document is required in graphical editors as in text editors. However, as the above examples show, this is not enough. While an overview could help users be instantly informed about changes made throughout the document, it can also be easily overloaded with information and therefore become unusable. Only if the proper amount of information is presented through an overview, will this visualisation be useful to the users. To do so, proper mechanisms should exist to “filter” the available information and show only the part currently needed by the user. A way to do this would be to visualise the available awareness information at different levels of detail, for instance, visualise part of the document, or visualise separately objects that belong on different layers or group of objects, etc., all of them corresponding to levels that are related with the document's underlying structure. Finally, the available information should also be filtered based on the severity of the changes and their effect on the document at its different levels.

2.6 Collaborative authoring of web documents

Until recently, the World Wide Web was used mainly as a read-only medium where authors publish to many readers. Now the trend is towards multiple interacting authors as seen in the collaborative authoring of websites and webpages. Authors often perform their tasks in a distributed way, with regard to time and space, which can make the coordination of their work difficult.

Changes to a webpage may affect other parts of the page or pages linked to it and hence lead to semantic inconsistencies within or between pages. Keeping users aware of changes made by other users can help to resolve such inconsistencies or even prevent them. While many popular tools for the collaborative authoring of webpages such as Wikis, blogs and WebDAV applications succeed in enabling collaboration between users, they unfortunately provide little awareness to users about the changes made by their collaborators, and therefore do not prevent semantic inconsistencies. In this section we present in detail some systems that were built to inform users when a webpage is edited, i.e. notification tools, as well as more advanced systems that provide information about a document's modifications between different document versions.

Wiki systems

Two of the definitions given for a wiki are the following. The Oxford English Dictionary [12] defines wiki as:

> *"A type of webpage designed so that its content can be edited by anyone who accesses it, using a simplified markup language."*

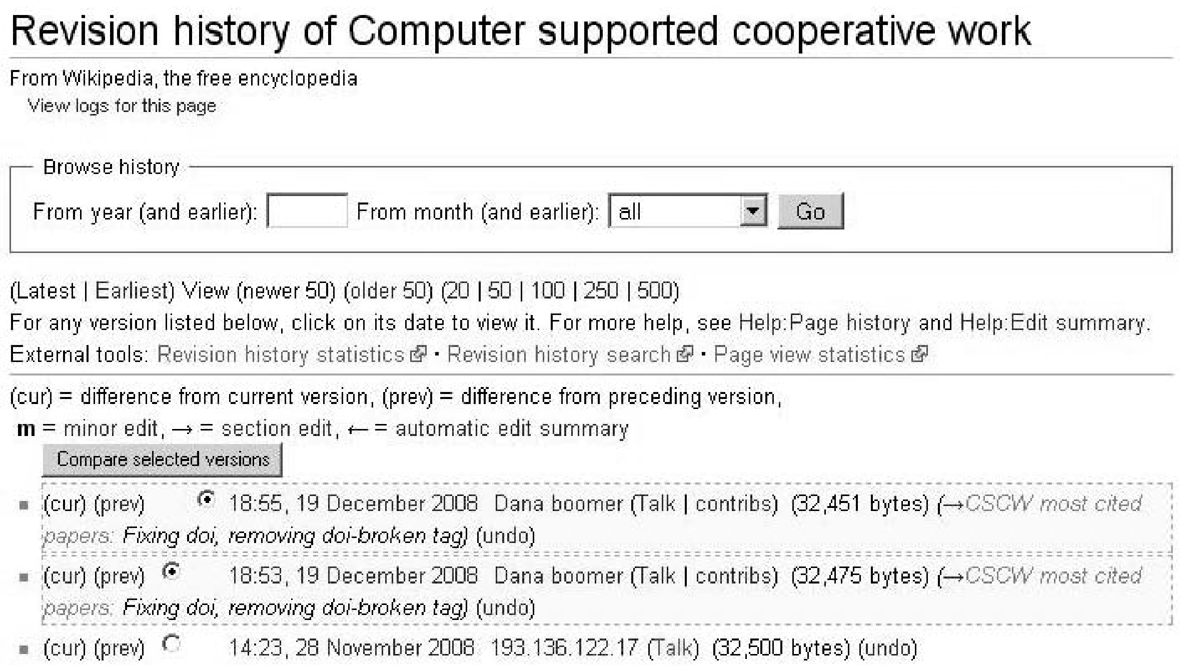
Encyclopedia Britannica [10] defines wiki as:

> *"World Wide Web (WWW) site that can be modified or contributed to by users. Wikis can be dated to 1995, when American computer programmer Ward Cunningham created a new collaborative technology for organizing information on Web sites."*

Wiki pages or wiki sites are essentially databases of knowledge where authorised users can search, insert or edit their contents. For instance, wikis are very often used by companies as a replacement of intranets to easily manage and make any information among colleagues widely available. They are often also used for management of projects, because they enable the project members to easily access, edit and share any project-related information. However, the most common use of wikis is on the Internet, where wikis are available to a larger set of users that can contribute to the authoring of a webpage. A web browser is the only requirement for users to access a wiki and editing its content is fairly easy. Most of the wikis use a markup language to format the content and link pages to each other. The possibility of linking pages is a useful feature of wikis, since it enables users to also navigate between semantically related pages. WikiWikiWeb [18], written by Ward Cunningham, is the first wiki application to have been written, but Wikipedia [8] is one of the most well-known and widely used wiki sites.

In terms of change awareness, Wikipedia provides users with the possibility of tracking changes performed in a wiki page by means of a revision history attached to the page. It consists of the old versions and a record of the date and time when each version was created, as well as the user who created it as shown in Figure 2.10. The users can then choose to see a specific revision, or can choose two revisions and be informed about the differences between them.

Upon selection of two revisions, a diff algorithm is used to calculate the differences between them. This means that a state-based-approach is used to compute the differences, and part of the actual modifications might be ignored. The differences are shown as indicated in Figure 2.11. The old version is presented on the left side and the new on the right. Inserted and deleted text is highlighted in red. The user is required to scroll through the document to be informed about all modifications. Although all modifications are depicted through this visualisation, unfortunately there is almost no means of quantifying them. As shown in Figure 2.10 the only information offered to users about the number and the severity of modifications is a number representing the length of each


Revision history of Computer supported cooperative work

From Wikipedia, the free encyclopedia

View logs for this page

Browse history

From year (and earlier): From month (and earlier): all Go

(Latest | Earliest) View (newer 50) (older 50) (20 | 50 | 100 | 250 | 500)

For any version listed below, click on its date to view it. For more help, see Help:Page history and Help:Edit summary.

External tools: Revision history statistics · Revision history search · Page view statistics

(cur) = difference from current version, (prev) = difference from preceding version,

m = minor edit, → = section edit, ← = automatic edit summary

Compare selected versions

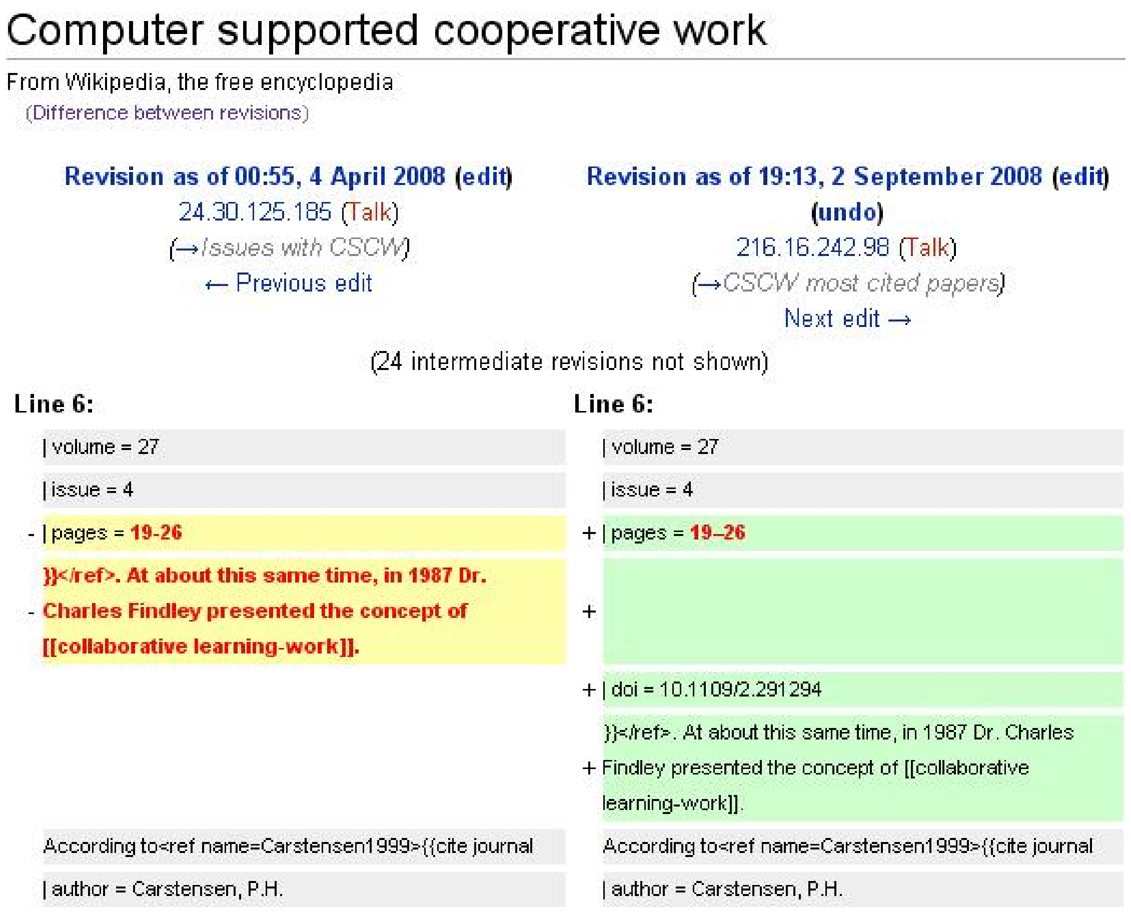
- (cur) (prev) 18:55, 19 December 2008 Dana boomer (Talk | contribs) (32,451 bytes) (→*CSCW most cited papers: Fixing doi, removing doi-broken tag*) (undo)
- (cur) (prev) 18:53, 19 December 2008 Dana boomer (Talk | contribs) (32,475 bytes) (→*CSCW most cited papers: Fixing doi, removing doi-broken tag*) (undo)
- (cur) (prev) 14:23, 28 November 2008 193.136.122.17 (Talk) (32,500 bytes) (undo)

Figure 2.10: Revision history of the Wikipedia webpage on the term CSCW

page in number of bytes for every revision. By comparing the length of the revisions, the user is expected to have an estimation of the severity of modifications. However, as a webpage evolves through the different revisions, the length of a document can not always provide enough information about the modifications between two non-successive revisions. For instance, the fact that revision number 3 has a similar length to revision number 10, does not necessarily mean that there were very small modifications in the in-between revisions. There might have been a large amount of content inserted, and an equal quantity deleted.

Wikipedia tags small changes, like corrections of spellings, as “minor” changes. It also specially tags revisions that include changes to only one section, which corresponds to the level of the paragraph for usual text documents. However, no other information is collected about the exact document parts where modifications are made, or the importance of the changes based on the syntactic level of the modified elements and their importance in the webpage. Moreover, no overview of the modifications throughout the page is given. The users are required to search for modifications inside the document.

Wikipedia also offers the “watch pages” feature by which users receive email notifications when specified pages change. The notification mechanism provides, for each page, the number of added or deleted bytes



Computer supported cooperative work

From Wikipedia, the free encyclopedia
(Difference between revisions)

Revision as of 00:55, 4 April 2008 (edit) 24.30.125.185 (Talk) (→Issues with CSCW) ← Previous edit	Revision as of 19:13, 2 September 2008 (edit) (undo) 216.16.242.98 (Talk) (→CSCW most cited papers) Next edit →

(24 intermediate revisions not shown)

Line 6:	Line 6:
| volume = 27	| volume = 27
| issue = 4	| issue = 4
- | pages = **19-26**	+ | pages = **19–26**
- **}}</ref>. At about this same time, in 1987 Dr. Charles Findley presented the concept of [[collaborative learning-work]].**	+
	+ | doi = 10.1109/2.291294
	+ }}</ref>. At about this same time, in 1987 Dr. Charles Findley presented the concept of [[collaborative learning-work]].
According to<ref name=Carstensen1999>{{cite journal	According to<ref name=Carstensen1999>{{cite journal
| author = Carstensen, P.H.	| author = Carstensen, P.H.

Figure 2.11: Difference between two revisions of the Wikipedia webpage on the term CSCW

to the page. This notification algorithm, however, does not provide detailed information about the exact location and the importance of the modifications. Additionally, it does not provide any information at all about modification made on webpages that are related to the "watched" pages. As described before, the possibility of navigation and linking between pages of a wiki is a very important feature. The content of a webpage is very often related to that of the linked pages. For instance, when a concept is defined in page A using other terms that are described in other pages, for instance page B. As a result, a modification in page B might also affect semantically the content of page A. Unfortunately, a Wikipedia user would not be informed about such situations.

History Flow Visualisation

Although the idea of presenting an overview of a project's evolution has not yet been adopted by Wikipedia, overviews are present in other systems. By analysing differences between multiple revisions of a webpage, history flow visualisations [117] provide an overview of the page's evolution. They provide information about how a group has contributed to a webpage in terms of the amount of text that was altered. The visualisation mechanism shown in Figure 2.12 works as follows. Every revision of a webpage is represented by a "version" vertical line. The length of the line corresponds to the length of the page and the colour of the line corresponds to the user that added the page, or the page part. In Figure 2.12 user Mary created the first revision of the webpage, and users Andrew and Suzanne made modifications to it by inserting and deleting document parts. The amount of modifications can be extracted by the relative length of the document subparts to the document length.

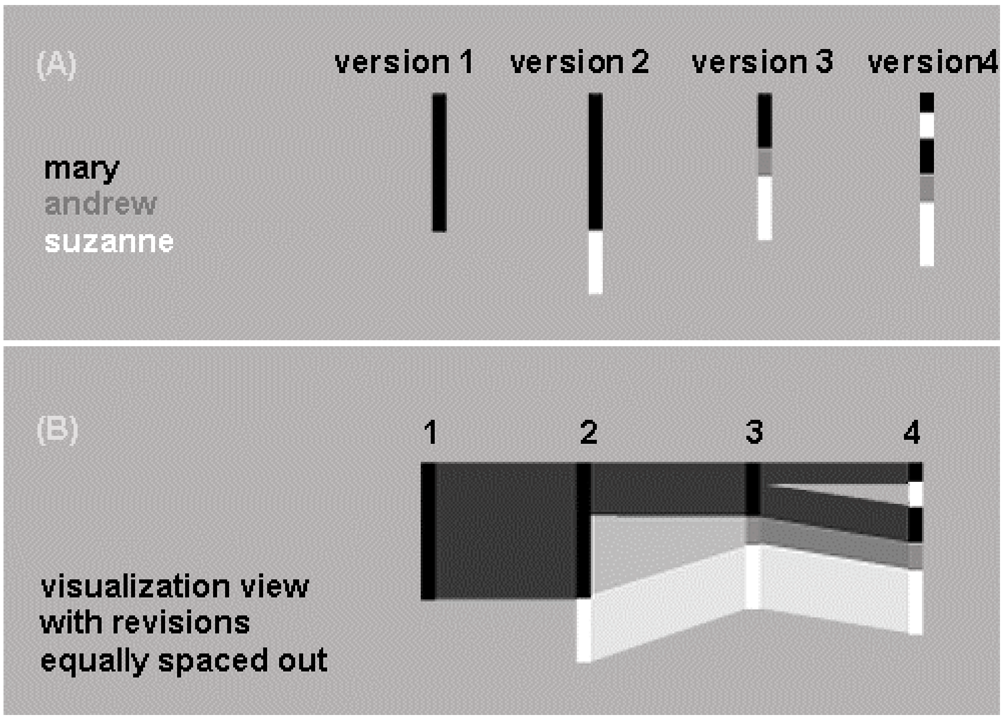


Figure 2.12: History flow visualisation for the modifications of three users over four versions of a document

Unfortunately, the granularity of modifications that are extracted is the sentence level, where sentence is defined as "pieces of text delimited by periods or html tags" [117]. Although this granularity of pinpointing modifications is more flexible than the one provided by Wikipedia, which is the paragraph level, it is still not flexible enough. The fact that only a specific level is provided and that there is no possibility for the users to be informed in different ways and on different granularity levels, depending on their preferences and current needs, is very restrictive.

Watch That Page, ChangeDetect, Website Watcher

Notification tools, such as Watch That Page [6], ChangeDetect [1] and Website Watcher [7], exist to track changes performed on specified webpages. These tools require the user to set up a list of "interesting pages" and some personal preferences as to the detail and frequency of the updates they will receive. The tools monitor the selected pages and notify users, usually through emails, about the modified pages and the actual modifications. Web notification tools are based on the same principles as the document comparison tools presented at the beginning of this chapter. As a result, they also have the same disadvantages concerning the detailed change awareness they provide to the users. The compared documents or webpages are considered to be a list of characters. No structure is identified in the documents, making it impossible to monitor modifications on document parts, or to monitor modifications that altered document parts of specific syntactic levels. Moreover, there is no mechanism that provides information about the severity of modifications on a document or classifies modifications according to their importance. Finally, the state-based-approach is employed here as well, which might lead to loss of some modifications and therefore decreased level of awareness.

Concerning the links between webpages, some notification tools, for instance Website Watcher [7], go one step further and allow the possibility to automatically add the pages linked to the main webpage to the list of monitored pages. Users are then notified if changes are performed on one of the monitored pages including the linked ones. However, no information is provided about the relation between the monitored pages, the severity of modifications in the linked pages and how these changes might influence the changes in the main webpage.

Notification mechanism for bitmap-modelled webpages

In [52], a notification mechanism is proposed for tracking changes performed on webpages. Users can mark regions of interest on a webpage that are stored as bitmaps. The system periodically checks if the selected regions visually differ from the stored ones with a certain severity index fixed by the user. If a region of interest is modified, the user is notified by the system. This approach is image-based and assumes the webpage retains its spatial layout. The system consists of a "notification editor" and a "notification viewer". The first provides users with an environment where users select interesting 2D areas inside webpages and specify the layout of the notification they would like to receive, while the second is the environment in which modified document parts can be visualised.

Summary of collaborative systems for websites

Summarising, there are not many available collaborative web applications in comparison to other collaborative applications. With the exception of some wiki systems, web applications provide almost no change awareness to users since they only inform users whether a webpage has been modified or not. No detailed awareness information is computed by them. Some wiki systems provide change awareness by means of document overviews, or modification logs, but, unfortunately, they consider webpages only as a set of lines and almost ignore the underlying document structure. Finally, none of the above systems provide detailed information about modifications made on pages linked to other pages and how the changes of one may affect the rest.

2.7 Collaborative development of software

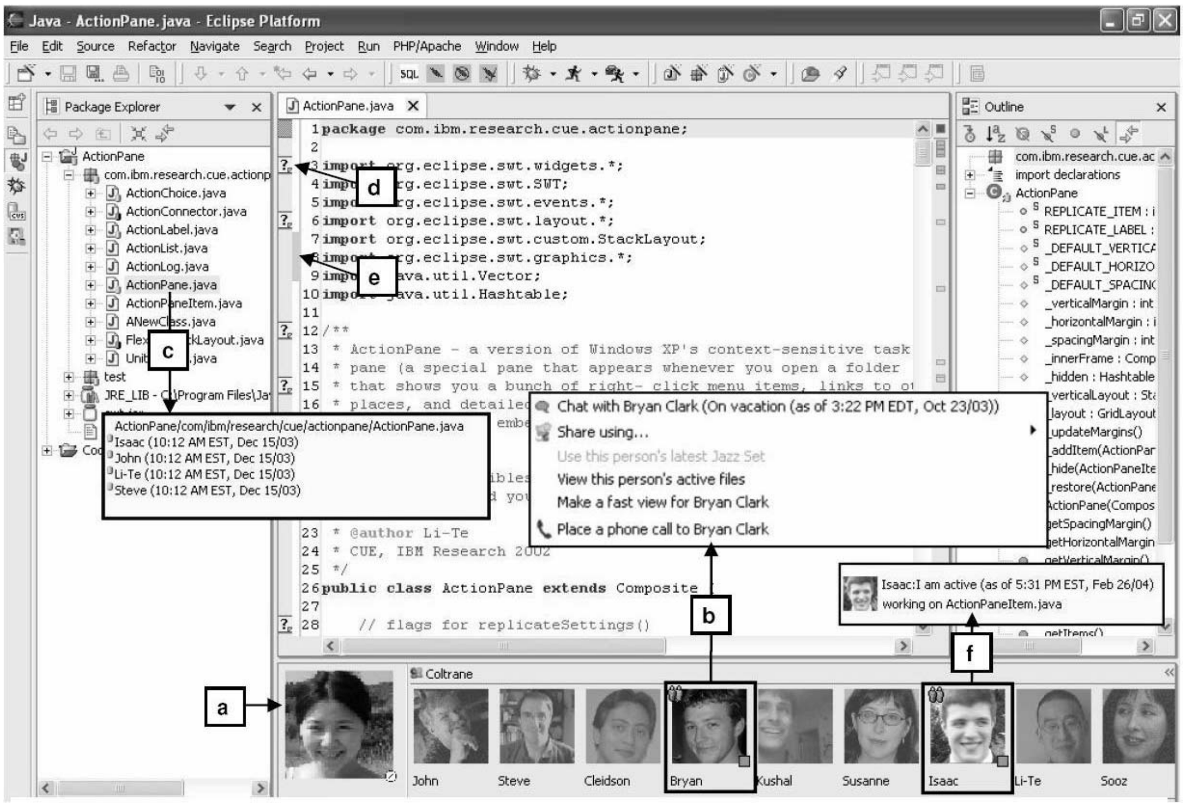
Software development is an area with collaborative tasks where large groups of people are involved. The collaboration may span a large period of time and a large set of documents. People involved in such projects usually work asynchronously to ensure that the project will always compile without errors. Collaborating over such a large set of documents with strong dependencies in code inside or between documents requires a considerable effort for coordination and awareness. Being aware of changes made by other users to any of the documents involved is a difficult task if done manually.

Flexible awareness mechanisms are needed, to collect and compute the required information on a granularity level depending on the user needs and current task to be accomplished. Additionally, visualisation mechanisms are required to show the current status of the project and the editing activity of people over time. These views of the project should have a variable level of detail.

Various research groups in the CSCW domain acknowledge the need for awareness in software development. Their research resulted in the development of various awareness mechanisms. Some of them are Jazz [31, 66], Seesoft [38], Tukan [107, 108], Advizor [37], Polymetric Views [78], Spyware [102], CollabVS [34], Xia/Creole [120], Palantìr [105], Soft-Change [48, 49], Augur [45], Spectograph [119] and Evolution Matrix [77]. In this section, we describe some of the most popular tools.

Jazz

Jazz is a collaborative development environment "designed to support small, informal teams; anyone can create a team and add or remove members" [66]. Jazz is implemented as an extension of the Eclipse Java Development Tools (JDT) [11]. The system extracts activities from the environment's user interface and the local history to monitor how the source code is manipulated and to provide a useful knowledge base and context for communication. Several features are available in Jazz to support awareness of team member activities in addition to screen sharing.



```
package com.ibm.research.cue.actionpane;

import org.eclipse.swt.widgets.*;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.custom.StackLayout;
import org.eclipse.swt.graphics.*;
import java.util.Vector;
import java.util.Hashtable;

/**
 * ActionPane - a version of Windows XP's context-sensitive task
 * pane (a special pane that appears whenever you open a folder
 * that shows you a bunch of right- click menu items, links to ot
 * places, and detailed
 ...
 * @author Li-Te
 * CUE, IBM Research 2002
 */
public class ActionPane extends Composite {

    // flags for replicateSettings()
```

ActionPane/com/ibm/research/cue/actionpane/ActionPane.java
Isaac (10:12 AM EST, Dec 15/03)
John (10:12 AM EST, Dec 15/03)
Li-Te (10:12 AM EST, Dec 15/03)
Steve (10:12 AM EST, Dec 15/03)

Isaac:I am active (as of 5:31 PM EST, Feb 26/04) working on ActionPaneItem.java

Java - ActionPane.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run PHP/Apache Window Help
Package Explorer
ActionPane
com.ibm.research.cue.actionp
ActionChoice.java
ActionConnector.java
ActionLabel.java
ActionList.java
ActionLog.java
ActionPane.java
ActionPaneItem.java
ANewClass.java
FlexStackLayout.java
UnitTest.java
test
JRE_LIB - C:\Program Files\Ja
ActionPane.java
Outline
com.ibm.research.cue.ac
import declarations
ActionPane
Chat with Bryan Clark (On vacation (as of 3:22 PM EDT, Oct 23/03))
Share using...
Use this person's latest Jazz Set
View this person's active files
Make a fast view for Bryan Clark
Place a phone call to Bryan Clark
Coltrane
John
Steve
Cleidson
Bryan
Kushal
Susanne
Isaac
Li-Te
Sooz
a
b
c
d
e
f

Figure 2.13: Jazz Band, the key visualisation tool of Jazz

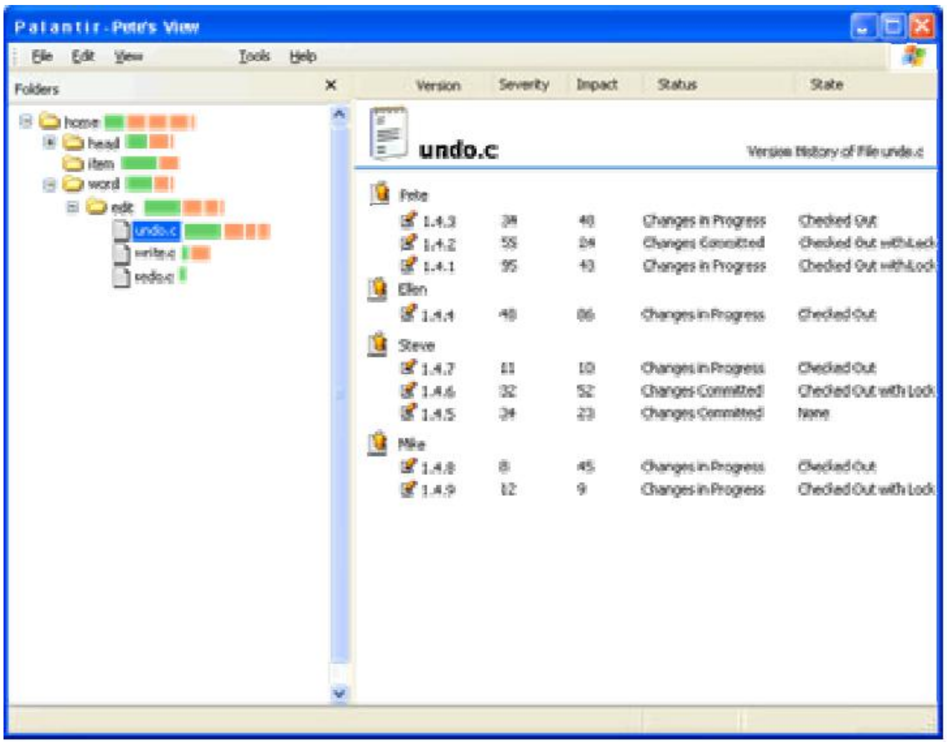
The key visual feature of Jazz is the "Jazz band", shown in Figure 2.13, which provides peripheral awareness of the status and activities of other team members. Members are represented by images, decorated by status icons at the bottom right indicating whether the person is online, away, or busy (a). A menu offering communication options with users (b) and their status message is also offered (f). Files in the explorer are enhanced using colour and icons to show the status of the resources (what are users currently doing with the files). Hovering over the resource brings up a tooltip displaying who is responsible for the changes (c). Finally, a chat can be initiated and stored for a specific part of the code (d) and a modifications' indicator (e) for a piece of code that was modified in the local version of a remote user offers a view of the remote copy of the code. Summarising, Jazz offers advanced mechanisms for workspace and presence awareness, but information about change awareness is restricted to whether an artefact was modified and by which users. No details about the modifications are available to the users.

Palantìr

Palantìr [105] provides awareness information about concurrent modifications performed in isolation in the context of configuration management systems. It extracts information from version control tools and computes the difference between files by comparing the number of lines changed. Palantìr captures a number of events, for instance, "added", "removed", "changes in progress", "changes committed", etc. The effect of the changes is computed and presented to the users by means of two metrics – the severity and impact metrics. The severity metric measures how much a component in a user's workspace has changed when compared to its latest checked version in the repository, or its version on the collaborators' workspaces. It can be binary, indicating that a change of any kind occurred, or a number indicating the percentage of lines of code that were modified in any way. The impact measure takes into consideration the code dependencies and computes how much a component in a local workspace is affected by changes to related components in remote workspaces.

Palantìr offers different ways of visualising the computed changes, for instance the tabular, the explorer and the fully graphical visualisation. The explorer visualisation is shown in Figure 2.14. On the left hand side is an expandable tree view. Colour annotations on file names are used to show the user currently editing the files. The tree view is enhanced with horizontal bars indicating the severity of ongoing and committed changes. The longer the bar, the higher the severity of the changes.

Another view is the hierarchical view of Figure 2.15, which shows a view of a user's workspace and the artefacts included in every artefact. For every artefact, the local user is informed of which other users have a



undo.c Version History of File undo.c

	Version	Severity	Impact	Status	State
Pete					
	1.4.3	39	40	Changes in Progress	Checked Out
	1.4.2	55	24	Changes Committed	Checked Out withLock
	1.4.1	95	43	Changes in Progress	Checked Out withLock
Ellen					
	1.4.4	48	86	Changes in Progress	Checked Out
Steve					
	1.4.7	11	10	Changes in Progress	Checked Out
	1.4.6	32	52	Changes Committed	Checked Out with Lock
	1.4.5	24	23	Changes Committed	None
Mike					
	1.4.8	8	45	Changes in Progress	Checked Out
	1.4.9	12	9	Changes in Progress	Checked Out with Lock

Figure 2.14: The explorer view, a visualisation tool of Palantir

copy of the artefact on their local workspace. For instance, three users, Ellen, Pete and Mike have a local copy of the “spell” artefact. Pete and Mike each have version 1.0 in their workspace, and their changes are still in progress as indicated by the question mark. Ellen, on the other hand, already has checked in a new version of the artefact, as indicated by the exclamation mark, resulting in her having version 1.1 in her workspace. Finally, the severity of a user’s changes on an artefact is indicated by a progress bar: the fuller the bar, the higher the severity.

While the severity and impact metrics were conceived to detect potential direct and indirect conflicts in concurrent changes made by the users, the information they deliver is unfortunately insufficient to determine whether there exist potential conflicts. The percentage of lines changed in a document is not an indicator of conflicting changes in the document. Concurrent changes of users on one and only line of code could be a potential conflict if changes are committed, while numerous changes of various users in different parts of the same artefact would not lead to a conflicting situation and therefore should not be presented as such. Although we favour the idea of metrics, the granularity of information provided by Palantìr is still at the document level, and documents are treated simply as a set of lines ignoring the semantics inside a document.

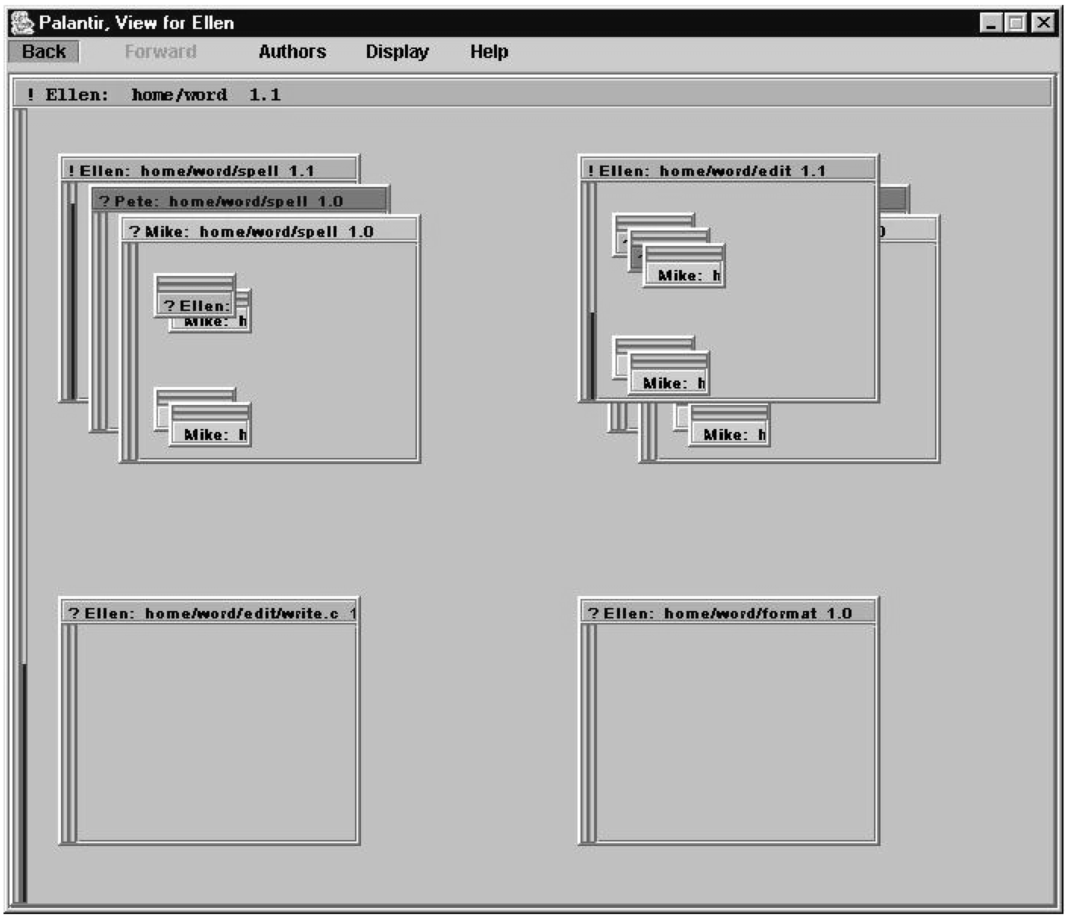

Palantir, View for Ellen
Back
Forward
Authors
Display
Help
! Ellen: home/word 1.1
! Ellen: home/word/spell 1.1
? Pete: home/word/spell 1.0
? Mike: home/word/spell 1.0
? Ellen:
Mike: h
Mike: h
! Ellen: home/word/edit 1.1
Mike: h
Mike: h
Mike: h
? Ellen: home/word/edit/write.c 1
? Ellen: home/word/format 1.0

Figure 2.15: The hierarchical view, a visualisation tool of Palantir

Tukan

In asynchronous collaboration over a big set of documents, it is often the case that users are not aware of work made in parallel by their collaborators. This results in conflicting and diverging work, which needs extra effort to be handled. Tukan suggests a different way of working, by introducing different collaboration modes, ranging from users working in offline mode, to users working very tightly in synchronous mode. Constraints and specific transition paths are defined for and between the collaboration modes. For instance, users need to relinquish part of their privacy if they need to work with other users in a tightly-coupled mode.

An evaluation of the system has shown that users worked mainly with only two collaboration modes; the “presence aware” and the “tightly-coupled” mode. This shows that although users do need support in both asynchronous and synchronous collaboration, a very detailed hierarchy of collaboration modes can be too complicated and not useful for users. However, there is another feature of Tukan that makes it useful for users.

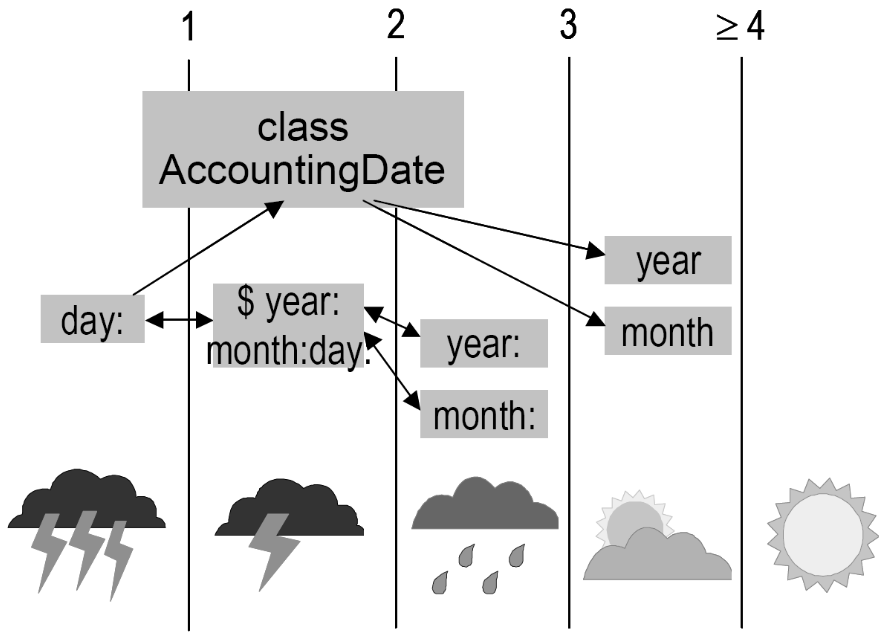


Figure 2.16: Tukan uses weather symbols to denote the existence of direct and indirect conflicts in concurrent changes made by collaborators

Tukan provides orientation and activity awareness. It is particularly useful at helping developers find knowledgeable people to work with and at avoiding conflicts. Tukan analyses the artefacts in a software project and determines which artefacts are semantically related. Additionally, it considers the foci and nimbi of the users that work on the project, aggregates all of the above information and determines the severity of potential conflicts between artefacts. This information is visualised as shown in Figure 2.16. A graph of artefacts and weighted relationships between them (e.g. composition, inheritance) is presented to the users. The graph is annotated with weather symbols showing where direct and indirect conflicts between artefacts occur. Figure 2.16 shows some artefacts of a project. A user has created a new version of the method “day:” resulting in the heavy lightning symbol being displayed under the method. All other displayed methods are related to “day:”, which causes other bad weather conflict icons to appear in front of their names. If there is no near conflict, a sun is shown to indicate that everything is up-to-date [107].

SoftChange

SoftChange is a tool that “retrieves the history of a project, analyses and enhances it by finding new relationships in it, and allows users to navigate and visualise this information" [48]. It was designed to help developers understand how a software project has evolved since its conception. SoftChange extracts metadata and the different revisions of each of the project's files from a version control system (CVS) and analyses the data. It then classifies changes based on the available information.

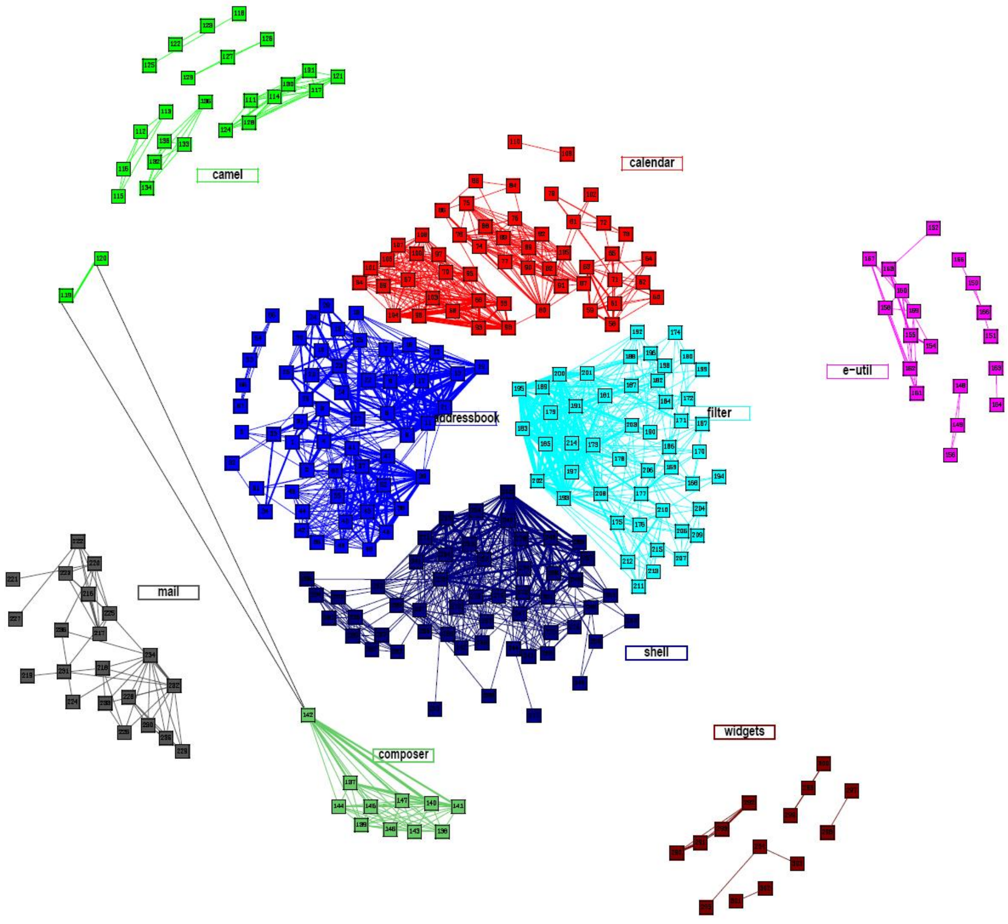


Figure 2.17: Modification-coupling graph for SoftChange

The visualisation mechanism of SoftChange is composed of two main parts: a hypertext browser and a graphical viewer. The hypertext browser allows the user to navigate and visualise who made a given change, when they made it, the modified files and why the change occurred. The graphical viewer is also composed of two parts. The first generates static plots of modifications of the software, where modifications are defined on the level of lines, made by users to specific files. For instance, the number of files that are part of the project at a given point in time or the frequency with which a file is modified can be computed and presented in histograms as an overview of the evolution of the project. The second part of the graphical viewer provides graphs that show files, users and their interrelationships, for instance, which files have been modified together, or which authors modify which files, as shown in Figure 2.17.

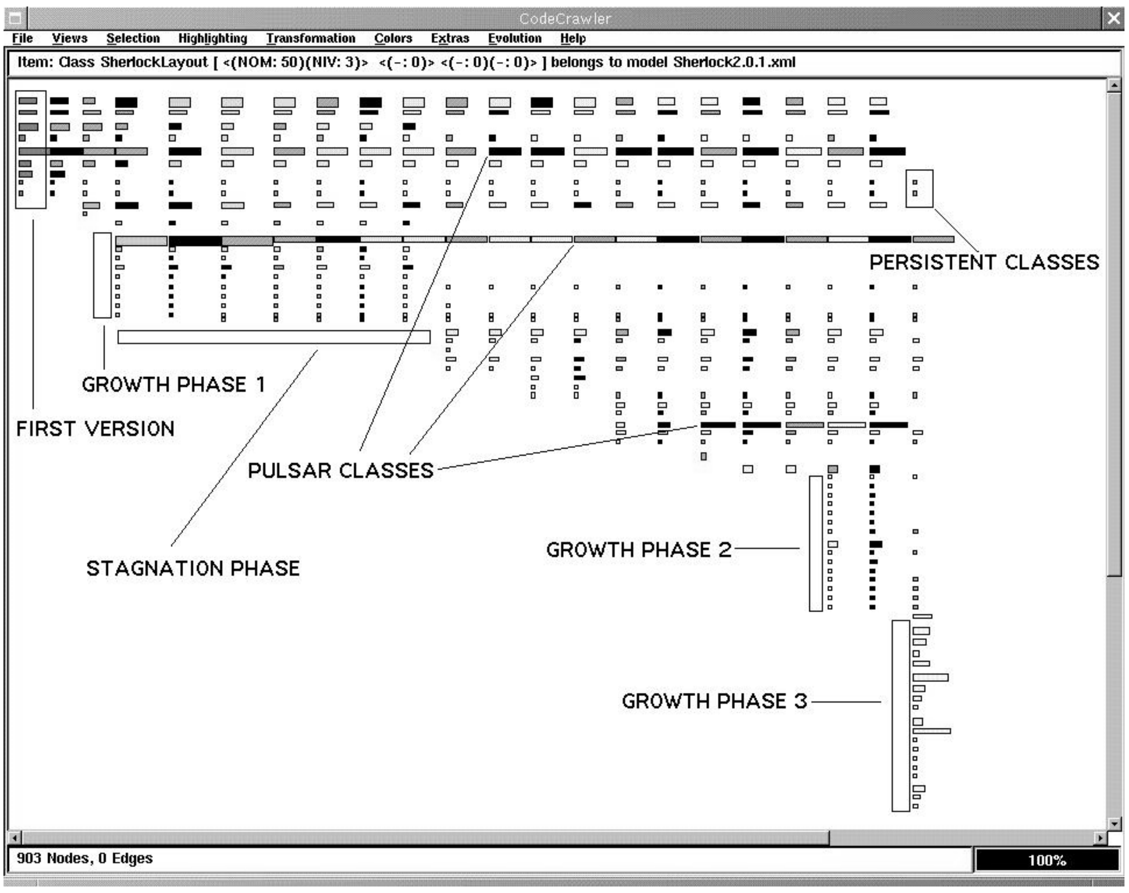
Evolution Matrix, Polymetric Views and Spyware

The idea of metrics defined to calculate how many changes have been made to a document is also known in reverse engineering. Lanza and Ducasse [78] and Robbes and Lanza [102] have defined various metrics in order to inform users about the evolution of programming code.

CodeCrawler is a single-user tool within which various visualisation tools are implemented to show software evolution. The evolution Matrix [77] and the Polymetric Views [78] are examples of them. They aim to help users understand the structure and detect problems of software systems in the early stages of reverse engineering [32]. They combine the concepts of software visualisation and software metrics, where the latter defines the information to be collected and the former the way to visualise the information. Lanza and Ducasse introduced a set of software metrics at different granularity levels, i.e. class metrics, method metrics and attribute metrics [78].

The evolution matrix displays the evolution of the classes of a software system. Each column of the matrix represents a version of the software, while each row represents the different versions of the same class. The columns are sorted alphabetically. A schematic evolution matrix is shown in Figure 2.18. The "height", "width" and "colour metric" of each rectangle can be used to display the number of specific types of modifications. The type of modification displayed by each of the metrics can vary based on user preferences. The evolution matrix enables users to make statements on the evolution of an object-oriented system at two granularity levels – the system level and the class level. At the system level, the metrics used are: "size of the system", "addition and removal of classes" and "growth and stagnation phases". At the class level examples of metrics are the "number of methods" and the "number of variables".

Polymetric views could be seen as more efficient views of the evolution matrix, since they also consider the project's hierarchical structure. An example view is the "system hotspots view" shown in Figure 2.19a. The nodes represent the classes, while the size of the nodes represents the number of methods they define. The gray nodes represent metaclasses. A second example, is the "inheritance classification view" shown in Figure 2.19b. The width and height of the class nodes represent the number of added methods and the number of overridden methods, while



Item: Class SherlockLayout [<(NOM: 50)(NIV: 3)> <(-: 0)> <(-: 0)(-: 0)>] belongs to model Sherlock2.0.1.xml

903 Nodes, 0 Edges

Figure 2.18: The Evolution Matrix visualisation

the colour represents the number of extended methods. The "system complexity view" shown in Figure 2.19c is a third example, where the nodes represent the classes, while the edges represent inheritance relationships. The metrics used are the number of attributes for the width, the number of methods for the height, and the number of lines of code for the colour. Finally, The "inheritance career view" is shown in Figure 2.19d. The width and the colour of the class nodes represent the number of descendants, while the height represents the number of methods.

While all of the above tools present information calculated from various metrics, they unfortunately concentrate on presenting the status of the project at specific versions and not the changes themselves. Additionally, they do not provide any information about the severity of changes made to a project and they visualise the available information solely on the project and class level. No detailed information is given about where in a class the changes have been made.

Spyware, a system for forward and reverse engineering of software

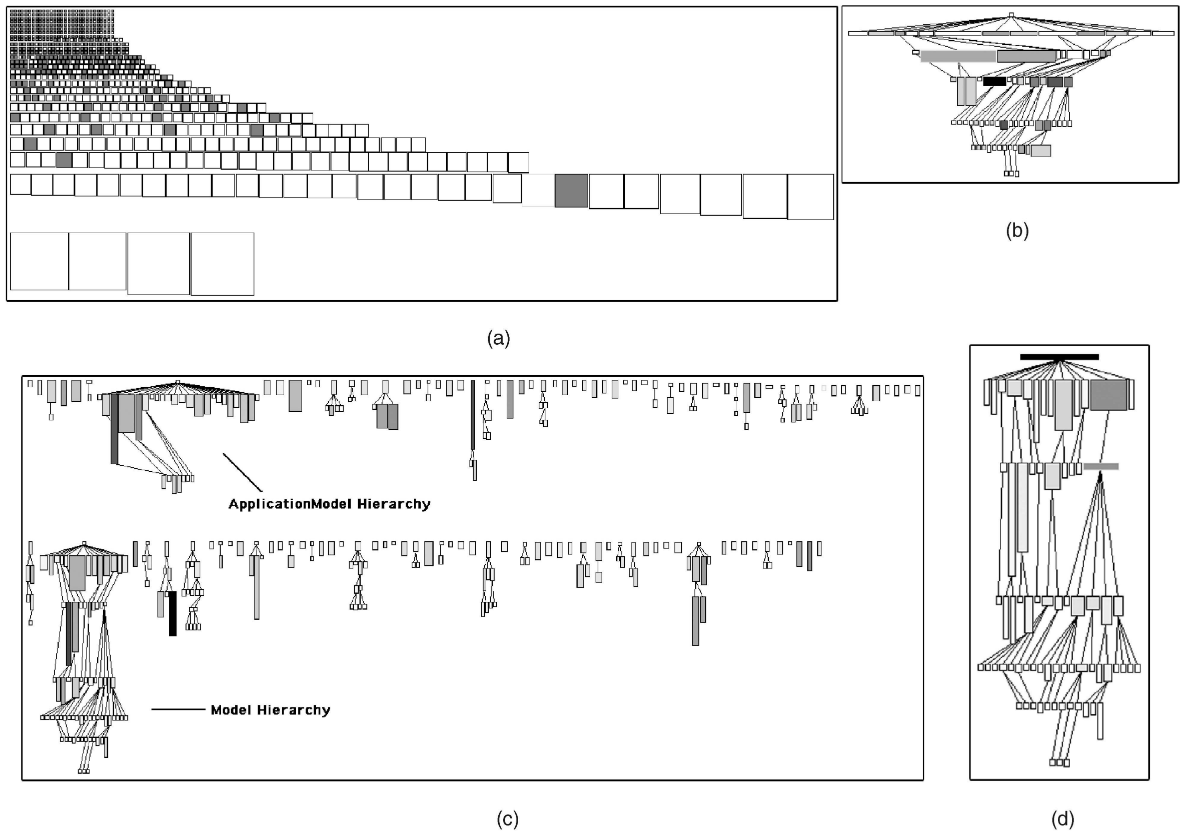


Figure 2.19: Visualisation of Polymetric Views

projects, concentrates on the actual changes made to the project. Contrary to the two systems above that implement a state-based approach, Spyware implements an operation-based approach. Therefore, it presents more accurate information about the evolution of a project [102]. The actual changes made to elements of different levels, for instance packages, classes, methods and variables are tracked. However, this approach concentrates on the project's current status and evolution. Therefore the information presented through the available visualisation tool does not concentrate on the modifications made to the project, for instance "two method names were modified in class foo", but rather on the current status of the project, for instance "class foo contains four methods".

CollabVS

CollabVS is an extension of the interface of Visual Studio software development environment. It provides real-time awareness information to users working asynchronously[34]. The approach is adapted for collaborative software development and provides developers with warning messages concerning concurrent activity on shared program elements. The users can choose the granularity of program elements for which the dependency checking is done and therefore the granularity of the possible conflicts as well. The identified possible conflicts are presented to the user through a notification mechanism and also in a separate list of conflicts. For a selected conflict, if the user cannot act appropriately to immediately resolve the conflict, there is the possibility to switch to communication sessions, or to set watches for the concurrently edited element. When the collaborator has finished editing the element, the users are notified and can resolve any conflicts. This approach is very promising because it provides information on different granularity levels. Users can choose the required level of information based on their tasks. However, a quantitative measure for concurrent operations or for conflicts is still missing.

2.8 Shared workspaces

The collaborative editors presented until now have a common feature. They aim to help users collaboratively authoring a document, or documents related to each other by providing information about other users' modifications. However groups frequently collaborate over different documents, possibly with different groups, and need to be informed about the status of all of them. To assist users involved in such collaborative activities, a number of shared workspaces have been developed. Their common feature is that they enable users to be aware of the status of a set of documents and their collaborators activity over all documents. There are many shared workspaces available, such as SubEthaEdit [17] and MS-SharePoint [16], but only a few provide at least some basic change awareness information. Here we report on some of them – BSCW [20, 25, 26], Stories [95], Groove [13] and GroupDesk [46]. We also report on State-Treemap [85], a visualisation mechanism for awareness information in shared workspaces.

BSCW

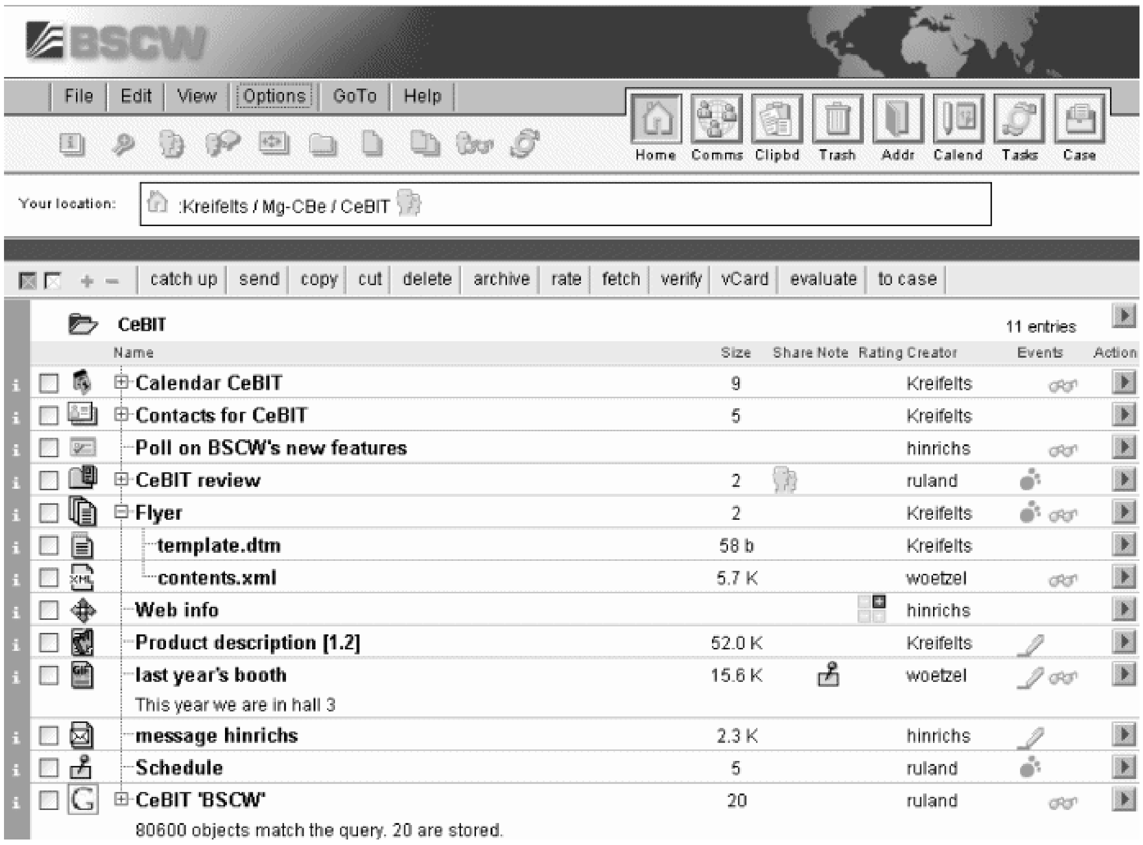
The BSCW (Basic Support for Cooperative Work) Shared Workspace system was developed to support the work of widely-dispersed workgroups, particularly those involved in large research and development projects. The system integrates simple facilities such as the storage and retrieval of documents, with more sophisticated features such as group and member administration, check-in/-out facilities and access to meta-information regarding documents and members. The system aims to support collaborative information sharing, therefore, it provides a simple, event-based awareness service to inform users, at-a-glance, of the current status and past changes to information held in the workspace.

The objects supported by BSCW are documents, links (to normal W3 pages and other workspaces), folders, groups and members. Modifications made on objects are marked as events, the most important ones being the "read", "re-named", "moved", "touched" and "deleted" events. Events can be tracked at different levels, from individual objects to complete workspace folder hierarchies. For instance, the "touch" event on a container shows that something has happened to an object inside the container. The events, once collected, are delivered to users through a notification mechanism. Users are kept aware of events, either through email notifications, or an icon based mechanism that assigns event-icons next to the modified objects according to the type of modifications, as shown in the column "Events" of Figure 2.20. For instance, the file "Calendar CeBIT" has been read (glasses icon), the file "Product description [1.2]" has been edited (pen icon) and changes were made somewhere in the folder "CeBIT review" (footstep icon). Users are hence, instantly notified about changes made to the objects of a workspace.

Although BSCW successfully delivers information about the changes made by various users to many documents, it provides neither any information about changes made to a document by different users in parallel (i.e. it focuses only on asynchronous collaboration) nor any information about in-document changes. Although it is necessary to offer information about the modified documents and the type of modification, this is only the first step that a user would take when investigating what has happened in a shared workspace. After finding the documents with "interesting changes", users very often need information about the severity of changes and the exact part in the document where they are made. BSCW does not offer any information about in-document changes, their severity, or the location of them in the document. Therefore, when users are informed through BSCW that a document has been modified, they need to open the document and manually compute the changes.

Stories for NESSIE

"Stories" have been introduced by Pankoke-Babatz et al. as an awareness mechanism to visualise the activities of users. This approach is based on



CeBIT 11 entries

Name	Size	Share Note	Rating	Creator	Events	Action
Calendar CeBIT	9			Kreifelts		
Contacts for CeBIT	5			Kreifelts		
Poll on BSCW's new features				hinrichs		
CeBIT review	2			ruland		
Flyer	2			Kreifelts		
template.dtm	58 b			Kreifelts		
contents.xml	5.7 K			woetzel		
Web info				hinrichs		
Product description [1.2]	52.0 K			Kreifelts		
last year's booth This year we are in hall 3	15.6 K			woetzel		
message hinrichs	2.3 K			hinrichs		
Schedule	5			ruland		
CeBIT 'BSCW' 80600 objects match the query. 20 are stored.	20			ruland		

Figure 2.20: The BSCW shared workspace

NESSIE, the awareNESS envIronmEnt with an underlying event and notification infrastructure (ENI). User actions, such as create, read or modify operations on a shared document, result in events, which are stored in a database. This information can later be filtered and the events of a specific time-frame can be visualised through different visualisation tools. The proposed visualisation mechanisms are the Daily Activity Report, the Time-Object-Activity (TOA) diagram and the DocuDrama Conversation.

The Daily Activity Report presents activity information in a textual form and chronological order. It is realised as an HTML-formatted email that is sent daily to all collaborators and is a tool used also in the BSCW workspace. The Docudrama Conversation aims to present the interaction between team members which took place while collaborating on documents in a shared workspace. It displays the sequence of activities, which have been performed on a document. The events are played out by avatars, which perform symbolic actions. Finally, the TOA diagram is a two dimensional representation to present user actions on objects as shown in Figure 2.21. The x-axis denotes the time. The y-axis is used to

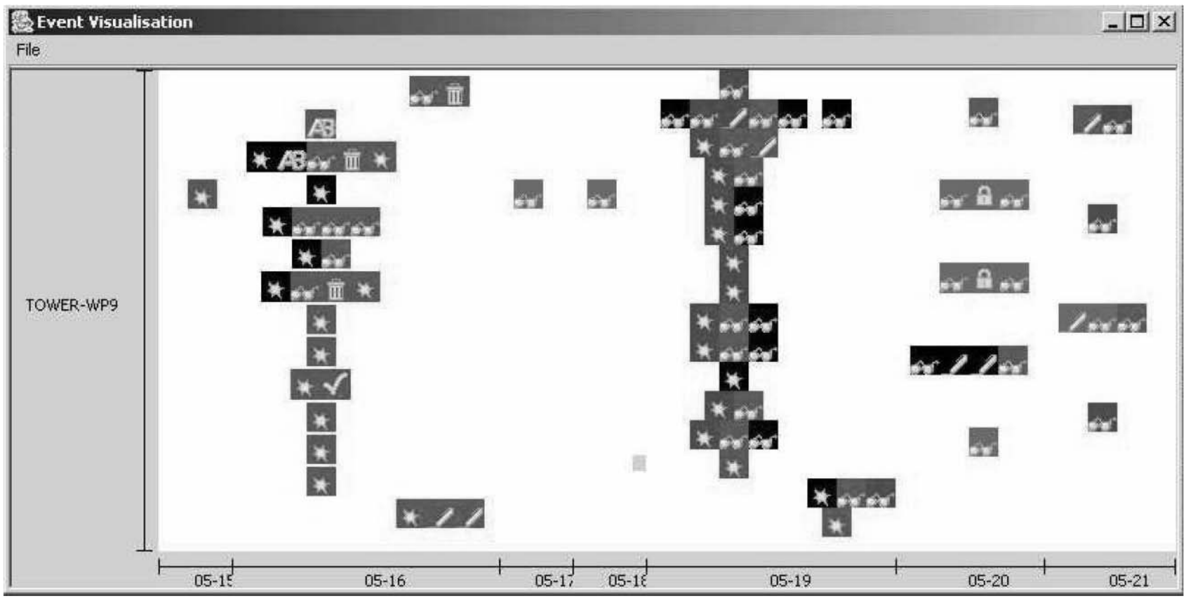


Figure 2.21: The Time-Object-Activity diagram

indicate the folders in which the actions occurred. For each user action, an icon is added to the diagram. The icon itself indicates the type of the action, and the colour is determined by the user's name. If more than one action occurred on the same object on the same day, then the icons are positioned on a single line.

The TOA diagram can be seen as an overview of the changes on documents in a workspace. However, users can only be informed about the types of other users' actions, but not their effect and importance. Moreover, the user actions, as in the case of BSCW workspace, are only at document-level, which prohibits the computation of multi-level awareness information.

Groove

Groove is another popular application for shared workspaces. Users have the possibility to create a new workspace at will and invite other users to share documents published in it. The workspace can include files, discussions or calendar items. Users can work on the items and modify them in both offline and online modes. Modifications made by a user on the workspace items are sent to their collaborators automatically by a mechanism that synchronises the workspaces of all users. Users are kept aware of changes made to the workspace items through a text notification mechanism that generates a message for every modification. Built-in communication tools are also offered to assist users. No other advanced mechanism for computation and visualisation of awareness information at various levels of granularity is offered by Groove.

GroupDesk

A system very similar to the BSCW system is GroupDesk. The system provides a simple environment for the coordination of cooperative document production. Support for awareness is achieved by visualising the event information using the desktop metaphor. A user, upon selecting a workspace, is presented with a graph of icons representing the objects in the workspace and the users working on them. Relations are drawn between users and the objects currently modified by them. Two types of events are defined in GroupDesk, “modifications” and “activities”. Modification events are generated by the system, each time the state of an object changes due to some action of a user. Activities describe synchronous events, related to the users in the system. Their creation marks the starting point and their deletion the end point of the corresponding action. GroupDesk offers awareness information for both asynchronous and synchronous collaboration. Events are distributed to the users through an event notification mechanism. They are visualised by changing the colour of the modified objects according to a colour code.

GroupDesk, similarly to BSCW and most shared workspaces, provides information only at the document level. Although changes inside documents are tracked, the information stored in the system is still at the document level. For instance, a change that corrects a spelling mistake, and a change that deletes a paragraph, are both marked as a “modification” of the document. No detailed information about the severity of the modifications, or the exact place in the document where they were made is stored in the system. As a result, GroupDesk cannot offer multi-level awareness information.

State Treemap

State Treemap was introduced as a mechanism that displays private workspace objects as a treemap where rectangles are decorated with colours indicating the state of the objects. Objects being modified by more than one user are marked as objects where potential conflicts may arise. When users publish their changes, their collaborators are informed that their local copy is in conflict with the latest published version. This approach visualises the status of objects in a workspace, using the inherent hierarchical structure of the workspace. The workspace is represented by a rectangle vertically split into subrectangles equal in number to the child elements of the workspace root. Each of the subrectangles is then further horizontally split in subsubrectangles if the corresponding object in the workspace includes other objects, etc., as shown in Figure 2.22. A rectangle's area is determined by the size of the corresponding objects in the workspace.

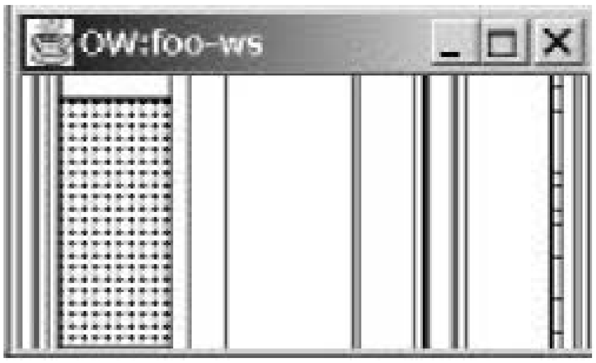


Figure 2.22: State Treemap

The treemap is an efficient way of visualising changes made in workspaces with a lot of objects. It offers a way to quickly locate whether changes have been made to the workspace objects while delivering information about the objects' size as well as their place in the hierarchy. However, the continuous splitting of rectangles into subrectangles can lead to a complicated scene, where neither the hierarchy of the workspace can be easily inferred by the visualisation tool, nor the place of a modified object in that hierarchy. This limits the use of treemap exactly to the situations for which it was initially designed.

We favour the idea of a visualisation tool that takes into account the structure of a shared workspace or object. However, we feel there is the need for a simpler tool that conveys the same information faster to the users. Additionally, a mechanism that would show how much an object has changed would be needed. Currently, treemaps show the objects that have been modified and the ones where (possible) conflicts arise. However, no comparison is made between the objects based on the number of modifications. A document heavily modified is not differentiated from one with slight modifications. This is mainly due to the fact that modifications are tracked only at the document level. Intra-document changes are not monitored and a document can only be in either the "modified" status or the "not modified" status. Thus, no multi-level awareness can be computed for workspaces implementing the treemap approach.

Summary of shared workspaces

All of the above systems aim to assist users in situations where many people collaborate over many documents and they need to share information about the status of the documents. They succeed in providing a large amount of information about which user modified which document, but fail to provide information about in-document changes. Modifications inside a document are all treated in the same way, without distinguishing their level or evaluating their effect on the document part that they alter. Therefore, it is not possible to compute awareness information on different granularity levels based on the needs of individual users and applications. Since awareness information is computed only at the document level, it is not possible to present an overview of the document with all the changes made to it. Therefore, if a user needs to be informed in detail about changes made to a document, they need to manually search for the changes inside it.

2.9 Summary

In this section, we revisit the requirements introduced at the beginning of this chapter. Using them, we have created Table 2.1 where all of the above-presented systems are listed and the degree of their compliance to each of the requirements is presented. We now describe each of the columns in turns. The "Domain" column describes where the system belongs according to the domains introduced at the beginning of the chapter. The system's name is given in the "Tool" column. The "ML" (Multi-Level) column contains information on whether the system offers computation of awareness information at various structural document levels. The "Vis" column contains information on whether the system offers an overview as a visualisation tool for awareness information and whether information is visualised at different levels. The next column is the "Met" column that refers to the computation of awareness information and whether it is done for different metrics. Finally, the "Mode" column contains information about the collaboration mode the system is designed for, asynchronous (asynch), synchronous (synch) or semi-synchronous (semi), and "Reuse" tells whether the system has been reused for a different domain, or a different type of documents.

A brief look at the currently implemented collaborative applications is sufficient to realise that there is no common approach, followed by developers, with respect to the change awareness mechanisms that the

Domain	Tool	ML	Vis.		Met.	Mode			Reuse
			overview	levels		asynch	synch	semi	
DCT	WinMerge	-	+	-	-	+	-	-	-
DCT	DiffDoc	-	-	-	-	+	-	-	+
DCT	DiffDog	-	-	-	-	+	-	-	+
VCS	CVS	-	+	-	-	+	-	-	-
VCS	SVN	-	+	-	-	+	-	-	-
CAT	Edit Wear	-	+	-	-	+	-	-	-
CAT	PREP Editor	-	-	-	+	+	-	-	-
CAT	SASE	-	+	-	-	-	+	-	-
CAT	SASSE	-	+	-	-	+	+	-	-
CAT	TeNDaX	-	+	-	+	-	+	-	-
CAT	CoWord	-	-	-	-	-	+	-	+
CAT	Divergence	-	-	+	-	-	-	+	-
CAG	MSWord	-	-	-	-	+	-	-	-
CAG	RationalRose	-	+	-	-	+	-	-	-
CAG	Chimera	-	-	-	-	+	-	-	-
CAG	Bitmap-based	+	-	-	-	-	+	-	-
CAG	PastDraw	-	+	-	-	+	-	-	-
CAW	Wikipedia	-	-	-	-	+	-	-	+
CAW	HistoryFlow	-	+	-	-	+	-	-	-
CAW	Bitmaps	-	+	-	-	+	-	-	-
CAW	WatchThatPage	-	-	-	-	+	-	-	-
CAW	ChangeDetect	-	-	-	-	+	-	-	-
CAW	WebsiteWatcher	-	-	-	-	+	-	-	-
DCD	Jazz	-	+	-	-	-	+	-	-
DCD	Palantìr	-	+	-	+	-	+	-	-
DCD	Tukan	-	-	-	-	+	+	+	-
DCD	SoftChange	-	+	-	-	+	-	-	-
DCD	EvolutionMatrix	-	+	-	+	+	-	-	-
DCD	PolymetricViews	-	+	+	+	+	-	-	-
DCD	Spyware	-	+	+	+	+	-	-	-
DCD	CollabVS	-	-	+	-	-	-	+	-
DCW	BSCW	-	+	+	-	+	-	-	-
DCW	Stories	-	+	-	-	+	-	-	-
DCW	Groove	-	-	-	-	+	+	-	-
DCW	GroupDesk	-	+	-	-	+	+	-	-
DCW	StateTreemap	+	+	+	-	+	-	-	+

Table 2.1: Summary of existing awareness-enabled collaborative systems

applications offer. Systems from different domains offer different awareness information and fulfil different requirements. This was expected, since each of the systems presented in this chapter was built to satisfy very specific user needs, which were different from (or only a subset of) the user and system requirements we address in the frame of this thesis. However, users' feedback from studies that tested some of the systems presented in this chapter, show that the users' expectations, with regard to the change awareness information a system should offer, are much higher. Additionally, users' needs on change awareness appear to be the same independently of the type of the authored documents, or the users' working modes. Unfortunately, the currently implemented systems concentrate only on a small set of features and therefore, fail to offer the advanced information that users need.

For instance, very few approaches are reusable for other document types or document modes and almost none can be applied to all of the domains, or all three collaboration modes. Many of the systems offer overviews of changes made to the documents, but only a few consider the severity of the changes and compute awareness information using different metrics, i.e. considering the current user's interest, role, or needs.

Finally, the most important drawback of current collaborative applications is that different document models are used for different document types. Even though hierarchically structured documents are widely used in various applications, there is no commonly accepted document model in the CSCW community and only a few systems exploit the document structure. As presented in this chapter, there are still many approaches in the CSCW community that simply consider the documents as an array of characters or lines and provide awareness either on the whole document, or, if applicable, on various document parts that have no semantic meaning since they are defined as blocks with a specified range of characters. Unfortunately, this allows almost no flexibility for the computation and presentation of awareness information.

We believe that a collaborative editing system should monitor user activity and record all changes made to various parts of the document. A summary of all edits would then form a global editing profile. The scale for the representation of changes should be flexible, allowing it to be adapted according to various document levels, for instance sections, paragraphs, sentences, etc. Users should also be allowed to use the global editing profile to select parts of the document and visualise the changes made there. In the next chapter we show how all the above can be materialised. We present the metamodel of an awareness framework that uses an underlying structured document model to enable the computation and presentation of enriched change awareness information.

3 General awareness framework

In this chapter we describe an awareness framework that fulfils all the requirements presented in Chapter 2. The proposed mechanism computes and visualises, at different granularity levels, the severity of the changes made to a co-authored document. To achieve this, the modifications made to the document are collected and their severity is measured. Since modifications can be made to different document parts at different syntactic document levels, the notions of a document part and various document levels are defined. Finally, to achieve generality of the approach, the process of the computation of the awareness information is designed so as to be independent of the document type and mode of collaboration.

We begin with an introduction to the structured document model used to represent the co-authored documents and the concept of operations used to describe the modifications made on document parts. Then we introduce the awareness framework in terms of the main components of its layered architecture and finally we detail the metamodel on which the framework is based.

3.1 Concepts

In this section, we present the general concepts of a structured document model and an operation, as well as some background required to explain the awareness framework introduced in the following section.

3.1.1 Structured document model

For an awareness mechanism to deliver the awareness information mentioned above, a flexible document model is required that gives access to document parts of various syntactic document levels. Furuta et al. [47] presented the concept of a logically structured document model a long time ago. The components introduced in this approach were related either hierarchically or through cross-references. Such a flexible model allowed the creation of document "fragments" to enable different users to access different parts of a document through the assignment of access rights on the fragments. Additionally, different views of the document were created and presented, based on the granularity of the fragments chosen.

Ignat and Norrie [67] have also used a hierarchical document model allowing collaboratively edited text documents to be accessed on different levels. We adopt this hierarchical structure and extend it to represent other kinds of documents with multiple levels as well. The reason for adopting this structure is its generality, as it encompasses a large class of documents such as textual and XML documents. For instance, a book contains chapters composed of sections. Each section is composed of paragraphs, each paragraph of sentences, each sentence of words and each word of characters. In this case, the granularity levels associated with the hierarchical model would be book, chapter, section, paragraph, sentence, word and character.

In Figure 3.1, an example of a document with 5 levels of granularity – document (level 0), paragraph (level 1), sentence (level 2), word (level 3) and character (level 4) – is illustrated. A history is assigned to each node, containing the modifications referring to children of that node. For example, the log of modifications associated with a paragraph will include insertions and deletions of sentences in that paragraph.

We represent the node of a document as described in [67].

Definition 6. A node N of a document is a structure of the form $N = <$*level*, *children*, *history*, *content*$>$, where

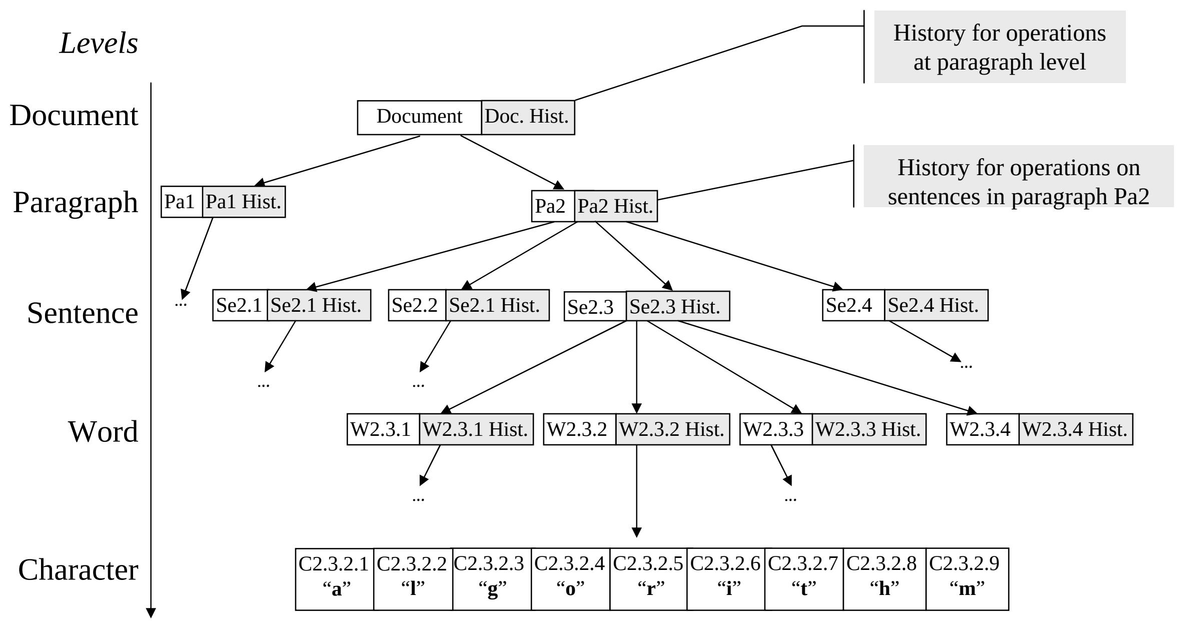


Figure 3.1: The hierarchical document model

- *level* is the structured document level, $level \in \{0, 1, \cdots, n\}$ corresponding to node N,
- *children* is an ordered list $\{N_1, ..., N_m\}$ of child nodes,
- *history* is an ordered list of operations referring to child nodes,
- $content = \begin{cases} \text{object stored in node,} & \text{if } N \text{ is a leaf node} \\ \sum_{i=1}^{m} content(child_i), & \text{otherwise} \end{cases}$

Note that a distributed history is not a requirement for the definition of a *node*. A document modification log where all operations are stored can be used as well.

3.1.2 Operations

We define operations representing changes made to the hierarchical structure in a similar way to that described in [67].

Definition 7. An operation is a structure of the form op= <*type*, *level*, *position*, *content*, *length*, *user*>, where

- *type* is the type of the operation, for instance *insert, delete or move*,
- *level* is the level of the operation's *content*,

- *position* is a vector of positions specifying the path from the root to the node where the operation is applied,
- *content* is a node representing the content of the operation,
- *length* is a vector with the number of units of each document level inserted, deleted, moved, etc. by the operation,
- *user* is the user who generated the operation.

The vector position specifies the indexes that compose the path in the tree where the operation is applied. For instance, the operation that inserts the word "algorithm", an insert operation of word level, to the text document of Figure 3.1 has a *position* vector equal to $< 2, 3, 2 >$ since it inserts the word in the second position of the third sentence of the second paragraph, i.e. information about the paragraph and the sentence in which the word is located, as well as the position of the word inside the sentence, are included in the position parameter. Finally, if the sentence "CSCW stands for Computer Supported Cooperative Work." is inserted in the document, the insert operation will have a *length* vector equal to $< 0, 1, 7, 45 >$ since the inserted text can be described as 0 units of paragraph level, 1 unit of sentence level, 7 units of word level or 45 units of character level. Note that the inserted text is a node of a fixed level, i.e. sentence level. However, its length varies and can be equal to any of the numbers provided by the *length* vector, depending on the level of the units we are interested in. For instance, the operation's length, it terms of the number of words inserted in the document, will be equal to 4.

We realise that there might not be any obvious reason to include the *length* vector in the definition of the operation. This information can be easily computed using the *content* of the operation. However, there exist collaborative situations where privacy concerns are raised concerning the amount of data exchanged between users. In such an environment, including the length of an operation in the operation while hiding its content might reveal enough information about users' modifications without showing the actual modifications. A more extended discussion on collaboration in privacy-sensitive environments is provided in Chapter 8. To keep the definition of an operation universal, we choose to include the length vector in it.

Our awareness mechanism can be adopted by existing collaborative authoring applications to increase the change awareness of users. *Nodes* and *operations* are defined in a generic way, in order to facilitate the adoption of the mechanism by a wide range of applications.

It can be argued that the use of a structured document model constrains the framework's generality, since there exist collaborative applications that still use a linear document model. A linear document model could also be mapped to the structured document model presented above, where only one document level exists. However, the awareness information computed for such a document would be less valuable for users, since they require multi-level awareness. Additionally, a large amount of research in the CSCW community has already moved towards structured document models as a result of research conducted by the document engineering community, which we believe supports our choice of a structured document model.

3.1.3 Operations versus diff algorithms

To collect information about the changes made to a document, two different approaches have been developed – the *state-based* and *operation-based* approaches. While we briefly mentioned them in the previous chapter, here we discuss them in detail.

In the *state-based* approach, given two different states of a document, the application computes the difference between them. This approach is used by many document comparison tools, for example [2, 3, 5, 9], CSCW editors [91] and also document engineering approaches [98]. In order to produce the most accurate "difference" between the two document states, many "diff" algorithms have been introduced for both linear [27] and hierarchical [19, 91] documents.

The *operation-based* approach keeps track of the actual modifications made to a document, i.e. operations applied to the document, at the moment of their creation. In this way, the set of operations that transformed a document from one state to another is stored and can be reused to rebuild a document state from its preceding state when needed. Examples of existing awareness mechanisms that adopt an *operation-based* approach are the work by Pankoke-Babatz et al. [95] and by Mangano et al. [82] where users' modifications are recorded and replayed at will.

A disadvantage of the *state-based* approach is that it only takes into account the initial and final states of the document and loses information about the process of transforming one state into another [101]. Additionally, the *state-based* approach has the disadvantage of not supporting merging and conflict resolution in the way that *operation-based* approaches do [81]. However, the *operation-based* approach comes with shortcomings as well, since access to the actual modifications is assumed.

Unfortunately, this requires that collaborators use the same or compatible authoring applications and no strict privacy rules apply, so that access to the document's history, i.e. modification logs, is granted.

Being aware of the above issues we favour and adopt an *operation-based* approach for computing and storing awareness information in collaborative authoring. This implies that the awareness framework we propose provides the most accurate awareness information since all the changes that transform the document from one state to another are captured and that the mechanism can be used as an extension of existing collaborative editors. It does not impose any further requirements on many of the currently available collaborative editors since the operations that transform a document from one version to another are already created by the tools and therefore can easily be recorded and used for the computation of awareness information. However, we are aware that, in cases where the authoring procedure cannot be monitored, the *state-based* approach should be used instead. Examples of such situations are described in Chapter 8 where being able to offer awareness information, even if operations do not capture the exact changes, is far more beneficial than not delivering any information at all.

At this point we would like to note that the choice of either of the two approaches described above does not influence the process of computing and visualising awareness information in the proposed framework. The awareness mechanism uses a set of modifications, described by a set of operations, independently of the approach that is used to extract this set. However, the exact values of the computed awareness information might vary and developers using the awareness mechanism that we propose should be aware of this issue.

3.2 Framework's architecture

The layered architecture shown in Figure 3.2 presents the main components of the proposed awareness framework. In order to apply our awareness mechanism to an existing collaborative application some prerequisites need to be satisfied, in terms of components that need to be present in the application. These are shown in the lower half of the architecture. If these components are present, the awareness mechanism can be applied on top of them as shown in the upper half of Figure 3.2. We now briefly discuss each component in turn.

The component that can be considered as the first requirement is

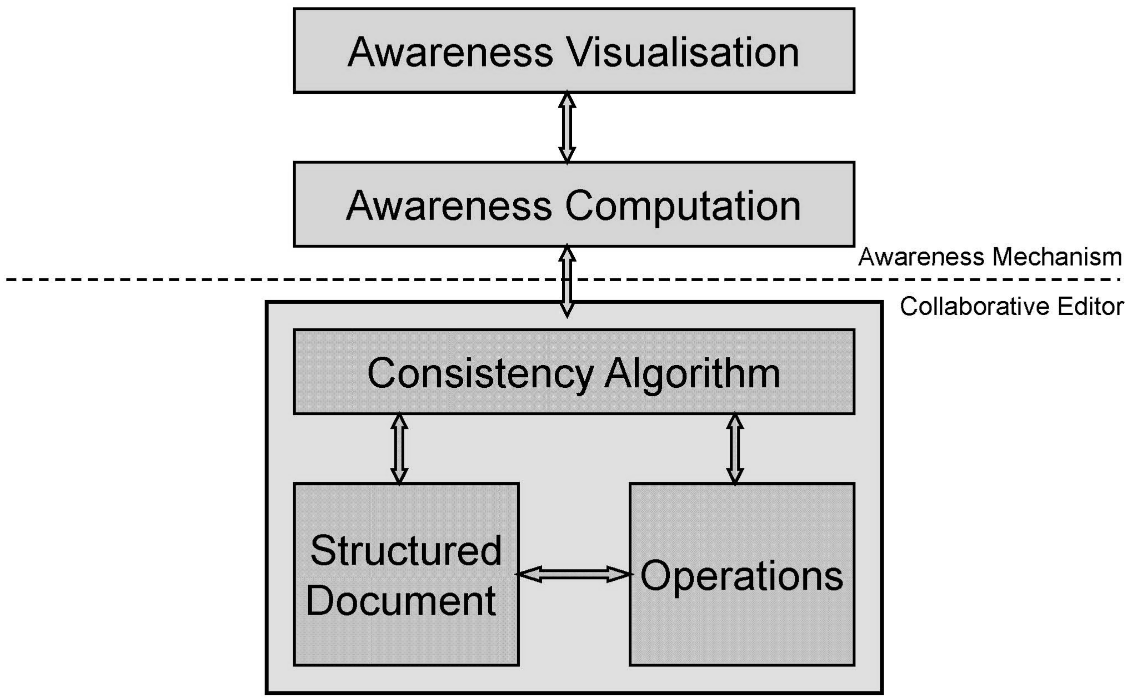


Figure 3.2: Basic components of the awareness framework

the existence of an underlying *structured document model.* As already described, in order to provide multi-level awareness, access to the document's parts of different syntactic document levels is required. This information should be made available by the collaborative application.

A second component of the framework which can also be considered as a requirement to maximise awareness information is the existence of *operations* modelling the modifications made by users to specific document parts. Therefore, operations should refer to the structure of the document. As already discussed, operations can either be created by the collaborative application, or by a diff algorithm.

In the next layer, the consistency algorithm is considered. This component includes any algorithms used to handle operations applied by different users when they collaborate. Such an algorithm could be a turn-taking protocol, a locking algorithm, an operation transformation algorithm, a merging algorithm or a conflict resolution algorithm. We assume that this component is part of the existing collaborative application.

The second half of the architecture concerns the *computation* and *vi-*

sualisation of awareness through our awareness mechanism. Local[1] and remote[2] operations are used in this layer. When an operation is applied to a specific document node, its severity is computed using different metrics and it is added to the awareness information of the corresponding node. The same procedure is followed for all local and remote operations, independently of the collaboration mode.

While local operations are instantly applied to the user's local copy of the document, remote operations are first managed by the consistency algorithm and then applied to the document. To correctly compute the severity of changes made on a user's local document copy, we consider local changes as they are created and applied in the local copy and remote changes only after they have been through the consistency algorithm. Since the consistency algorithm does not interfere with the awareness computation, any algorithm can be used.

In a similar manner, the computation of awareness information is independent of the collaboration mode. We consider operations at the time they are applied to a document copy. These can be changes just made by the local user, or changes made by remote users at any time.

The information computed in the computation layer is further filtered according to the application needs and user preferences and presented in the next layer through a visualisation tool. The visualisation tool can vary according to the collaborative tool. An *editing profile* is an example of a flexible interactive visualisation tool for collaborative editors. We used this in the integration of the awareness framework and a collaborative text editor, as described in Chapter 4. However, it is important to note that other visualisation tools could be used.

3.3 Framework's metamodel

The core of the metamodel on which the framework is based is shown in the UML model in Figure 3.3. The main concepts included in the metamodel are the *document*, the *node*, the *operation*, the *value*, the *metric* and the *visualisation tool.* The definition of the core concepts is intentionally generic, to enable the extensibility of the framework and its use with collaborative applications that handle various types of documents. After the basic core is set, the main concepts can be more specifically defined based on the requirements of the application and the document

[1]operation created by users and applied to their local copies of the co-authored document

[2]operation created by remote users at their local copies, sent via some network, received and applied by the rest of the users at their local copy

types handled by it. In this section, we describe the framework's main concepts and the associations between them. In the next chapters we present example applications where the framework can be used with existing collaborative authoring tools of various document types, to enhance the awareness provided by the applications to the users.

3.3.1 Document and node

A document of any document type is modelled as a *document* object. It consists of a root element modelled as a *node* object. A *node* represents a document part of any syntactic document level. *Nodes* are related to each other with the *hasChildren* and *hasParent* associations and build hierarchies of nodes. The fact that a document's root is a node, allows the representation of documents of different levels. A research paper, for instance, is modelled as a document with a root node of “article” level. As such, it contains nodes of “section” level, which contain nodes of “paragraph” level, etc. A document with a deeper hierarchy though, for instance a book, is represented as a document object with a root element of “book” level. This allows for the correct representation of a book that contains chapters, which contain sections, etc.

Note that a document's type affects the available node types and syntactic levels. For instance, a text document, as in the above examples, consists of nodes that include text and are of level book, chapter, section, paragraph, sentence, word and character. However, a graphical document has different nodes in terms of their type and syntactic level. For instance, “graphical objects”, “groups” of graphical objects, and different “pages” and “layers” can be the document parts of a graphical document and the document levels are relative to the document structure. Generally, the types and levels of the *node* objects are defined based on the characteristics of the collaborative application where the awareness mechanism is applied.

3.3.2 Operation

To include all editing operations in the framework, we consider all possible ways users interact with a document. Examples of interactions are the insertion or move of a document part in a document, the change of an object's colour, etc. Due to the large number of possible operations, we separate them into groups. For instance, we group together operations that result in modifications of the document structure, such as deleting,

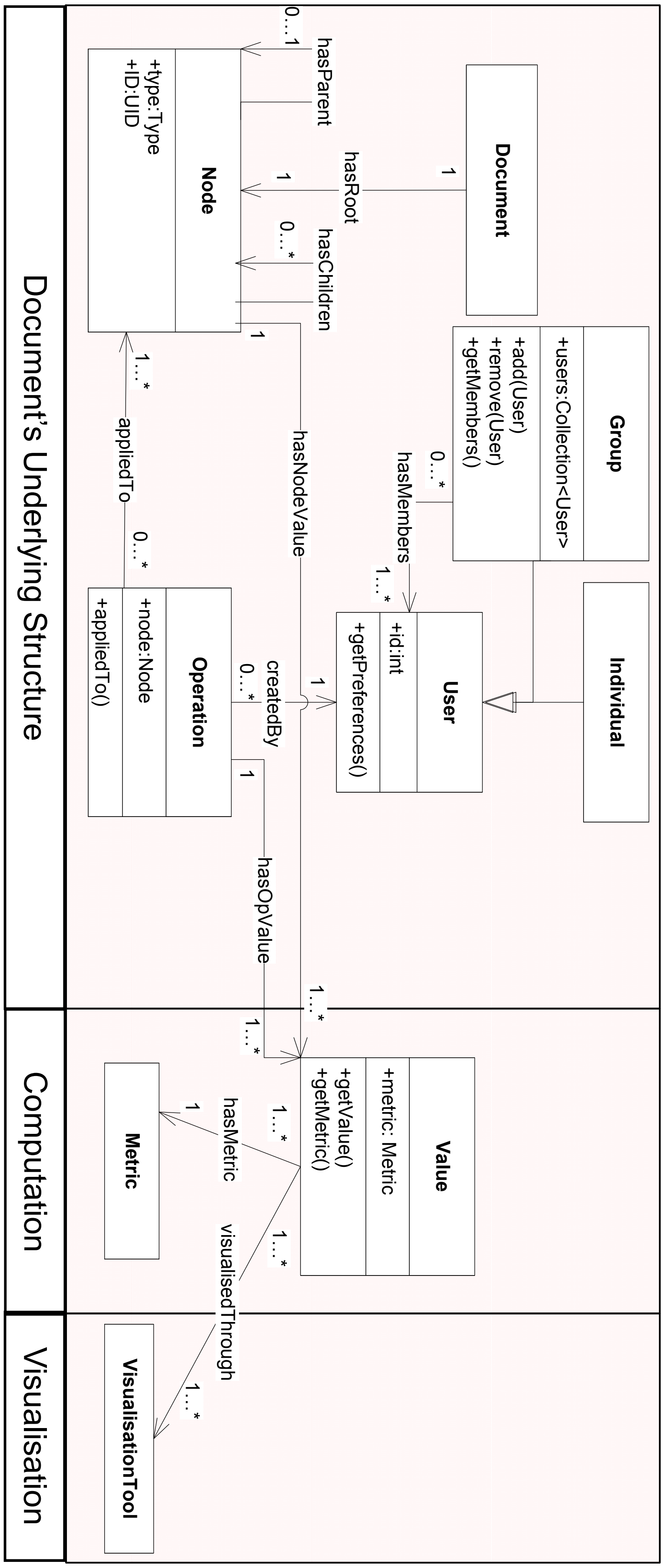


Figure 3.3: Main concepts of the awareness framework

inserting or moving document parts and refer to them as *structural operations*. Similarly, we model modifications that change the format of a document part as *formatting operations*, etc.

We model the above types of operations as specialisations of the general *operation* concept. The model can easily be extended to add other types of operations as well. Since the possible modifications that a user can make to a document are defined and also restricted solely by the authoring tool and the types of the documents handled by the application, the different operation types included in the metamodel vary. We choose to present only the concept of an *operation* at this time. For this reason, the different types for operations are not included in Figure 3.3. We describe them in detail in the next chapters, when the framework is used to enhance existing collaborative editors with awareness functionalities.

Every *operation* is related through the association *appliedTo* to a set of *nodes* where it is applied. The *operation* is stored in the modification log kept for the document or the specific *node*. Finally, the value of the *operation* is computed, as described below, and it is added to the corresponding *nodeValue*.

3.3.3 Operation value and node value

In our framework, each *node* and each *operation* is related to one or more *value* objects. The *value* of an *operation* reflects the severity of the *operation* when it is applied to a *node*, while the *value* of a *node* reflects the total severity of the changes made to the *node* and its children. We model them under the same concept of a *value* but will refer to them as *opValue* and *nodeValue* and have therefore introduced the associations *hasOpValue* and *hasNodeValue* respectively. In the next section, we describe how these are computed based on the metrics selected for a collaborative text editor.

The computation and storage of the *value* objects associated with a *node* varies, since it depends on the application. In asynchronous collaboration, the computation of the *value* objects is usually done every time a user updates their copy, i.e. receives a newer version of the document. Since the operations are applied on the document only once, it is sensible to perform the awareness computation only after an update and keep the computed values stored for further filtering and presentation.

However, in synchronous and semi-synchronous collaboration, operations are received continuously and it is time consuming and inefficient to store all the computed values for every operation, local or remote, when it is applied to the document. Therefore, in these working modes, it is advisable to only compute the current awareness values on-the-fly for presentation to the user rather than storing them. However, on-the-fly computation of awareness can be too restrictive in specific situations. For example, assume we want to offer the user the possibility of viewing awareness information for previous document states. Since the awareness values would be updated every time a new operation is applied, the previously computed values would be lost and it would not be possible to support such a requirement. To resolve such problems, we propose storing the status of the document and the current awareness values from time to time and recomputing the awareness information at any point in the future by reapplying the operations to the document. With the method *getValue* in the class *Value*, we indicate the aforementioned application-specific procedures that can be used to retrieve the corresponding *value* objects.

3.3.4 Metric

The existence of multiple *value* objects associated with both *operation* and *node* objects is due to the various metrics used. The severity of an operation can vary depending on the desired granularity of awareness. If a user is interested in recording all of the changes made to a text document, including even the spelling mistakes, this implies that all operations have to be recorded. The values associated with an operation will then be computed based on the total number of characters inserted or deleted. However, if an editor in a publishing company wants to check whether a document is still undergoing major changes or approaching its final version, they probably want to be informed only about those changes that include insertions and deletions of whole paragraphs. In this case, the awareness information would be based only on operations that inserted or deleted nodes of at least paragraph level, i.e. paragraphs or sections.

To enable this functionality, we support the existence of various metrics, each of which defines the level of detail that needs to be recorded for each modification and the exact information that needs to be extracted and delivered to the user. For a text document, an *operation* that inserts a sentence with content “Supporting Group Work.” has a *value* equal to 1 if the *metric* defined was “sentences”, which means that users are interested in the number of whole sentences inserted, a *value* of 3 if the *metric* was “words”, a *value* of 19 if the *metric* was “characters” and a *value* of 0 if the *metric* was "paragraphs". The various *value* objects associated with the *operation* objects applied to a *node* are then summed up to compute the corresponding *value* objects for the *node*. The detailed procedure of the above computations can be found in Section 4.2.2 where a set of example metrics is introduced for a specific text application.

3.3.5 Visualisation tool

We model the different ways in which the *value* objects can be presented to the user under the concept of a *visualisation tool*. The tools used for visualisation can vary depending on the application, the document type and the users' needs. In order to satisfy all the requirements set in Chapter 2, we favour the use of tools that present an overview of modifications throughout the whole document. However, overviews can also vary based on the type of the co-authored document. For instance, an overview of the modifications made in a text document is expected to be different from an overview of modifications made on a 2-dimensional scene of objects or an overview of changes made on a set of related documents.

The common characteristic, though, of all the above, is that they present a subset of the computed *nodeValues*. To retrieve this set of values, a filtering mechanism is needed, that takes into consideration the users current need for awareness, filters the available information and retrieves and presents to users only the relevant information. In the next chapters we propose several visualisation tools, for different document types. The filtering mechanism we used when applying the framework to a collaborative text editor is also presented in the next chapter.

3.3.6 User

Finally, we also include the notion of a *user* in our model. A *user* can either be an *individual* or a *group*. Each *user* has some preferences, such as a colour used for the visualisation of the awareness information. Additionally, we define that each *operation* has a specific *individual* as a creator. We use this information to select the *operation* objects according to their creator and compute the awareness based on this criterion as well. For instance, we might want to only present information about the changes made by a specific user. The model can easily be extended so that access rights between *user* objects and *node* objects are set, but since we are not currently interested in this information, we omit it from

Figure 3.3.

3.4 Summary

We presented in this chapter all the concepts included in the metamodel of an awareness mechanism that computes and presents multi-level awareness information to users. The definition of the concepts was kept general, to make the framework applicable in a large number of existing collaborative applications. In order to apply the framework to a given application, the application-specific document nodes, document levels, operations, metrics and visualisation tools need to be defined and modelled as specifications of the framework's general concepts. For an existing collaborative application where a hierarchical representation of the co-authored model is available, the application of the awareness mechanism is expected to be a straightforward procedure.

To prove the generality of the awareness mechanism, we applied it to an asynchronous collaborative text editor and a semi-synchronous collaborative text editor as presented in Chapters 4 and 6 respectively. We also conducted user studies with the asynchronous editor to test whether the concepts included in the awareness mechanism were well accepted and correctly interpreted by the users. The results are presented in Chapter 5. Finally, we considered ways to apply the framework to a graphical editor and an editor of webpages. We present our proposition in Chapter 8.

4
Awareness enhanced asynchronous text application

To validate the awareness framework presented in the previous chapter, we tested it with an existing collaborative editor developed in our group [67]. In this chapter we present the basic functionality of the editor, its underlying document structure, the realisation of the framework's core concepts for the specific editor and the procedure followed to integrate the computation and visualisation of awareness information into the editor. At the end of this chapter, we present the enriched editor along with the selected visualisation tool and comment on its new functionalities.

4.1 Asynchronous text editor

We start this section with a brief presentation of the existing functionality of the text editor that we used. We present its underlying document structure and the operations defined for it.

4.1.1 Editor's initial functionality

The first application into which we integrated our awareness framework is a collaborative text editor. It supports users that collaboratively author text documents in asynchronous mode. The editor implements the copy-modify-merge paradigm. A repository is maintained to store the different versions of the co-authored document. Users can download the latest version of the document and make their modifications on their local document copy. Upon completion of the modifications, the users can upload their local version to the repository by sending their modifications. The new version of the document is stored as an XML file containing all the operations that transformed the document from one version to another. After the new document version is successfully uploaded, it becomes available to the rest of the users. If more than one user downloads the same version of a document, modifies it and tries to upload their modifications, a merge algorithm is employed to merge the modifications of all users and store a version of the document containing all the alterations into the repository.

4.1.2 Document model and operations

Documents edited in this application are modelled according to the hierarchical document structure presented in Section 3.1.1. There are only 5 document levels available in the application, the "document", "paragraph", "sentence", "word" and "character" levels. When users make a modification, this is translated by the application into the creation of an operation applied to a document mode. Two types of operations are supported – the "creation" and "deletion" of document parts. Operations are also defined in a similar way to this in Section 3.1.1. Following the notation introduced in that section, the operation $op = \{create,\ 3,\ \{2,3,4\},\ awareness,\ \{0,0,1,9\},\ Bob\}$ inserts the word "awareness", i.e. a document node of word level (3), at position 4 of the third sentence of the second paragraph. The modification is made by user "Bob". The vector {0,0,1,9} represents the number of parts of the different document levels that are created by this operation. Hence, 0 paragraphs, 0 sentences, 1 word or 9 characters are created by this operation.

Note that the above representation of operations is the one that follows the notation introduced in the previous chapter. The representation of operations in the text application that we used was slightly different. The *length* vector including the number of parts inserted by the operation was not present in the definition of an operation for the text application. The computation of this information was introduced into the editor during the integration of the awareness mechanism into the editor. As we explained in the previous chapter, the computation of multi-level awareness for the above editor is not affected by the existence (or not) of the *length* vector in an operation, since this information is available to all users and can easily be computed by the *content* of an operation. However, in privacy-sensitive environments, where the content of an operation might not be visible by all users, the *length* vector would give additional helpful information. For the sake of consistency throughout this thesis, we assume that operations in this chapter contain the *length* vector and we present them as above.

4.2 Framework's extension for the text application

4.2.1 Document model and operations

We use the metamodel presented in the previous chapter to describe the concepts included in the collaborative editor. For this, we extend the core concepts of the framework's metamodel and introduce the application-specific concepts as specialisations of the general concepts. Details can be seen in Figure 4.1.

The available *nodes* are the ones that correspond to the document levels introduced above, for instance *document*, *paragraph*, *sentence*, etc. They are all modelled as specialisations of the *node* concept. Since only the deletion and creation of document nodes are supported by the application, example operations are *createWord*, *deleteWord*, *createSentence*, *deleteSentence*, etc. Note that since operations are applied to nodes of different levels, we define create and delete operations for each node level. The above operations are grouped under the notion of a *structural operation*, since they modify the structure of a document. A *structural operation* is a specialisation of the general *operation* concept.

Having explained how the part of the metamodel referring to the underlying document structure is materialised for the text application that we used, we will go on to present in detail how the rest of the framework's concepts were materialised in the application.

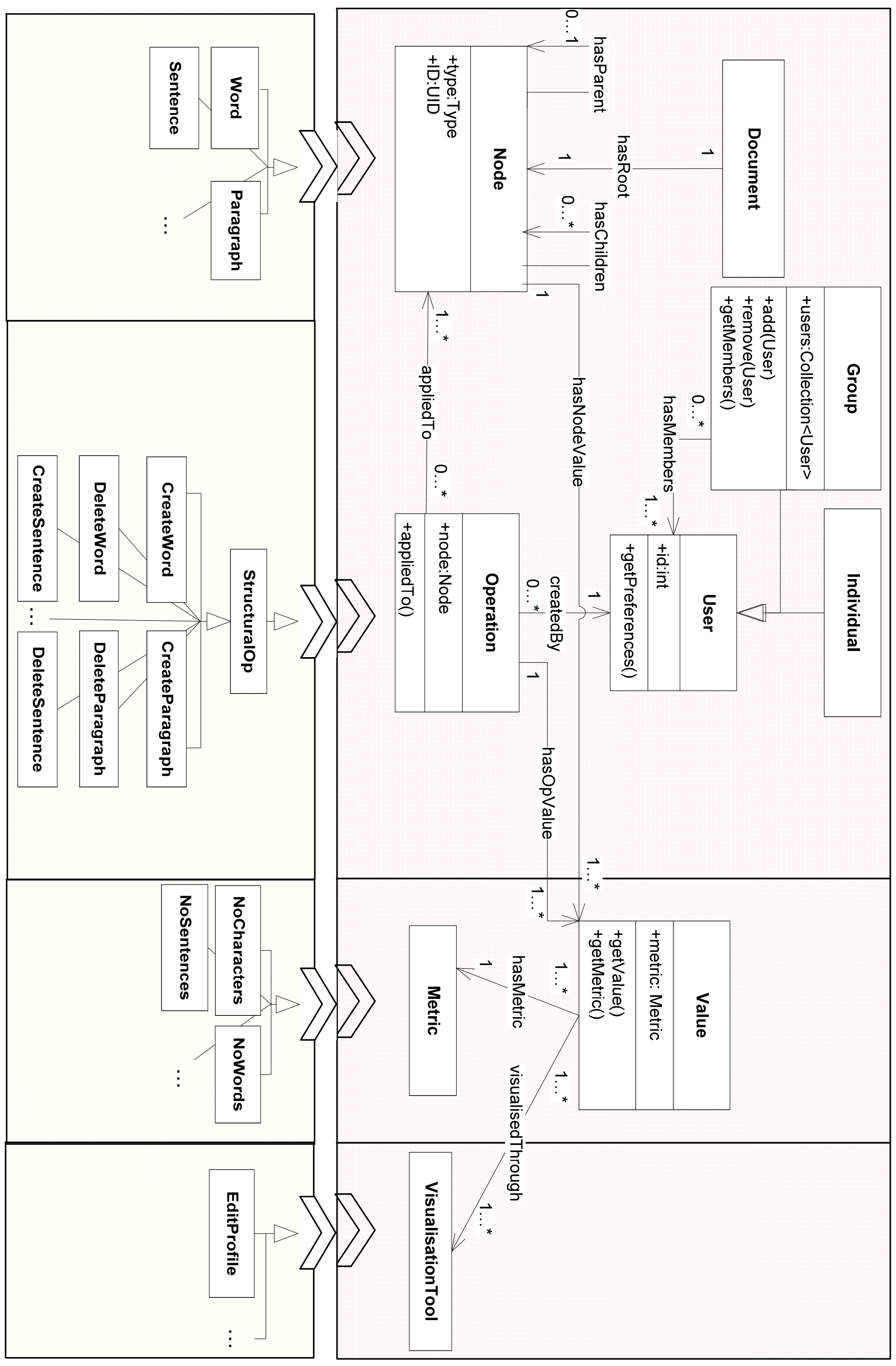


Figure 4.1: Extension of the framework's core concepts for a text asynchronous editor

4.2.2 Definition of metrics

The definition of metrics for a given application aims at specifying the kind of information that users are interested in and the level of detail with which the information is collected. The severity of an operation is then computed for each of the metrics. However, the definition of metrics is not only user-related. It is also application-related, since the available information depends on the features of the application, the collaborative activity supported by the application and the details of the underlying document model.

The collaborative application we use is a text editor, which means that all operations refer to some piece of text. The supported operations are the creation and deletion of a textual document part. Although there exist different syntactic levels for the document parts, all operations can be seen as a creation / deletion of a piece of text in / from a document. To classify operations according to their severity, we need to closely look at the details of operations that distinguish one from the other. These are the "level" and the "length" of an operation as described in Chapter 3. Both attributes need to be taken into consideration, since they reflect the severity of an operation.

For instance, the creation of a new paragraph in a document section, might possibly reflect the addition of a new concept. The deletion of a sentence in a paragraph, might reflect some repetition in the description of a concept, or some useless comment, while the deletion of a character would probably be a spelling correction. The level of the above operations indirectly gives information about the severity of a modification.

Similarly, the length of an operation might also affect our perception about the severity of the operation. Has an operation that adds a paragraph of 2 sentences the same severity as an operation that adds 20 sentences? Finally, two operations could be compared based on both their level and their length. For instance, which operation has the highest severity? The creation of a new section of two paragraphs, or the addition of 4 paragraphs in an existing section?

Unfortunately there is no clear answer to the above questions. The answers vary depending on the collaborative task and the users' roles and preferences. For instance, spelling mistakes might be more interesting for an author who received a co-authored document from a proofreader, than modifications that alter a whole section. However that would not apply to users who need an overview of a document's status to decide whether a document could be delivered before a deadline. Taking the above concerns into consideration, we decided to provide the users with a flexible mechanism that offers the required level of awareness based on their current preferences. Therefore, we consider both the "level" and the "length" of an operation in our computations.

As a result, the severity of an operation, described by the concept of the *opValue*, is computed based on the number (reflecting the "length") of units of different syntactic document *levels* that are created or deleted by the operation. The metrics are shown in Figure 4.1 as specialisations of the core *metric* concept. In detail, they are the "number of characters", the "number of words", the "number of sentences" and the "number of paragraphs", i.e. the number of syntactic document units of a specific level *ulevel*, inserted or deleted by an operation. As a result, an operation that creates a new sentence in a paragraph has a different *opValue* if users are interested in the number of sentences inserted, or in the number of words. We define the $opValue$ as follows.

Definition 8. The $opValue$ of an operation op is the number of units of level $ulevel$ that are inserted or deleted by the operation.
$opValue(op, ulevel) =$

$$\begin{cases} 0 & \text{if } level(op) > ulevel \\ 1 & \text{if } level(op) = ulevel \\ Count(unit_j)_{\substack{unit_j \in content(op)\,\&\\ level(unit_j) = ulevel}} & \text{if } level(op) < ulevel \end{cases}$$

If the operation level is the same as the level $ulevel$ of the units that we count, the $opValue$ of the operation is 1. For example, if the operation op is of level sentence, it means that it inserts or deletes a whole sentence. If $ulevel$ is also the sentence level, we are interested in counting the insertions or deletions of whole sentences, which means that the $opValue$ is equal to 1. If $ulevel$ is the word level, then the $opValue$ of a sentence level operation will be equal to the number of words that belong to the sentence inserted or deleted by the operation. If $ulevel$ is the paragraph level, the $opValue$ of a sentence level operation is not defined. We will use the value of 0 to denote such an $opValue$.

Note that the computation done above is the same as the one required to produce the *length* vector of an operation when the general notation of an operation is used. Therefore, if an operation's *length* is also available, the *opValue* could be defined using the information kept in the vector as below.

Definition 9. The *$opValue$* of an operation *op* is given by the element in the *$length$* vector that corresponds to level *$ulevel$*.
$opValue(op, ulevel) = length(op, ulevel)$

The $opValue$ shows the severity of only one operation when this is applied to a document part. To compute how much a document part has changed, all operations applied to it need to be taken into account. We introduce the $nodeValue$ to represent the severity of all modifications made on a node. This includes the values of all the operations applied to the node or its children. Due to the fact that the editor we extend keeps a distributed history of operations, operations concerning changes to a node are kept in the node's history and operations concerning the node's children are kept in the children's history. For that reason, we add a second factor to the $nodeValue$ metric, defined below, representing the changes made to each of the node's children.

Definition 10. We define $nodeValue$ of a node N to be the sum of two components: the sum of the $opValue$ of all the operations applied to the node and the sum of the $nodeValue$ of all the node's children.

$$\begin{aligned} nodeValue(N, ulevel) = & \sum_{op_i \in history(N)} opValue(op_i, ulevel) \\ + & \sum_{N_j \in children(N)} nodeValue(N_j, ulevel) \end{aligned}$$

$NodeValue$ of a node at the character level is equal to zero, since its history and children are empty sets. For every node, we additionally define a $createNodeValue$ and a $deleteNodeValue$ as the $nodeValue$ computed for only the create and delete operations, respectively. The $createNodeValue$ and $deleteNodeValue$ are the values to be later presented through the visualisation tool.

In the rest of this section we describe, with the help of a simple example, how the above values are computed. Assume a user is working on a document where the first sentence of the third paragraph has the content "We assign a valu every document node". The user edits the sentence so that its final content becomes "We assign a value to every document node". The modifications made by the users are the insertion of the character "e" at the end of the word "valu" and the insertion of the word "to" after the word "valu". In terms of operations, they can be written as:
$op_1 = \{create, 4, \{3, 1, 4, 5\}, e, \{0, 0, 0, 1\}, userName\}$

$op_2 = \{create, 3, \{3, 1, 5\}, to, \{0, 0, 1, 2\}, userName\}$

The first operation will be applied to the word “valu” and therefore will be stored in its history and the second operation will be applied to the sentence “We assign a valu every document node” and will be stored in its history. The tree structure representing the document can be seen in Figure 4.2. Next to the document nodes each node's history is shown with the operations that are applied to the node.

Depending on the selected metric, the *createNodeValue* computed for every document node varies. If the metric “number of characters” is used, then the computed values are the ones shown in Figure 4.3. If the metric “number of words” is used then the values are shown in Figure 4.4. For any other metric, all values remain equal to zero. Note that the values of all “affected” nodes are updated in the figures, including the parent nodes of the nodes where the operations are applied.

4.3 Visualisation of awareness

A novelty of our approach is the flexibility of presenting awareness for the various structural document levels and document parts. For that reason, a visualisation tool that allows the users to define the granularity of awareness that they want to view was built.

4.3.1 Filtering procedure

The visualisation tool, through its graphical user interface, allows users to define in detail the information that they need. This, in terms of the preceding definitions, can be translated to users filtering the exact *nodeValues* that need to be presented through the visualisation tool. For this, a filtering mechanism is used, that traverses the structured document and the *nodeValues* computed for each node and feeds the visualisation tool with the correct set of *nodeValues* based on user requirements.

The filtering mechanism provides two ways to traverse the document nodes. One is at a document level and one for a document node. To illustrate the need to provide awareness information for a specific level or document node, let us consider the following examples: when a user receives a new version of a document, they need to see what has happened in the document since the last version they were aware of. For a document handled by the given text application, this means how much each paragraph has changed, since this is the next level after the document level. To get this information, users only need to specify the document

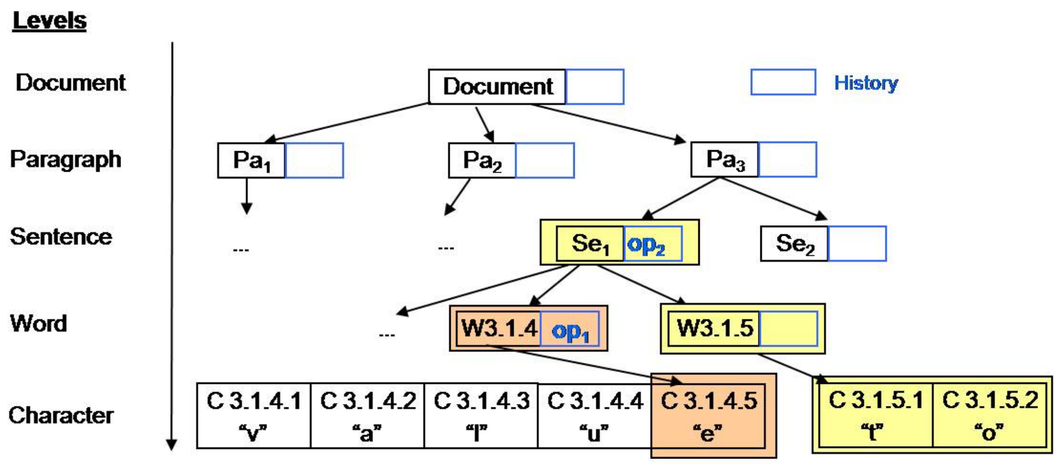


Figure 4.2: Example of a tree-structured document with a set of operations applied to it

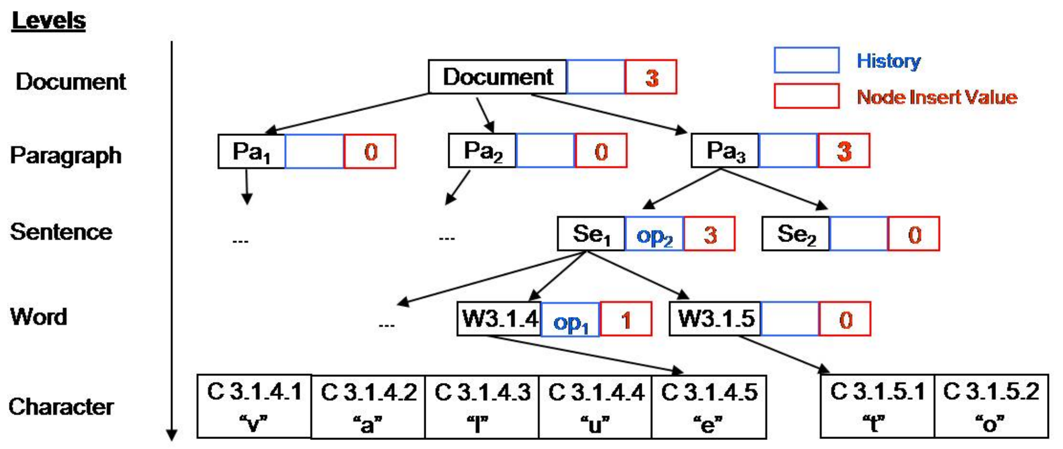


Figure 4.3: The metric "number of characters" is used for the computation of the *nodeValues*

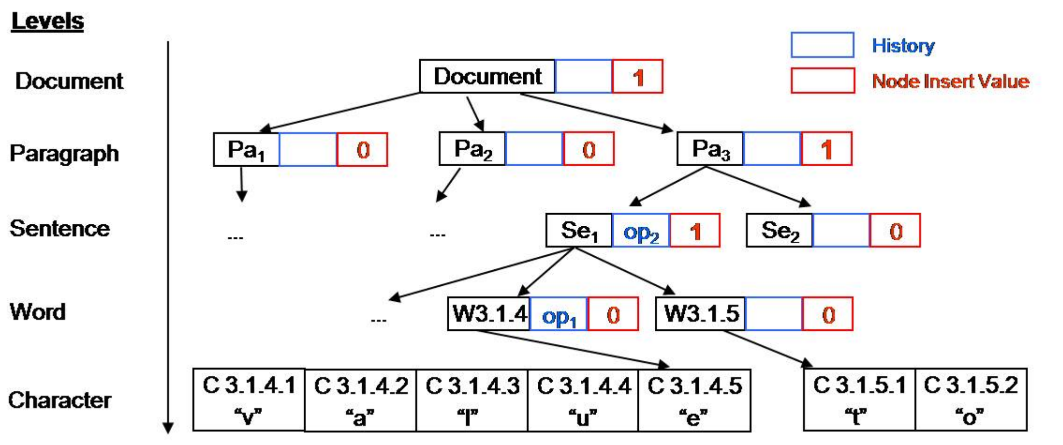


Figure 4.4: The metric "number of words" is used for the computation of the *nodeValues*

level in which they are interested, in this example the paragraph level. However, if a user is responsible for a specific paragraph they might be interested in changes made to that paragraph only. For instance, they might be interested in changes made on sentences of the first paragraph. Filtering the computed information on the sentence level would provide the user with the *nodeValues* of all sentences in the document. To avoid this, we allow the user to filter the computed information by specifying only a certain document node. The *nodeValues* computed for the node's children are then presented through the visualisation tool. We modelled this behaviour with two functions, the *awarenessOnLevel* function and the *awarenessOnNode* function.

In Figure 4.5 and Figure 4.6 we present a graphical representation of the filtering mechanism. Then, we also present the formal definitions of the functions. In the document of the previous example with the operations op_1 and op_2, the filtering of the computed information on different document levels can be graphically represented as in Figure 4.5. A request of all modifications made on the paragraph level would traverse the structured document and return only the paragraph nodes and their *nodeValues*, as highlighted with the upper shaded group of nodes. A request for the modifications through all the document sentences would return the lower shaded group of nodes. However, if information about a specific document part is required, for instance paragraph three, then only the sentence level nodes that are direct children of the specified paragraph will be returned as shown in the upper shaded group of nodes in Figure 4.6. The rest of the document sentences will not be returned. In a similar manner, if the modifications made to the first sentence of the third paragraph are requested, the lower shaded group of nodes will be returned.

We now present the formal definitions of the *awarenessOnLevel* and the *awarenessOnNode* functions. As can be seen, the information returned from the functions can be narrowed down if users provide a complete set of requirements instead of only a document level or a document node. An example of such a request could be to search for the severity of the deletions of whole sentences that user Bob did throughout all paragraphs. In this request, apart from the document level we also specify the type of the operation, the metric, as well as the user that made the modifications.

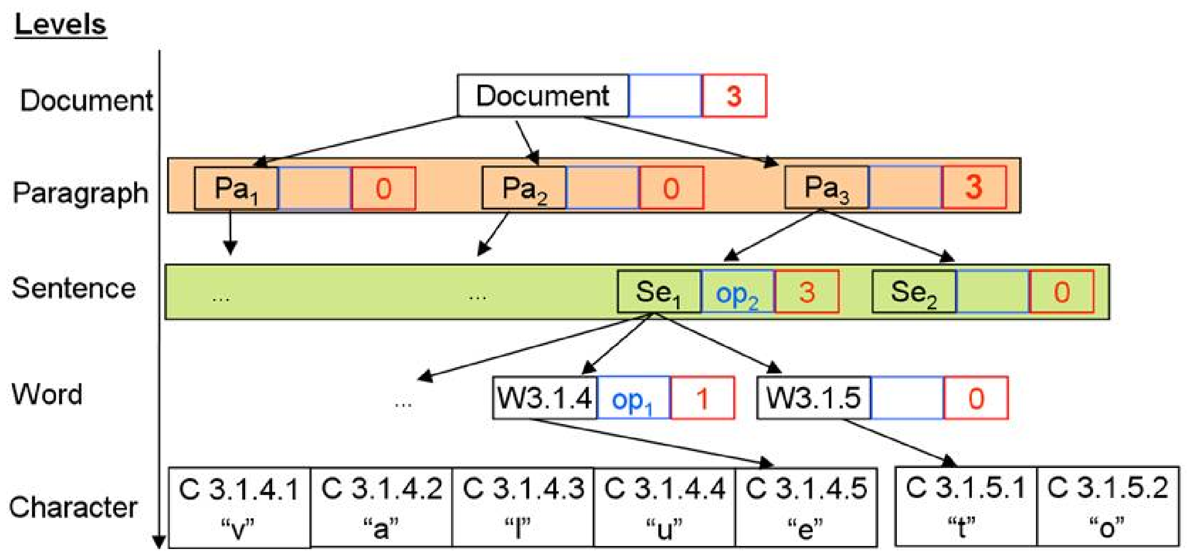


Figure 4.5: Access to nodes of a given level

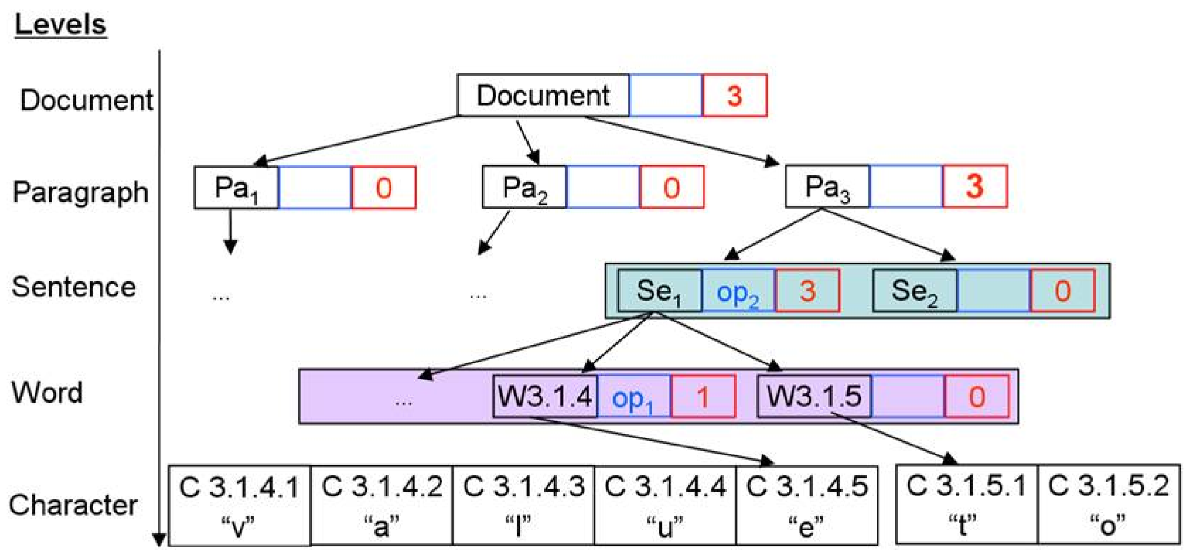


Figure 4.6: Access to children nodes of a given node

Definition 11. We define the function $awarenessOnLevel(opType, level, userList, metric)$, where

- $opType \in \{creation, deletion, both\}$ is the type of the operations that the user wants to visualise,
- $level \in \{0, 1, 2, 3\}$ is the level of the nodes, in the tree document, whose changes should be visualised,
- $usersList$ is a list of users whose modifications should be visualised,
- $metric$ is the metric used for the computation of the *nodeValues.*

This function retrieves the *nodeValues* of the nodes on the given $level$ and returns the *nodeValues* corresponding to users in $usersList$ and operations with type $opType$.

Definition 12. In a similar manner we define the function $awarenessOnNode(opType, node, userList, metric)$, where

- $opType$, $metric$ and $usersList$ are defined as in the function $awarenessOnLevel$,
- *node* is the document node whose changes should be visualised.

This function filters the *nodeValues* of the *node*'s children and returns the ones computed with the given metric and corresponding to users in $usersList$ and operations with type $opType$.

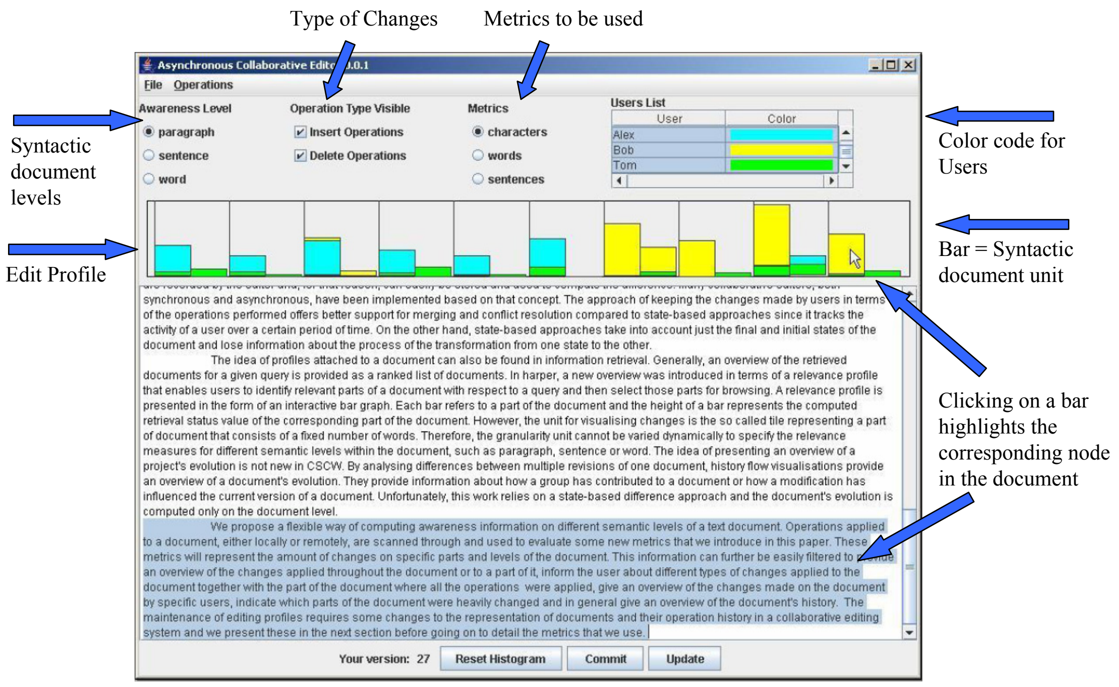
The functions defined above can be used to extract information about users or roles in collaborative situations. The information extracted directly is the modifications that a specific user has made. However, the information extracted alongside is equally important. Consider a collaborative situation, where each user has to write one section of the document and review another. It could be interesting to recognise users or user roles based on the parts of the document that they are editing, or based on the patterns that the users use when authoring or reviewing a document. An extensive discussion on this issue is presented in the next chapter, where the editor's awareness features are tested and users' feedback is collected.

The above computed information is more direct and can be extracted more easily in the case that the computation of *nodeValues* is applied in a synchronous editor where the *nodeValues* are continuously updated and presented to the user in real-time. Noticing, for instance, that the *nodeValues* of the second section are increasing, the user becomes aware that another user is working on that part of the document. If the *node-Values* of all the sections are slightly increasing one after the other, it may mean that a reviewer is going through the whole document making small changes everywhere. We realise that the potential use of the awareness information that we compute is greater than the one presented in this chapter, especially concerning synchronous and semi-synchronous collaboration. For that reason, we have investigated the possibility of applying the framework on a semi-synchronous collaborative editor and to applications that handle different document types. More on that is presented in the next chapters.

4.3.2 Edit profile

Using the above functions, a group of *nodeValues* is selected and visualised through the visualisation tool. To satisfy the users' requirement of an overview, we created a visualisation tool that provides an editing overview throughout the whole document, for the different document levels, or across different document parts. That tool is called *Edit Profile* and a screenshot of it integrated into the text editor is shown in Figure 4.7. The graphical user interface of the edit profile consists of a group of radio buttons, a table and a histogram shown in details in Figures 4.8, 4.9 and 4.10 respectively. By setting the value of the radio buttons, users specify the exact information that they need. In essence, they set the input parameters *opType*, *level* and *metric* for the filtering functions. The parameter *usersList* is set through the table.

With the leftmost group of radio buttons, we provide the user with the possibility of being aware of changes at different document levels. They can choose between "paragraph", "sentence" and "word" levels, i.e. visualise the total number of changes that were made on document nodes of the corresponding levels. If, for instance, the "paragraph" level is selected, then the values computed for each of the document paragraphs will be chosen and presented through the edit profile. These values picture the severity of all the changes made to each paragraph and its children nodes. The computed values are presented in the histogram, shown in Figures 4.10 and 4.7. We decided against adding the "document level", since for this editor it would return only one value at a time. We believe the information on a document level is needed in collaborative situations where users are working with multiple documents


Type of Changes
Metrics to be used
Syntactic document levels
Edit Profile
Color code for Users
Bar = Syntactic document unit
Clicking on a bar highlights the corresponding node in the document
Asynchronous Collaborative Edito .0.1
File
Operations
Reset Histogram
Commit
Update

Awareness Level: [✓ paragraph] sentence word

Operation Type Visible: ✓ Insert Operations ✓ Delete Operations

Metrics: [✓ characters] words sentences

Users List

User	Color
Alex	🟦
Bob	🟨
Tom	🟩

synchronous and asynchronous, have been implemented based on that concept. The approach of keeping the changes made by users in terms of the operations performed offers better support for merging and conflict resolution compared to state-based approaches since it tracks the activity of a user over a certain period of time. On the other hand, state-based approaches take into account just the final and initial states of the document and lose information about the process of the transformation from one state to the other.

The idea of profiles attached to a document can also be found in information retrieval. Generally, an overview of the retrieved documents for a given query is provided as a ranked list of documents. In harper, a new overview was introduced in terms of a relevance profile that enables users to identify relevant parts of a document with respect to a query and then select those parts for browsing. A relevance profile is presented in the form of an interactive bar graph. Each bar refers to a part of the document and the height of a bar represents the computed retrieval status value of the corresponding part of the document. However, the unit for visualising changes is the so called tile representing a part of document that consists of a fixed number of words. Therefore, the granularity unit cannot be varied dynamically to specify the relevance measures for different semantic levels within the document, such as paragraph, sentence or word. The idea of presenting an overview of a project's evolution is not new in CSCW. By analysing differences between multiple revisions of one document, history flow visualisations provide an overview of a document's evolution. They provide information about how a group has contributed to a document or how a modification has influenced the current version of a document. Unfortunately, this work relies on a state-based difference approach and the document's evolution is computed only on the document level.

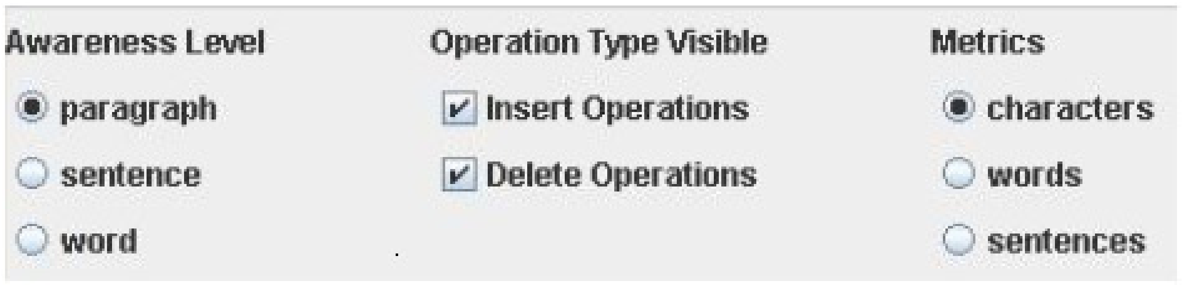
We propose a flexible way of computing awareness information on different semantic levels of a text document. Operations applied to a document, either locally or remotely, are scanned through and used to evaluate some new metrics that we introduce in this paper. These metrics will represent the amount of changes on specific parts and levels of the document. This information can further be easily filtered to provide an overview of the changes applied throughout the document or to a part of it, inform the user about different types of changes applied to the document together with the part of the document where all the operations were applied, give an overview of the changes made on the document by specific users, indicate which parts of the document were heavily changed and in general give an overview of the document's history. The maintenance of editing profiles requires some changes to the representation of documents and their operation history in a collaborative editing system and we present these in the next section before going on to detail the metrics that we use.

Your version: 27

Figure 4.7: Awareness enhanced GUI

to inform them whether they need to open a document and further check it. We discuss examples of such collaborative situations in Chapter 6. However, for the current editor, we believe that this functionality would not offer any additional information to the user.

The next two radio buttons enable the selection of each node's *node-Values* that are required by the users. The buttons refer to the type of the operations and the metric. We provide a selection of three metrics at this edit profile, namely "sentences", "words" and "characters". If, for instance, the metric "sentences" is chosen, then the values are computed based on the number of entire sentences inserted or deleted. Since the document level is not visualised in this edit profile, the "number of paragraphs" metric is also not offered to the users. Note that not all metrics are available for every document level. For instance, if information on the sentence level is visualised, which means that all the modifications for each sentence are presented, the paragraph metric is not available. This results from the fact that there cannot be any operation inserting or deleting a whole paragraph inside a sentence. Further, referring to the type of changes to be visualised, insertions, deletions or both can be chosen. Finally, we also provide the table shown in Figure 4.9 with an entry for every user who has made changes to the document. Users are assigned colours to distinguish changes made by different users. The set of users to be visualised can be selected dynamically.

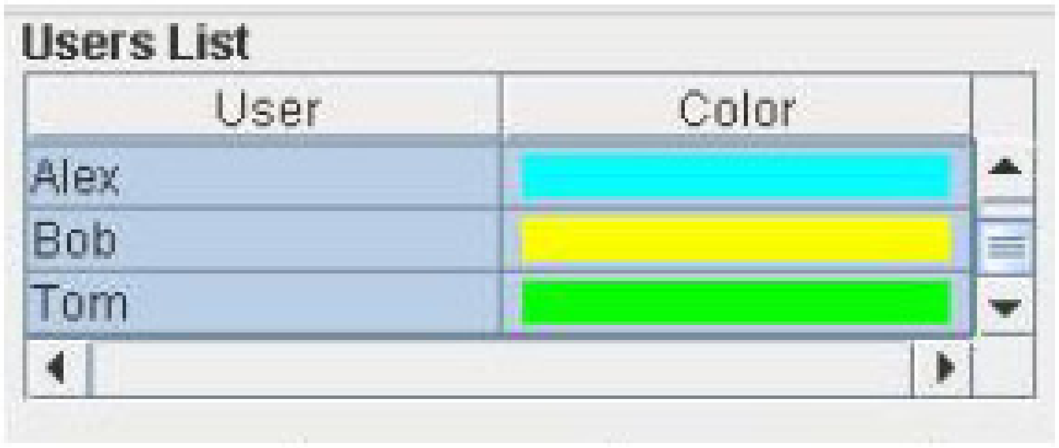


Awareness Level
● paragraph
○ sentence
○ word

Operation Type Visible
✓ Insert Operations
✓ Delete Operations

Metrics
● characters
○ words
○ sentences

Figure 4.8: GUI snippet presenting the radio buttons of the edit profile



Users List

User	Color
Alex	🟦
Bob	🟨
Tom	🟩

Figure 4.9: GUI snippet presenting the users table of the edit profile

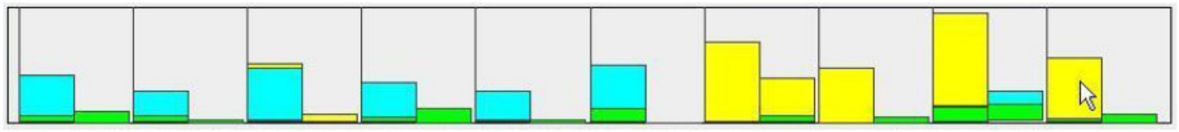


Figure 4.10: GUI snippet presenting the histogram of the edit profile

After specifying the document level, operation type, metric and list of users through the GUI, the awareness values computed throughout the entire hierarchical document are filtered, and the results are presented in the profile. A bar is drawn for every node showing the number of changes made to that node and its children. If only insertions or only deletions are selected, each bar shows the number of insertions or deletions, respectively, made to the corresponding node. Each bar is coloured with the colour of the user who made the changes. If more than one user is selected, each bar is horizontally split into subbars to represent the different users.

The height of a subbar is relative to the proportion of the total number of changes made to the corresponding node. Each subbar is coloured with the corresponding user colour. If both insertions and deletions are selected, each bar is vertically separated into two subbars to present the insertions and deletions separately. The left subbar shows the number of insertions and the right subbar the number of deletions. If more than one user is selected, the subbars are further horizontally split and coloured as presented in Figures 4.10 and 4.7.

In the example of Figure 4.7, awareness is provided at the paragraph level, for both insertions and deletions, and "characters" is the selected metric. As a result, the edit profile (shown also in Figure 4.10 shows 10 bars, corresponding to the 10 paragraphs of the document. Each of them is vertically split into two subbars, since information for both insertions and deletions is required. The left subbars reflect insertions on a paragraph, while the right reflect deletions. Since "characters" is selected, a bar's height reflects the total number of characters inserted or deleted to / from each paragraph. Users Alex, Bob and Tom are selected from the users table. Therefore, the modifications of all three users are presented.

Note that the above procedure uses the filter *awarenessOnLevel*. The procedure to activate the filter *awarenessOnNode* is similar to that described above, with the only difference being that, except for the case of document level, a document node needs to be selected by the user. We refer to this as the *zoom-in* functionality and discuss it in the following subsection.

4.3.3 Advanced awareness features

One expedient feature of edit profiles is that they are not separated from the main document. On the contrary, a user can be directed from the edit profile to the document with a *single left click* on the node of interest. If, for instance, the paragraph level is selected and a user clicks on the bar corresponding to the last paragraph, that paragraph will be highlighted as shown in Figure 4.7. The same applies to all nodes, at all document levels.

Finally, the edit profile is enhanced with a feature that returns information about a specific document part. Consider, for example, the case where a user notes that a specific paragraph seems to have undergone major changes and wants to see how the total number of changes in the paragraph is distributed to the sentences of that paragraph. By double clicking on the corresponding bar in the histogram, the user zooms in one level and sees the total number of changes computed for the direct children i.e. the sentence-level children nodes. This action activates the *awarenessOnNode* function.

4.4 Integration process

In this section, we present the integration process of the awareness mechanism into the asynchronous text editor. The upper half of Figure 4.11 presents the editor before applying our awareness framework and the lower half after applying the framework. Below, we discuss in detail the state of the editor before applying the awareness framework followed by the changes made to the editor to enable the framework's integration and the editor's functionality after applying the framework.

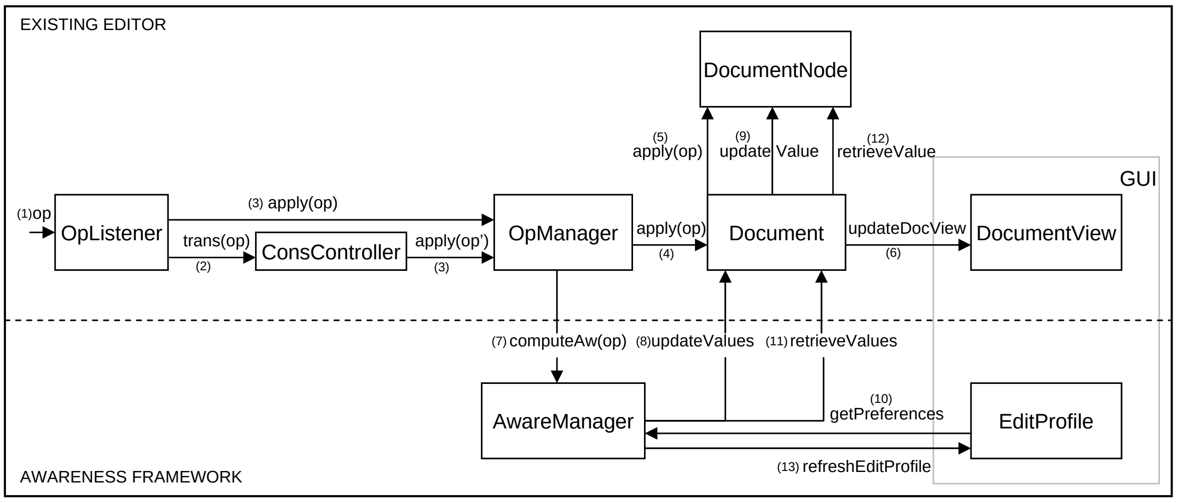


Figure 4.11: Editor's functionality before and after the integration of the awareness framework

Editor's functionality before applying the framework: Changes are made to the document both from the local and remote sites. A listener, *OpListener*, registers itself to listen for all the operations created (1). Local operations are applied to the *Document* (4) through the *OpManager* (3), while the remote operations are initially transmitted to the *consistency controller* (2), where they may be transformed according to the consistency algorithm used and finally are sent to the *OpManager* (3) to be applied to the *Document* (4). The *Document* component presented in Figure 4.11 represents the root of the hierarchical document. For an operation to be applied to a document, the hierarchical document needs to be scanned to find the *Document Node* where the operation should be applied (5). Once the *Document* is updated and the severity of the operation(s) included into it, the *Document View* in the *GUI* is also updated (6).

Changes made to the editor to allow the framework's integration: To include the computation of the enriched awareness information, the *Document Node* objects were extended to include the *value* objects introduced in the previous section. Further, an awareness visualisation tool, namely the *Edit Profile*, was added to the *GUI*. To have the editor communicate with the awareness framework as seen in Figure 4.11, *OpManager* holds a reference to the *AwareManager* and the *AwareManager* holds a reference to the *Document* object.

Editor's functionality after applying the framework: In the extended collaborative editor, the computation of awareness is triggered by the *OpManager* for each new operation received (7). The severity of the operation is computed for all available metrics, and the *value* objects of the changed *Document Node*s are updated by the *AwareManager* (8, 9). Finally, the *AwareManager* updates the *EditProfile* (13) to present the *value*s retrieved (11, 12) from the *DocumentNode*s according to user preferences (10).

4.5 Summary

In this chapter, the integration of our awareness framework into an existing collaborative editor is described. To enable this integration, the selected editor satisfied the requirement corresponding to the underlying document structure. The documents handled by the editor are represented by a *structured document model* and modifications made to them are modelled as *operations* applied to the document parts.

Based on the editor's available functionality and our expectations about user preferences and needs, we extended the core concepts of the framework metamodel. That included the materialisation of the *node*, *operation*, *metric* and *visualisation tool* concepts. Of course the metrics and the visualisation tool proposed in this chapter need to be tested to see whether they can easily be used by the users, whether they present the expected information and whether this information is interpreted in the expected way by the users. We used the resulting editor, presented in this chapter, to conduct a qualitative user study testing all the above. The set of our research questions, details on the study, as well as the users' feedback are presented in the following chapter.

5 Qualitative user study

In the previous chapter, we presented a prototype of an asynchronous collaborative editor enhanced with awareness, through the integration of our awareness mechanism. An edit profile was introduced as the visualisation tool used for the prototype. We expect that the resulting enhanced editor will provide multi-level change awareness to users working with it. To test this assumption, we used the editor to conduct a qualitative user study. We begin this chapter with a presentation of the research questions to which we wanted to provide an answer through the user study. Then we present the study design and finally the study results and a discussion on them.

5.1 Research questions

We start with a summary of the new features offered by the editor. The enhanced collaborative editor

- presents information about, not only the location of a change, but also its severity to a specific document part and to the document as a whole,
- presents an overview of all changes made to the document using an edit profile,

- presents information about different types of changes, on various document levels and document parts,
- presents information about who made the changes,
- provides an interactive graph for the user to easily access "problematic" document parts, or zoom into different levels and document parts, and
- allows different metrics to be defined and used according to application requirements and user needs.

We believe that such an editor provides users with more awareness information than existing editors. In the user study, we investigated the following issues.

- *Does the presence of an edit profile increase the awareness provided to the users?* More specifically, we investigated whether the presence of an edit profile assists users to quickly be informed about the modifications in a document, and whether it helps them to spot "problematic" document parts, i.e. document parts that have undergone many modifications and therefore may need to be checked.
- *Do users find it useful to be informed about changes made to the document on different document levels?* In detail, we investigated whether awareness about changes at different document levels increases the awareness of users about the details of modifications and whether it helps them track modifications through the different document levels. In addition, we investigated whether information on all different levels is needed.
- *Do users find it useful to be informed about changes made to the different document parts?* Similarly, we tested the reaction of users on receiving awareness information about different document parts, whether they believed that they needed this detailed information and how it could be used.
- *Do users find it useful to decide on the granularity of the changes that will be reported through the overview?* We investigated how users interpret the information returned by the edit profile for the various metrics. Are all metrics needed by users? In which collaborative situations would users use each metric?

- *Could the information presented through the edit profile be used to judge a document's evolution and quality, the quality of a person's writing skills, or any writing patterns?*

Given the restricted functionality of the editor prototype and the nature of research questions to be answered, we decided to conduct a qualitative rather than quantitative user study. Our aim was to get some initial feedback from users working with the enhanced editor and use it to redesign, if needed, the awareness framework and the prototype.

Additionally, we decided against conducting a user study where the awareness features of the prototype would be compared to the awareness features of existing awareness-enhanced systems. In such a study, users could be questioned about the level of another user's involvement in the authoring of a document's session using our prototype and another system. The correctness of their answer as well as the time needed to extract this information from each system could then be compared. We believe that such user studies are beyond the scope of our work at this point. However, once the concepts of the framework are accepted by the users and the implementation of our systems is beyond the level of initial prototypes, such comparative user studies would be very interesting.

5.2 Study design and methods

We carried out a qualitative user study based on a usability test. We defined as target user group for the asynchronous editor, people who were familiar with collaborative authoring in a scientific working context. Ten persons volunteered to participate in our study. All of them had an academic background in the field of engineering, sciences or economics.

The number of participants required for a qualitative user study is an issue that attracts much attention from researchers conducting usability studies [28, 92, 93]. While researchers agree that the number of subjects required for a qualitative user study is much lower than for a quantitative user study, there is no number of participants accepted by all researchers as generally adequate for such a study. We believe that this is reasonable since different user studies can have different requirements with regard to the research questions to be answered and the detail of the required results. Therefore, the number of participants required for user studies with different requirements will also differ.

It was not the intention of the user study presented in this chapter to collect statistical data, but rather to obtain the initial users' feedback and the level at which they understand, accept and correctly interpret the core concepts included in the prototype and the awareness framework. We therefore believed that the testing of ten participants would return adequate feedback.

Each session of the user study began with an introduction to the test procedure and a brief verbal tutorial to the editor since none of the participants had worked with it before. After filling in a short pre-questionnaire, each participant worked on five tasks that included the use of all of the editor's features. No time limit was set for completing this part and it lasted, on average, around 20 minutes. Participants were asked to speak aloud their intentions and any remarks they had while performing the tasks. After the task completion, participants filled in a post-questionnaire. The questions in all of the questionnaires had one of the following formats:

- statements to be ranked on a five point Likert scale
- binary (yes/no) questions
- multiple choice questions
- open-end questions

In some of the binary or multiple choice questions, the users were also asked to give reasons for their answer. The open end questions were mainly used to ensure that the participants interpreted correctly the information displayed through the edit profile. For instance, they were asked to assign roles to the users shown on the profile or decide on the document parts each user owns based on their editing activity, or to find hot spots in the document. After the task part and the questionnaires, there followed a brief informal and unstructured interview to discuss specific results from the observations and questionnaires.

Each session took about one hour and was guided by an experimenter who observed the test and answered the participants' questions. The whole session was recorded on video and, during the task part, the screen was captured as well, as shown in Figures 5.1 and 5.2. The results from the questionnaires, observations and informal interviews were later transcribed into a detailed report.

During the task part, the participants were presented with a typical situation where three persons, i.e. two students and their professor, collaboratively write a document using the asynchronous text editor. Participants were asked to assume that they had joined the group as students

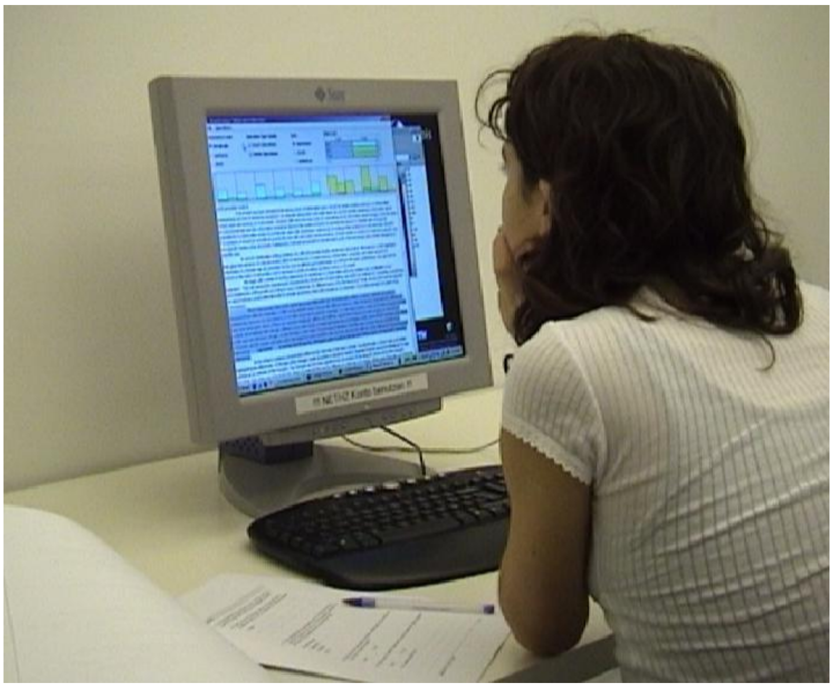


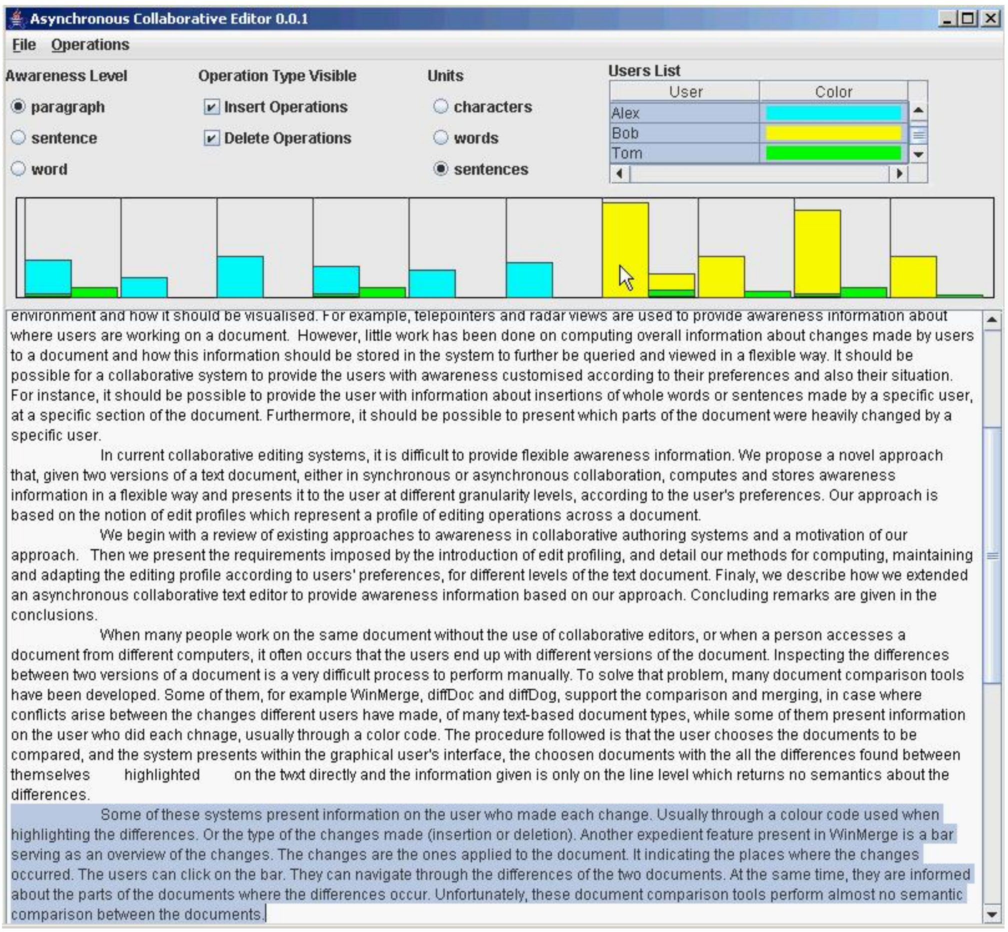
Figure 5.1: Example of a user interacting with the awareness enhanced editor in the frame of a user study

in order to help write the paper. The given tasks included interaction with the editor to explore different parts of the document and to interpret the overview in order to catch up on the document's evolution from its creation to the current version.

One pilot test with one out of the ten participants was performed. As the pilot revealed no inadequacies of the test setup, we decided to also take these results into account. The questionnaires used for this study can be found in Appendix A.

5.3 Results

The study considered the research questions presented in the beginning of this chapter. In summary, we checked whether the users could quickly and easily be informed, through the document overview, about the changes that each user made, as well as the role of each user and what parts of a document were "owned" by each user. Additionally, we investigated how users interpreted the information computed based on each of the metrics and which metrics they found more useful. Furthermore, we checked whether they liked the idea of an interactive visualisation tool as a means of navigating through the modifications made to the document



Awareness Level: [✓ paragraph] sentence word

Operation Type Visible: [✓ Insert Operations] [✓ Delete Operations]

Units: characters words [✓ sentences]

User	Color
Alex	🟦
Bob	🟨
Tom	🟩

environment and how it should be visualised. For example, telepointers and radar views are used to provide awareness information about where users are working on a document. However, little work has been done on computing overall information about changes made by users to a document and how this information should be stored in the system to further be queried and viewed in a flexible way. It should be possible for a collaborative system to provide the users with awareness customised according to their preferences and also their situation. For instance, it should be possible to provide the user with information about insertions of whole words or sentences made by a specific user, at a specific section of the document. Furthermore, it should be possible to present which parts of the document were heavily changed by a specific user.

In current collaborative editing systems, it is difficult to provide flexible awareness information. We propose a novel approach that, given two versions of a text document, either in synchronous or asynchronous collaboration, computes and stores awareness information in a flexible way and presents it to the user at different granularity levels, according to the user's preferences. Our approach is based on the notion of edit profiles which represent a profile of editing operations across a document.

We begin with a review of existing approaches to awareness in collaborative authoring systems and a motivation of our approach. Then we present the requirements imposed by the introduction of edit profiling, and detail our methods for computing, maintaining and adapting the editing profile according to users' preferences, for different levels of the text document. Finaly, we describe how we extended an asynchronous collaborative text editor to provide awareness information based on our approach. Concluding remarks are given in the conclusions.

When many people work on the same document without the use of collaborative editors, or when a person accesses a document from different computers, it often occurs that the users end up with different versions of the document. Inspecting the differences between two versions of a document is a very difficult process to perform manually. To solve that problem, many document comparison tools have been developed. Some of them, for example WinMerge, diffDoc and diffDog, support the comparison and merging, in case where conflicts arise between the changes different users have made, of many text-based document types, while some of them present information on the user who did each chnage, usually through a color code. The procedure followed is that the user chooses the documents to be compared, and the system presents within the graphical user's interface, the choosen documents with the all the differences found between themselves highlighted on the twxt directly and the information given is only on the line level which returns no semantics about the differences.

Some of these systems present information on the user who made each change. Usually through a colour code used when highlighting the differences. Or the type of the changes made (insertion or deletion). Another expedient feature present in WinMerge is a bar serving as an overview of the changes. The changes are the ones applied to the document. It indicating the places where the changes occurred. The users can click on the bar. They can navigate through the differences of the two documents. At the same time, they are informed about the parts of the documents where the differences occur. Unfortunately, these document comparison tools perform almost no semantic comparison between the documents.

Figure 5.2: Captured screen during the user study

and zooming into different parts of it. The results that we gathered are organised based on these issues and presented below.

Participants' previous experience

In the pre-questionnaire, we asked participants to list the different tools they had used when involved in collaborative authoring tasks. They reported to have used MSWord (n=7), various editors for Latex files (n=4), SVN (n=3), CVS (n=2) and Excel (n=2). We observed that similar results were presented in previous studies as well [72, 94, 109]. We have chosen not to report the editors used by only one user. During the past five years, all participants had worked on five or more collaborative authoring tasks. Most of the participants (n=9) assessed their computer literacy to be good and one assessed it to be average.

Edit profile

All participants liked the idea of an overview and had no problem interacting with it. A characteristic reaction was: "It is really easy to use. If you have the basic knowledge of a word processor, this is an interactive kind of it."

The participants reported that they would like such a feature in a collaborative editor, "to get a first feeling" about the changes made to the document. As presented in Figure 5.3, most of the users (n=9) either agreed or strongly agreed on the statement that it was useful to present the changes made to the document through the overview. Only one participant said that he disagreed, but later in the discussion, he specified that he meant that he did not like the way in which the visualisation tool was built in the GUI. When he was asked whether he would like to be informed about the changes made to a document through the overview, he said: "Yes. That would be ok. The overview is nice at the beginning ... when you get the newest version (from the repository or from another user)." This remark was made by other users as well. One of them said: "I would use it when I log in to notice where someone has worked."

One participant reported: "(The overview) was really useful because you can see the changes in a matter of seconds and you can eliminate any user you want." Another user said: "Overall, I would say the graph gives me a good sense to quickly locate, if I am in a hurry ..., where the changes are made." A third user, when asked if he would like to have the awareness information displayed in the overview, compared the editor with the one he used most and said: "In general it is much more useful than when you use ... (he mentioned the other editor) which has all these balloons ... it is really a mess. This ... (our editor) is much friendlier. You can see on a more abstract level ... not just focusing on every small detail."

All of them reported that they would not be able to get the same information without the use of the overview and that the overview offered sufficient information on the exact place in the document where the changes were made. However, 4 of them said that the information about the amount of the changes was not enough. This was mainly due to the fact that there was no information provided by the tool about the actual changes on the text document itself. This confirms our expectation that the edit profile provides the needed overview of all changes made to a document, before checking them into detail. In a second step, if the user wants to check the changes in more detail, a tool that shows the changes on the text is required.

Document levels

A novel characteristic of our approach is the fact that awareness information is computed not only for the document as a whole [86], but also for document parts which correspond to meaningful syntactic units and not just lines [64]. We tested the above argument in our study. The participants were asked to visualise the changes made to the document at the different document levels and try to find the parts of the document that had undergone major changes by zooming in on the different document levels. As presented in Figure 5.3, five participants *disagreed* and five *strongly disagreed* with the statement that it was not useful to present awareness information on different document levels.

Document parts

Another expedient characteristic of our approach is that, in addition to the information computed and presented on different document levels, we provide the user with information about different document parts as well. In our study, we asked the participants to find the paragraph where insertions but no deletions were made, to zoom in, see how the total amount of changes in the paragraph were distributed in the paragraph's sentences and interpret the information displayed.

All of them found the correct paragraph, zoomed in successfully and commented on the changes presented on the sentence level. In the example paragraph used for this task, the users saw that the increased amount of changes presented for the specific paragraph was due to changes that modified only a specific part of it. The corresponding sentences at the specific paragraph part were heavily changed by all users. The rest of the sentences were edited by only one user. The participants reported that this could mean that either the other users had not read that part or they agreed with what was written. When asked whether they liked the zoom in functionality, they all answered affirmatively. Figure 5.3 additionally shows how much they liked the zoom in feature.

Metrics

To test whether our approach delivers more information in comparison to other approaches due to the use of multiple metrics, we required that the participants locate, with the help of the overview, the paragraph with

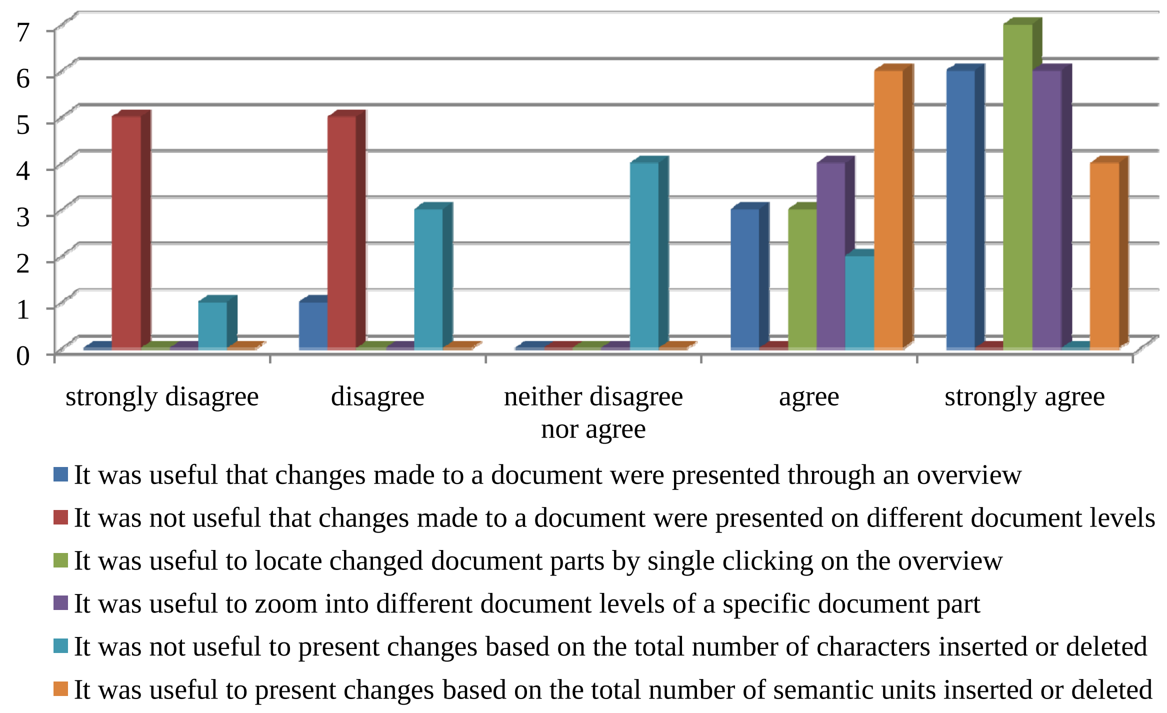


Figure 5.3: Users' feedback on Likert scale questions from the post-questionnaire

the most changes in terms of complete sentences inserted or deleted and the one with the most changes in terms of the total number of characters inserted or deleted. All participants navigated easily between the profile views for the two metrics.

After successfully locating the required paragraphs, they were asked to comment on the results. They reported that: "the paragraph with the most complete sentences inserted was not accurate in a higher level of understanding" while the second one "was not accurate in orthography". Another user mentioned: "on a structural level, it makes me assume that the changes on the second paragraph were minor, whereas in the first paragraph they were major and probably had to do with its structure." This verifies our hypothesis that depending on the metric chosen, the information presented can be interpreted in different ways and lead to different conclusions.

One of the questions we included in the post-questionnaire was whether the participants would like to be informed about changes made to a document based on the total number of characters, or based on the total number of syntactic units of a higher level, i.e. words and sentences, or based on both. We observed that, although all of them were able to see the difference in the results returned from the different metrics, only three of them chose to be informed in both ways. Six participants would prefer to be informed only based on the numbers of higher level syntactic units and only one chose the characters only. This answer justifies the results in Figure 5.3 on how useful they found the fact that awareness information was computed according to the number of characters and according to other syntactic document units.

A user said that he would find the information delivered for the paragraph with the most complete sentences inserted or deleted "much more useful semantically, because when you write a paper, you want to see if some additional information has been delivered in that paragraph. If you have just more characters inserted it probably meant you have used different terminology." Another participant pointed out: "If I see changes on the sentence level it means I should read the paragraph again because the meaning has changed. If I see it on the character level, it means it changed slightly."

We conclude that the users, for the specific task given, would prefer to get information about "major" changes made to a document, i.e. changes that had to do with insertions and deletions of units on a higher level than the character level. However, they were aware that their decision was based on the specific task given and they mentioned: "If somebody has to do a proofreading or spell check it would be interesting to have (the information computed on) characters."

Working patterns, user roles

In addition, we were interested in investigating whether information about user roles, the quality of a document or a user's writing skills could be extracted through the edit profile. We believe that if a user study were to supply detailed information on these queries, it should be constructed with relevant tasks. Although our study was not built on such a concept, we obtained some important feedback from the participants and therefore report on it. When the participants were introduced to the tasks, they were not informed about the other users' roles. However, only by being aware:

- of the number of participants with each role,
- what each role included in terms of changes made to the document or ownership of document parts and
- by checking the overview,

they were all able to judge successfully the role of each of the users and also the owner of each document part. For instance, they all reported that Tom, as shown in Figure 5.2, was the professor because "he made few changes to all paragraphs" and although "he was not responsible for a specific paragraph, he edited all." Here, the users demonstrated that they were able to assign roles to users based on their editing activity given that the roles and each role's tasks were specified. It would be interesting to further investigate whether a user's editing activity observed through an edit profile can deliver additional information about users and their working patterns.

Usability Results

Alongside the results relevant to our research queries, the user study provided us with usability results as well. Here we report some of our observations. None of the users showed difficulty in using the available GUI features. They navigated easily between the document levels and easily visualised the changes of different users computed with all of the available metrics. In addition to that, they also liked the fact that different types of changes, i.e. insertions and deletions, could be visualised separately. We observed that they made heavy use of this feature in an attempt to see, for instance, the insertions without being distracted by the bars corresponding to the deletions. Additionally, as reported in Figure 5.3, they rated the *single click* functionality as very useful.

However, we observed that in certain cases, the users were not able to easily interpret the information given by the overview. This was due to the fact that there was no labelling and no absolute values on the x-axis of the overview. The problem arose when users navigated between different document levels or when they zoomed in on different document parts. For instance, they could not compare the values visualised in the overview on the paragraph level with those on the sentence level. In addition to that, some users lost orientation when they zoomed in and had to go back to a higher level and zoom in again to ensure that they zoomed in on the desired document area.

5.4 Discussion

In what follows we discuss our study results presented in the previous section.

Edit profile

As shown by our user study and also reported by Gutwin et al. [58], users like the idea of a document profile that presents the sum of changes made to a document. It provides a simple means of making users aware of "hot areas" and also who is or has been active in various parts of the document. Our hypothesis that the overview would help users to get information quickly and on an abstract level before going on to check changes in detail was confirmed by the users. All of them were able to spot the paragraph or the sentence with the most changes and report successfully on the document parts that each user had worked on. Additionally, we observed that they interacted with it and got all the information needed in a very intuitive way. That is reasonable if we consider that the overview is just a linear representation of a document.

Document levels

We observed that the feature of presenting awareness information on different document levels received positive feedback from the users. They demonstrated no difficulty in navigating between levels. We estimate that the reason for this is the fact that the document levels defined in our approach correspond to the syntactic elements used in natural language. However, the different levels available are application-dependent. For instance, when asked how useful it is to display the awareness information at both the paragraph and the sentence levels, one participant answered: "*neither disagree nor agree* for the sentence level and *strongly agree* for the paragraph level". This raises another point, also mentioned by another user: "I don't know how (the overview) would be in a document with more paragraphs." The same could occur even in a small document, when trying to visualise the changes at the word level. This is a proof that, as with any feature, the overview should be used with care and the developers of a system should be careful not to load the system and the user with too much information. Under this precondition, the awareness information provided by a system can be heavily ameliorated by the use of an overview.

In order to be able to provide all the information computed by the metrics without overloading the user with meaningless information, we believe that some improvements could be made to the system. For instance, it is necessary to extend the document model and the editor to include also "higher" document levels, i.e. section or chapter levels, to efficiently compute and present changes made to longer documents as well. If information on lower levels, for instance the word level, is still needed, then the overview could be filled with information only about the nodes that had changed. In such a case, the position of a bar in the overview may not reflect the position of the node in the text, but this information could additionally be delivered by extending the visualisation tool.

Document parts

An advantage of our approach over existing ones such as TenDaX [65] is that the document parts correspond to syntactic parts of a document defined as in natural language, and not in document regions of no syntactic meaning. For example, a user can request information about changes made on all the paragraphs of a section or all the sentences of a paragraph and so on. This information is provided to the users of our system by allowing them to zoom in on different parts of a document. It was shown that, by zooming in, the participants could extract more detailed information about the changes made to the document part under inspection, which means that the users were given the possibility to easily be informed, at different levels of detail, about the way document parts were changed.

Metrics

We believe that our approach delivers more information in comparison to other approaches since the user decides on the granularity of the information to be computed and presented according to application needs as well as user roles or circumstances. For instance, if, in a publishing company, an author is looking for spelling mistakes corrected by a proofreader, they should choose to compute the awareness values based on the total number of characters and then visualise the results at the word document level. On the contrary, if a professor of a research group wanted to check whether a research paper is still under major reconstruction, they would probably choose to monitor the number of complete paragraphs or sentences inserted or deleted.

The flexibility of our approach in computing awareness by monitoring and counting changes on units of different syntactic levels was appreciated by all participants. It was reported that each of the metrics defined could be used for specific tasks and user roles. This shows that depending on the metric chosen, the information presented can be interpreted in different ways and lead to different conclusions.

Working patterns, user roles

Evaluating the study's results, we conclude that, according to users, the overview could be used to judge the quality of a document and also individual authors. It would be interesting to further investigate this and find ways to present this information according to application requirements.

Usability Results

The usability test showed that users had no difficulties when using the editor. This suggests that the tool is intuitive and easy to use. Additionally, users rated the single-click functionality as very useful because they could easily navigate through the edit profile to "interesting document parts". That was expected since such a feature was reported as missing by participants of the study conducted by Gutwin et al. [58]. The fact that some users lost their orientation while zooming in on the different document levels could be solved by providing the user with some additional information through the edit profile to help them orientate themselves.

5.5 Extensions based on users feedback

In addition to the possible extensions of our editor to address the issues that arose from the usability results, we intend to extend our approach by providing even more functionality. Since we were interested in investigating how the users would react to some of the extensions that we have already planned, we included some relevant questions in the post-questionnaire. Here we concentrate on the results collected about two possible extensions. The first concerned the computation and visualisation of awareness information on different versions of the document and the second on enhancing the asynchronous collaborative editor with synchronously updated information about changes made by other users. Here we report on the first one. Details on user feedback about the second extension are given in the next chapter, since they were used as part of the requirements set defined for the definition of the "shadow document sets" and the "BeAware" workspace.

The information presented by the overview in its current form represents all of the changes made to the document from the moment of its creation up to the current version. We asked the users whether they would like to “be informed through the overview about how the document evolved from one version to another”. They all answered affirmatively. Most of them had even noticed before completing the post-questionnaire that this functionality was missing and reported that they would find it very helpful.

5.6 Summary

Based on the users’ feedback through questionnaires and discussions, we have reported our comments and observations with regard to the research questions introduced in the beginning of this chapter. The users appreciated and highly rated all of the features proposed by our approach. They gave positive feedback about the use of an overview that presents the sum of the changes made on a document. They appreciated the possibility of quickly accessing all of the “important”, i.e. heavily changed, document parts as well as being able to be informed about the changes made on a document at a more abstract level before going into details.

The presentation of the changes made to a document at different document levels and document parts was also highly rated, as well as the zoom in functionality. By combining all of the above features, the users could filter the computed awareness information and visualise the part of it that was appropriate to the current situation. Additionally, we observed that the users benefited from the various metrics available. They realised that, depending on the application and their roles, they could specify the granularity of the changes to be monitored and presented through the edit profile.

The above results suggest that edit profiles and in general our awareness framework can successfully provide multi-level awareness. However, by no means do they show that the current implementation and the features provided by the prototype are enough for users. We believe that the awareness information computed by the prototype can be increased if the awareness features included in the prototype are combined with features of other existing awareness mechanisms. User studies where our prototype is combined with other awareness-enhanced applications would then be required to identify the advantages for the users deriving from the combined features.

6 Semi-synchronous collaboration

In the implementation of the asynchronous editor presented in Chapter 4, users work in isolation and are not informed about changes being made concurrently on the local copies of other users. The only way users can be informed about changes from other users is by updating their copies against the newest (if any) committed version of the document. However, this means that if more than one user has worked on the same document part, possible conflicts to be resolved may arise. In the worst case, the changes of one user might have to be discarded, which could be disappointing and time-consuming for the users.

To prevent these issues from arising, we propose a new feature that asynchronous editors could provide. While the users work on their local copy, in privacy, each user's overview is updated synchronously to include the severity of the changes made by other users on their local copies. However, the remote changes are not applied on the local copy of the main document since the users have chosen to work asynchronously on a document version that reflects only their changes.

In this chapter we introduce the concept of "shadow document sets" that allows for synchronously-updated awareness in asynchronous environments. We present our approach in detail, along with a prototype of a shared workspace for asynchronous collaboration based on "shadow document sets", the "BeAware" workspace. We start with an example of a collaborative activity that motivates the need for synchronously-updated awareness information in asynchronous collaboration and presents in detail the information that users often need. Using the example, as well as users' feedback from a user study we conducted, we collect the full set of requirements in terms of awareness features an application should provide to successfully address the above problems. We continue with the introduction of shadow document sets, the implementation of the BeAware workspace and a presentation of its advanced features.

6.1 Motivation and requirements

To further motivate our work and establish a set of requirements, we use an example of three users, Mary, Bob and Tom collaboratively authoring a document using the enhanced asynchronous editor of Chapter 4. The users work in privacy but are connected to a network. We assume the existence of a central repository where the document versions are kept, available for download by all users. The application implements the copy-modify-merge paradigm. As a result, when users want to edit a document, they download the latest version from the repository, make the changes and finally publish the changes to the repository. If more than one user is working in parallel, when a user publishes their changes, the application searches for any recent published versions of which the user is not informed and performs a merge of the user's changes with the changes of other users reflected in the version already published. Only after the merge has been performed will the user changes be published. Every time a user downloads a version from the repository, or updates their local version, they are informed through the edit profile about the changes made to the document by other users.

As explained in the previous section, we want to support users working in parallel by providing awareness information in real-time. We assume that all three users work in privacy, modifying their local copy of version V_n downloaded from the repository. While Mary works on her copy, Tom and Bob also make changes and Tom publishes his changes. Mary needs to be informed about the following:

- Changes that were made to the document since the last time she logged in. This is information that needs to be presented when Mary logs in and should be available at any time throughout the current authoring session. If version V_{n-k} was the last version that Mary is aware of, the awareness information presented to her when she logs in has to include all the changes made on the last k document versions.

- Changes (not yet committed) made by Tom and Bob at their local copy of version V_n, while Mary is working on her copy of the same version, i.e. concurrent uncommitted changes.
- Any new version published on the repository which Mary's document copy is not informed of, i.e. concurrent committed changes.

The three different kinds of changes are presented in Figure 6.1. There are concurrent uncommitted changes made by Bob, concurrent committed changes made by Tom and the changes that transformed the document from version V_{n-k} to version V_n. When Mary downloads the document version V_n she can be informed about the changes made to the document since her last login through an edit profile as already presented in Chapter 4. To inform Mary about changes made by Bob and Tom after her login, we need to allow changes made by a user to be sent in real-time to all other users. Such a requirement can be easily fulfilled and is already implemented for synchronous collaborative tools. However, tracking all the changes and applying them to Mary's copy is not desired, since the advantages of asynchronous work will be lost. Since Mary chooses to work in privacy, her document copy should not be modified. Similarly, since Bob and Tom choose to work in privacy, their changes should not be published to Mary unless they explicitly choose to do so. The solution is to make Mary aware of the *severity* of the changes of other users without visualising the actual changes in the document.

This could be implemented by computing the severity of each operation, and informing the awareness mechanism about the operation applied to a specific part of the document, without actually applying the operation, i.e. without modifying the document. Unfortunately, such an approach would not be satisfactory since it could lead to a corrupted document. If, for instance, Bob inserted a whole new paragraph in the document and made further changes to it, the awareness information computed for these operations could not be assigned to any document part in Mary's document copy, since the new paragraph would not exist. In addition, such an approach would create a document copy including all changes, as presented in Figure 6.1, and would make it difficult to present them separately through the awareness mechanism.

As a result, different kinds of changes need to be handled separately and kept in different document instances. Additionally, special care needs

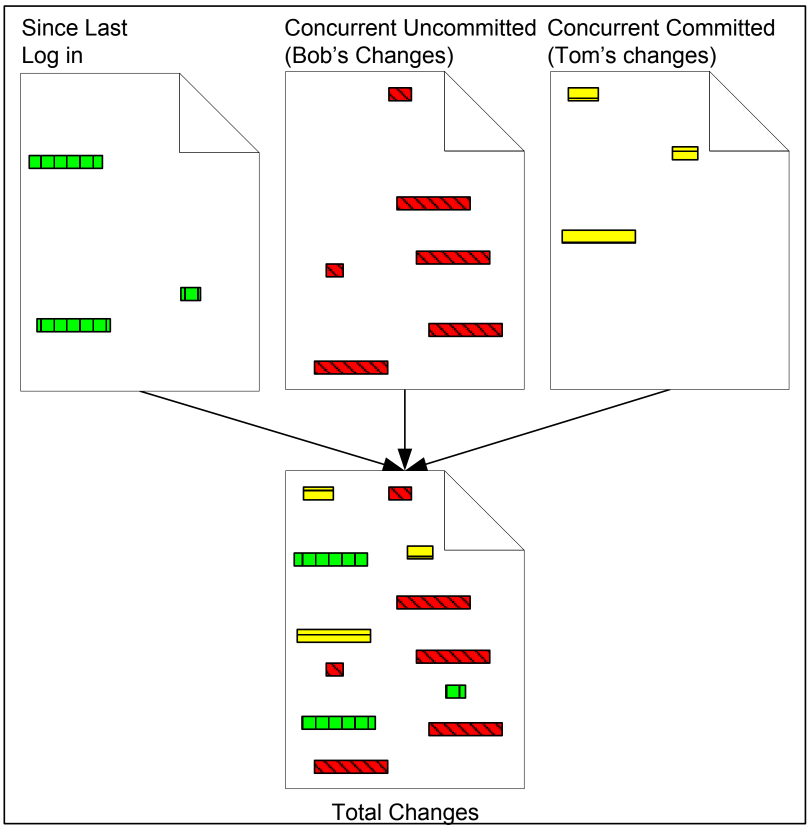

Since Last Log in
Concurrent Uncommitted (Bob's Changes)
Concurrent Committed (Tom's changes)
Total Changes

Figure 6.1: The current status of the document is composed by the sum of all users' changes

to be taken to ensure that the awareness mechanism will inform Mary about all the changes made by other users, without modifying her own copy of the document.

The importance of awareness information provided by an asynchronous collaborative tool was also highlighted by the users of the user study presented in the previous chapter. The asynchronous collaborative editor used in the study included the computation and visualisation of awareness information based on the notion of *edit profiles*. The awareness information presented to the users concerned only changes that users made to the document in the past, while transforming it from its initial to its final version. The users were not informed about changes made to the document by other users in real-time. In the post questionnaire, we described to users how we envisioned a synchronously-aware asynchronous editor, asked for their feedback and collected any new requirements set by them.

In response to the question of whether they would like to be provided with real-time awareness information about changes made by other users to their document copies, most of the users (7 out of 10) said they would like to have such functionality in an asynchronous editor. They added that if they were informed that some other users were working on a document, or some specific areas of it, they would probably not interfere "to avoid conflicts and therefore decrease the time necessary to edit that part". This confirms our initial assumptions that such an approach would support users working asynchronously and gave us a strong motivation for the work presented in this chapter.

Further requirements were formulated from the discussion with those participants who were sceptical about the proposed feature. The participants reported that although they would like such a feature, they were worried what it could cause confusion or disturbance. If, for instance, a user was informed that many other users were simultaneously editing the same document, they might be frustrated and stop working. Other participants mentioned that they would not like to be informed about changes that other users had made and were not yet committed.

We examined all of these issues and formulated our approach. Our goal is to enable asynchronous collaborative editors to additionally:

- offer the computation and visualisation of awareness information about changes being made to a document in real-time,
- inform a user about the severity of other users' changes without modifying the user's document copy,
- present information concerning changes currently made but not yet published separately from those currently made and already published,
- inform the users without causing disruption or loading them with too much information.

All the above should be combined with the more general requirements which we presented in Chapter 2 and can be summarised under

- offer all of the above information on a level of granularity specified by the users.

In the following section, we describe our approach by introducing the notion of *shadow documents*.

6.2 Shadow documents

When users asynchronously edit a document, parallel copies of it are created for the users to work on. In our example, when user Mary logs into an asynchronous application, she downloads a copy of the latest version of the document published in the repository. This local document copy presented in Figure 6.2a is the view of the document presented to Mary and will be used when Mary makes any further changes to the document.

Awareness information about changes made to the document by other users is provided through an edit profile, the visualisation tool used by the application. The awareness information provided by existing collaborative applications concentrates on published changes which have been made to the document since the user last logged in. In our example, these are the changes that transformed version V_{n-k}, the last version Mary is aware of, to version V_n. The awareness information computed from these changes will be presented as indicated in the left profile of Figure 6.2b, called *Since Last Login (SLL) profile*. However, Mary also wants to be informed about changes, committed or uncommitted, made by other users, while she is working on her local copy. We propose the use of two more edit profiles, the *Current Uncommitted (CU) profile* and the *Current Committed (CC) profile* presented in Figure 6.2b, to visualise this information.

The existence of three different *edit profiles* results from the existence of three different types of changes. For the information presented in an *edit profile* to be computed, a structured document is needed. Therefore, three structured documents are required to hold the changes. These are not visible to Mary. We therefore call them *shadow documents*. When Mary logs in, the *Since Last Login (SLL) shadow document* is updated to hold all the changes Mary is not aware of. At this moment, this shadow document is Mary's local document copy. Later, it will additionally hold the changes made by her during the current session. While she is working on her document copy, the *Current Uncommitted (CU)* and the *Current Committed (CC)* shadow documents will be updated according to changes made by Bob and Tom as presented in Figure 6.2d,e,f.

When Bob and Tom start editing the document, Mary is instantly informed. The changes created by Bob and Tom are sent to Mary's site where they are applied to the *CU shadow document* as presented in Figure 6.2e. This shadow document will always present information about changes made concurrently by other users while Mary is working on

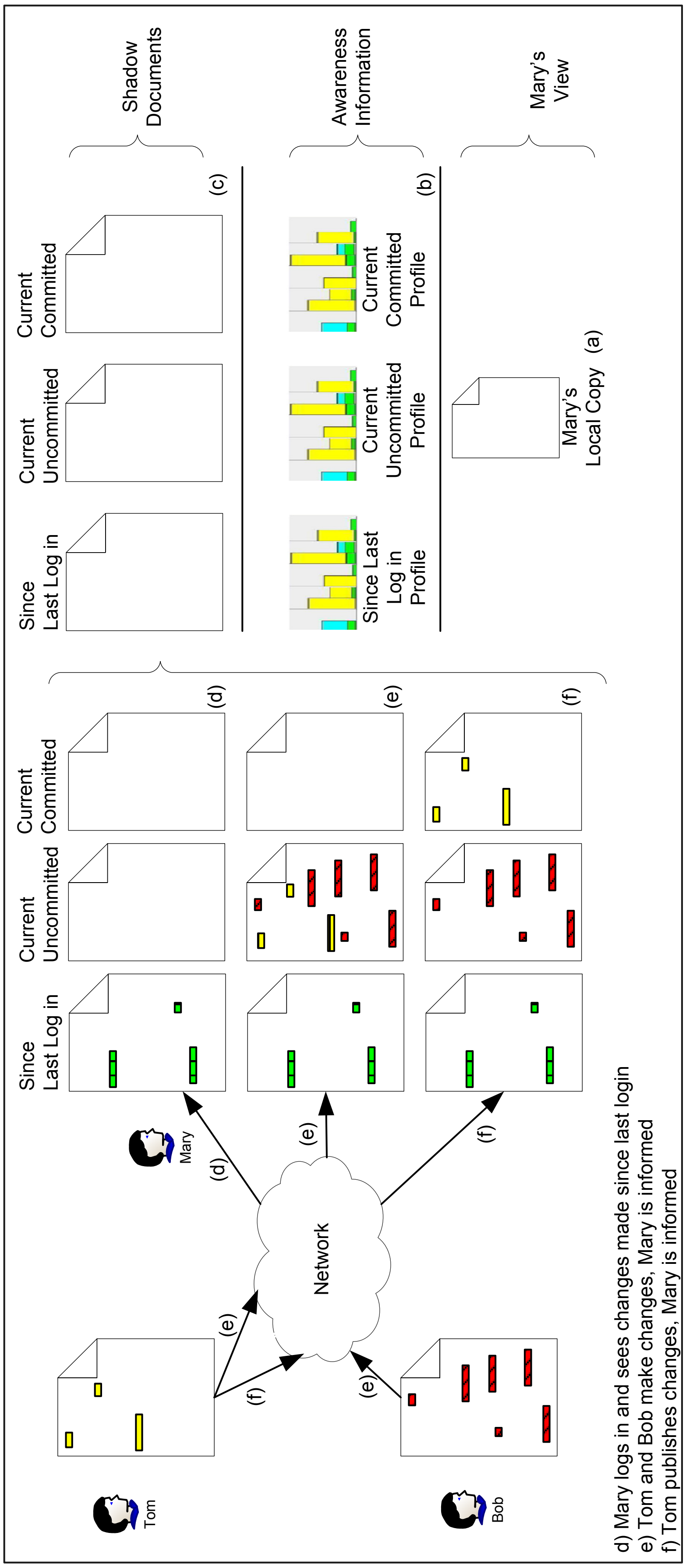

Tom
Bob
Mary
Network
(d)
(e)
(f)
Since Last Log in
Current Uncommitted
Current Committed
Shadow Documents
Awareness Information
Mary's View
Since Last Log in Profile
Current Uncommitted Profile
Current Committed Profile
Mary's Local Copy
(a)
(b)
(c)
d) Mary logs in and sees changes made since last login
e) Tom and Bob make changes, Mary is informed
f) Tom publishes changes, Mary is informed

Figure 6.2: Content of shadow documents in a typical scenario of asynchronous collaboration

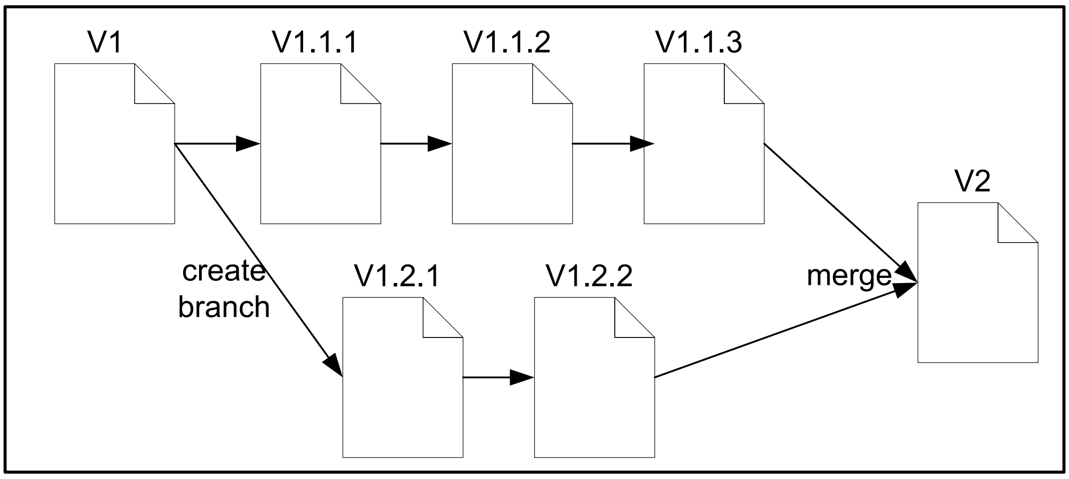


Figure 6.3: Example of a typical versioning system

her copy and which are not yet published. If some changes are published to the repository, Mary is informed and the *CC shadow document* holds this information as shown in Figure 6.2f. Changes that are published by other users and are committed have to be “extracted” from the *CU shadow document* and applied to the *CC shadow document* so that the *CU shadow document* will constantly present only uncommitted changes. Propositions on how this can be done are given in Section 6.5.

Classical document versioning approaches were our inspiration for the approach presented above. Versions are used extensively in database applications, computer-aided engineering [71] and software development [5, 27]. In all of these applications, a version is created every time a user makes a set of changes to the document being authored. The most up-to-date information is delivered by the most current version. If a document is being edited concurrently by more than one user or application, then branches can be created to trace the parallel editing as shown in Figure 6.3. When the parallel editing is finished, if only one final version is required to describe the current status of the document, the branches are merged. If the existence of multiple document variants is allowed, then the latest version of every branch is used to describe the document.

In a similar way, different types of changes are applied in our approach to different instances of a document, i.e. the various *shadow documents*. The main difference between our approach and document versioning approaches is that the current status of a document copy being worked on by a specific user is not described by only one version at a time but rather by a set of three shadow documents, which we refer to as the *shadow document set* associated with a document copy.

An advantage of our approach in comparison to versioning systems is that there is no need to merge or save any of the shadow documents, which would put some extra load to any application implementing the concept. The information they present is needed only for the current session and has no value for another session. The committed changes are already stored in the repository, i.e. Tom's changes, and those not committed are discarded when Mary logs off, since the user that created them, i.e. Bob, chose not to publish them. This approach satisfies the second and third requirements from the list presented in Section 6.1.

6.3 Implementation of shadow documents

In this section we propose two possible implementations of the concept of *shadow documents* using the previous example to illustrate them. For both implementations, we assume that a structured document model is used for the development of the collaborative editor. Under this assumption, the fifth requirement set in Section 6.1 will also be satisfied because the idea of edit profiles can be adopted.

We assume again that Mary, Bob and Tom work in parallel. The hierarchical structure of the document being authored is presented on the left side of Figure 6.4a. For the sake of simplicity, we additionally assume that Bob makes only one change to the document, which is the deletion of the second sentence of the second paragraph, i.e. sentence $sen2.2$. The operation is received at Mary's site and needs to be integrated into the shadow documents.

Create a new document by copying the whole document hierarchy. The first implementation we propose is to copy the whole document hierarchy. The *SSL shadow document* is created when Mary logs in. At the moment when the first operation created in parallel arrives at Mary's site, a shadow document is created. The operation can be committed or non-committed and the copy created will be used as the CC shadow document or the CU shadow document, respectively. Once the document hierarchy is copied and the required shadow document is created, the operation that arrived can be applied to this document. Simultaneously, the severity of the operation will be computed and presented through the awareness visualisation tool which comes with the collaborative application. The shadow document representing the latest document copy at the moment of its download is the one on the left side of Figure 6.4a and the shadow document where all the operations created

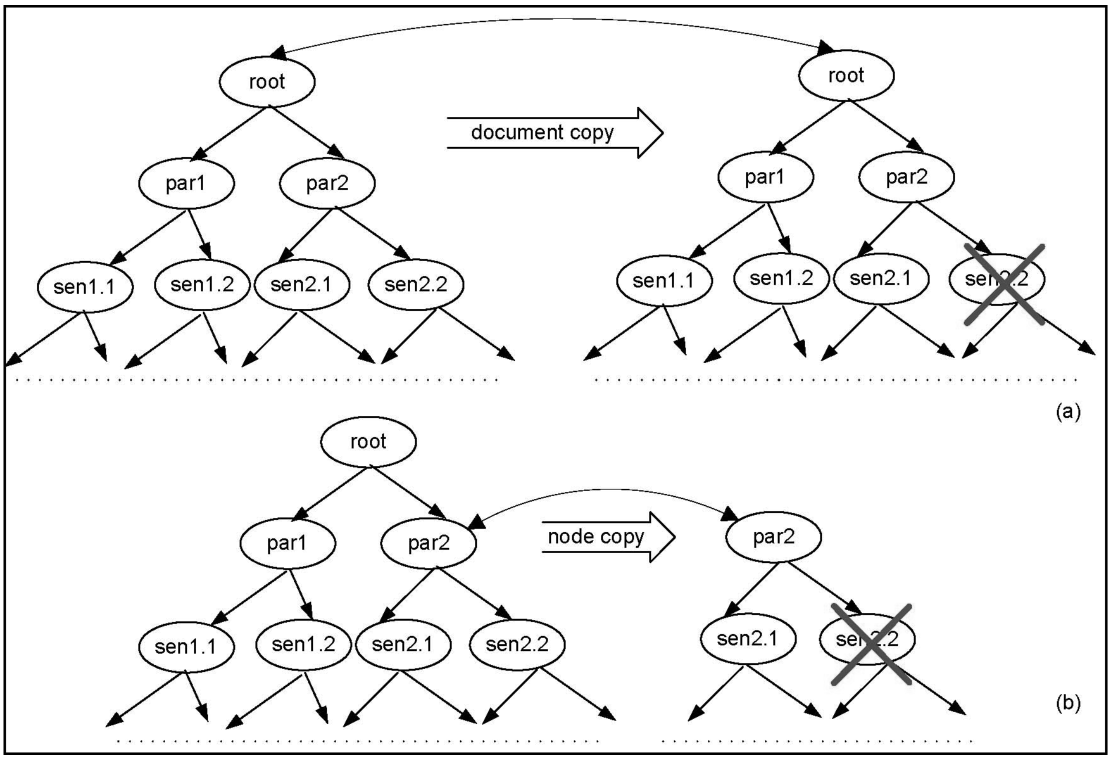


Figure 6.4: Visualisation of two proposed implementations for shadow documents

in parallel by other users will be applied is represented by the document on the right side of the same figure. One such document will be created for each of the two shadow documents.

While extending an asynchronous editor to develop a shared workspace, we implemented the notion of *shadow documents* by copying the whole document hierarchy. Such an implementation is easy, since it only requires the creation of a copy of the document's instance, but can be proved memory inefficient when working with large documents. Therefore, in what follows, we propose a second variation for the implementation.

Create a new document by copying document nodes on demand. An alternative implementation is to copy only some document nodes, i.e. parts of the hierarchy. This implementation is less likely to cause memory problems, but requires a more sophisticated mechanism for the copying and handling of documents and document parts. We propose to copy only the document part affected by the operation to be applied. In our example, the deletion of sentence $sen2.2$ will affect its parent node, i.e. the paragraph $par2$ since it will change its set of children nodes. As a result, only paragraph $par2$ will be copied. The resulting document is presented in Figure 6.4b. For a hierarchical document model, a node consists of its children. For instance, a paragraph consists of sentence-level document nodes. As a result, copying a document node in a hierarchical document can be translated to a creation of a new set of children nodes. For the first shadow document, the children of paragraph $par2$ will be the set $\{sen2.1, sen2.2\}$ while for the shadow document holding the concurrent uncommitted changes it will be the set $\{sen2.1\}$.

The existence of three shadow documents results in the existence of three different sets of children for every document node being edited. The definition of a document node, as presented in Chapter 3:

Definition 13. A node N is a structure of the form

$$N =< level, children, history, content >$$

is now extended to:

Definition 14. An extended node N_e is a structure of the form

$$N_e =< level, children_e, history_e, content_e >$$

where

- $children_e$ is an ordered list $\{children_1, children_2, children_3\}$ of children elements,
- $history_e$ is an ordered list $\{history_1, history_2, history_3\}$ of history elements,
- $content_e$ is an ordered list $\{content_1, content_2, content_3\}$ of content elements.
- The elements $children_1$, $history_1$, $content_1$ describe the SSL shadow document, the elements $children_2$, $history_2$, $content_2$ describe the CC shadow document and the $children_3$, $history_3$, $content_3$ describe the CU shadow document.

The element $level$ is still the syntactic level of the node, while the rest of the elements have been extended to hold three instances each, corresponding to the three shadow documents. For instance, $history_2$ will hold the set of uncommitted operations applied to the node and the node's content will be computed in the same way as previously, but based on the set $children_2$ and not $children_1$. Therefore, the $content_2$ of a node will be the sum of the *content* elements of all the children nodes in the set $children_2$.

Additionally, the path in the document's tree hierarchy where an operation is applied will also depend on the particular shadow document to which the operation is to be applied. If, in our example, the deletion of sentence $sen2.2$ was made by the operation $op = deleteSentence(2.2)$ and the operation is committed, then the notation will change to $op = deleteSentence(2.2c)$ meaning that the second child in the set of $children_3$ of the second paragraph will be deleted. This notation can be extended and be a lot more complicated in the case of longer documents to which many operations are applied.

The complexity of the above methods that implement the shadow documents might increase depending on the consistency maintenance algorithm that is used to maintain the shadow documents that hold the concurrent edits. Therefore, when implementing the notion of shadow documents, the issues to consider are the length and the type of a document, as well as the complexity of the algorithms employed to enable the maintenance of the shadow documents, for instance, a consistency algorithm or an undo algorithm. A lot of effort is currently devoted by the CSCW research community to improve the correctness and reduce the complexity of consistency and undo algorithms. We believe that the above issues and implementation methods of shadow documents should be re-evaluated in the light of more efficient algorithms. A detailed discussion on consistency algorithms and how they are expected to affect the usability and value of shadow documents is presented in Section 6.5.

6.4 BeAware

We implemented the notion of *shadow document sets* in the context of a shared workspace for asynchronous collaboration. In order to provide maximum awareness information to users, we also adopted and implemented the notion of *edit profiles* presented in Chapter 4. Therefore, our prototype of the asynchronous shared workspace provides awareness information for changes made to various documents, by various users, and at different granularity levels depending on a document's structure and the users' needs.

We provide a repository where all versions of all documents being collaboratively authored are stored. We filter this information once from the document point of view and once from the user point of view. Therefore, when a user logs into the application, they are provided with information about all the documents they are authoring and all other users they are collaborating with. Awareness information is computed based on all the changes made to the documents by all users either in the past or in real-time.

The difference between an asynchronous shared workspace and an asynchronous authoring tool is presented in Figure 6.5. While an asynchronous authoring tool provides information about only one document at a time, a workspace provides a view on more than one document. Consider, for instance, an authoring tool that monitors changes made to a document and provides awareness information through a visualisation tool as shown in Figure 6.5a,b. If a user is using this tool to work on more than one document, they need to separately open each of them (Figure 6.5a,b) using the authoring tool to get information about all documents. In addition, if a user works together with another user on more than one project, they cannot directly be informed about that user's activities across projects. Consider, for instance, that User2 of Figure6.5 is authoring Document1 in collaboration with User1. When User2 needs to check if and where in Document1 User1 has worked, User2 must open the document and search for changes made by User1. If the two users collaborate in more documents, the procedure of finding where User1 has worked will be more complicated, since User2 needs to open separately all the co-authored documents, to locate changes from User1 as shown in Figure 6.5a,b. There is no flexible way to inform User2 whether a document has changed, or at which document a collaborator has worked without opening the documents.

However, the workspace we implemented provides simultaneously a view on all the documents that a user works on. The users can easily navigate from one document to another. They are provided with awareness information even without opening the documents and can see all of the document changes and the users that worked on each document as shown in Figure 6.5c. Consider, for instance, User1 who is working on all three documents of Figure 6.5c. When User1 logs in the framework, they are presented with an edit profile for each of the three documents without opening the documents. In this way, User1 is instantly informed about the amount of changes made at each of the documents and which user worked on which document. Similarly, since the repository also holds information for the users, the awareness information can also be presented based on a user perspective. In this case, when users log in,

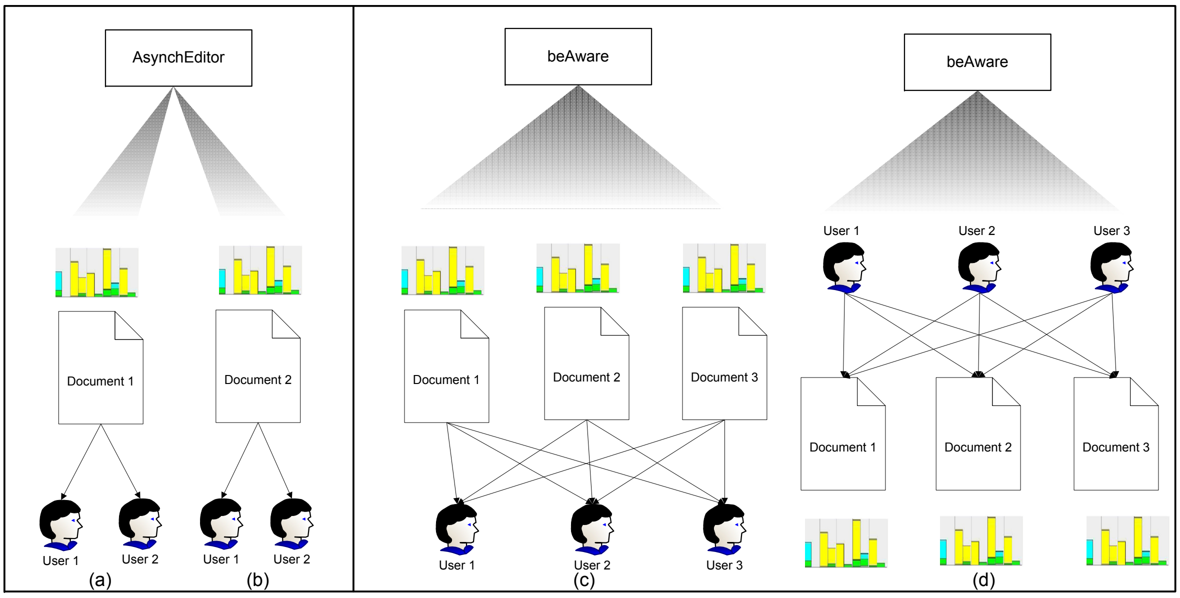


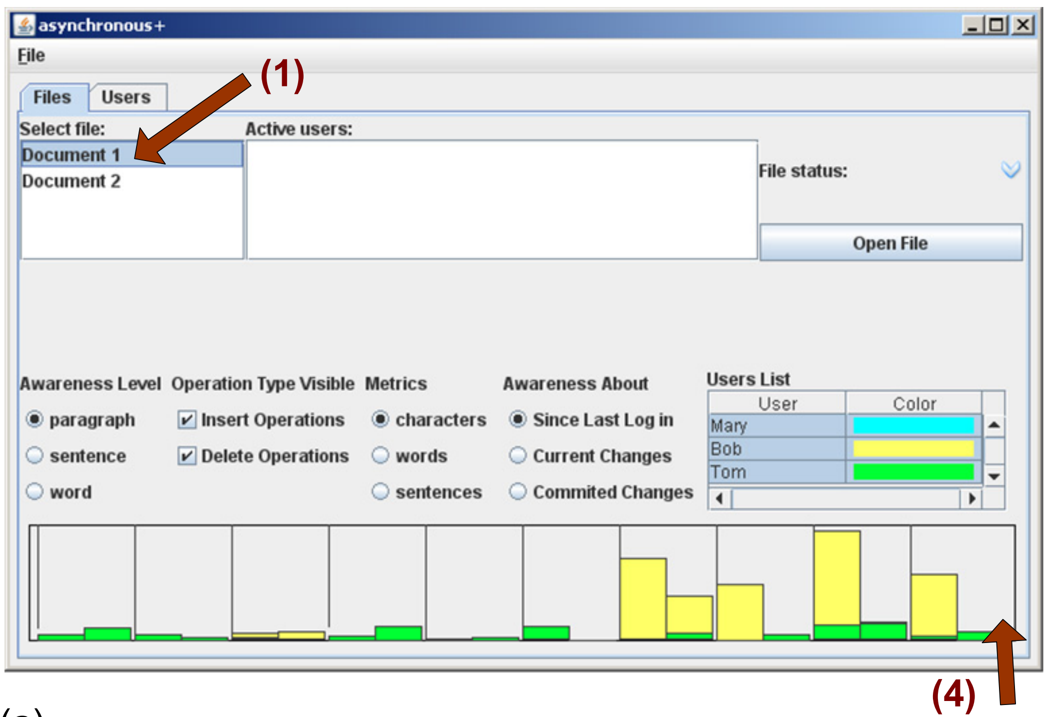
Figure 6.5: The view on documents and users defined by an asynchronous collaborative editor and BeAware

they are presented with a list of their collaborators. If the name of a user is chosen, then a list with the documents on which the user worked is presented. If requested, the detailed changes made on every document can also be presented through the visualisation tool (Figure 6.5d).

In the rest of this section, we present in detail the features implemented in our prototype with the help of Figures 6.6, 6.7 and 6.8. The awareness information presented by the application corresponds to our example with users Mary, Bob and Tom.

Detailed awareness information about changes made to a document on any of its three shadow documents are presented through *edit profiles*. This visualisation tool gives detailed information at different granularity levels, without the need to visualise the document itself. The user obtains a first impression of the amount of changes made to a document, the part of the document where they were made and by which users. If the user wants to visualise the changes superimposed on the document or make further changes to it, this is possible since an asynchronous collaborative editor is integrated into the application and the user can open any of the documents directly for editing.

We show the two panels provided by the application GUI. The first, shown in Figures 6.6a, 6.7a and 6.8a, returns awareness information filtered from the document view. The second, shown in Figures 6.6b, 6.7b and 6.8b is filtered from the user view. The numbers in the figures correspond to the numbers in the list provided below.



asynchronous+

File

Files | Users

Select file:
- Document 1
- Document 2

Active users:

File status:

Open File

Awareness Level: [✓ paragraph] sentence word

Operation Type Visible: [✓ Insert Operations] [✓ Delete Operations]

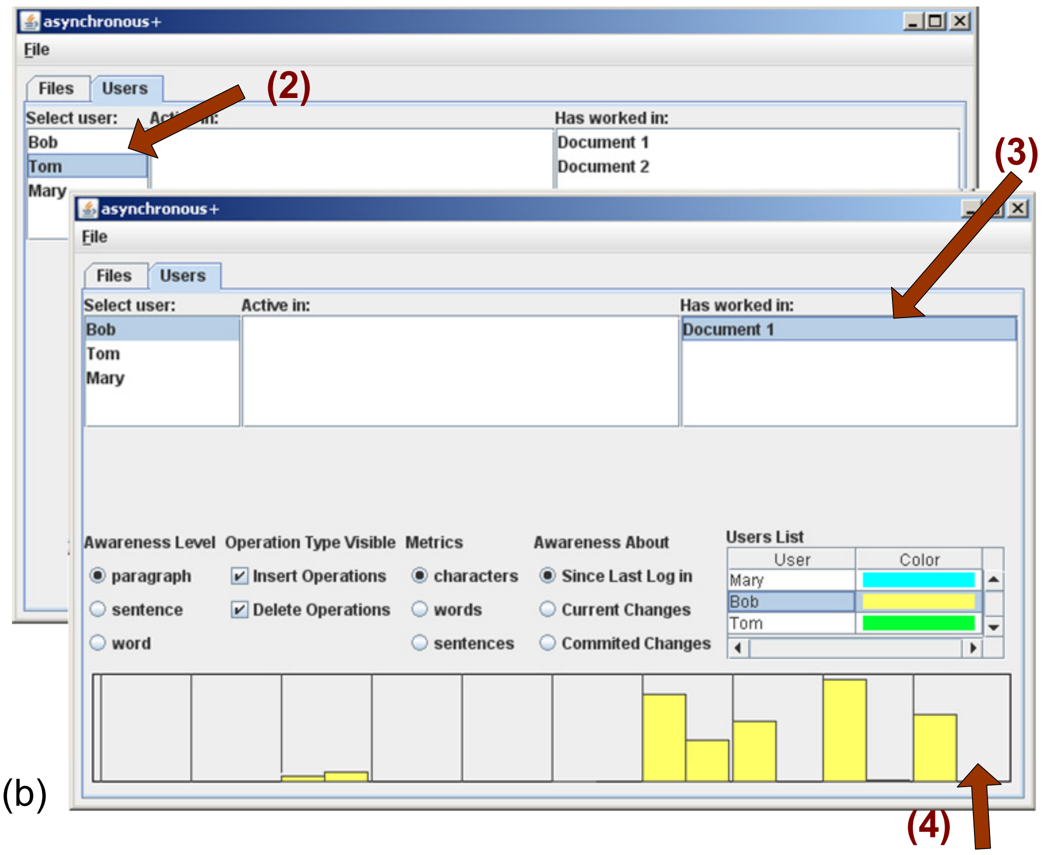
Metrics: [✓ characters] words sentences

Awareness About: [✓ Since Last Log in] Current Changes Commited Changes

Users List

User	Color
Mary	🟦
Bob	🟨
Tom	🟩

(a)



asynchronous+

File

Files | Users

Select user:
- Bob
- Tom
- Mary

Active in:

Has worked in:
- Document 1
- Document 2

asynchronous+

File

Files | Users

Select user:
- Bob
- Tom
- Mary

Active in:

Has worked in:
- Document 1

Awareness Level: [✓ paragraph] sentence word

Operation Type Visible: [✓ Insert Operations] [✓ Delete Operations]

Metrics: [✓ characters] words sentences

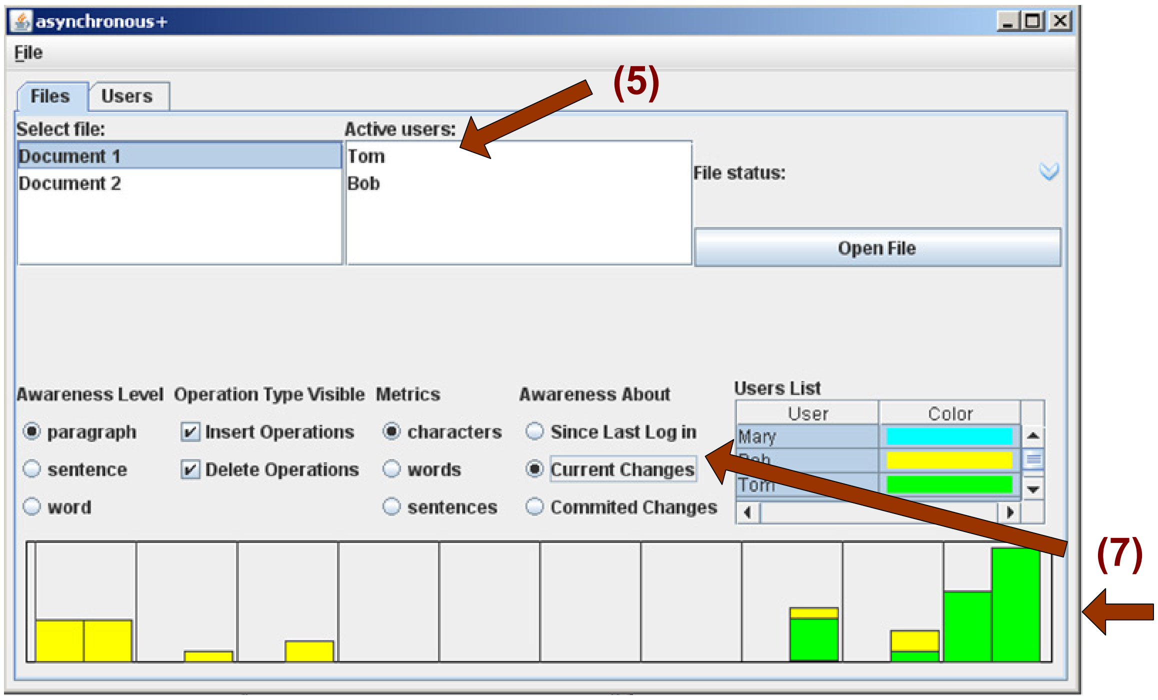
Awareness About: [✓ Since Last Log in] Current Changes Commited Changes

Users List

User	Color
Mary	🟦
Bob	🟨
Tom	🟩

(b)

Figure 6.6: The BeAware workspace. (a) Files Panel when the user first logs in. (b) Users Panel when the user first logs in



Select file: Document 1, Document 2

Active users: Tom, Bob

File status:

Awareness Level: [✓ paragraph] sentence word

Operation Type Visible: [✓ Insert Operations] [✓ Delete Operations]

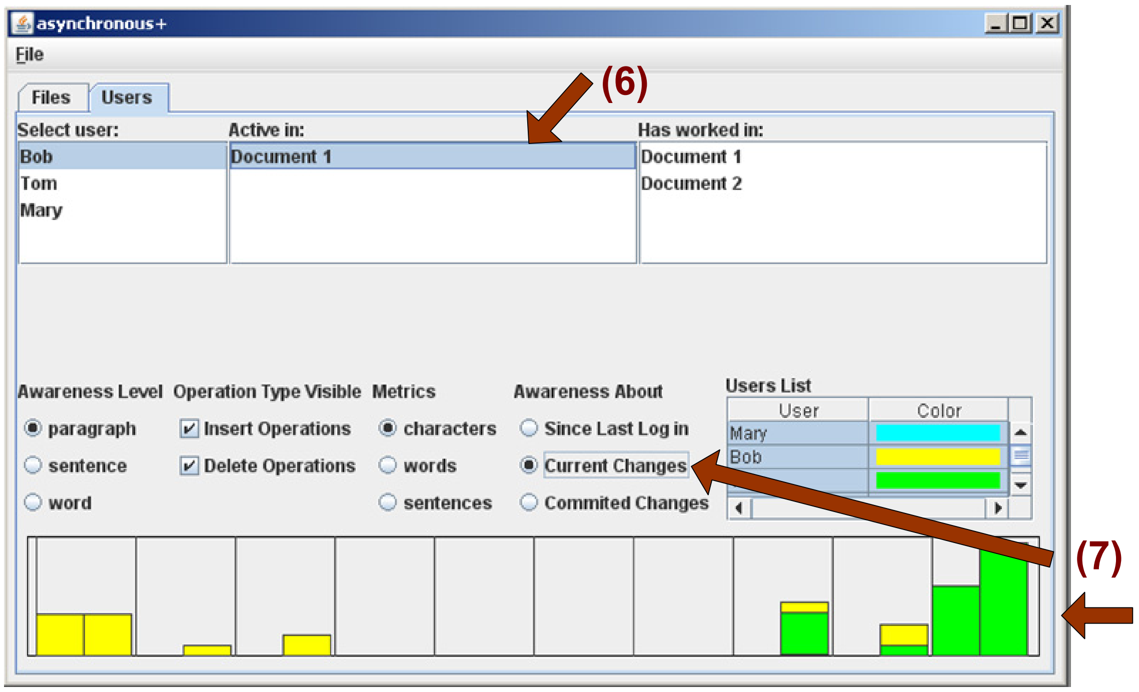
Metrics: [✓ characters] words sentences

Awareness About: Since Last Log in [✓ Current Changes] Commited Changes

Users List

User	Color
Mary	🟦
Bob	🟨
Tom	🟩

(a)



Select user: Bob, Tom, Mary

Active in: Document 1

Has worked in: Document 1, Document 2

Awareness Level: [✓ paragraph] sentence word

Operation Type Visible: [✓ Insert Operations] [✓ Delete Operations]

Metrics: [✓ characters] words sentences

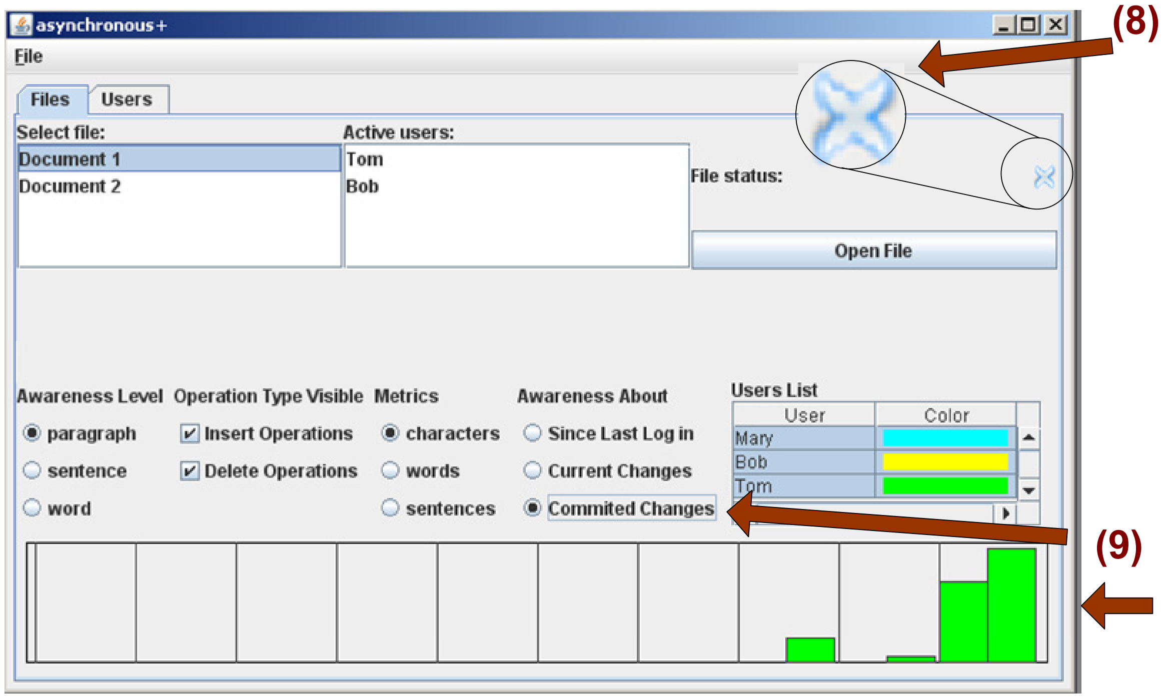
Awareness About: Since Last Log in [✓ Current Changes] Commited Changes

Users List

User	Color
Mary	🟦
Bob	🟨
	🟩

(b)

Figure 6.7: The BeAware workspace. (a) Files Panel with concurrent uncommitted changes. (b) Users Panel with concurrent uncommitted changes



Select file:
- Document 1
- Document 2

Active users:
- Tom
- Bob

File status:

Awareness Level: [✓ paragraph] sentence word

Operation Type Visible: [✓ Insert Operations] [✓ Delete Operations]

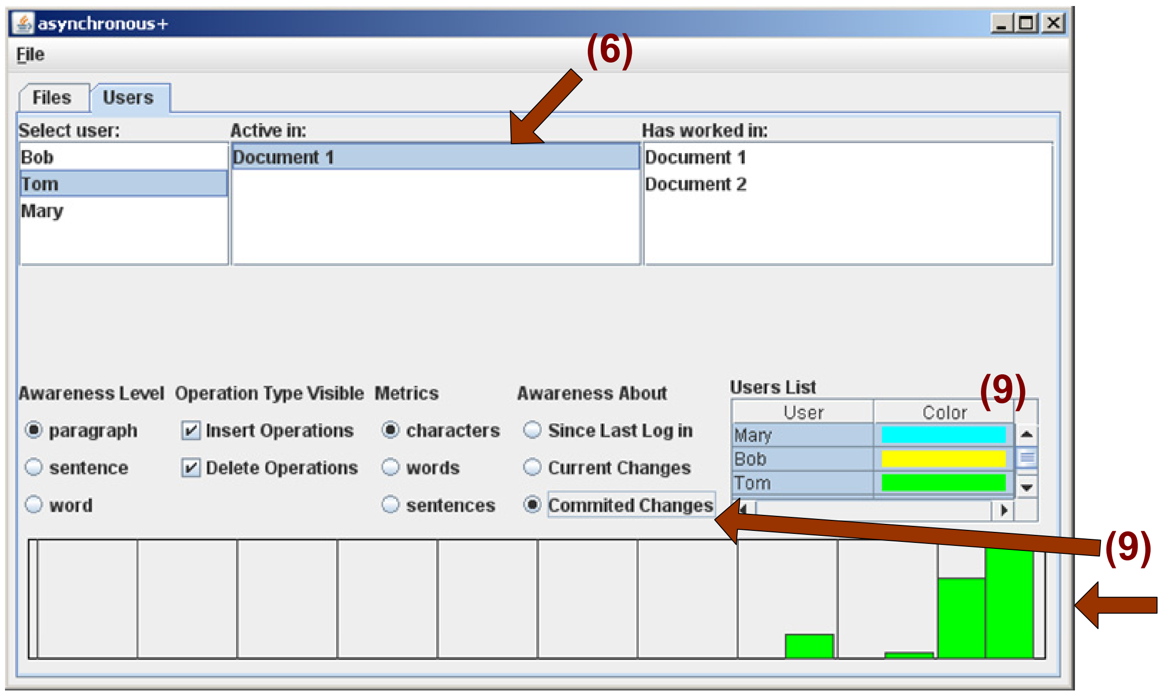
Metrics: [✓ characters] words sentences

Awareness About: Since Last Log in Current Changes [✓ Commited Changes]

Users List

User	Color
Mary	🟦
Bob	🟨
Tom	🟩

(a)



Select user:
- Bob
- Tom
- Mary

Active in:
- Document 1

Has worked in:
- Document 1
- Document 2

Awareness Level: [✓ paragraph] sentence word

Operation Type Visible: [✓ Insert Operations] [✓ Delete Operations]

Metrics: [✓ characters] words sentences

Awareness About: Since Last Log in Current Changes [✓ Commited Changes]

Users List

User	Color
Mary	🟦
Bob	🟨
Tom	🟩

(b)

Figure 6.8: The BeAware workspace. (a) Files Panel with concurrent committed changes. (b) Users Panel with concurrent committed changes

When users log into the application they are provided with:

1. **A list of all the documents in the repository for which the user has a task**. Assuming that Mary is the first user that logs in, she is presented, as shown in Figure 6.6a, with a list of documents on which she is working, namely "Document 1" and "Document 2". Clicking on any of them returns through the edit profile, all the changes that all other users, i.e. Bob and Tom, made since the last time Mary logged in (see item 4 as well).

2. **A list of all the collaborators**. The users panel shows all the users that are collaboratively working on documents present in the repository (Figure 6.6b).

3. **A list of the documents each user has worked on since the user's last login**. By selecting one user in the users panel, Mary is informed about the documents this user has worked on since her last login. In this example, Tom has worked on both documents, while Bob only on "Document 1".

4. **Detailed awareness information about the changes made to every document since the user's last login**. This information corresponds to the first of the shadow documents and is provided by both panels as an *edit profile*. Mary can further select to visualise the changes from any user separately by selecting the user's name from the "Users List" in either of the two panels. In Figure 6.6b, Bob is selected and his changes throughout "Document 1" are presented in the *edit profile*.

5. **A list of the documents currently being authored by other users**. We assume that Bob and Tom log into the application and both open the file: "Document 1" to work on it. To open a file, a user can select it in the files panel and then press the button "Open File". Immediately, Mary is informed about the intention of both users, through the list "Active Users". The list presents the names of the users currently working on the selected document.

6. **A list of the documents currently being authored by each user**. The information presented in item 5, can also be presented from the users point of view. Mary, in this case, can select a user and be informed about the documents on which the user is currently working.

7. **Detailed awareness information about all the changes currently made to a document by other users**. The changes made in real-time by Bob and Tom are presented through the *edit profile* in both panels. This information represents the second of the shadow documents. Mary is informed that Bob is working at the beginning and at the end of the document, while Tom only at the end of the document. Please note that for changes currently being made to be visible, Mary needs to set the "Awareness About" radio button in the GUI to "Current Changes". The default selection is "Since Last Login". In this way, Mary is not distracted by concurrent changes while working. Only if she chooses to do so, will she be informed about all the details of the changes. In this way, the fourth requirement presented in Section 6.1 is satisfied.

8. **A notification for every document when a new committed version is available in the repository**. When Tom commits his changes, a new version is created in the repository and Mary is informed about it through a change of an icon. This icon reflects the status of Mary's copy for the selected document (as specified in the files panel). We decided to inform users in this way to ensure they are not distracted from their work. Of course any other notification mechanism could be used.

9. **Detailed awareness information about all the changes reflected in a newly committed version on the repository**. Finally, Mary is informed about the committed changes for the given document by selecting "Committed Changes" in the "Awareness About" radio button in either of the two panels. The information presented is the one computed for the third of the set of shadow documents.

We presented in detail all the features provided by BeAware, a prototype of a shared workspace. BeAware implements the notions of *shadow document sets* and *edit profiles*. Therefore, it provides the computation and visualisation of changes made to many documents, by various users working in privacy, in the past and in real-time.

6.5 Discussion

At this point we would like to differentiate BeAware from existing shared workspaces. BSCW [20] and GROOVE [13] are two of the most widely used. Both systems successfully deliver information about changes made by various users to many documents. However, they do not provide any information about either changes made to a document by different users in parallel or in-document changes. On the contrary, BeAware informs the user about changes made in real-time by other users while they are asynchronously working on their local copy, as well as about the severity of these changes on the various document parts.

In the rest of this section we report on some issues related to the approach we have presented as well as possible extensions. An important issue that may arise is how to ensure consistency of the shadow documents regarding the concurrent committed and uncommitted changes. Changes to be applied to the CC shadow document are always sent from the repository and consist of all changes made by a user to modify the document from one version to another. Since a precondition for publishing one's changes is that the user's copy is informed about the latest copy in the repository, and the CC shadow document always reflects the latest version on the repository, applying the current committed changes will not cause any conflict with the corresponding shadow document. The use of any algorithm used for consistency maintenance in asynchronous collaboration is allowed.

The current uncommitted changes also need to be applied to a shadow document. Since they are made from various users and are applied in real-time, the use of an algorithm that supports synchronous collaboration is needed. Although we currently handle operations from various users, there are situations where we cannot guarantee consistency. Concurrency and consistency maintenance issues are not within the scope of this thesis. However, we realise that the use of a sophisticated consistency maintenance algorithm would support the correct maintenance of the shadow documents [110, 112, 113]. If consistency maintenance algorithms are not used in combination with the notion of shadow documents, or if algorithms that fail to always support the correct integration of real-time editing operations are used, the correct maintenance of the shadow documents can not always be guaranteed.

This shows the dependence of our approach on the correctness of the existing maintenance algorithms. This is inherent from the layered architecture of our approach which was presented in Chapter 3. As already discussed, the computation of awareness information follows the integration of the editing operations in the document, which follows the handling of the operations by the maintenance algorithm. As a result, an incorrect consistency maintenance algorithm will create inconsistent documents, which would diminish the value of awareness information to the user.

A further issue arises from the fact that some of the concurrent uncommitted changes may at some point be published. This results in a set of operations applied in the CC shadow document and the same set of operations being "undone" from the CU shadow document. We propose two different approaches to handle this issue. One is the development of a new, or adoption of an existing, algorithm that supports the "undo" of operations. A second solution is to clear the document that holds the uncommitted changes, extract all published operations from the log file and reapply the remaining changes to the initial copy that was downloaded by the user while logging in.

Finally, every time Mary updates her copy of the document by downloading a new version from the repository, all three shadow documents need to be cleared and recomputed. This issue requires special attention since there may exist changes created on a previous document version, i.e. the changes made by Bob that are sent to Mary's updated copy and need to be integrated. Further investigation of this issue is required as well.

6.6 Summary

In this chapter we presented an approach that addresses the problems resulting from modifications made to a document by various users in parallel when working asynchronously. We introduced *shadow document sets* to represent changes made by other users to a document being edited. The shadow document set associated with a local document copy can be considered as an awareness preview of concurrent changes. Depending on the required information, the changes stored in each of the shadow documents can be presented separately. We adopted this notion in the development of BeAware, a shared workspace for asynchronous collaboration. We provide the users with information about changes made by their collaborators since their last login and, additionally, information about changes made to the document in parallel by other users working on their own local copies. Changes are separately stored and presented depending on whether they have already been published to the repository or not.

Although we provide users with all the above information, the local copies of the users are not modified to ensure that they continue working in privacy. Additionally, only the severity of each user's changes is presented to other users, keeping the actual changes hidden until they are published to the repository. The awareness information computed in BeAware is visualised through *edit profiles* as an overview of the changes made to each of the shadow documents. If the need arises, users can open any document with an asynchronous editor also provided by BeAware and visualise the changes in the document itself, or even make new changes.

Finally, we would like to note, that the work presented in this chapter could also be classified as awareness on semi-synchronous collaboration, since part of the information about users modifications is revealed to the users in real time. However, note that almost no privacy issues are taken into consideration. We did consider ways to inform users without disrupting them, but we did not consider whether any issues exist concerning their modifications log that is sent to their collaborators. Such issues are investigated in the next chapter, where another approach is proposed for privacy-sensitive collaborative environments.

7 Multi-level change awareness in privacy-sensitive environments

When involved in collaborative tasks, people often choose to use semi-synchronous applications in order to concurrently work in isolation. Hence, privacy of their changes is maintained until they decide to publish their contributions. Not being aware of changes made by their collaborators, they often create concurrent modifications which might generate conflicts or redundancies. In this chapter, we propose an awareness mechanism that solves this problem by computing and providing awareness in semi-synchronous collaboration. The characteristic that mainly differentiates this approach from the one presented in the previous chapter is that this approach allows users to specify the level of detail of their uncommitted modifications that will be made available to their collaborators, i.e. it takes privacy issues into consideration. A detailed discussion on this is presented at the end of the chapter.

7.1 Awareness in the light of privacy issues

People very often decide to work using a semi-synchronous environment because they wish to work in privacy and review their work before publishing it. Although providing awareness about modifications made by other users in real-time can be advantageous and therefore required by users, as explained in the previous chapter, it can also be problematic if privacy-sensitive information is exchanged between collaborators. Therefore, there is the need for an advanced mechanism that keeps users aware of other users' modifications, while respecting their privacy concerns.

However, which data is considered private cannot be uniquely defined. It depends on the users, their collaborators, their roles, the collaborative situation and the task to be accomplished. Therefore, the level of privacy that each user desires should be set by the users themselves depending on their current needs and situation. The computation of awareness information in privacy-sensitive environments should respect any private uncommitted data and compute awareness information based on the available information only.

Ghost operations, introduced by Ignat et al. [69], is a concept that can be applied to the above collaborative situations. *Ghost operations* represent filtered operations according to the settings of a user with respect to other users. Instead of transmitting the actual (from now on referred to as *real operations*) operations created by a user, the operations are filtered and only part of the information that they contain is transmitted to other users.

The awareness mechanism described in Chapter 3 is general enough to be applied to any kind of collaborative application with a structured document model, independently of the document type or the mode of collaboration. However, it assumes that all the required information for the computation of awareness is sent by users to their collaborators. This is described by the definition of the *operation* concept, where all information is included. Therefore, it might be problematic to apply such a mechanism to privacy-sensitive collaborative situations.

In this chapter we concentrate on extending this awareness mechanism to compute the *maximum available* information in the light of privacy issues. We decided to adopt the concept of *ghost operations* and investigate whether it can be handled by our awareness mechanism. The results, being a small modification of the awareness computation and the edit profile GUI, are presented in this chapter.

In what follows, we describe an overview of the general procedure for the creation and receipt of ghost operations and the computation of the awareness information provided by *ghost operations*. To analyse in detail the generation of *ghost operations*, we first give their definition by means of the possible filters applied to the attributes of a real operation. Then, we present some of the masks that can be created by combining the various filters. We also discuss example situations where each mask can be used. We continue by presenting the extended version of our framework's metamodel that includes the notion of *ghost operations*. Finally, by means of edit profiles, we show how the information concerning the *ghost operations* is presented to the users in the case of an example situation and discuss in detail the computation of the values presented in the visualisation tool.

7.2 Architecture

The architecture shown in Figure 7.1 presents the procedure followed from the creation of an operation at a user's local site to the receipt of it by another user and the computation of awareness information. The steps followed are:

- Generation of a real operation at a user's site.
- Filtering of the operation attributes by using various masks to generate a ghost operation.
- Transmission of the ghost operation through the network to all collaborators.
- Receipt of an operation from a collaborator and extraction of the available information. The document node where the operation will be applied is found, the appropriate *opValues* are computed and the corresponding *nodeValues* are updated.

In what follows we analyse in more detail the second and fourth steps of the above procedure, i.e. the *ghost operation generator* and the *awareness computation*.

7.3 Ghost operations

In [69], a *ghost operation* was defined as $g(operation) = < filter(type), filter(parameter)* >$ following the definition of the real operation given by a type and a list of parameters $operation =< type, parameter* >$.

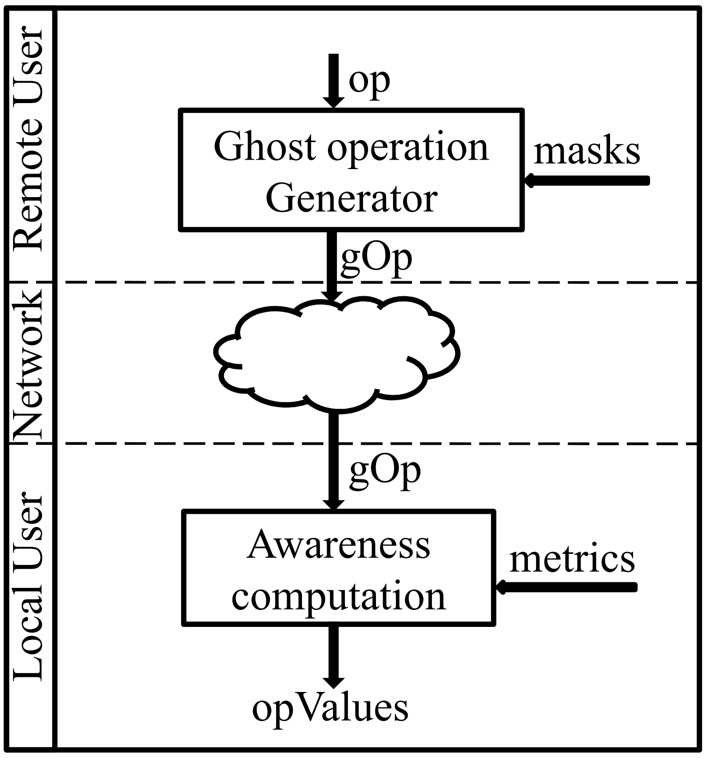


Figure 7.1: Generation of ghost operations, transmission through the network and computation of awareness after the operations' receipt

Hence, the ghost operation is the operation obtained by filtering the type and the parameters of the original operation according to user privacy preferences.

The definition of *ghost operations* for the structured documents we consider, where a *real operation* is defined as $op =< type, level, position, content, length, user >$, results as well from the application of filters to the attributes of the *real operation.* Therefore, we define *ghost operations* as below.

Definition 15. We define a ghost operation $g(op)$ as $g(op) =< filter(type), filter(level), filter(position), filter(content), filter(length), filter(user) >$, where

- filter(type)= $\begin{cases} insert, & \text{if } op \text{ is an insert and} \\ & \text{its type is not masked} \\ delete, & \text{if } op \text{ is a delete and} \\ & \text{its type is not masked} \\ edit, & \text{if } op \text{ is a delete or insert} \\ & \text{and its type is masked} \end{cases}$

- filter(level)= $\begin{cases} level, & \text{if the level of } op \text{ is not masked} \\ null, & \text{if the level of } op \text{ is masked} \end{cases}$

- filter(position=$[V_0, V_1, \cdots V_n]$)= $[V_0, \cdots V_i]$, where $0 \leq i \leq n$

- filter(content)= $\begin{cases} content, & \text{if the content of } op \text{ is not masked} \\ null, & \text{if the content of } op \text{ is masked} \end{cases}$

- filter(length)= $\begin{cases} length, & \text{if the length of } op \text{ is not masked} \\ null, & \text{if the length of } op \text{ is masked} \end{cases}$

- filter(user)= $\begin{cases} user, & \text{if the user identity of } op \text{ is not masked} \\ null, & \text{if the user identity of } op \text{ is masked} \end{cases}$

The set of available types presented in the definition above are only indicative. The set of types of ghost operations is only restricted by the available types of real operations. The vector of positions of the original operation can be filtered by providing the positions of the higher level of granularity of the node, but hiding the positions of the lowest level of granularity. For instance, for an original operation of insertion of a word, only the position of the paragraph where the word is to be inserted can be provided and the sentence and word level positions can be hidden.

Since the difference between a ghost and a real operation is not the set of their attributes, but rather the values of their attributes, both kinds of operations are fundamentally the same. Therefore, the general definition of an *operation* included in the metamodel of Chapter 3 can be used to model the concept of ghost operations as well. This shows that the core concepts of the awareness mechanism can be used in privacy-sensitive environments as well.

Note that if none of an operation's attributes is masked, then the ghost operation will be identical with the real operation and all information will be transmitted between collaborators. If, on the contrary, all filters are applied, then no information is transmitted between users. A semi-synchronous collaborative environment using the fully masked ghost operations would offer as much awareness as an asynchronous environment. The above are the two extreme situations. There exist others where only a subset of the attributes is masked. The set of masked attributes is then defined by the desired level of privacy. In the next section we present example situations where combinations of the above filters are used depending on the information users need to keep private.

7.4 Privacy levels

As discussed above, the level of privacy requested from users may be task or user specific. Therefore, we decided to make the filtering/masking mechanism that creates the ghost operations quite flexible by offering various levels of privacy. Here we present the most important ones in terms of the operation attributes that get masked in each level. We also give examples of collaborative situations where each level could be selected. We realise that the possible masks to be used are equal to the maximum number of combination of the filters presented above and therefore there are more combinations possible than the ones presented here. We believe that some of them would not make any sense and possibly would not be used at all, while others could be used frequently. Therefore we decided to introduce the ones we expect to be needed most often by users.

No privacy: We start from the first level where no privacy issues arise, and therefore no information is masked. These ghost operations hold the maximum information and the computation of awareness is not restricted at all. All the *opValue*s are computed and the *nodeValue*s of the document nodes affected by the operation are updated.

Mask the user and/or type: Here the *type* or the *user*, or both of these attributes can be filtered to produce the ghost operation. This level of ghost operations will be created in collaborative situations where the users want to keep their anonymity until they commit their changes, or when they want to inform their collaborators that the document is being edited without specifying the type of changes. The exact document part that is modified is available to the collaborators and information about the extent of the changes as well. Users receiving such ghost operations will be notified about the document parts being edited and as a result are likely to avoid working on the same document areas. We believe it would also be reasonable to mask the *content* as well if the type is masked. An example of such a ghost operation is:
$gOp1 = < edit, 3, [2,3,4], null, [0,0,1,4], null >$.

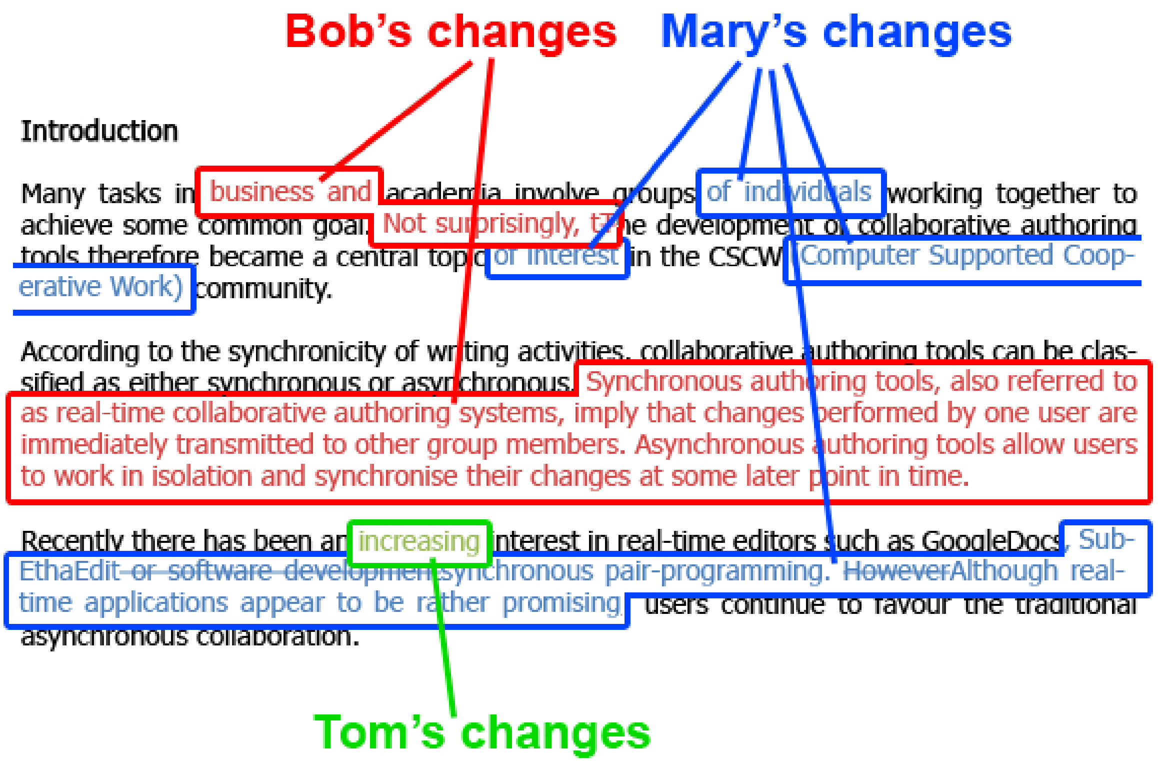
Independently of the *type* and/or *user* being filtered, the next levels concentrate on the filtering of other attributes.

Mask the changes: This level holds operations where the *content* is filtered. Such ghost operations might be produced in collaborative situations where the users allow the exact document part being edited and the extent of the changes to be available to their collaborators, but do not publish the actual changes. An example of such a ghost operation is:

$gOp2 = < insert, 3, [2, 3, 4], null, [0, 0, 1, 4], null >$.

Mask the changes and their severity: Ghost operations that conform to this level of privacy have the *content* and the *length* masked. Users who intend to make changes to a specific paragraph in a document but do not yet know what exactly to write, often produce a lot of changes which they later discard. To mask such phenomena but at the same time inform their collaborators where they are currently active, ghost operations of this level could be used. The remaining unmasked attributes, i.e. the *level* and *position* can be used to inform other users about the document part being edited without showing the actual modifications. These two attributes can also be used by the user who receives the ghost operation to check if the position is also filtered and distinguish this case from the next one. For instance, an operation of word level, where the position is not masked, should have a position attribute, where a complete path from the document level to the word level is given. An example of such a ghost operation is:
$gOp3 = < edit, 3, [2, 3, 4], null, null, null >$.

Mask the changes, their severity and part of the position: Finally, we consider collaborative situations where the user also wants to keep information about the exact document part where the changes are made private. This can be achieved either by hiding the *position* attribute (part of it or all of it), or by hiding the *level* attribute, or both. Filtering the *level* attribute will make it impossible to verify whether the position is incomplete and hence the position will also be considered masked. Filtering part of the *position* means making available only part of the path in a structured document where the operation will be applied. For instance, a proofreader might want to inform their collaborators about the document part they are working on, without giving details about the exact sentences or words being edited. Such information could be enough to inform collaborators about their activity but at the same time respect their privacy. Finally, masking all the *position* attributes would deliver similar information to a mechanism showing that a user is present in a document. If additional information is given, for instance the *level* or the *length*, the severity of user changes could be computed for the document level. Although such an operation is correctly handled by the proposed mechanism, it requires that at least a part of the position attribute is provided to render the computed information interesting for the users. Examples of ghost operations of this privacy level are:
$gOp4 = < edit, null, [2], null, null, null >$.
$gOp5 = < edit, 3, null, null, null, null >$.



Introduction

Many tasks in business and academia involve groups of individuals working together to achieve some common goal. Not surprisingly, t~~T~~he development of collaborative authoring tools therefore became a central topic of interest in the CSCW (Computer Supported Cooperative Work) community.

According to the synchronicity of writing activities, collaborative authoring tools can be classified as either synchronous or asynchronous. Synchronous authoring tools, also referred to as real-time collaborative authoring systems, imply that changes performed by one user are immediately transmitted to other group members. Asynchronous authoring tools allow users to work in isolation and synchronise their changes at some later point in time.

Recently there has been an increasing interest in real-time editors such as GoogleDocs, SubEthaEdit ~~or software development~~synchronous pair-programming. ~~However~~Although real-time applications appear to be rather promising users continue to favour the traditional asynchronous collaboration.

Figure 7.2: Concurrent changes generated by Mary, Bob and Tom

$gOp6 = < edit, null, null, null, null, user1 >$.

7.5 Visualising ghost operations through edit profiles

Here we show by means of an example, how various types of ghost operations are visualised. For that, we use edit profiles with a modified GUI that presents the "maximum available" *nodeValues*. Afterward, we provide more details about the computation of the awareness information.

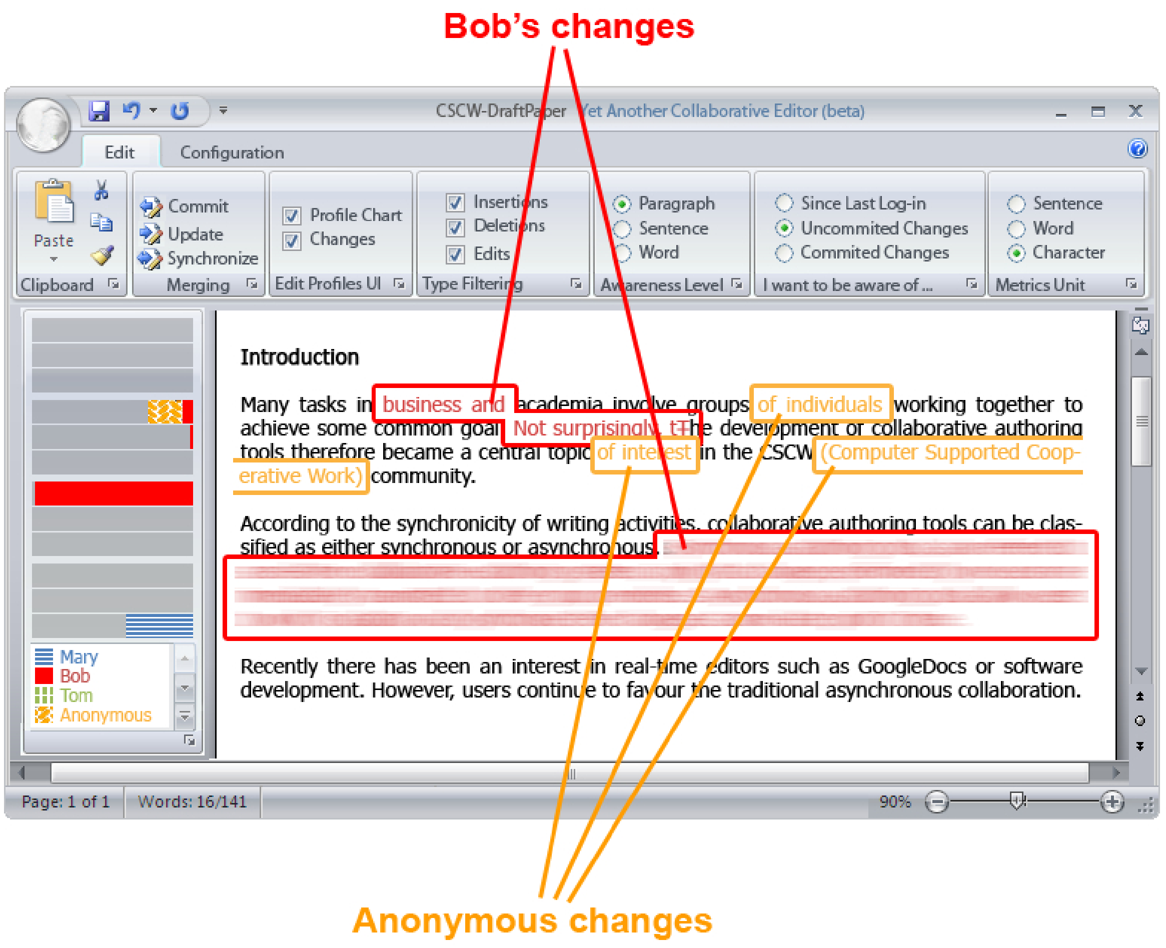
Consider the case of some researchers authoring a research paper. For simplicity, we consider that the document being edited contains 4 paragraphs. Consider that users Mary, Bob and Tom work on the same version of the document and they concurrently generate the changes illustrated in Figure 7.2.

Mary wants to let the others know about the changes she did in the second paragraph (the title of the section is considered as a paragraph itself), but does not want at this point to reveal her identity for those changes. She therefore generates 8 ghost operations gOp_{1Mary}, $\ldots$, gOp_{8Mary} corresponding to the insertion of words 'of', 'individuals', 'of', 'interest', '(Computer', 'Supported', 'Cooperative', 'Work)'. For instance, the ghost operation for the insertion of the first word 'of' is $gOp_{1Mary} =< insert, 3, [2,1,7], `of', [0,0,1,2], null >$. Note that the characters "(" and ")" could be considered as part of a word or word separators in the implementation of such an editor. Here we consider them as part of a word. Concerning the changes Mary did on paragraph 4, she wants to let the other users know that she is editing that paragraph but still needs some time to review what she has edited. She chooses to send her changes with a masked content, masked type and partially masked position (only the paragraph level node is given). However, she does not filter the length of her changes, providing in this way a measure of the changes that she made. She generates 17 ghost operations $gOp_{9Mary}, \ldots, gOp_{25Mary}$ corresponding to the insertion of character ',', insertion of word 'SubEthaEdit', deletion of words 'or', 'software' and 'development', insertion of words 'synchronous', 'pair-programming', deletion of word 'However' and insertion of words 'Although', 'real-time', 'applications', 'appear', 'to', 'be', 'rather', 'promising'. For instance, $gOp_{9Mary} =< edit, 4, [4], null, [0,0,0,1], Mary >$.

Bob decides to send to the other users the changes that he made in paragraph 2 of the document unmasked. He therefore generates 6 ghost operations $gOp_{1Bob}, \ldots, gOp_{6Bob}$ corresponding to the insertion of words 'business', 'and', 'Not', 'surprisingly,', deletion of character 'T' and insertion of character 't'. For instance, $gOp_{1Bob} =< insert, 3, [2,1,4], 'business', [0,0,1,8], Bob >$. As he wants to review the changes he did in paragraph 3, he will mask their content, but send all the other information required. He generates two ghost operations gOp_{7Bob} and gOp_{8Bob} corresponding to the insertion of sentences 'Synchronous authoring tools, also referred to as real-time collaborative authoring systems, imply that changes made by one user are immediately transmitted to other group members.' and 'Asynchronous authoring tools allow users to work in isolation and synchronise their changes at some later point in time'. For instance, $gOp_{7Bob} =< insert, 2, [3,2], null, \{0,1,25,184\}, Bob >$.

Tom just started working on the document; he did just a small change. He just wants to let the others know that he has started working on the document and therefore he masks all information about his changes except his identity. He therefore generates the ghost operation $gOp_{1Tom} =< edit, null, null, null, null, Tom >$.

Figure 7.3 represents the edit profiles in the presence of the above described masked concurrent changes as seen by a fourth researcher who just opened the document containing the paper. The researcher wants to be informed about the uncommitted changes of his colleagues presented by means of a profile chart (left side of the figure) that gives him a quick overview of the changes along with the changes themselves in the document. He therefore selects the *ProfileChart* and *Changes* options. He selects to see all types of changes, i.e. insertions, deletions and edits presented at the level of paragraph, changes being measured in characters. The profile chart includes three bars for each paragraph, showing in a top-to-down order the number of insertions, deletions and edits made on that paragraph. Changes by one user are identified by a unique colour, in the lower left side a legend of users with their associated colours is presented.



[✓ Profile Chart] [✓ Changes]

[✓ Insertions] [✓ Deletions] [✓ Edits]

[✓ Paragraph] Sentence Word

Since Last Log-in [✓ Uncommited Changes] Commited Changes

Sentence Word [✓ Character]

☰ Mary
🟥 Bob
▥ Tom
▨ Anonymous

Introduction

Many tasks in business and academia involve groups of individuals working together to achieve some common goal. Not surprisingly, tThe development of collaborative authoring tools therefore became a central topic of interest in the CSCW (Computer Supported Cooperative Work) community.

According to the synchronicity of writing activities, collaborative authoring tools can be classified as either synchronous or asynchronous.

Recently there has been an interest in real-time editors such as GoogleDocs or software development. However, users continue to favour the traditional asynchronous collaboration.

Figure 7.3: Edit profiles in the presence of ghost operations

From the profile chart, we can deduce that no changes were made in the first paragraph, that the changes in the second paragraph were made by Bob (insertions and deletions) and an anonymous user (insertions), that Bob contributed also by inserting content in the third paragraph and that Mary edited the last paragraph. We can also see that another user Tom is active in the document, but no information about the changes he made is provided. Inside the document, changes made by users are included if their content and position were not masked. If the content of changes was masked but their positions and lengths were specified, the changes are blurred.

Visualising the changes superimposed on the document could be an optional feature and be deactivated by the users if needed. This feature could be offered by adopting and implementing the concept of shadow document sets. Additionally, if users decide to work in privacy they can deselect the *ProfileChart* and *Changes* options.

The above awareness mechanism offers users the possibility to work in privacy as in the case of traditional semi-synchronous communication or to view in real-time the changes made in the system. Moreover, it offers users the flexibility of filtering, according to their privacy preferences, the information about their changes that are transmitted to their collaborators. If none of the users filters the transmitted changes and all users select the *Changes* option to integrate user changes as soon as they occur, the system simulates the functionality of a real-time system enhanced with edit profiles.

In the collaborative example presented in this section, the users defined their preferred level of privacy for each of the edited paragraphs. We believe that it is unlikely that users would like to define their privacy levels in such fine grained granularity. However, we presented such a collaborative situation to illustrate many possible alternatives with regard to the privacy levels that a user could use when privacy concerns are raised. We expect that a system offering the above functionality would offer an automatic or semi-automatic process of defining the above levels. For instance, default privacy levels could be set by the system. An alternative would be to have project- or document-specific privacy levels set for each user through a configuration menu. A user might then select one privacy level for a co-authored document and another privacy level for a second document. It would also be interesting to investigate whether privacy levels could be set automatically by the system based on predefined user roles and tasks.

7.6 Awareness computation for ghost operations

We now consider the above examples of ghost operations and explain how our mechanism computes awareness information. We assume that the operations arrive at a collaborator's site and are about to be evaluated.

Operations such as $gOp_{1Bob} =< insert, 3, [2,1,4], 'business', [0,0,1,8], Bob >$, belong to the first level that we introduced, where no privacy issues are taken into consideration. For the computation of awareness, the document node where the operation will be applied is found first. An *opValue* is then computed for each of the defined metrics and its value, together with information about the type of the operation and the user who created it, is attached to the node by being added to the corresponding *nodeValue*. The resulting *nodeValues* are visualised through the edit profile. The changes are also superimposed on the text. The colours used in the chart and in the text respectively show the user who made the change.

A procedure similar to the above one is followed when the ghost operation has the user attribute masked as in operation $gOp_{1Mary} = < insert,\ 3,\ [2,1,7], 'of', [0,0,1,2], null >$. It only differs at the step where the computed values are attached to the node. Since the user is not known, the values are attached to the node and marked as being made by user "Anonymous". This "new" user is introduced to hold all the changes made from users who want to keep their anonymity.

If the ghost operation arriving at a site provides no content, then it is impossible to annotate the document. If, however, adequate information is given about the severity of the operation and the position where it is applied, then our mechanism informs the user about these details of the change. For instance, the severity of operation $gOp_{7Bob} = < insert, 2, [3,2], null, [0,1,25,184], Bob >$ can easily be computed since the length is provided. Additionally, the level can be used to check whether the position is partially masked or not. In this operation, the position is not masked, since the operation refers to a new sentence inserted at the third paragraph in position 2. This means that the exact node where the operation will be applied is given. All of the above information will be provided through the edit profile, but the document will not be annotated with the correct text, since the content is not given. To illustrate, however, that the position and the length of the change is known, the document will be annotated with blurred text.

If the content is masked and the combination of level and position show that the position is partially masked, then the exact node where the operation is applied cannot be found. Using the given part of the position, an approximation of the exact position can be found, i.e. the closest ancestor of the changed node is found. For instance, the operation $gOp_{9Mary} =< edit, 4, [4], null, [0, 0, 0, 1],\ Mary >$ is of level character, but only the paragraph is given in the position element. Paragraph 4 is the deepest we can reach in the hierarchical document and hence we assign all the computed information to this node. Since we do not know exactly where the changes are made in the paragraph, the document is not annotated at all. This helps the user to distinguish this case from the previous one.

7.7 Summary

We have presented an awareness mechanism which, in the light of privacy issues, computes real-time awareness in semi-synchronous collaboration. Users can choose through a set of privacy levels the one that best fits their role, collaborative task, current needs and situation. The detail of information sent to other users about their uncommitted changes is defined by these levels. Based on the privacy level, various filters are applied to the original operations to mask some of the operation attributes and create ghost operations. Upon reception at a remote site, ghost operations are specially handled by our awareness mechanism to extract the maximum available information. The computed awareness information is finally visualised by means of edit profiles which give users a quick overview of the hot areas that contain concurrent changes made by their collaborators.

We believe that the examples used in this chapter constitute a representative, but by no means exhaustive, list of the most common activities and privacy concerns that users have when involved in collaborative tasks. User studies would be needed on a fully implemented version of the above, to analyse whether our awareness approach achieves improvements over traditional semi-synchronous collaborative applications. Additionally, we need to investigate whether there exist other privacy levels that need to be taken into consideration to extend the awareness mechanism towards satisfying more users in various collaborative situations. Finally, the users' ability to correctly interpret the information presented through the new edit profiles and the process of masking the attributes of an operation need to be investigated.

We would like to differentiate the concept of ghost operations and the work presented in this chapter with the work presented in the previous chapter and the notion of shadow document sets. Although both approaches aim to enhance semi-synchronous collaborative environments with the computation of real-time awareness, they concentrate on different aspects of these collaborative situations.

The introduction of ghost operations and the masking mechanism presented in this chapter address privacy issues raised by users when working with semi-synchronous environments. Therefore, it deals with the "encryption" of the information stored in real operations and the "decryption" of the information stored in ghost operations. All the operations handled by this approach are concurrent uncommitted modifications.

However, shadow document sets address a different issue. They concentrate on the correct handling of operations, when received at a user site, based on whether it is urgent for a user to be informed about the specific operations, i.e. based on whether the operations are committed or uncommitted.

Concluding, we would like to mention that both techniques could be merged and applied as one in existing semi-synchronous or asynchronous collaborative applications to enable them to provide awareness for both committed and uncommitted modifications while respecting any privacy issues that could arise concerning uncommitted work.

8 Further applications

The generality, in terms of working modes, of the awareness framework presented in Chapter 3 has been shown in the previous chapters, where the awareness mechanism was applied to a collaborative text editor for the enhancement of both asynchronous and semi-synchronous collaboration. In this chapter we show that the computation of awareness in our mechanism is not only independent of the collaboration mode, but also of the document type handled by the collaborative application. For that, we investigate if and how the awareness mechanism could be used to enhance collaborative applications that handle documents other than text. We chose a graphical editor and an editor of webpages, because although there is a lot of collaborative activity concerning these document types, there is a lack of awareness-enabled collaborative applications.

In the first part of this chapter, we consider the collaborative authoring of a website and investigate how our awareness mechanism could be extended to support change awareness about both *intra-document* and *inter-document* modifications. Although intra-document changes were successfully addressed in the previous chapters, this was not the case for inter-document modifications, i.e. modifications on one webpage that may affect a page linked to it. We propose a change awareness mechanism that monitors intra- and inter-document edits, taking into account changes made to a page and pages connected to it through *html* or *transclusion links.* The severity of all the changes is computed based on various metrics and on different semantic levels according to user preferences. A visualisation tool indicates how much a document and documents linked to it have changed. An edit profile allows users to easily spot parts with "interesting" changes within webpages.

In the second part of this chapter, we consider a graphical collaborative editor and how the awareness framework could be extended based on the operations supported by the application, as well as proper metrics and visualisation tools to compute the severity of modifications made to a graphical document and the presentation of this information to the user within a graphical editor.

8.1 Introduction on collaborative authoring of websites

Systems that support the co-authoring of websites often allow users to freely edit pages. Changes to a webpage may affect other parts of the page or pages linked to it and hence lead to semantic inconsistencies within or between pages. Keeping users aware of changes made by other users can help to resolve such inconsistencies or even prevent them. While many popular tools for the collaborative authoring of webpages such as Wikis, blogs and WebDAV applications succeed in enabling collaboration between users, they unfortunately provide almost no awareness to users about the changes done by their collaborators, and therefore do not prevent semantic inconsistencies.

We address the need of users to track changes made over time to co-authored webpages. We identify two categories of changes that may appear: *intra-document changes* and *inter-document changes*. The first category includes changes made to a single webpage. Users need to be informed in detail about the type of changes, which part of the page is affected, in which way and how much. An overview of the changes should be provided to the users so that they are aware of the group activity and can react quickly to changes that might affect their work. Webpages often contain links to other pages and the target pages may be semantically connected. Additionally, parts of webpages are often reused in other webpages resulting in "transcluded" webpages [89] indirectly linked to "compound" webpages [74] that include the reused information. In both cases, changes to a target/transcluded page may directly or indirectly influence the content of the source/compound page resulting in *inter-document changes*. In the process of collaborative authoring of a website, it is therefore important that a user is also notified about concurrent changes made to linked documents.

In Chapter 3, we proposed a mechanism that computes awareness information about *intra-document* changes made to structured documents. In this chapter, we show how we applied the same mechanism to collaborative situations where websites are authored. We also show how we extended our awareness mechanism to deal with links between webpages.

We propose a mechanism that based on *html* and *transclusion links*, informs authors of source/compound webpages about changes made to the target/transcluded webpages. The mechanism additionally computes and visualises change awareness about *intra-document* changes at the different syntactic document levels of one or more webpages. We begin with a motivating example to show the kinds of information that would be useful for users when co-authoring webpages. We then present our approach to computing change awareness for *intra-* and *inter-document* modifications and finally introduce a visualisation tool to show the computed awareness information.

8.1.1 Motivating example

We use an example to motivate the need for users to be informed about intra-document and inter-document changes to a website. Consider the case of a research group that maintains a website about their activity. The structure of the website is shown in Figure 8.1.

The group is involved in several projects (Project 1, Project 2, Project 3) with a subset of group members participating in each. The group website is composed of a main page and a set of project pages. The main page contains a description of the group research topics, a list of the group members, a summary of the group projects together with links to the internal webpages describing the projects, and links to an external webpage of Project 1 maintained by a collaborating research group. We assume that the summaries of the group projects are transclusions of the short descriptions of the projects found on the associated webpages.

Figure 8.1 shows the *html* and *transclusion links* from the group main page to the projects' internal and external pages. To keep the group website consistent, users working on the main page would like to be continuously informed about changes made to:

- *The main page itself.* Users need to be informed in detail about the type and location of changes, as well as their severity on the parts of the page where they were made and the page as a whole.

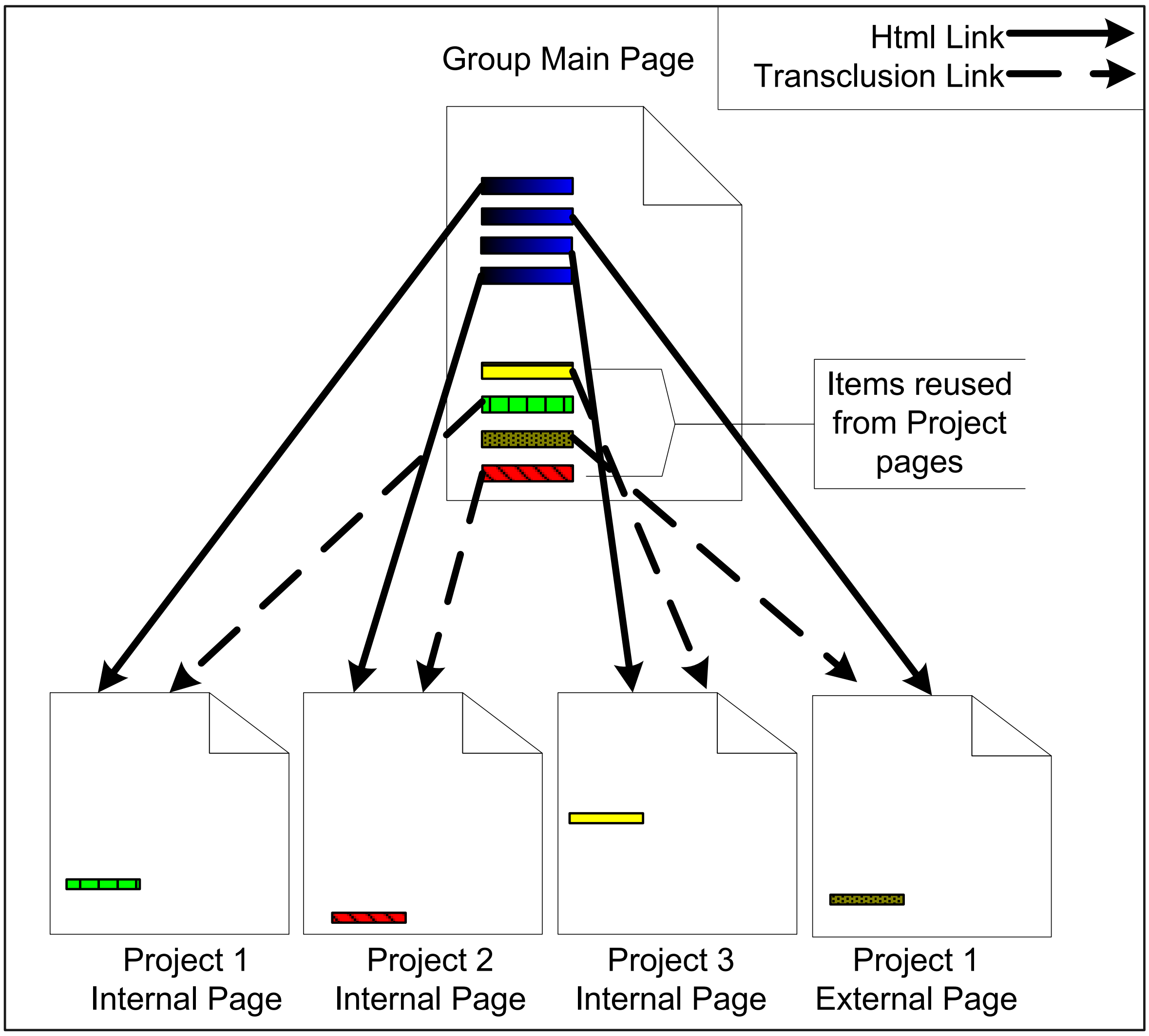


Figure 8.1: Structure of a group website

- *The transcluded parts.* Any change made to the source document of a transcluded part must be tracked and the compound document must be notified. After refreshing the transcluded part to show its latest status, awareness information about the detailed changes made to it needs to be given as well.

- *The linked pages.* Users editing the main page should be informed about changes made to the linked webpages as these changes might produce inconsistencies in the main page. For instance, if a new project member is added to an internal project, the complete list of participants on the group main site should be updated. In the same way, if a new software release is announced on the external page of Project 1, this information could be used to update the group main page.

The visualisation mechanism that presents change information to the users should provide an overview of changes as well as details at a selected document level such as a section or paragraph, the transcluded parts and the linked pages.

Some tracking tools allow the possibility to follow links from a source page, but they do not relate changes made to the linked pages with the source page. Therefore, a user working on the source page is not aware about changes made to the linked pages. In the following sections we present our awareness mechanism for intra and inter-document changes. Users working on a webpage are informed about changes made to the main page and linked pages without visiting those webpages.

8.1.2 Special issues in co-authoring of web documents

In this section, we take a closer look at the co-authoring of websites and search for any special characteristics that this collaborative activity or the authored documents may have. To do so, we compare the co-authoring of a website to the co-authoring of a text document. After highlighting any differences and special issues that arise when users are collaborating over websites, we describe how an awareness mechanism could be used for this collaborative activity and whether any extensions to the mechanism previously proposed would be required. We consider the following issues.

- What are the document parts of a webpage? How are webpages linked to each other?
- What are the possible modifications a user should be aware of when co-authoring webpages?
- How are modifications generated and how can they be modelled?
- Are there any privacy issues we need to take into consideration?

For the sake of simplicity, we will consider webpages that consist only of text. Therefore, the document parts included in a webpage are identical to the ones included in a text document. Additionally, each webpage is viewed as a structured document. Through the structured document model, we can address different parts of a webpage at various syntactic document levels and compute awareness information about how much they have changed at different granularity levels. We adopt the definition of a document node introduced in Chapter 3.

The modifications that can be made to webpages are the insertion or deletion of text and the creation or deletion of links. They are mapped to operations applied to specific parts of the page. Examples of operations are the insertion/deletion of a new paragraph to/from a section or of a sentence to/from a paragraph or of a word to/from a sentence etc. Operations are also defined as in Chapter 3. These modifications alter either the structure of a webpage, i.e. *structural operations* or the structure of a website, i.e. *linking operations.* They are presented in Figure 8.2 as specialisations of the *operation* concept.

Following the document's structure, the severity of changes made to document parts at lower syntactic levels, such as word or sentence in the case of text, are aggregated to show the total changes made to the document at a higher syntactic level such as a paragraph. Monitoring changes made to different parts of the document and evaluating their severity returns the required intra-document change awareness. Until now, everything is identical to the collaborative authoring of text documents, with the existence of links being the only exception.

Links added to, or removed from, a webpage are important actions of website authoring. It determines the interdependence of webpages and therefore the structure of a website. The structure of the authored sites is a graph and not a tree as in the case of text editing. Since the awareness mechanism presented in Chapter 3 considers only hierarchical documents, it cannot be applied in its current form to the co-authoring of more complex structures. Therefore, we extend the metamodel describing the awareness mechanism as shown in Figure 8.2 to include the notion of *html links* and *transclusion links.* This is modelled through the associations *HtmlTo*, *HtmlFrom*, *TransTo*, *TransFrom* that relate node objects to each other.

An alternative way to model links between webpages would be to model *links* as a separate notion. *Node* objects would be connected to each other with links to form a graph. *Links* could then be separated into *intra* and *inter* links. The *hasParent* and *hasChildren* presented in Figure 8.2 would then be examples of *intra links* and the *HtmlTo*, *HtmlFrom*, *TransTo* and *TransFrom* examples of the *inter links.* Such a metamodel is more general than the one presented in Figure 8.2 since it enables the creation of more complex graphs. Additionally, since links are modelled as separate concepts, they can hold additional information. We present some details of such an extended model in Section 9.2. We realise that the potential of extending our framework towards this direction is great, since our awareness mechanism could be applied to more applications

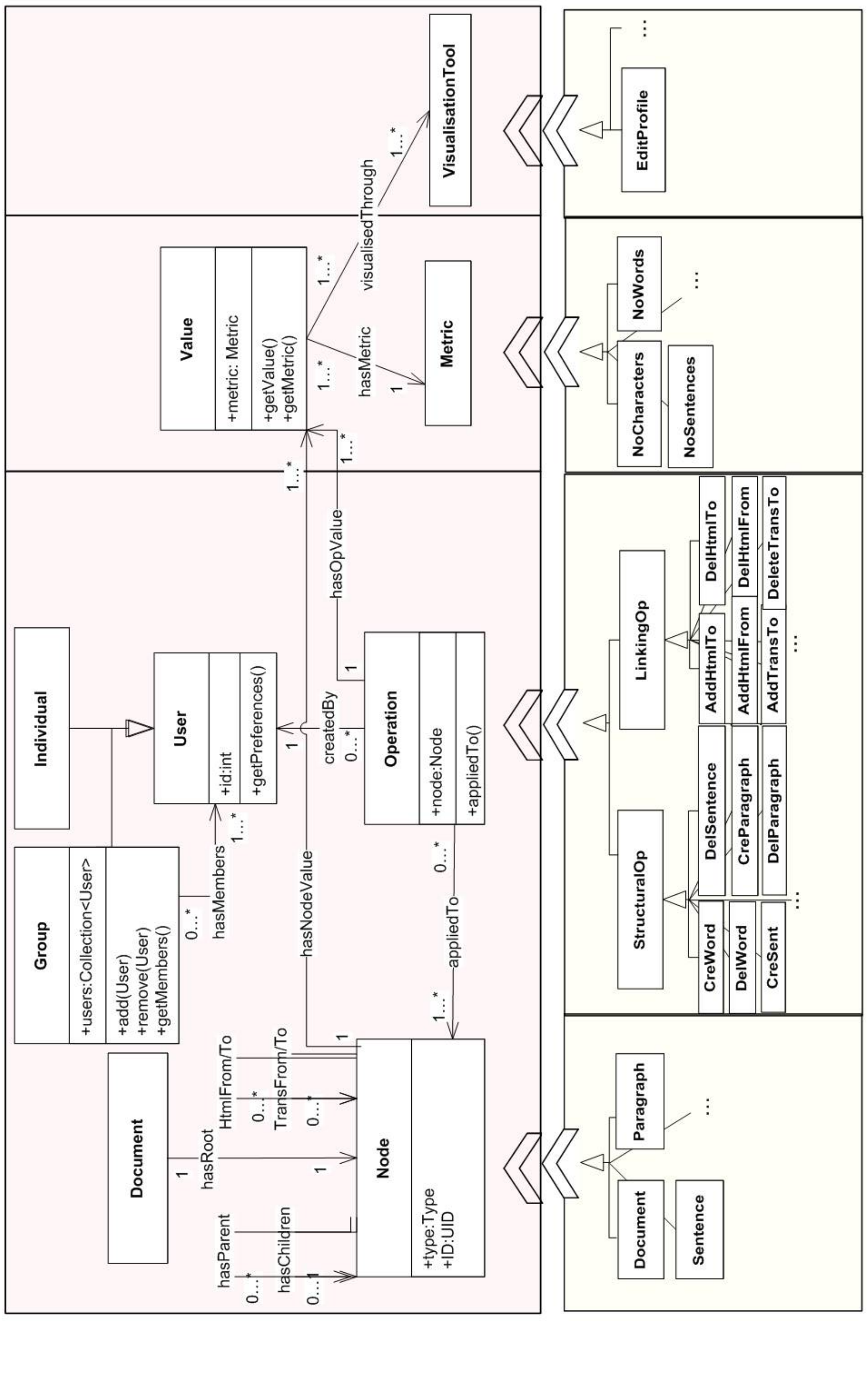


Figure 8.2: Extension of the framework's core concepts for co-authoring of websites

that support collaborative or single user editing tasks. However, building a framework that could be integrated into as many applications as possible is not the goal of this thesis. Before extending the framework into this direction, research would need to be conducted to investigate in detail the needs of users of these applications and the need for multi-level awareness on documents with a more complex structure.

Note as well, that in the model we use, there is no distinction between internal and external pages to a website. Therefore, a webpage that belongs to a website where a user has authoring access might be linked to external webpages that are not authored by the user. This means that, although users need to be informed about modifications made to the external pages, there is no way to track the changes made to them. This raises again the issue of the *operation-based approach* versus the *state-based approach* to compute the changes between two versions of a document. With the first approach, the operations that transform a webpage from one version to another can be created by using a special web application when authoring the page that captures the changes. With the second approach, they can be computed by using a diff algorithm [88] between the two versions.

When an application that captures user changes is used by all users, for instance when working with internal pages, the captured operations reflect the actual changes made to the page. When a diff algorithm is used, a set of operations that transform the initial state of the webpage to its final state is created. Unfortunately, this set may not be identical to the set of actual changes made by users, resulting in loss of some information. The advantage, though, of a diff algorithm is that it can be used to compute the difference between two versions of any webpage without monitoring the authoring procedure, which applies to the situation described above with the external pages.

Being aware of these issues, we chose to use a diff algorithm in this chapter to enable the computation of awareness information for both internal and external pages. When addressing such problems, we believe that being able to offer awareness information, even if operations do not capture the exact changes, is far more beneficial than not delivering any information at all. Of course, in situations where only internal pages are authored and the accuracy of computed awareness is crucial, an authoring application that captures user changes can be used instead of a diff algorithm.

8.1.3 Tracking of modifications and computation of awareness

The procedure followed to compute intra- and inter-document changes and the corresponding awareness information is shown in Figure 8.3. A *PageCache* is maintained locally for each user, including a copy of each of the webpages of interest to the user, as well as a copy of all the pages linked to them. We refer to all of these as *monitoredPage*s. Additional information is kept in each *monitoredPage* showing how much each part has changed. A background process periodically calls the method *updateValues* for each *monitoredPage* to check whether any intra- or inter-document changes occurred and compute their severity.

To compute awareness information for the intra-document changes, we call the method *getIntraValues* for the currently monitored page. *getIntraValues* takes the cached (old) version of the page and the online (current) version from the web and compares them. If they are not identical, the procedure in the block "condition", shown in Figure 8.3, is followed. Initially, a diff algorithm is used to compute the changes that transform the old version of the page into the current one in terms of operations denoted as *changes*. The *changes* are used by the method *computeIntraValues* to compute the *intraValues* attached to the changed parts of the page. This procedure is detailed in Subsection 8.1.4. Finally, these values are stored in the *monitoredPage*, the new version of the page is stored in *PageCache* and *getIntraValues* returns the computed intra values associated with parts of the document.

To compute the awareness information for the inter-document changes, the linked pages need to be accessed. Therefore, the list *listOfLinks* with all links of the *monitoredPage* is retrieved. Note that both *html* and *transclusion links* are included in the list and *listOfLinks* is accessed after the current version of the *monitoredPage* has been stored locally. Therefore the list contains the links included in the current version of the page. Any link that was present in the old version but removed in the current version is not included. Similarly, any link that was not present in the old version but was added in the current version is included. We believe it is reasonable to search for changes only in pages currently linked to the *monitoredPage* as the removal of a link is already depicted as a change in the *monitoredPage*.

In the loop that follows, we handle each of the links separately. We initially retrieve the *linkedPage* and compute the intra-document awareness for it by calling the method *getIntraValues* as for the *monitoredPage*. As

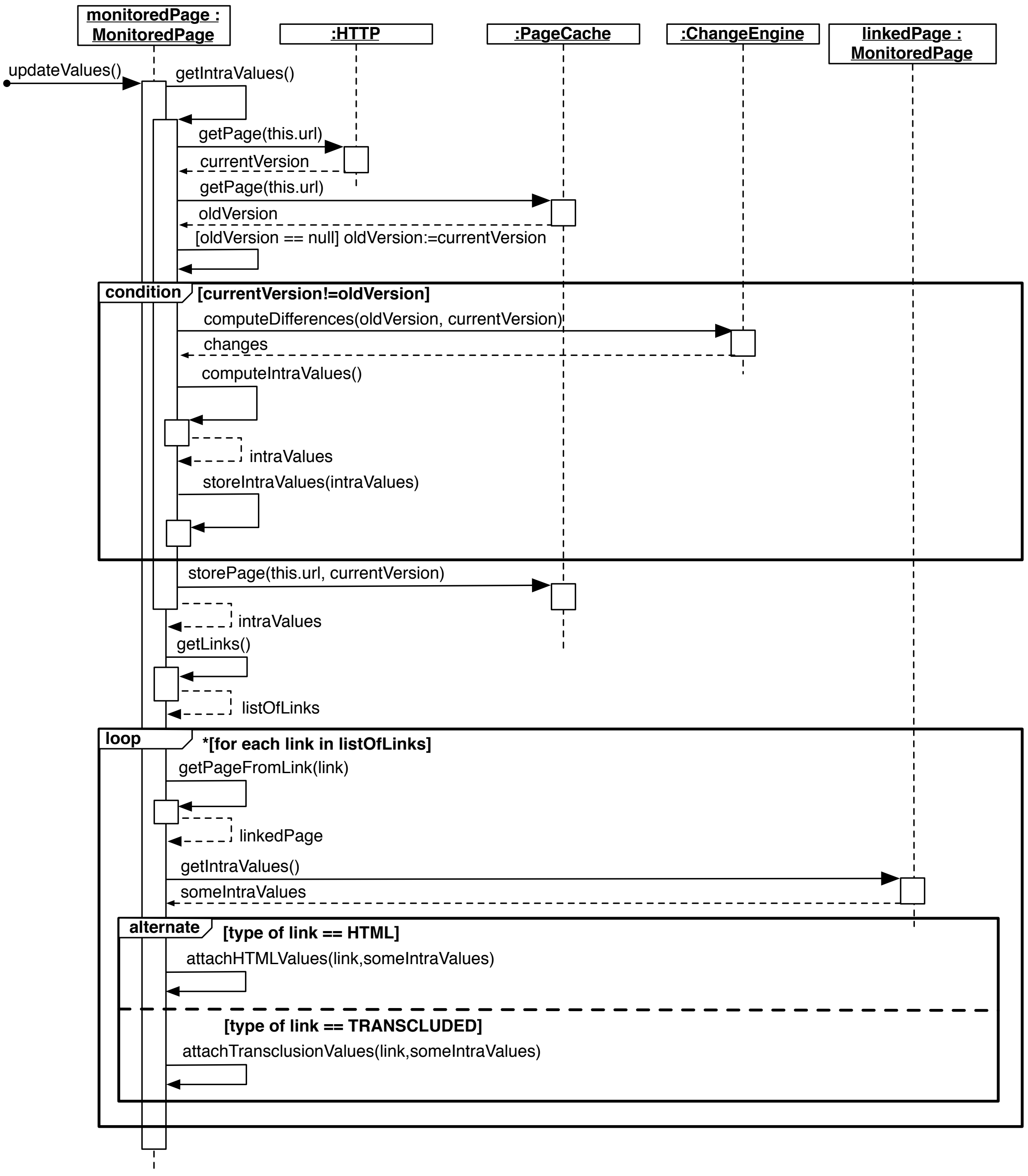

monitoredPage : MonitoredPage
:HTTP
:PageCache
:ChangeEngine
linkedPage : MonitoredPage
updateValues()
getIntraValues()
getPage(this.url)
currentVersion
getPage(this.url)
oldVersion
[oldVersion == null] oldVersion:=currentVersion
condition
[currentVersion!=oldVersion]
computeDifferences(oldVersion, currentVersion)
changes
computeIntraValues()
intraValues
storeIntraValues(intraValues)
storePage(this.url, currentVersion)
intraValues
getLinks()
listOfLinks
loop
*[for each link in listOfLinks]
getPageFromLink(link)
linkedPage
getIntraValues()
someIntraValues
alternate
[type of link == HTML]
attachHTMLValues(link,someIntraValues)
[type of link == TRANSCLUDED]
attachTransclusionValues(link,someIntraValues)

Figure 8.3: Process for computing intra- and inter-document awareness

explained before, any new version of the *linkedPage* is detected and the changes made to it as well as the corresponding awareness information are computed. Finally, the object *someIntraValues* is returned, including the computed awareness information. *someIntraValues* may be null if the current version of the *linkedPage* is identical to the old version and different than null if the versions differ. Note that if the link was added to the current version of *monitoredPage*, the *linkedPage* is inserted in the *pageCache*. The old version of the *linkedPage* is identical to the current one, so the *someIntraValues* will be null. Changes made to the new linked document will be monitored and reported starting from the moment the link is added. This part of the loop is executed for both *html* and *transclusion links*. The intra-awareness information computed for the *linkedPage*s is then attached to the structure of the *monitoredPage* based on the type of the link. The procedures *attachHTMLValues* and *attachTransclusionValues* are described in Section 8.1.5.

It is obvious from the procedure described here that we compute inter-document awareness information based on first level links only, i.e. we consider changes made only to pages directly linked to the current page. Although the procedure can easily be adapted for other levels, i.e. links included in linked pages, we believe that the information delivered in this case would not be of additional value to the collaborative situations we address in our current work.

8.1.4 Intra-document awareness

As described in Section 8.1.2, the document model and the operations defined for the editing of webpages are similar to the ones defined for text editing. Therefore, the computation for awareness information about intra-document modifications is identical to the one described in Chapter 4. The metrics and the visualisation tools used are also the same, as shown in Figure 8.2.

8.1.5 Inter-document awareness

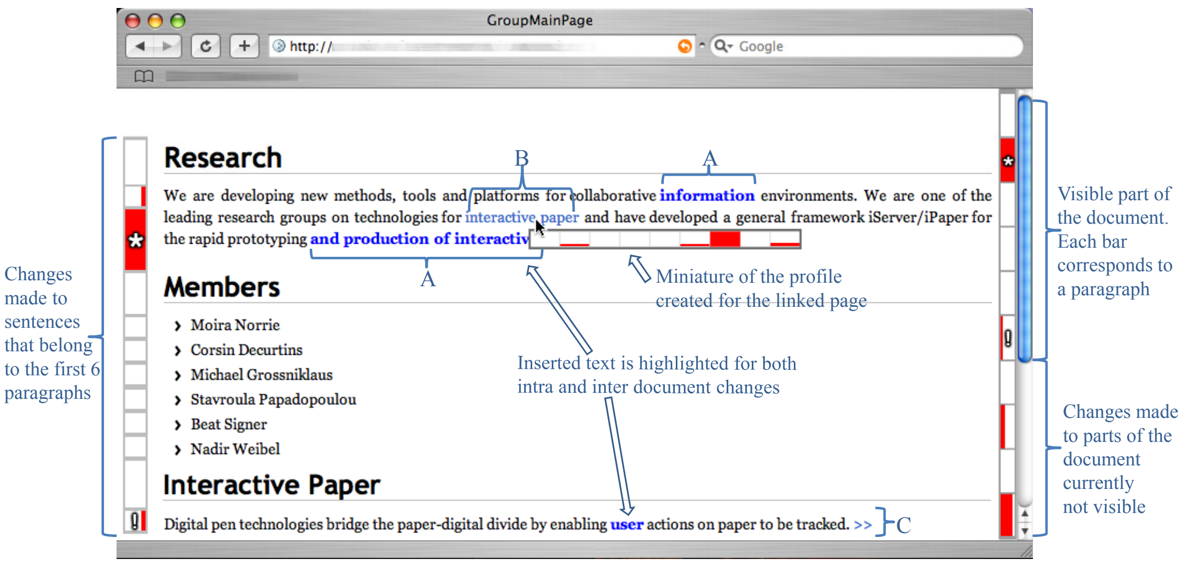
We now describe the procedure followed to attach to a *monitoredPage* the awareness information computed for a *linkedPage*. As described in Section 8.1.3, the awareness information is computed in the same way for all *linkedPage*s linked through either an *html* or a *transclusion link* to a *monitoredPage*. This awareness information comprises a set of *nodeValue*s computed for each of the parts of the *monitoredPage*. To keep the awareness information delivered through the *monitoredPage* up-to-date, we need to attach the computed values to the parts of the *monitoredPage* that are affected.

If the link to the *linkedPage* is an *html link*, the part of the *monitoredPage* where the link is included needs to be updated. We decided to attach to this part all the awareness information computed for the *linkedPage*. In this way, we provide the user with the possibility while browsing through the *monitoredPage* and watching intra-document changes, to also be informed about the changes made to the *linkedPage* without loading the page. Our aim is to deliver to the user an edit profile with the quantity of the changes in the *linkedPage*, as well as the parts of the page that changed. In this way, the user can decide whether they need to revisit the linked page and update the information included in the *monitoredPage*. The way the awareness information is presented to the user is shown in Section 8.1.6.

If the link to the *linkedPage* is a *transclusion link* then only part of the awareness information computed for the *linkedPage* needs to be transferred to the *monitoredPage*. Through the transclusion mechanism, the part of the *monitoredPage* that is transcluded from a *linkedPage* is updated to include the latest changes made to it. However, updating the transcluded part in the *monitoredPage* is not enough. Awareness information that is attached to this part, needs to be transferred to the *monitoredPage* as well so that users are informed about the amount of changes. To do so, the awareness information computed for the *linkedPage* is scanned and only the values computed for the transcluded part are kept. This mechanism is heavily dependent on the transclusion mechanism and the metadata that is created during the authoring process of the *transclusion link*. This metadata is usually new tags inserted in the transcluded or the composed pages, or other anchors that can be used to define the beginning and end of the transcluded part [73]. Using this data, we identify the transcluded part in the *linkedPage*, extract the awareness information computed for it and copy it to the corresponding part of the *monitoredPage*.

8.1.6 Visualisation of intra/inter-document awareness

We revisit our motivating example to present the tools used to visualise the computed awareness information. We use only a representative sample of changes made to two of the example pages to demonstrate how the visualisation tool informs users about different kinds of changes. We assume there are some intra-document changes (annotated with "A" in Figures 8.4 and 8.5) made to the main page and the internal page of project 1 (interactive paper project). The main page has an *html link* to the project page through the phrase "interactive paper" in the first paragraph (B) and the paragraph about the interactive paper project is a transcluded paragraph (C) from the project page. The changes made to the main page are insertions of text in the paragraph about group research. The changes made on the project page are an insertion of a word in the first paragraph (the transcluded paragraph), the insertion of two new paragraphs (the sixth and seventh paragraphs) and the insertion of a new member in the list of group members (paragraph number nine). Note that titles are handled as paragraphs of one sentence and changes of the project page are considered as inter-document changes for the main page. Finally note that an external page linked through the main page could also be used instead of the internal project page and the computed awareness information would be visualised in the same way.


GroupMainPage
http://
Google
B
A
A
C
Changes made to sentences that belong to the first 6 paragraphs
Miniature of the profile created for the linked page
Inserted text is highlighted for both intra and inter document changes
Visible part of the document. Each bar corresponds to a paragraph
Changes made to parts of the document currently not visible

Research

We are developing new methods, tools and platforms for collaborative **information** environments. We are one of the leading research groups on technologies for interactive paper and have developed a general framework iServer/iPaper for the rapid prototyping **and production of interactiv**

Members

- Moira Norrie
- Corsin Decurtins
- Michael Grossniklaus
- Stavroula Papadopoulou
- Beat Signer
- Nadir Weibel

Interactive Paper

Digital pen technologies bridge the paper-digital divide by enabling **user** actions on paper to be tracked. >>

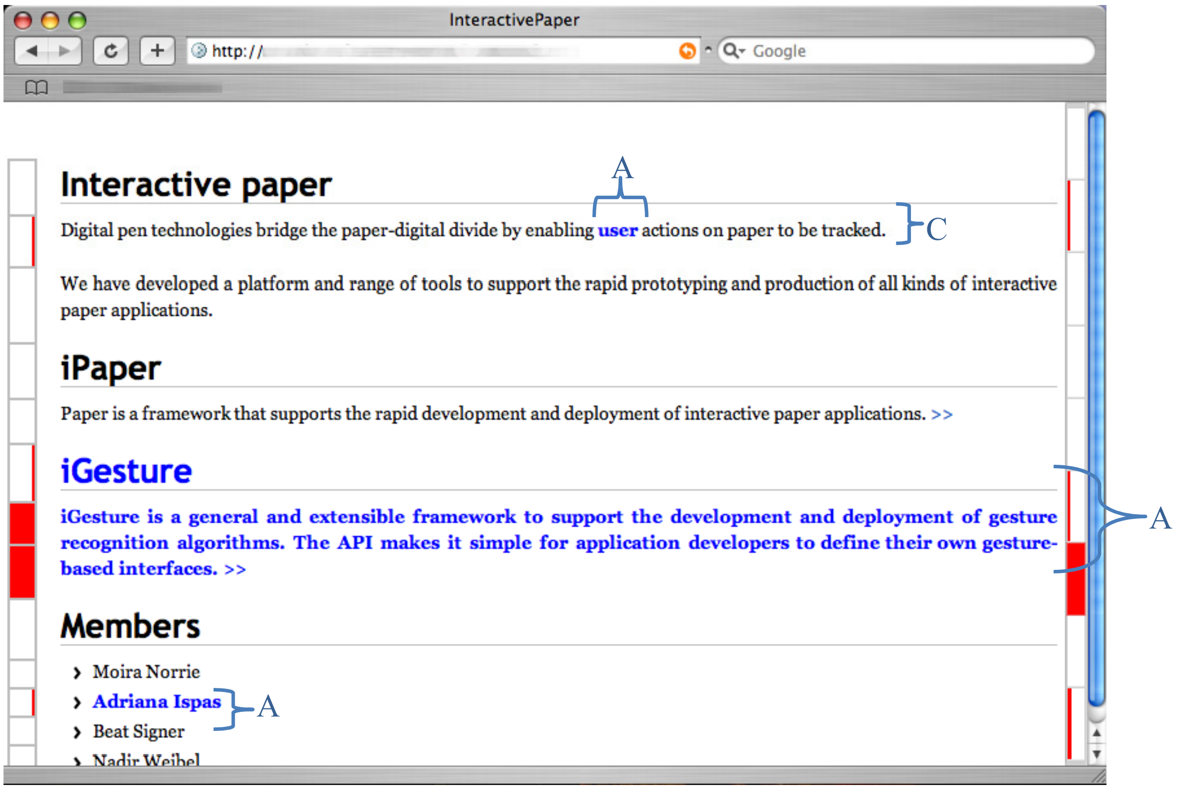
Figure 8.4: Visualisation of intra- and inter-document changes made to the main page

We assume that the main page contains 10 paragraphs. The window in Figure 8.4 presents the first 6 paragraphs. In this figure, we also show how we visualise the intra-document changes of the main page. Two edit profiles are introduced on the sides of the page. The profile on the right side shows an overview of all changes made throughout the page, while the profile on the left shows all changes made to the portion of the page currently shown. In order to construct the profile on the right side presented in Figure 8.4, the awareness information computed for each paragraph is used. By using the profile, the user can instantly spot that there have been many changes on the second and the last paragraph and some changes in the sixth and eighth paragraph. The left profile can be thought of as a zoomed version of the right profile. The information provided by the left profile corresponds to the first six paragraphs, which is the part of the right profile marked by the scrollbar.

The reason for the existence of the left profile, is that the information provided by it can be at a different syntactic level. In Figure 8.4, for instance, the left profile shows changes made to each of the sentences of the first six paragraphs. In this way, a user that would notice the large amount of changes made to the second paragraph, could further filter the awareness information and find how the changes in the paragraph are distributed to the paragraph sentences. The height of the bar corresponding to each sentence is adjusted to the length of the sentences for the left profile, to ease the mapping of changes to document parts. The width of a bar in each profile represents the normalised number of words inserted or deleted in the corresponding document part. Note that users are able to configure the syntactic levels used in the profiles as well as the detail of the information provided through them to visualise information about changes of various types on a user-defined granularity. The intra-document changes made to the project page are also visualised in the same way, as shown in Figure 8.5.

Since the first paragraph of the main page includes an *html link* to the project page, changes made to the project page might influence the content of the main page. For instance, the introduction of a new prototype (iGesture) in the project page might need to be included in group research summary (first paragraph) in the main page. Hence, when a user points with the mouse on the *html link*, an overview pops up that shows the changes made in the project page. We expect this feature to give the user a rough idea of the amount of changes made to the project page as well as the parts of the page that changed the most to help the user decide whether they need to visit the project page to see the changes in detail and whether the content of the main page should be modified as well. With the "*" symbol on the profiles we inform users if there are changes made to pages linked through *html links*.

The changes made to transcluded parts are presented as intra-document changes through the edit profiles. The sixth paragraph of the main page is a transcluded paragraph. The changes made to it are presented through the two profiles in the main page, exactly in the same way as



InteractivePaper

http://

Interactive paper

Digital pen technologies bridge the paper-digital divide by enabling user actions on paper to be tracked.

We have developed a platform and range of tools to support the rapid prototyping and production of all kinds of interactive paper applications.

iPaper

Paper is a framework that supports the rapid development and deployment of interactive paper applications. >>

iGesture

iGesture is a general and extensible framework to support the development and deployment of gesture recognition algorithms. The API makes it simple for application developers to define their own gesture-based interfaces. >>

Members

- Moira Norrie
- Adriana Ispas
- Beat Signer
- Nadir Weibel

Figure 8.5: Visualisation of intra-document changes made to the project page

they are presented in the project page. The distinction between the transcluded and compound page is made through the “!” symbol added to the compound page at the parts of the profiles that correspond to the transcluded parts.

Our awareness mechanism and the visualisation tool introduced above aim to increase the amount of awareness information presented to users about changes made to a page and the pages linked to it. In the absence of adequate awareness information, a number of problems could appear. For instance, a user editing the main page would not be informed about changes made to the linked page (interactive paper project). They would not know that the list of members in the interactive paper project has been updated since a new member was added. As a result, the list of members in the main page would not be updated to include the new member. Additionally, users would not be informed about the new prototype that is released in the interactive project and therefore, information about the released frameworks on the main page would remain obsolete. Our awareness mechanism supports the users to avoid all of the above situations.

8.1.7 Discussion on co-authoring of websites

We have presented a change awareness mechanism that tracks intra- and inter-document edits in the co-authoring of webpages. We described how metrics are computed to quantify the changes made inside a document as well as on transcluded parts of the document and linked documents. Moreover, we have described a visualisation tool based on edit profiles that enable users to have an overview of changes done on a webpage and any pages linked to it. This allows users to easily spot "interesting" changes done on document parts and browse these changes at finer levels of granularity.

It would be interesting to test the usability of our approach by conducting user studies on the prototype that implements the ideas described in this section. The extensibility of our approach would also enable the experimentation with various visualisations. Finally, it would be possible to extend our awareness mechanism to take into account not only physically linked documents, but also semantically linked documents in the context of semantic wikis or more general semantic web.

8.2 Introduction on collaborative authoring of graphical documents

Although change awareness is an important factor in collaboration, there has been relatively little work to date on ways to compute and visualise summaries of changes made using graphical authoring tools. In this section, we address change awareness in graphical applications. Since we concentrate on co-authoring activities, the basic user needs for change awareness remain the same. Therefore, the set of requirements presented in Chapter 2 applies in co-authoring of graphical documents as well.

This implies that the awareness framework presented in Chapter 3 can be used to model the computation and visualisation of awareness information in collaborative graphical applications. For this, the only requirements that the application should fulfil, is to provide a structured document model and the notion of operations to model the authored document and the modifications made to it respectively.

For the applications that fulfil the requirements, the integration of the awareness framework into them should be a straightforward task. In the rest of this section, we describe how the awareness mechanism could be integrated into a collaborative graphical editor. To do this we revisit the metamodel of the framework and define the form that its core concepts would need to take for a graphical application.

8.2.1 Issues related to graphical applications

A first issue that arises when trying to integrate the awareness framework into a graphical application is that an underlying structured document model is required. This might not seem straightforward in the beginning, since there is no direct resemblance to a structured document as in the case of a text document. For a text document, it is clear that a section consists of paragraphs and a paragraph consists of sentences, since this hierarchy resembles the syntactic levels with which users are already familiar. Does this apply to graphical documents as well? If we take a closer look at graphical documents and the way they are authored, very few graphical documents have no structure. On the contrary, many of them include graphs that span many pages and may include graphs split into layers, each of them including graphical objects, or groups of objects, or templates of objects, etc. This proves that although a hierarchical structure may not be instantly noticeable in a graphical document, it is still there.

Note, however, that while a hierarchy can be defined in both text and graphical documents, in the case of text documents, the children of a node are ordered. Therefore, any node can be uniquely identified by a path in the document hierarchy. However, in graphical documents, nodes are not ordered. For instance, there is no order between objects that belong to the same group of objects. Objects are therefore identified by unique identifiers. Additionally, the place where a graphical object will be drawn in the scene of objects is not related to its place in the hierarchy representing the document. This is specified explicitly by the users, by setting the object *position* attribute.

The visualisation tool is the next issue and is also related to the position of objects in a document. Since objects are drawn in a 2 dimensional (or 3D) scene of objects, an overview presenting the modifications throughout the document cannot be linear, i.e. of one dimension, as in the case of a text document. The lack of order between objects is another reason that renders the edit profiles unusable in the form they were already presented. Therefore, an appropriate visualisation tool must take these issues into consideration. In the next sections we present a mock up of such a visualisation tool and the functionalities we envision that it will have.

In the next section, we start with a presentation of the extended

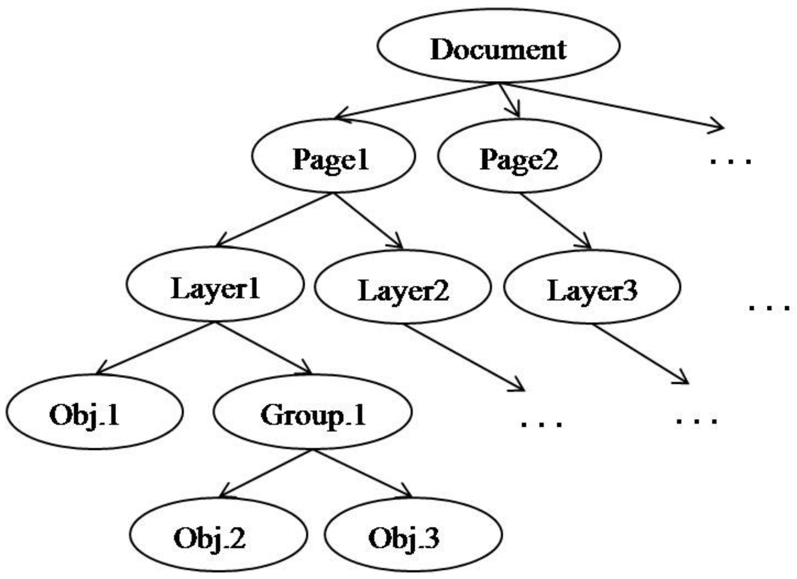


Figure 8.6: A hierarchically structured graphical document

framework metamodel describing the computation and visualisation of awareness information in collaborative graphical authoring tools. We detail how its general concepts have been specialised for the case of graphical applications.

8.2.2 Extension of the awareness framework for a graphical application

We chose to work with Draw-Together [68], a collaborative graphical editor developed in our group. Draw-Together has the hierarchical document structure shown in Figure 8.6. Documents handled by the application consist of nodes of different document levels, i.e. *pages*, *layers*, *groups* and *objects*. The application also supports the concept of operations modelling modifications made to the document.

In the rest of this section, we present the extension of the metamodel that describes our awareness mechanism. The metamodel is shown in Figure 8.7. It is composed of the core model shown in the upper half of the figure, which is the one introduced in Chapter 3, and the specialisation of its concepts for graphical authoring tools shown in the lower half of the figure. Alongside our description of how the main concepts are extended for graphical tools, we also discuss the similarities and differences between text and graphical authoring tools and some issues that emerge when working in these environments.

The concept of a node is further specialised for graphical documents

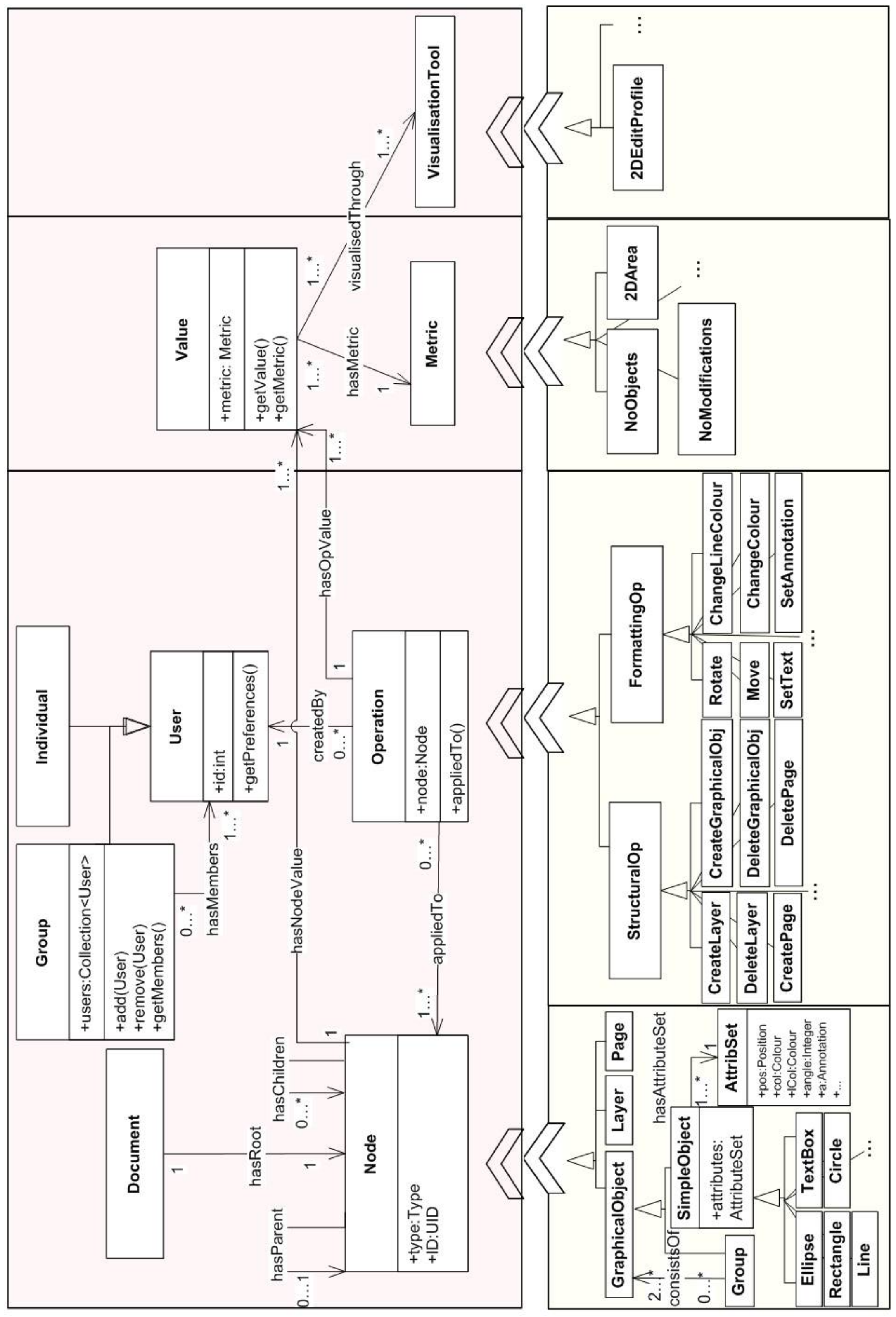


Figure 8.7: Extension of the framework's core concepts for a graphical editor

as shown in Figure 8.7. The notions of a *page*, a *layer* and a *graphicalObject* are defined as specialisations of a node. A *graphicalObject* can either be a *simpleObject* or a *group*. As *simpleObjects*, we define any of the objects that a graphical collaborative authoring tool supports, for instance an *ellipse*, a *rectangle* or a *textbox*. A group consists of a set of any *graphicalObjects*, as specified by the association *consistsOf*, thereby also allowing the creation of subgroups. Through the *hasChildren* and *hasParent* associations, the hierarchy of Figure 8.6 is created in a similar manner to the hierarchy of paragraphs-sentences-words-characters defined in text documents. Finally, attributes are assigned to each *simpleObject* as an *attributeSet* through the association *hasAttributeSet*. For example, an attribute can be the object's *colour*, *text* in case of a textbox, *annotation*, *angle* representing the rotation angle of an object, or the object's *xy* position in the 2D scene of objects, denoted as *position*.

We classified the operations applied to document nodes in graphical applications into *structural* and *formatting* operations. Structural operations modify the document structure. The creation or deletion of a node are typical examples of structural operations. Other structural operations could also be defined. Formatting operations modify a node's attributes. *Move*, *changeColour*, *rotate*, *setAnnotation* and *setText* are only some of many formatting operations defined in our model to describe collaborative activity in graphical authoring tools. The set of operations used in the text authoring tool of Chapter 4 consists only of structural operations that create and remove nodes. The much richer set of operations defined for graphical documents in comparison to the set of operations defined for text documents, renders the computation and visualisation of awareness information more complex in graphical applications.

Since modifications are made to nodes through operations applied to them in both text and graphical documents, the concept of metrics is defined for both document types in a similar manner. An operation's value depends on the metric used. In text editors, for instance, a *deleteSentence* operation delivers different information, i.e. has different value, when a user is interested in the total number of deleted sentences or the total number of deleted characters in a document. Our proposition of metrics and visualisation tools for graphical documents is influenced by the issues presented in the previous section.

Our proposal of an example metric would be to count the number of modifications made at each *xy* position of the 2D scene. Since modifications to a document or document part are made in the form of operations applied to graphical objects and not to xy positions, the number of modifications at a specific xy position is identical to the number of modifications made to the graphical objects that occupy the corresponding position. Therefore, the metric is defined as the number of modifications made to a graphical object. This justifies the metric *NoModifications* shown in Figure 8.7. Note, however, that there exist various types of operations applied to graphical objects. To compute and present awareness information as required according to users' focus and granularity, we allow the computation of node values for each of the operation types. As a result, a user interested in only a specific type of changes could visualise the corresponding values, without being distracted by node values that reflect other types of changes.

Definition 16. Based on the metric presented above, the value $opValue$ of an operation op of any type, is simply defined to be equal to 1 since it represents one modification. $opValue(op) = 1$

Definition 17. In a similar manner, the value $nodeValue$ of a node N for a given type of operation $opType$ is defined as the sum of the $opValues$ of the set of operations, denoted as $operations(N)$ applied to the node, all of them being of type $opType$.

$$nodeValue(N, opType) = \sum_{op_i \in operations(N)} opValue(op_i)$$

where $op_i(type) = opType$

The definition of appropriate metrics is a very demanding task due to its dependence on user preferences. The proposed metrics would need to be further enriched possibly by conducting user studies and collecting the users' feedback. Examples of new possible metrics could be the number of objects affected by an *operation*, as well as the area in the 2D scene that will be modified. Both are shown in Figure 8.7 as *NoObjects* and *2DArea*. It would also be interesting to investigate ways of combining the two metrics and their effectiveness.

Finally, we present our proposition of a visualisation tool for awareness in graphical authoring tools. It is presented in Figure 8.8 and in Figure 8.7 as *2DEditProfile*. The proposed tool is based on the concepts of heat maps [39] to provide a highlighting mechanism suitable for large documents. We consider a 2D layer drawn upon the 2D scene of objects. The area around each graphical object is coloured according to its *nodeValue* computed for a specific type of operation applied to it. If the user wishes to visualise the severity of many different kinds of modifications,

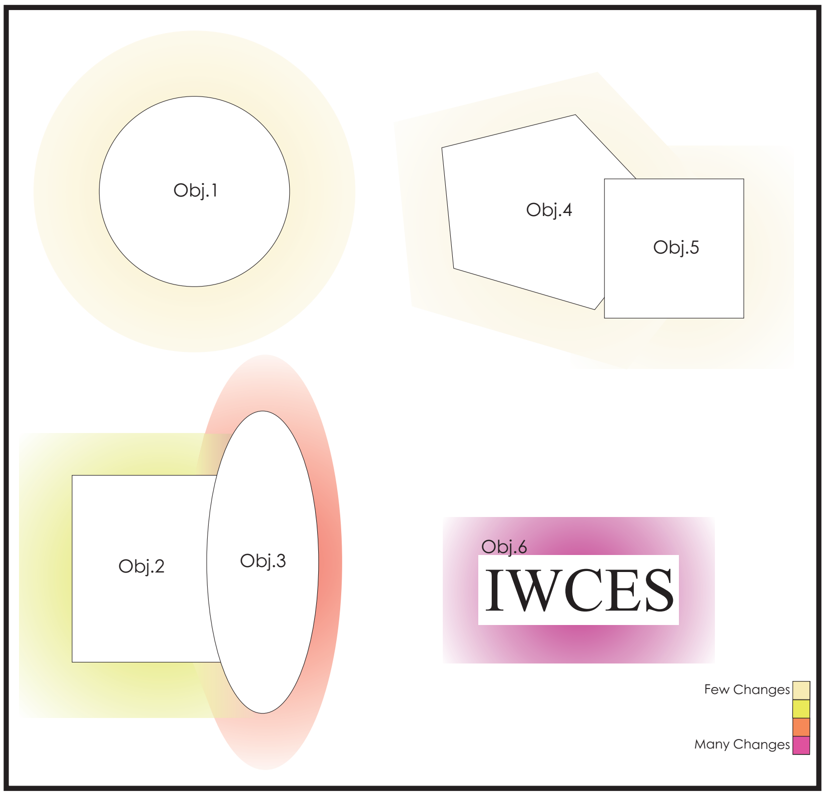


Figure 8.8: Example of a 2D edit profile as a visualisation tool for modifications over graphical documents

i.e. different types of operations, then the sum of the object's *nodeValues* is computed and the area around the object is coloured based on this value. Since an object's colour is one of its attributes and can be modified by users, the area inside a graphical object is not affected by the visualisation tool. The greater the number of modifications of each type made to the object, the greater the corresponding object's *nodeValue* and the darker the colour used to highlight the area around the object. The tone of the colour is more intense around the border of the object, while it gets less intense further away from it.

Consider, for instance, the example of Figure 8.8. Objects 1 and 6 have a different number of modifications made to them. Object 1 has fewer modifications, hence the colour around it is lighter than the one around object 6. Group objects are specially handled. If all objects in a group have undergone the same number of modifications, the colour around them is uniform as with objects 4 and 5. On the contrary, if the number of modifications of each of the group's objects differs, then different colours are used to highlight the areas around the objects. At the intersection of these areas, the colour is a combination of both colours.

In this example, object 3 has undergone more modifications than object 2.

The above visualisation tool offers an overview of modifications made to a 2D scene of graphical objects. Most of the time, such a scene represents a page of a graphical document. It would be interesting to investigate how such a tool could be extended to present awareness information about modifications made to objects at various layers, or to present more than one page at a time or zoom into a specific document part and present information about it. Additionally, it would be interesting to experiment with a slight change in the use of colours in the visualisation tool by assigning specific colours to specific kinds of changes. This would help users easily distinguish the type of a change only based on the colour used for highlighting the modified object.

A further issue to investigate could be the granularity of awareness information required by users with different roles and collaborative tasks. We expect that visualising the available information at the different document levels will be useful for users, however, we believe that information would be required at different document levels depending on the users' roles and collaborative tasks. Note that this issue was already raised for the text editor, by the users of the user study presented in Chapter 5.

The next example illustrates the above issue. Consider a co-authored graphical document that describes the plan of a building that has multiple floors and multiple apartments pro floor. We assume that there is a document page for every floor. A user that is working on the whole document, would most probably need to have an overview of all the document, to see where in the document modifications were made by the collaborators. The document parts, for instance the pages, where "many" changes have been made could be located from the overview. As a next step the users could zoom in on the specific pages and be informed about the modifications made to each apartment, or the rooms of the apartments. However, in other collaborative situations, a user might not need to have an overview over the whole document. If for instance a user is responsible for the design of a specific apartment, or floor, then they might need to directly zoom in and check for changes at the desired document level or document part. We believe that user studies could be a means of analysing and understanding the users' needs for each of the above collaborative situations and therefore guide the extension of the concepts included in the metamodel of Figure 8.7.

Finally, considering the implementation of the awareness framework that we proposed, we believe it is necessary to offer users the possibility of filtering the computed awareness information by selecting the type of modifications they wish to visualise. In a similar way to the implementation of this awareness mechanism for text documents, a flexible GUI that will enable users to select any combination of operation types to be visualised, at any granularity level could be built. Then a user would be able to visualise, for instance, only deletions of objects that belong to a specific layer of a specific page, or all of the rotated objects of a given set of pages.

8.2.3 Discussion on co-authoring of graphical documents

In this section, we have described ways in which our awareness mechanism could be applied to the co-authoring of graphical documents. We proposed an extension of our framework metamodel to enable the computation and presentation of an overview of changes in collaborative graphical authoring tools. A collaborative application implementing this approach would offer customised awareness information according to user preferences. It could compute and present awareness information at different levels of granularity such as pages, layers, group of objects and objects and it could filter the information according to operation types and users.

9 Conclusion

In this thesis, we have motivated the need for multi-level change awareness emerging from users' feedback from a number of previous user studies. We looked at collaborative applications built to support collaborative authoring in different domains and pointed out their strengths and weaknesses with respect to the amount of awareness information that they provide to users. The findings of this survey were summarised in the Table 2.1 where the features offered by the applications were compared to the list of features which users reported that they need. From this process, we concluded that most of the awareness features that users wish for are not offered by current systems. Finally, the need for a universal approach that would compute and provide the required awareness information independently of the collaborative application domain was highlighted.

The work conducted in the frame of this thesis aims at providing a generic mechanism for the computation and visualisation of multi-level awareness in applications that enable the co-authoring of documents. This mechanism uses the structured model of the co-authored documents to provide awareness at different granularity levels. We described the basic concepts included in our awareness mechanism and the framework resulting from it. To test the framework's applicability, we integrated it into an asynchronous text editor, implemented a prototype and used it to conduct user studies. Based on users' feedback, we extended the awareness mechanism to compute real-time awareness information into asynchronous environments. We also implemented a prototype of a shared workspace that provides flexible awareness information for a set of documents and investigated ways to adjust the computation of awareness information in situations where privacy issues arise.

Further, we have shown how the awareness mechanism could be used to enhance the awareness information provided by collaborative applications that handle different types of documents. For that, we examined the editing activity of graphical documents and of websites and investigated whether the framework could be applied to such collaborative applications. We showed that for the case of graphical editing, the framework can be applied as it is, by only materialising its core concepts based on the information handled by the editor. In the co-authoring of websites, a small extension to the model was needed to include the notion of links between webpages.

To conclude this thesis, we critically evaluate our awareness mechanism and the metamodel that describes it in the next section. Since we have already commented in detail on possible extensions to the work presented in each chapter, we choose to address here only the main issues. Then we provide an outlook on possible future research ideas emerging from the work presented in this thesis.

9.1 Discussion

Combination of the awareness framework with other approaches. Throughout this thesis we discussed extensively on the detailed information our awareness mechanism computes and the way that it satisfies the list of awareness requirements introduced in the beginning of Chapter 3. However, this by no means implies that an application implementing the above framework would provide all the awareness information that might be required. The framework that we propose should be combined with other mechanisms that help users preserve other types of awareness information. Additionally, the framework itself needs to be further extended to provide additional functionalities for change awareness. The way of visualising the computed awareness is one example. While the current version of the framework includes visualisation tools that provide editing overviews, the implemented prototypes do not present the changes superimposed on the document. This is a feature that users in our user study also reported as missing and should be provided.

Application-specific extensions of the metamodel. While the core model of our awareness mechanism stays the same for different applications and document types, the way that each of the concepts in the core model is materialised depends on the document type and the application. For instance, a document *node* can be defined for both text and graphical collaborative applications, but what exactly that node describes differs. In this sense, our framework could be seen as a template that can be used to compute and visualise application-specific awareness information. To integrate our framework into an application, a matching of the framework's concepts to the application's concepts needs to be made. While the matching of the document *nodes* and *operations* might be a straightforward procedure, the definition of the application-relevant metrics and visualisation tools might require more effort.

Reusability. Although the definition of metrics and the design of visualisation tools might be demanding, they offer the advantage that the defined concepts and their implementation can be reused by other applications that offer similar features and have similar needs. We already made use of this framework's advantage, when investigating the co-authoring of websites. The metrics and the visualisation tool used for the *intra-document* modifications were the ones defined for the authoring of text documents. We envision that the use of the framework to enhance the awareness information provided by some editors, could lead to the creation of a rich set of metrics and visualisation tools that could be applied to other applications as well.

User studies. The prototypes we implemented in the framework of this thesis were designed to enable us to experiment with additional metrics and visualisation tools by testing them with users. The study we conducted aimed at collecting a first feedback on the way users interpret the information provided through edit profiles and whether the concepts being implemented are reasonable for users. The feedback that we collected, showed us that the main concepts included in our framework were reasonable and that the information provided by the prototype helped the users maintain multi-level awareness of their collaborators' editing activity. We then tested the generality and applicability of our approach by extending the framework's metamodel for different collaborative editors with regard to the type of the co-authored documents and the working mode of the collaborators. Through the above, this thesis reached its set of goals.

It is clear that for each of the collaborative applications that we addressed in this thesis there is a set of interesting issues for discussion. For instance, user studies could be conducted to investigate the applications in more detail and try to provide a richer set of application specific awareness requirements, metrics and appropriate visualisation tools. Users' feedback could then be compared to feedback collected from studies on other awareness-enabled applications to find which of the features are mostly preferred by the users. However, all the above, are well beyond the scope of a single thesis.

9.2 Outlook

Apply the framework to broadly used applications. The editors used in this thesis were prototypical implementations of collaborative applications to allow for experimentation of our concepts and user testing. The functionalities they provide are by inferior to the ones provided by the authoring applications that people often use when they work. Therefore, the awareness information that can be computed for documents being authored with these applications is restricted. Applying the awareness mechanism to an editor like Open Office [14] or CoWord [121] would be a very interesting next step, since the need of new metrics and visualisation tools may arise due to the rich set of available modifications. When trying to enrich the above applications with awareness information, various issues might arise. Here we discuss some of them. Consider, an application that in addition to the deleteNode and createNode operations offers also the moveNode, colourNode, renameNode, mergeNodes, etc. Visualising the changes of each of the document types separately might overload the visualisation tool. Another interesting issue would be to define the severity of the new types of operations in terms of the metrics that will be used to compute the severity. For instance, how would the severity of a move operation be defined? Maybe by using the level of the moved document part? Or by using the distance between the initial and final positions? How would then such a distance be defined in text documents? Additionally, the notation used in the visualisation tool to show a move operation would also be an interesting issue to investigate.

Awareness for more complex document types. Integrating our awareness mechanism into systems like Open Office might also be challenging due to the complexity of the document types. A simple example could be a document that includes more than one type of document nodes, such as text, graphs, tables, etc. To compute and present awareness information for such a document, we would need to define the structural document model that includes all the types of document nodes and the way they relate to each other and then try to combine the different metrics and visualisation tools that are available for each document type. We discuss here some of the issues that would have to be taken under consideration. For instance, would it be reasonable to define a table or a figure as a child element of a section, or a paragraph in a text document? We next present another issue. Consider that in a co-authored document there is a graph that is a child element of a section. If a graphical object is deleted from the graph, or if the colour of an object in the graph is changed, would it be reasonable to present this information through the visualisation tool at the level of the node's parent elements? For instance would it be reasonable to sum the opValue of the "deleteTextNode" operation with the "deleteGraphicalObject" operation and present the result as the deleteNodeValue of a section? Or would it be reasonable to compute the "changeColourValue" of a section? Issues such as the above would be very interesting for further investigation.

Awareness in single-user applications. Although our awareness framework is described as a mechanism that provides people with knowledge about the status and the actions of their collaborators, another possible application for it would be in single-user applications. People who are involved in long-lasting tasks often need to keep an overview of their own progress and actions over time. For instance, an author writing a book, or a student writing a thesis, would usually author such documents for a long period of time and often go through iterations of correcting and rewriting various document parts. Being aware of all the modifications of the different revisions is a very complicated task to be done manually. In such authoring situations, it might be useful for a user to easily find the document part that was heavily edited at a specific day, for instance after the meeting of the student with their supervisor. Similarly, it would be interesting to find what modifications were made after a new document part was inserted. A system implementing our approach could help users of single-user applications easily collect information about their documents in above situations.

Additionally, it might also be interesting to investigate whether people follow specific working patterns when authoring documents, whether they keep these patterns over many documents and if they depend on user roles, document types and the specific tasks to be accomplished. For that, the editing activity of various people would need to be recorded and the resulting editing profiles could be compared to find similarities in the authoring process followed by each of the users. If such patterns are found, this information could be used to automatically recognise the users and their roles by their working patterns and configure the authoring applications to better serve the users' needs.

Awareness for more complex document structures. Finally, a very interesting and challenging extension of our mechanism would be its use for more complex document structures. We briefly mentioned this idea in Chapter 8 while presenting the computation of awareness for *inter-document* modifications. The idea of links between documents was modelled through the introduction of new associations between the *node* objects. An alternative would be to introduce the concept of a *link* and the associations *hasLinkTo* and *hasLinkFrom* relating *node* objects with *link* objects. As explained in Chapter 8 *intra-links* and *inter-links* would be modelled as specialisations of the link concept and would include all the possible links between *node* objects, including parent-child relationships and html or transclusion links. New types of links could be defined to allow for more flexible structures of documents. For instance, the content of a document could vary depending on the user that works on it. Finally, modelling the links as separate entities would enable the storing of additional information for *link* objects. For instance, it would be possible to search for patterns of following links between documents. Associations between *link* objects could enable the above. An example of such an association could be a *followedAfter* association, relating two links when one is usually followed right after the other. We believe that the extension of our awareness mechanism towards this direction would be very promising.

A
User study questionnaires

Global Information Systems Group, ETH Zurich
Professor Moira Norrie
Stavroula Papadopoulou
Elke Reuss

Asynchronous Collaborative Editor 0.0.1: User Test

A. Introduction to the test.
B. Pre-Questionnaire.
C. Brief introduction to the asynchronous editor.
D. Task part with introduction, tasks and questions.
E. Post-Questionnaire.

B. Pre-Questionnaire

1. How do you assess your computer literacy:

   ☐ inexperienced

   ☐ average

   ☐ well experienced

2. How often have you been involved in a collaborative authoring task during the last five years?
   Answer: I have done it for about ......times

3. What kind of software did you use when involved in such tasks?
   Answer: ....

4. What is your profession?
   Answer: ....

D. Task part with introduction, tasks and questions

Introduction

Three users are already collaboratively authoring a document of 10 paragraphs. They are a professor and two students. Each student is mainly responsible to author specific paragraphs of the document, but can also make changes to the rest of the document. The professor has to ensure that the whole document is well written. Therefore, he can make changes to all the paragraphs.

You are a student joining the group in order to help writing the paper. You log into the asynchronous text editor used and see the last version of the paper. The information displayed in the graph concerns the total number of changes made to the document from the time that it was first created until the currently available version.

Tasks: Please process the following 5 tasks.

1. Study the graph presenting an overview of the changes made to the document and answer the next questions:

   (a) How many users made changes to the document?
   Answer: ....

   (b) Which are the paragraphs of the document that each student is mainly responsible for? Fill the following table accordingly, using the appropriate paragraph numbers. (More than 1 paragraph number can be entered in each table cell)

student's name	paragraph numbers

   (c) Which user is the professor?
   Answer: ....

   (d) How did you reach that conclusion?
   Answer: ....
   ....

   (e) Please indicate in the following scale how much you agree with the statement:

   The information displayed in the graph was useful.

strongly disagree	disagree	neither disagree nor agree	agree	strongly agree
1	2	3	4	5

2. Studying the information displayed in the graph answer the following questions:

   (a) From all the sentences of the document, which is the sentence with the most insertions? Locate it in the text and write down the first and last word of it.
   Answer: ....

   (b) Which is the paragraph with the most insertions? Locate it in the text and write down the first and last word of it.
   Answer: ....

   (c) Which paragraph had the least deletions? Locate it in the text and write down the first and last word of it.
   Answer: ....

   (d) Please indicate in the following scale how much you agree with the statement:

   It was useful that information about changes made to the document was displayed on both sentence and paragraph level.

strongly disagree	disagree	neither disagree nor agree	agree	strongly agree
1	2	3	4	5

3. Zooming

   (a) For the paragraph with the most insertions, zoom in one level so that you visualise the changes made to each one of the paragraph's sentences.

   In terms of amount of changes as well as of the place where the changes were made, how do you interpret the information displayed?
   Answer: ....
   ....
   ....
   ....
   ....

   (b) For the paragraph with the least deletions, zoom in one level so that you visualise the changes made to each one of the paragraph's sentences. At which part of the paragraph were the most changes done? Please mark one:

   ☐ Beginning
   ☐ Middle
   ☐ End

   (c) Did you like the functionality of locating modified parts of the document by zooming in through the graph?

   ☐ Yes
   ☐ No

   (d) Was it easy to locate the part with the most changes within each paragraph?

   ☐ Yes
   ☐ No

   If no, please specify why : ....
   ....
   ....
   ....
   ....
   ....

4. Find the 3rd paragraph in the document. Zoom in and choose its 3rd sentence. Zoom in again to see the changes on the word level.

   (a) Locate and write down the words that had spelling mistakes.
   Answer: ....
   ....
   ....
   ....
   ....

   (b) Was the procedure to locate spelling mistakes easy?

   ☐ Yes

   ☐ No

   If no, please specify what you found difficult: ....
   ....
   ....
   ....
   ....

5. Semantic levels.

   (a) Return on the paragraph level. Set the unit to “Sentences” so that you see in the graph how many complete sentences were inserted and deleted from the paragraphs. Find the paragraph with the most complete sentences inserted, locate it in the text and write down the first and last word of it.
   Answer: . . . .

   (b) Set the unit to “Characters” again and locate the paragraph with the most insertions. Is this paragraph the same with the one returned in step 5i?

   ☐ Yes

   ☐ No

   (c) Does this result give you any additional information about the changes done on the two paragraphs?

   ☐ Yes

   ☐ No

   If yes, please specify what information you get. . . . .
   . . . .
   . . . .
   . . . .
   . . . .

E. Post-Questionnaire

Please answer the following questions 6) to 9). If you don't understand a question, please let us know so that we can clarify it.

6. You have used until now two different ways to evaluate the changes made on the document. The first one takes into consideration the total number of characters deleted or inserted and the second one takes into consideration the total number of semantic units (words or sentences) deleted or inserted.

   (a) Which of the two ways would you choose to be informed about the changes that other users have made on the document?

   ☐ Characters
   ☐ Semantic units
   ☐ Both

   i. If you have answered "Both" in the previous question, please specify in which cases, in terms of kinds of changes that you would like to visualise, you would use each way.
   Answer: ....
   ....

   ii. If you have answered "Characters", please specify why you wouldn't like to be informed about the total number of semantic units (words or sentences) deleted or inserted.
   Answer: ....
   ....

   iii. If you have answered "Semantic Units", please specify why you wouldn't like to be informed about the total number of characters deleted or inserted.
   Answer: ....
   ....

   (b) Would you like any additional information to be displayed to you to inform you about the changes made on the document?

   ☐ Yes
   ☐ No

   If yes, please specify that kind of information: ....
   ....
   ....

7. Considering the visualisation tool (graph) used, please answer the following questions:

   (a) Does this visualisation tool give you sufficient information about where in the document have changes been made?

   ☐ Yes

   ☐ No

   (b) Does this visualisation tool give you sufficient information about the amount of changes made to the document?

   ☐ Yes

   ☐ No

   (c) Do you believe you could get the same information without the visualisation tool?

   ☐ Yes

   ☐ No

   If yes, please specify how else you believe you would get this information: ....
   ....
   ....

   (d) Did you get additional information about the changes made on the document by the fact that deletions were presented in the visualisation tool separately from the insertions?

   ☐ Yes

   ☐ No

8. For the following questions, please indicate using the scales, how much you agree with each of the statements.

   (a) It was useful that changes made to a document were presented through an overview (graph).

strongly disagree	disagree	neither disagree nor agree	agree	strongly agree
1	2	3	4	5

   (b) It was not useful that changes made to a document were presented on different document levels.

strongly disagree	disagree	neither disagree nor agree	agree	strongly agree
1	2	3	4	5

   (c) It was useful to locate changed document parts by single-clicking on the overview.

strongly disagree	disagree	neither disagree nor agree	agree	strongly agree
1	2	3	4	5

   (d) It was useful to zoom into different document levels of a specific document part.

strongly disagree	disagree	neither disagree nor agree	agree	strongly agree
1	2	3	4	5

   (e) It was not useful to present changes based on the total number of characters inserted or deleted.

strongly disagree	disagree	neither disagree nor agree	agree	strongly agree
1	2	3	4	5

   (f) It was useful to present changes based on the total number of semantic units inserted or deleted.

strongly disagree	disagree	neither disagree nor agree	agree	strongly agree
1	2	3	4	5

9. Considering possible extensions of the current asynchronous text editor:

   (a) While working on your local copy, would you like to have the overview synchronously updated according to the changes that other users are making to the document?

   ☐ Yes

   ☐ No

   If no, why? ....
   ....
   ....

   If yes, please answer to the two following questions.

   i. Please indicate in the following scale how much you agree with the statement:
   It would be useful to be synchronously informed about changes made to the document by other users.

strongly disagree	disagree	neither disagree nor agree	agree	strongly agree
1	2	3	4	5

   ii. If you were informed through the overview that another user was working at a specific part of the document would you start working at the same part?

   ☐ Yes

   ☐ No

   For either answer, please specify why: ....
   ....
   ....

   (b) The information presented to you on the overview concerns the total changes made to the document from the moment of its creation until the currently available latest version.

   Would you like to be informed through the overview about the way the document evolved from one version to the other?

   ☐ Yes

   ☐ No

   (c) Please indicate in the following scale how much you agree with the statement:

It would be useful to present, through the overview, all the words that had spelling mistakes throughout the whole document.

strongly disagree	disagree	neither disagree nor agree	agree	strongly agree
1	2	3	4	5

B Publications

The work presented in the frame of this thesis was also presented to the scientific community through the following list of publications.

1. "*Providing Awareness in Multi-synchronous Collaboration Without Compromising Privacy*"

   C. L. Ignat, **S. Papadopoulou**, G. Oster and M. C. Norrie, In Proceedings of CSCW 2008, Conference on Computer Supported Cooperative Work, San Diego, California, USA, November 2008

2. "*Intra/Inter-document Change Awareness for Co-authoring of Web Sites*"

   S. Papadopoulou, C. L. Ignat, G. Oster and M. C. Norrie, In Proceedings of WISE 2008, The Ninth International Conference on Web Information Systems Engineering, Auckland, New Zealand, September 2008

3. "*User Study of Edit Profiles in Collaborative Authoring Systems*"

   S. Papadopoulou, E. Reuss and M. C. Norrie, In Proceedings of CTS 2008, The 2008 International Symposium on Collaborative Technologies and Systems, Irvine, California, USA, May 2008, (Nominated for Best Paper Award)

4. "*How a structured Document Model Can Support Awareness in Collaborative Authoring*"

S. Papadopoulou and M. C. Norrie, In Proceedings of CollaborateCom 2007, The 3rd International IEEE Conference on Collaborative Computing: Networking, Applications and Worksharing, New York, USA, November 2007

5. "*Shadow Document Sets for Synchronously-Aware Asynchronous Collaboration*"

   S. Papadopoulou and M.C. Norrie, In Proceedings of CollaborateCom 2007, The 3rd International IEEE Conference on Collaborative Computing: Networking, Applications and Worksharing, New York, USA, November 2007

6. "*Awareness Model to Overview Modifications in Collaborative Graphical Authoring Tools*"

   S. Papadopoulou, C. L. Ignat and M. C. Norrie, In IWCES 2007, The Ninth International Workshop on Collaborative Editing Systems, GROUP 2007, International ACM SIGGROUP conference on Supporting group work, Sanibel Island, Florida, USA, November 2007

7. "*Document Profiling to Enhance Collaboration*"

   S. Papadopoulou and M. C. Norrie, In IWCES 2006, The Eighth International Workshop on Collaborative Editing Systems, CSCW 2006, Conference on Computer Supported Cooperative Work, Banff, Canada, November 2006

8. "*Increasing Awareness in Collaborative Authoring through Edit Profiling*"

   S. Papadopoulou, C. L. Ignat, G. Oster and M. C. Norrie, In Proceedings of CollaborateCom 2006, the 2nd International IEEE Conference on Collaborative Computing: Networking, Applications and Worksharing, Atlanta, USA, November 2006

Bibliography

[1] Changedetect - be the first to know.
http://www.changedetect.com.

[2] Diff doc - the comprehensive document comparison tool.
http://www.softinterface.com/MD/Document-Comparison-Software.htm.

[3] Altova diffdog - xml-aware differencing and merge tool.
http://www.altova.com/diffdog.

[4] Microsoft word. http://office.microsoft.com/word.

[5] Subversion - an open source version control system.
http://subversion.tigris.org.

[6] Watchthatpage - your monitor for changes on the web.
http://www.watchthatpage.com.

[7] Website-watcher - save time, stay informed. http://aignes.com.

[8] Wikipedia - the free encyclopedia that anyone can edit.
http://www.wikipedia.org.

[9] Winmerge - an open source visual text file differencing and merging tool. http://winmerge.sourceforge.net.

[10] Britannica online encyclopedia. http://www.britannica.com.

[11] Eclipse - an open development platform. http://www.eclipse.org.

[12] English oxford dictionary. http://dictionary.oed.com.

[13] Groove virtual office. http://www.groove.net.

[14] Openoffice - the free and open productivity suite. http://www.openoffice.org.

[15] Rational rose. http://www-306.ibm.com/software/awdtools/developer/rose/index.html.

[16] Microsoft office sharepoint server. http://www.microsoft.com/sharepoint/default.mspx.

[17] Subethaedit. http://www.codingmonkeys.de/subethaedit.

[18] Wikiwikiweb. http://c2.com/cgi/wiki.

[19] R. Al-Ekram, A. Adma, and O. Baysal. DiffX: an algorithm to detect changes in multi-version XML documents. In *CASCON '05: Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research*, pages 1–11. IBM Press, 2005.

[20] W. Appelt. WWW based collaboration with the BSCW system. In *SOFSEM '99: Proceedings of the 26th Conference on Current Trends in Theory and Practice of Informatics on Theory and Practice of Informatics*, pages 66–78, London, UK, 1999. Springer-Verlag.

[21] R. M. Baecker. *Readings in Groupware and Computer-supported Cooperative Work: Facilitating Human-Human Collaboration.* Mor-gan Kaufmann, 1993.

[22] R. M. Baecker, D. Nastos, I. R. Posner, and K. L. Mawby. The user-centered iterative design of collaborative writing software. In *CHI'93: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 399–405, New York, NY, USA, 1993. ACM Press.

[23] R. M. Baecker, J. Grudin, W. Buxton, and S. Greenberg. *Readings in Human Computer Interaction: Toward the Year 2000.* Morgan Kaufmann Publishers, 1995.

[24] J. Bair. The need for collaboration tools in offices. In *Proceedings of the 1985 Office Automation Conference-AFIPS*, February 1985.

[25] R. Bentley, T. Horstmann, K. Sikkel, and J. Trevor. Supporting collaborative information sharing with the WWW: The BSCW shared

workspace system. In *Proceedings of the 4th International WWW Conference*, pages 63–74, Boston, MA, USA, December 1995.

[26] R. Bentley, T. C. Horstmann, and J. Trevor. The World Wide Web as enabling technology for CSCW: The case of BSCW. *Computer Supported Cooperative Work*, 6(2/3):111–134, 1997.

[27] B. Berliner. CVS II: parallelizing software development. In *Proceedings of the USENIX Winter 1990 Technical Conference*, pages 341–352, Berkeley, CA, USA, 1990. USENIX Association.

[28] N. Bevan, C. Barnum, G. Cockton, J. Nielsen, J. Spool, and D. Wixon. The "magic number 5": is it enough for web testing? In *CHI'03: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '03 extended abstracts on Human Factors in Computing Systems*, pages 698–699, Fort Lauderdale, FL, USA, 2003. ACM.

[29] P. H. Carstensen and K. Schmidt. Computer supported cooperative work: New challenges to systems design. In *Handbook of Human Factors*, pages 619–636, 1999.

[30] S. Chan, M. Wong, and V. Ng. Collaborative solid modeling on the WWW. In *SAC '99: Proceedings of the 1999 ACM symposium on Applied computing*, pages 598–602, New York, NY, USA, 1999. ACM.

[31] L.-T. Cheng, S. Hupfer, S. Ross, and J. Patterson. Jazzing up eclipse with collaborative tools. In *eclipse '03: Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange*, pages 45–49, New York, NY, USA, 2003. ACM.

[32] E. J. Chikofsky and J. H. C. II. Reverse engineering and design recovery: A taxonomy. *IEEE Software*, 7(1):13–17, 1990.

[33] H. H. Clark and S. E. Brennan. *Grounding in Communication.* APA Books, Washington, 1991.

[34] P. Dewan and R. Hegde. Semi-synchronous conflict detection and resolution in asynchronous software development. In *ECSCW'07: Proceedings of the tenth European Conference on Computer Supported Cooperative Work*, Limerick, Ireland, September 2007.

[35] P. Dourish and V. Bellotti. Awareness and coordination in shared workspaces. In *CSCW'92: Proceedings of the 1992 ACM Conference on Computer Supported Cooperative Work*, pages 107–114, New York, NY, USA, 1992. ACM Press.

[36] P. Dourish and S. Bly. Portholes: supporting awareness in a distributed work group. In *CHI'92: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 541–547, New York, NY, USA, 1992. ACM.

[37] S. Eick, T. Graves, A. Karr, A. Mockus, and P. Schuster. Visualizing software changes. *IEEE Transactions on Software Engineering*, 28(4):396–412, April 2002.

[38] S. G. Eick, J. L. Steffen, and J. Eric E. Sumner. Seesoft-a tool for visualizing line oriented software statistics. *IEEE Transactions on Software Engineering*, 18(11):957–968, 1992.

[39] M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences of the USA*, 95(25): 14863V–14868, 1998.

[40] C. Ellis, S. Gibbs, and G. Rein. *Design and Use of a Group Editor*. Engineering for Human-Computer Interaction. North-Holland, 1990.

[41] C. A. Ellis, S. J. Gibbs, and G. Rein. Groupware: some issues and experiences. *Communications of the ACM*, 34(1):39–58, 1991.

[42] M. R. Endsley. Measurement of situation awareness in dynamic systems. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 37(1):65–84, March 1995.

[43] R. S. Fish, R. E. Kraut, and M. D. P. Leland. Quilt: a collaborative tool for cooperative writing. In *Proceedings of the ACM SIGOIS and IEEECS TC-OA 1988 conference on Office information systems*, pages 30–37, New York, NY, USA, 1988. ACM.

[44] M. Fracker. Attention allocation in situation awareness. In *Proceedings of the Human Factors Society 33rd Annual Meeting (Visual Performance: Spatial Awareness)*, pages 1396–1400, 1989.

[45] J. Froehlich and P. Dourish. Unifying artifacts and activities in a visual tool for distributed software development teams. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 387–396, Washington, DC, USA, 2004. IEEE Computer Society.

[46] L. Fuchs, U. Pankoke-Babatz, and W. Prinz. Supporting cooperative awareness with local event mechanisms: the groupdesk system. In *ECSCW'95: Proceedings of the fourth European Conference on Computer Supported Cooperative Work*, pages 247–262, Norwell, MA, USA, 1995. Kluwer Academic Publishers.

[47] R. Furuta, V. Quint, and J. André. Interactively editing structured documents. *Electronic Publishing Origination, Dissemination, and Design*, 1(1):19–44, 1988.

[48] D. German, A. Hindle, and N. Jordan. Visualizing the evolution of software using softChange. In *SEKE'04: Proceedings of International Conference on Software Engineering & Knowledge Engineering*, pages 336–341, New York NY, 2004. ACM Press.

[49] D. M. German. An empirical study of fine-grained software modifications. In *ICSM '04: Proceedings of the 20th IEEE International Conference on Software Maintenance*, pages 316–325, Washington, DC, USA, 2004. IEEE Computer Society.

[50] R. D. Gilson. Introduction to the special issue on situation awareness. *Human Factors*, 37(1), 1995.

[51] S. Greenberg. Sharing views and interactions with single-user applications. In *Proceedings of the ACM SIGOIS and IEEE CS TC-OA conference on Office information systems*, pages 227–237, New York, NY, USA, 1990. ACM.

[52] S. Greenberg and M. Boyle. Generating custom notification histories by tracking visual differences between web page visits. In *GI '06: Proceedings of the Conference on Graphics Interface 2006*, pages 227–234, Quebec, Canada, 2006. Canadian Information Processing Society.

[53] S. Greenberg, C. Gutwin, and A. Cockburn. Using distortion-oriented displays to support workspace awareness. In A. Sasse,

R. J. Cunningham, and R. Winder, editors, *Proceedings of HCI People and Computers XI*, pages 299–314. Springer-Verlag, 20–23 1996.

[54] S. Greenberg, C. Gutwin, and A. Cockburn. Awareness through fisheye views in relaxed-wysiwis groupware. In *GI '96: Proceedings of the Conference on Graphics Interface '96*, pages 28–38, Toronto, Ontario, Canada, 1996. Canadian Information Processing Society.

[55] J. Grudin. Computer-supported cooperative work: Its history and participation. *IEEE Computer*, 27(5):19–26, 1994.

[56] C. Gutwin and S. Greenberg. A descriptive framework of workspace awareness for real-time groupware. *Computer Supported Cooperative Work*, 11(3):411–446, 2002.

[57] C. Gutwin, S. Greenberg, and M. Roseman. Workspace awareness support with radar views. In *CHI'96: Companion Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 210–211, New York, NY, USA, 1996. ACM Press.

[58] C. Gutwin, M. Roseman, and S. Greenberg. A usability study of awareness widgets in a shared workspace groupware system. In *CSCW'96: Proceedings of the 1996 ACM Conference on Computer Supported Cooperative Work*, pages 258–267, New York, NY, USA, 1996. ACM Press.

[59] C. A. Gutwin. *Workspace awareness in real-time distributed groupware.* PhD thesis, Calgary, Canada, 1998.

[60] J. M. Haake and B. Wilson. Supporting collaborative writing of hyperdocuments in SEPIA. In *CSCW'92: Proceedings of the 1992 ACM Conference on Computer Supported Cooperative Work*, pages 138–146, New York, NY, USA, 1992. ACM Press.

[61] S. Hayne, M. Pendergast, and S. Greenberg. Gesturing through cursors: Implementing multiple pointers in group support systems. In *HICSS'93: Proceeding of the Hawaii International Conference on System Sciences*, pages 4–12, Maui, Hawaii, January 1993. IEEE Press.

[62] C. Heath and P. Luff. Collaborative activity and technological design: task coordination in london underground control rooms.

In *ECSCW'91: Proceedings of the second European Conference on Computer-Supported Cooperative Work*, pages 65–80, Norwell, MA, USA, 1991. Kluwer Academic Publishers.

[63] C. Heath, P. Luff, and G. Cambridge. Collaboration and control: Crisis management and multimedia technology in london underground line control rooms. *Computer Supported Cooperative Work*, 1:69–94, 1992.

[64] W. C. Hill, J. D. Hollan, D. Wroblewski, and T. McCandless. Edit wear and read wear. In *CHI'92: Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 3–9, Monterey, California, USA, May 1992. ACM Press.

[65] T. B. Hodel-Widmer and K. R. Dittrich. Concept and prototype of a collaborative business process environment for document processing. *Data & Knowledge Engineering*, 52:61–120, 2005.

[66] S. Hupfer, L.-T. Cheng, S. Ross, and J. Patterson. Introducing collaboration into an application development environment. In *CSCW'04: Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, pages 21–24, New York, NY, USA, 2004. ACM.

[67] C. L. Ignat and M. C. Norrie. Customizable collaborative editor relying on treeOPT algorithm. In *ECSCW'03: Proceedings of the eighth European Conference on Computer Supported Cooperative Work*, pages 315–334, Helsinki, Finland, September 2003. Kluwer Academic Publishers.

[68] C.-L. Ignat and M. C. Norrie. Draw-together: graphical editor for collaborative drawing. In *CSCW'06: Proceedings of the 2006 20th anniversary Conference on Computer Supported Cooperative Work*, pages 269–278, New York, NY, USA, 2006. ACM Press.

[69] C.-L. Ignat, G. Oster, P. Molli, and H. Skaf-Molli. Gasper: A collaborative writing mode for avoiding blind modifications. Research Report RR-6204, LORIA – INRIA Lorraine, May 2007.

[70] R. Johansen. *Groupware: Computer Support for Business Teams.* The Free Press, 1988.

[71] R. H. Katz. Toward a unified framework for version modeling in engineering databases. *ACM Computing Surveys (CSUR)*, 22(4): 375–409, 1990.

[72] H.-C. Kim and K. S. Eklundh. How academics co-ordinate their documentation work and communicate about reviewing in collaborative writing. Technical Report TRITA-NA-P9815, IPLab-151,KTH-Sweden., 1998. Interaction and Presentation Laboratory, Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm University.

[73] J. Kolbitsch and H. Maurer. Transclusions in an html-based environment. *Journal of Computing and Information Technology (CIT)*, 14:161–174, 2006.

[74] H. Krottmaier and D. Helic. Issues of transclusions. In *E-Learn'02: Proceedings of World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education*, pages 1730–1733, Montreal, Canada, 2002. AACE.

[75] D. Kurlander. Graphical editing by example. In *CHI'93: Proceedings of the INTERACT '93 and CHI '93 ACM SIGCHI Conference on Human Factors in Computing Systems*, page 529, New York, NY, USA, 1993. ACM Press.

[76] D. Kurlander and S. Feiner. Editable graphical histories. *IEEE Workshop on Visual Languages*, pages 127–134, October 1988.

[77] M. Lanza. The evolution matrix: recovering software evolution using software visualization techniques. In *IWPSE '01: Proceedings of the 4th International Workshop on Principles of Software Evolution*, pages 37–42, New York, NY, USA, 2001. ACM.

[78] M. Lanza and S. Ducasse. Polymetric views - a lightweight visual approach to reverse engineering. *IEEE Transactions on Software Engineering*, 29:782– 795, 2003.

[79] M. D. P. Leland, R. S. Fish, and R. E. Kraut. Collaborative document production using Quilt. In *CSCW'88: Proceedings of the 1988 ACM Conference on Computer Supported Cooperative Work*, pages 206–215, New York, NY, USA, 1988.

[80] Y. Li, J.-J. Bu, C. Chen, and X.-H. Xu. Research on awareness model on real-time collaborative graphics editing system. In *ICMLC'03: Proceedings of International Conference on Machine Learning and Cybernetics*, pages 2944– 2949, 2003.

[81] E. Lippe and N. van Oosterom. Operation-based merging. In *CDE-5: Proceedings of the fifth ACM SIGSOFT Symposium on Software Development Environments*, pages 78–87, Tyson's Corner, Virginia, USA, December 1992. ACM Press.

[82] N. Mangano, A. Baker, and A. van der Hoek. Calico: A prototype sketching tool for modeling in early design. In *Second International Workshop on Modeling in Software Engineering*, May 2008.

[83] L. McGuffin and G. Olson. ShrEdit: A shared electronic workspace. Csmil, Cognitive Science and Machine Intelligence Laboratory, University of Michigan, 1992.

[84] S. Minör and B. Magnusson. A model for semi-(a)synchronous collaborative editing. In *ECSCW'93: Proceedings of the third European Conference on Computer Supported Cooperative Work*, pages 219–231, Milan, Italy, September 2003.

[85] P. Molli, H. Skaf-molli, and C. Bouthier. State treemap: an awareness widget for multi-synchronous groupware. In *CRIWG'01: Seventh International Workshop on Groupware*, 2001.

[86] P. Molli, H. Skaf-Molli, and G. Oster. Divergence awareness for virtual team through the web. In *IDPT 2002: Proceedings of world conference on the Integrated Design and Process Technology*, Pasadena, California, USA, June 2002. Society for Design and Process Science.

[87] P. Molli, H. Skaf-Molli, G. Oster, and S. Jourdain. SAMS: Synchronous, asynchronous, multi–synchronous environments. In *CSCWD'02: Proceedings of the Conference on Computer Supported Cooperative Work in Design*, pages 80–85, Rio de Janeiro, Brazil, September 2002. Society for Design and Process Science.

[88] E. W. Myers. An $O(ND)$ difference algorithm and its variations. *Algorithmica*, 1:251–266, 1986.

[89] T. Nelson. *Literary Machines.* Mindful Press, 1982.

[90] C. M. Neuwirth, D. S. Kaufer, R. Chandhok, and J. H. Morris. Issues in the design of computer support for co-authoring and commenting. In *CSCW'90: Proceedings of the 1990 ACM Conference on Computer Supported Cooperative Work*, pages 183–195, New York, NY, USA, 1990. ACM.

[91] C. M. Neuwirth, R. Chandhok, D. S. Kaufer, P. Erion, J. Morris, and D. Miller. Flexible Diff-ing in a collaborative writing system. In *CSCW'92: Proceedings of the 1992 ACM Conference on Computer Supported Cooperative Work*, pages 147–154, New York, NY, USA, 1992. ACM.

[92] J. Nielsen. Why you only need to test with 5 users. Alertbox, June 2000.

[93] J. Nielsen. Quantitative studies: How many users to test? Alertbox, June 2006.

[94] S. Noël and J.-M. Robert. Empirical study on collaborative writing: What do co-authors do, use, and like? *Computer Supported Cooperative Work - CSCW*, 13(1):63–89, 2004.

[95] U. Pankoke-Babatz, W. Prinz, and L. Schäfer. Stories about asynchronous awareness. In *Cooperative Systems Design-COOP 2004*, pages 23–38, 2004.

[96] I. R. Posner and R. M. Baecker. How people write together. volume 4, pages 127–138, 1992. ISBN 0-8186-2420-5.

[97] W. Prinz, E. Hinrichs, and I. Kireyev. Anticipative awareness in a groupware system. In *COOP'08: Proceedings of the 8th International Conference on the Design of Cooperative Systems*, 2008.

[98] V. Quint and I. Vatton. Making structured documents active. *Electronic Publishing Origination, Dissemination, and Design*, 7(2): 55–74, 1994.

[99] G. K. Raikundalia and H. L. Zhang. Newly-discovered group awareness mechanisms for supporting real-time collaborative authoring. In *AUIC '05: Proceedings of the Sixth Australasian conference on User interface*, pages 127–136, Darlinghurst, Australia, 2005. Australian Computer Society, Inc.

[100] G. K. Raikundalia and H. L. Zhang. Document-related awareness elements in synchronous collaborative authoring. *Australian Journal of Intelligent Information Processing Systems*, 9(2):41–48, December 2006. Special issue based on the Eleventh Australian Document Computing Symposium, Queensland University of Technology.

[101] R. Robbes and M. Lanza. Versioning systems for evolution research. In *IWPSE '05: Proceedings of the Eighth International Workshop on Principles of Software Evolution*, pages 155–164, Washington, DC, USA, 2005. IEEE Computer Society.

[102] R. Robbes and M. Lanza. A change-based approach to software evolution. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 166:93–109, 2007.

[103] M. Roseman and S. Greenberg. Building real-time groupware with GroupKit, a groupware toolkit. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 3(1):66–106, 1996.

[104] H. Sacks, E. Schegloff, and G. Jefferson. A simplest systematics for the organization of turn-taking for conversation. *Language*, 50: 696–735, 1974.

[105] A. Sarma, Z. Noroozi, and A. V. D. Hoek. Palantir: raising awareness among configuration management workspaces. In *Software Engineering, 2003. Proceedings of the 25th International Conference on Software Engineering*, pages 444–454, 2003.

[106] N. Sarter and D. Woods. How in the world did we ever get into that mode? Mode error and awareness in supervisory control. *Human Factors*, 37(1):5–19, 1995.

[107] T. Schümmer. Lost and found in software space. In *HICSS '01: Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, volume 9, page 9066, Washington, DC, USA, 2001. IEEE Computer Society.

[108] T. Schümmer and J. M. Haake. Supporting distributed software development by modes of collaboration. In *ECSCW'01: Proceedings of the seventh European Conference on Computer Supported Cooperative Work*, pages 79–98, Norwell, MA, USA, 2001. Kluwer Academic Publishers.

[109] M. Sharples. Adding a little structure to collaborative writing. *CSCW in Practice: An Introduction and Case Studies*, pages 51–67, 1993.

[110] C. Sun and C. Ellis. Operational transformation in real-time group editors: issues, algorithms, and achievements. In *CSCW'98: Proceedings of the 1998 ACM conference on Computer supported cooperative work*, pages 59–68, New York, NY, USA, 1998. ACM. ISBN 1-58113-009-0.

[111] C. Sun, X. Jia, Y. Yang, and Y. Zhang. REDUCE: A prototypical cooperative editing system. In *HCI International '97: Proceedings of the Seventh International Conference on Human-Computer Interaction-Volume 1*, pages 89–92, New York, NY, USA, 1997. Elsevier Science Inc.

[112] C. Sun, S. Xia, D. Sun, D. Chen, H. Shen, and W. Cai. Transparent adaptation of single-user applications for multi-user real-time collaboration. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 13(4):531–582, 2006.

[113] D. Sun and C. Sun. Operation context and context-based operational transformation. In *CSCW '06: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 279–288, New York, NY, USA, 2006. ACM. ISBN 1-59593-249-6. doi: http://doi.acm.org/10.1145/1180875.1180918.

[114] J. Tam and S. Greenberg. A framework for asynchronous change awareness in collaborative documents and workspaces. *International Journal of Human-Computer Studies*, 64(7):583–598, 2006.

[115] J. Tam, L. McCaffrey, F. Maurer, and S. Greenberg. Change awareness in software engineering using two dimensional graphical design and development tools. Research Report 2000-670-22, Department of Computer Science, University of Calgary, Alberta, Canada, October 2000.

[116] J. R. Tam. Supporting change awareness in visual workspaces. Master's thesis, Department of Computer Science, University of Calgary, Alberta, 2002.

[117] F. B. Viégas, M. Wattenberg, and K. Dave. Studying cooperation and conflict between authors with history flow visualizations. In

CHI'04: Companion Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems, pages 575–582, Vienna, Austria, April 2004. ACM Press.

[118] P. Wilson. *Computer Supported Cooperative Work: An Introduction.* Kluwer Academic Publishers, 1991.

[119] J. Wu, R. C. Holt, and A. E. Hassan. Exploring software evolution using spectrographs. In *WCRE '04: Proceedings of the 11th Working Conference on Reverse Engineering*, pages 80–89, Washington, DC, USA, 2004. IEEE Computer Society.

[120] X. Wu, A. Murray, M.-A. Storey, and R. Lintern. A reverse engineering approach to support software maintenance: Version control knowledge extraction. In *WCRE '04: Proceedings of the 11th Working Conference on Reverse Engineering*, pages 90–99, Washington, DC, USA, 2004. IEEE Computer Society.

[121] S. Xia, D. Sun, C. Sun, D. Chen, and H. Shen. Leveraging single-user applications for multi-user collaboration: The CoWord approach. In *CSCW'04: Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 162–171, Chicago, Illinois, USA, November 2004. ACM Press.

Curriculum Vitae

Particulars

Name	Stavroula Papadopoulou
Date of Birth	November 22, 1980
Citizenship	Greek

Education

2004-2009	**ETH Zurich**, Switzerland Global Information Systems Group *Doctor of Sciences ETH Zurich*
1999–2004	**Aristotle University of Thessaloniki**, Greece *Diploma in Electrical and Computer Engineering (top 8% of class)*
1995–1998	**5th Lyceum of Kavala**, Greece *National High School Degree (ranked 1st in class)*

Work Experience

2004–2009	Research and Teaching Assistant, supervised by Prof. Dr. Moira C. Norrie, Global Information Systems Group, ETH Zurich
2008	Invited Researcher, LORIA-INRIA Nancy-Grand Est, Nancy, France