

Formalisation des fonctionnalités nécessaires pour un système d'édition collaborative

Projet de semestre de **Laurent Haan**

École Polytechnique Fédérale de Lausanne
IN Ecublens, CH-1015 Lausanne
Switzerland

laurent.haan@epfl.ch

Description:

L'internet et les nouvelles possibilités de communication ont permis à un nouveau domaine de s'établir: l'édition collaborative. Dans ce travail, j'essaie de présenter les solutions actuelles, énumérer leurs avantages et désavantages et analyser les fonctionnalités nécessaires qu'un tel système doit fournir pour faciliter l'édition d'un document au sein d'une communauté.

Je vais expliquer comment une approche avec des documents structurés peut offrir une base solide sur laquelle on peut définir le processus d'édition, les rôles et permissions de différents éditeurs ainsi que les contraintes imposables à un tel document et les définitions d'un workflow pour structurer les étapes d'édition

1. Introduction

Dans les dernières années, l'internet a changé non seulement la manière d'accéder à des informations mais aussi notre manière de travailler. Fini l'époque, où un écrivain scribouillait son histoire sur un bout de papier pour ensuite le faire recopier par une centaine de moines dans une abbaye. Aujourd'hui, on écrit des e-mails, on publie une histoire dans un blog et de plus en plus rare deviennent les personnes qui utilisent encore des applications d'édition standard comme Office.

Un nouveau domaine de recherche est l'édition collaborative: un système qui nous donne la possibilité d'éditer un document ensemble dans un groupe, et pour cela nous propose de nombreuses fonctionnalités informatiques.

***Collaborative editing** is the practice of groups producing works together through individual contributions. Most usually it is applied to textual documents or programmatic [source code](#). Such [asynchronous](#) contributions are very efficient in time, as group members need not assemble in order to work together. Generally, managing such work requires [software](#); the most common tools for editing documents are [Wikis](#), and those for programming, [version control systems](#). (Wikipedia, Collaborative Editing)*

Comme nous allons le voir, il existe plusieurs méthodes et fonctionnalités dans un système d'édition collaborative et le but de ce travail est de rechercher les solutions actuelles, en dégager les avantages et désavantages, ensuite montrer la direction dans laquelle s'engagent les scientifiques et finalement proposer une propre solution pour l'édition collaborative. Je vais expliquer pourquoi une approche avec des documents structurés (exemple XML) nous offre des possibilités intéressantes sur lesquelles on peut facilement intégrer un tel système et finalement décrire les fonctionnalités nécessaires mais aussi les fonctionnalités additionnelles qu'on pourrait imaginer dans un système plus élaboré.

2. Types d'édition collaborative

L'idée derrière l'édition collaborative, même si l'implémentation diffère d'une application à une autre, est toujours la même: un document commun réside « quelque part » sur un serveur qui en gère l'utilisation. Les utilisateurs ont un moyen (par exemple une application ou une page Web) de voir une projection de ce document sur leur écran. Après modification de sa projection (car une projection peut varier d'un utilisateur à un autre), le serveur met à jour le document commun qui se trouve sur le serveur.

Cependant, la recherche dans « Computer Supported Collaborative Work » (CSCW) a montré que cette infrastructure avec un document centralisé et de nombreuses projections permet seulement un nombre restreint de processus d'édition. Ceci a mené dans le passé de distinguer entre le « niveau de couplage » [Dewan 91] dans un système d'édition collaborative. C'est justement ce niveau de couplage qui varie dans les implémentations différentes et on peut subdiviser les applications en deux catégories:

2.1.1 Niveau de couplage fort

On parle d'un niveau de couplage fort lorsque les utilisateurs voient les modifications du document commun « en temps réel ». Chaque modification du document est immédiatement envoyé au serveur, qui valide la modification et en avertit les autres utilisateurs. Dans la plupart des cas, ces applications sont des éditeurs complets, plus ou moins performants, qui utilisent des protocoles propriétaires pour la communication entre l'application et le serveur.

Parmi les applications avec un niveau de couplage fort, on peut encore les diviser selon leur « sharing entity ». Cette entité est l'unité la plus petite qui est envoyé par le protocole afin de mettre à jour le document commun. Cette unité peut être de taille fixe ou variable. Selon la taille de cette unité, les effets sur la projection du document peuvent varier: dans SubEthaEdit [SubEthaEdit] et Ace [ACE], l'unité qui est transmis par le protocole est un caractère ce qui permet une modification quasi temps-réel des projections du document. Dans Duplex [Pacull 94], l'unité est associé avec des unités de latex marquant une nouvelle section. Dans Griffon [Decouchant 93], l'unité est

variable et correspond à une unité d'un document structuré. Cette unité n'est plus traitée comme texte mais comme un élément.

En regardant les rapports de ACE [Rapport-ACE], qui était un projet à l'université de Berne, on voit qu'il existe plusieurs algorithmes capable de gérer un niveau de couplage fort avec des unités très petites (un caractère ou une action d'une utilisateur). Dans le rapport, librement accessible, ils expliquent les différences et problèmes qu'ils ont rencontré lors de l'implémentation des treize algorithmes (dOPT, Jupiter, adOPTed, GOT, GOTO ... pour en citer que quelques).

En remarquant que les algorithmes sont capables de travailler avec des unités très petites, on peut facilement étendre cette idée pour aussi envoyer les actions des utilisateurs, la position de leur souris, les boutons qu'ils ont cliqué. Un tel système est appelé « What You See Is What I See » (WYSIWIS) car à cause des mises à jour très fréquentes de la projection du document, chaque utilisateur voit à tout moment la même projection. Comme on va le voir dans la section 3, cela peut mener à des problèmes pour les utilisateurs de ces systèmes.

2.1.2 Exemple d'applications avec couplage fort

ACE, a collaborative editor

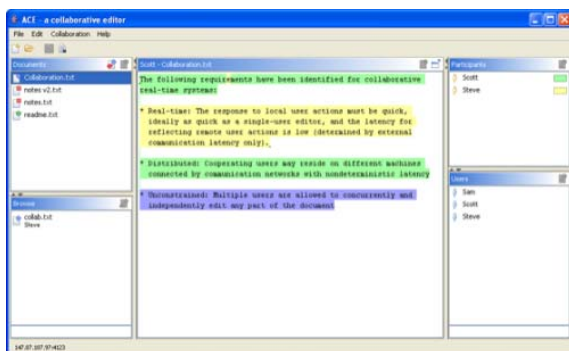


Illustration 1: Screenshot de la GUI de ACE

ACE est l'exemple parfait d'un éditeur WYSIWIS. La coloration d'une partie d'un texte, lorsqu'un utilisateur l'a sélectionné) fait partie des aides visuelles que ACE implémente pour faciliter l'édition en temps réel.

La GUI affiche les documents communs sur le serveur, la projection du document actuel, les utilisateurs sur le serveur et les participants à

l'édition du document projeté ainsi que la partie qu'ils modifient en couleur.

SubEthaEdit, collaborative text editing

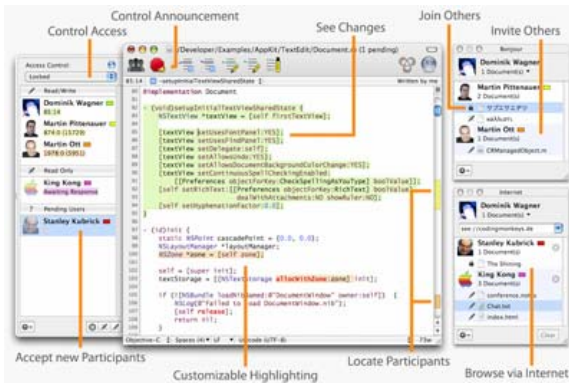


Illustration 2: Screenshot de la GUI de SubEthaEdit

utilisateurs ainsi que le texte qu'ils modifient dans la même couleur. On peut définir le contrôle d'accès à une partie de texte (par exemple mode « verrouille » qui empêche un autre utilisateur d'écrire dans la partie du texte sélectionné par un autre utilisateur). La scrollbar affiche une zone colorée pour montrer dans quelle partie de la projection se trouve un utilisateur. Finalement, on a une fenêtre qui affiche les utilisateurs dans le système qu'on peut rejoindre ou inviter pour travailler ensemble.

Writely, The Web Word Processor

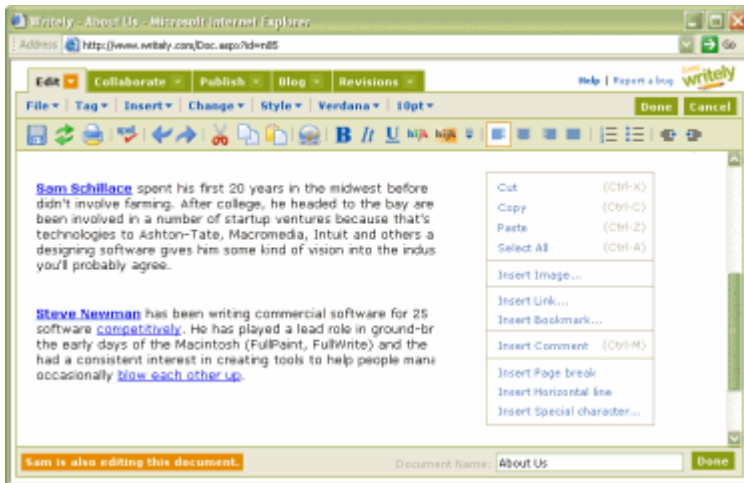


Illustration 3: Screenshot de la GUI (dans Internet Explorer) de Writely

au lieu de réagir à des événements, writely met à jour le document commun après un intervalle de temps pré-défini. Un système de collaborateur permet aussi de rajouter des personnes à l'édition d'un document. Un conflit dans l'édition d'une même partie de

SubEthaEdit est un éditeur en temps réel pour MAC qui utilise Bonjour [Bonjour] pour le partage de documents. Il possède un système de gestion d'utilisateur: on peut inviter ou exclure des utilisateurs d'un document.

La GUI affiche les utilisateurs en train de travailler sur le document, les utilisateurs en attente de permission, les couleurs des

Writely [Writely] est un éditeur en « temps-réel » en ligne qui a été acheté par Google le 9. Mai 2006 pour être intégré dans les applications Web de Google. Alors que les autres applications réagissaient sur des actions des utilisateurs pour mettre à jour le document commun, writely est soumis aux contraintes imposées par les pages Web ainsi que les serveurs

texte par plusieurs personnes est signalé à une des personnes. Un système qui sauvegarde automatiquement les anciennes versions des documents permet en plus de récupérer un état antérieur d'un document qu'un autre utilisateur a peut-être détruit avec des changements inadéquats.

2.2 Niveau de couplage lâche

Lorsque les utilisateurs ont aucune information sur les actions des autres utilisateurs, on parle d'un niveau de couplage lâche. Les utilisateurs travaillent tous dans une projection différente du document commun qui sont complètement indépendantes.

Un exemple classique d'une telle application est le BSCW [Horstmann 97] qui peut contenir des fichiers, des informations sur les utilisateurs et des messages électroniques. Le seul moyen d'interaction est de mettre une nouvelle version du document dans l'espace partagé entre les utilisateurs. Un simple système qui gère les versions des documents peut éviter des éventuels conflits. Un tel système peut être CVS [CVS], qui est capable non seulement de gérer les versions des fichiers mais aussi de mettre deux fichiers ensemble, d'extraire les différences entre deux fichiers etc.

Les versions les plus basiques d'espaces partagés utilisent les protocoles répandus comme FTP [Postel 85], HTTP [Fielding 99] ou webDAV [Goland 99]. Malgré la simplicité et les fonctionnalités très restreintes, ce système est utilisé très souvent sur internet car il permet aux utilisateurs d'utiliser leur propre éditeur et applications et l'utilisation de l'espace partagé intervient seulement au moment de la mise à jour du document sur le serveur.

2.3 Résultat

Actuellement, le deuxième type d'éditeur, ceux qui utilisent le niveau de couplage lâche est plus répandu. Cela est dû au coût qu'un changement de système impose aux utilisateurs: il doit évaluer les avantages et les désavantages du nouveau système.

Un nouveau système qui force un utilisateur de changer d'environnement de travail peut poser un problème à beaucoup d'utilisateur qui n'ont pas envi de

réapprendre la manipulation d'une nouvelle application. De plus, la mise en service d'un système avec niveau de couplage fort peut influencer la décision vers des solutions plus simples. L'utilisateur est demandé de choisir un serveur, configurer les droits d'accès, trouver et rajouter des utilisateurs (ou créer des comptes pour chaque utilisateur).

Dans un espace partagé, donc avec niveau de couplage lâche, l'utilisateur peut utiliser son éditeur personnalisé, peut garder son environnement de travail habituel et la mise en service d'une telle infrastructure est très simple (dans la plupart des cas, une adresse fixe suffit). Cependant, les désavantages envers les solutions avec couplage fort sont nombreux: l'utilisateur a qu'une vision des actions des autres personnes très limité et une simple gestion de version est peu suffisante à une édition collaborative massive entre plusieurs personnes. Une simple détection de conflit, résultant dans un « diff » en CVS demande une intervention manuelle d'une personne, résultant dans une subdivision classique des tâches, par exemple lors d'une édition d'un texte par mail (ou une personne doit gérer de mettre ensemble les parties écrites par les autres personnes).

3. Édition collaborative utilisant un groupware

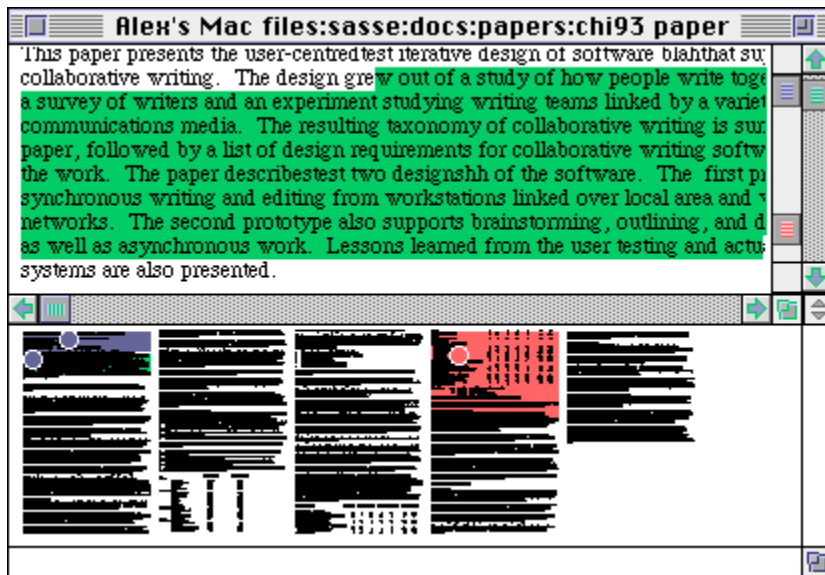


Illustration 4: Sasse [SASSE] normal text display for the green user and a gestalt view of the entire document

Dans ce chapitre, je vais présenter une étude réalisée par l'université de Toronto qui s'appelle « Learning to Write Together Using Groupware » [Mitchell 95]. Dans cette étude, deux groupes d'étudiants jeunes, sans expérience dans l'édition collaborative sont demandés de travailler sur un magazine pendant douze semaines. Ils utilisent une application qui s'appelle Aspects

[Aspects] et qui est un éditeur en temps-réel similaire à Sasse (Illustration2).

3.1 Écrire ensemble

Le problème initial était la compréhension du sens de « écrire ensemble ». Une première observation était que les deux groupes ont commencé immédiatement à écrire en parallèle, chacun écrivant indépendamment de l'autre, sans communication. La possibilité d'entrer du texte en même temps que les autres et de voir le texte entré par l'autre groupe les a irrité et il a fallu du temps pour s'y habituer.

Après avoir remarqué que le fait d'entrer du texte en même temps sans consultation peut mener à des conflits ou des duplications, ils se sont arrangés entre eux pour avoir une personne qui écrit tout et les autres qui font les suggestions. Cette technique traditionnelle ne demandait aucune adaptation des étudiants mais ne présente aucun avantage envers les environnements classiques d'édition.

Après quelques semaines, ils se sont habitués assez aux possibilités du nouveau environnement mais ont gardé le même type d'édition. Une seule personne écrivait alors que les autres suivaient sur leur propre écran et parfois mentionnaient des erreurs en utilisant leur souris. Le fait que les modifications n'ont pas été effectuées mais

seulement relevé à l'écrivain a posé beaucoup de problèmes: malgré le fait que plusieurs personnes peuvent écrire un même texte, ils ont trouvé que le résultat manquait de cohérence et qu'il fallait le récrire en entier par une même personne.

3.2 Utiliser la technologie

Même dans des éditeurs simples comme par exemple Word, on remarque que les utilisateurs se laissent facilement distraire par le formatage, des icônes clignotantes, des messages d'erreur ou des avertissements par l'application. Le risque qu'un utilisateur se laisse distraire par une modification de la projection est encore plus grande. Un nouveau utilisateur qui se connecte ou des variations de la coloration du texte peut mener à des pauses pour s'informer sur l'état du système. Dans le cas du groupe d'étudiants, la possibilité de voir la souris des autres utilisateurs et la possibilité de discuter présentaient des facteurs de distraction assez sévères.

De plus, la complexité des messages d'erreur (à cause des nouvelles technologies utilisées) a mené plusieurs fois à une perte des modifications faites. Ceci est pourtant dû au protocole utilisé par Aspects et peut être évité en grande ligne dans les nouvelles applications.

3.3 Collaboration et conscience

Aux problèmes techniques se rajoutent assez facilement des problèmes de collaboration et de conscience. Les étudiants ont facilement perdu la conscience personnel et ont fini par plus se retrouver dans le document ou d'oublier de savoir ce qu'ils voulaient faire. La perte de conscience peut provenir de plusieurs facteurs, soit la distraction par un changement dans la projection, soit par deux modifications parallèles sur l'écran ou il faut réfléchir qui a fait quel changement et si c'était ce qu'il fallait faire ou pas.

Dans un travail de groupe, comme l'édition collaborative, il peut y avoir plusieurs questions qu'un utilisateur peut se poser: où es-tu? que fais-tu ? qui a fait cela ?

La complexité avec laquelle une application arrive à répondre ces questions joue un grand rôle dans la facilité d'utilisation d'une application.

3.4 Contrôle du document

Un autre problème peut poser le contrôle du document. Les deux groupes d'étudiants ont utilisé Aspects dans le mode verrouillage de paragraphes, c'est-à-dire le paragraphe entier est verrouillé lorsque on a sélectionné une partie seulement. Les étudiants, qui ont voulu éviter qu'une modification soit faite à quelque chose qu'ils venaient d'écrire, ont délibérément pas mis des retours à la ligne pour éviter la création d'un nouveau paragraphe et ainsi permettant à d'autres utilisateurs de changer le contenu.

3.5 Conclusion

Comme cette étude à montré, les applications en temps-réel peuvent imposer, malgré les avantages envers des solutions classiques, des limitations dans la manière qu'une personne travaille. Alors qu'une telle application est favorable à un travail de groupe, l'individu risque de se perdre facilement ou de se heurter à des limitations techniques et finalement est freiné dans sa vitesse de travail.

Les fonctionnalités qu'un tel système offre ont tous leurs avantages et désavantages et demandent une utilisation mûre et bien structuré par les utilisateurs qui risquent de se distraire entre eux. La conscience des autres utilisateur dans le même document peut mener à des rivalités ou à un sentiment de surveillance: on finit par suivre les actions des autres utilisateurs pour savoir ce qu'ils font.

4. Approche documents structurés

Après avoir présenté quelques applications d'édition collaborative, aussi bien des éditeurs avec niveau de couplage fort qu'avec niveau de couplage lâche, je veux expliquer comment on peut utiliser les documents structurés pour réunir à la fois les avantages des deux niveaux de couplage.

J'aimerais rappeler les points forts et points faibles des deux méthodes:

	<i>Avantages</i>	<i>Désavantages</i>
Niveau de couplage fort	<ul style="list-style-type: none">- permet de définir les unités de partage- conscience des autres utilisateurs- plus de possibilités de collaboration	<ul style="list-style-type: none">- protocoles propriétaires- risque de distraction- utilisateur doit s'habituer à un nouveau éditeur
Niveau de couplage lâche	<ul style="list-style-type: none">- protocoles standards (HTTP, FTP)- éditeur personnel	<ul style="list-style-type: none">- unité de partage est le document entier- possibilité d'incohérence entre les mises à jour- moyens d'interaction très limités

Table 1: Avantages et désavantages des différents niveau de couplage

4.1 Extensible Markup Language (XML)

Le « langage de balisage extensible », dans la suite abrégé XML, est un standard du W3C [W3C] qui facilite le partage de texte et d'informations structurées en séparant le contenu (les données) du contenant (la présentation des données). XML est donc un langage qui décrit et contient des données.

Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML (ISO 8879). Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere. (W3C, XML website)

XML est à la base de nombreux autres langages, tous basés sur XML, qui ont de nombreuses applications. Le XML Schema est un langage qui peut décrire la structure d'un document XML, exprimé comme des contraintes sur la structure et le contenu des documents de ce type. En tant que tel, XML Schema est un outil puissant pour définir des types de document avec un niveau d'abstraction élevé.

XML Schema express shared vocabularies and allow machines to carry out rules made by people. They provide a means for defining the structure, content and semantics of XML documents. (W3C, XML Schema website)

4.2 XML et XML Schema pour la définition de type de document

En connaissant les performances de XML et les grands avantages de XML Schema pour la définition et structuration de document, on peut imaginer rajouter une étape supplémentaire au processus d'édition de document: définition de la structure du document.

Cette tâche très importante pour le déroulement de l'édition consiste à modulariser le document dans un ensemble de petits éléments. Pour cela, l'auteur du document ou un groupe de personnes, se mettent d'accord sur une structure d'un document en utilisant le mécanisme de composition-décomposition. Tout comme en programmation orienté-objets, on essaie de modulariser des parties du document et de les regrouper dans des structures de données.

Afin de définir une structure du document, l'auteur peut utiliser deux approches différentes:

- approche textuelle: il essaie de subdiviser le document en utilisant les données textuelles, exemple: paragraphe, titre, phrase, mot, lettre etc
- approche logique: la subdivision du document se fait sur le sens des différentes parties, dépendant bien sur du document qu'on veut définir. On peut imaginer par exemple: biographie, introduction, résumé, entête, signature, date de naissance etc

En utilisant les possibilités de XML Schema et les deux approches mentionnées, on arrive facilement à trouver une structure pour un document. Voici un court exemple pour une liste de liens:

```
<xs:element name="liste">  
  <xs:complexType>  
    <xs:sequence maxOccurs="unbounded">
```

```

<xs:element name="lien">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="URI" type="xs:anyURI"/>
      <xs:element name="text" type="xs:string"/>
      <xs:element name="date_validation" type="xs:date"/>
      <xs:element name="description">
        <xs:complexType>
          <xs:sequence maxOccurs="3">
            <xs:element name="parag">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:maxLength value="300"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

Cet exemple nous montre comment une courte phase de réflexion peut mener à une structure définissant la structure et le sens des différentes parties. En rajoutant aux restrictions de XML Schema un certain nombre de contraintes (par exemple des contraintes d'intégrité), que je vais expliquer plus en détail dans la partie 4.4, on peut arriver à un modèle de document qui peut faciliter l'édition collaborative par la suite.

4.3 XML pour une édition collaborative plus efficace

Cette idée, qui a déjà été traitée en partie dans le concept des document farms [Sire 01], se base sur le fait qu'on arrive, même avec un niveau de couplage lâche, à utiliser différentes unités de partage grâce à la structure du document XML.

Prenant l'exemple de notre liste de liens, dont le Schema se trouve en haut. On peut imaginer qu'une instance d'un document pourrait être:

```

<?xml version="1.0" encoding="UTF-8"?>
<liste xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="link_list.xsd">
  <lien>

```

```

<URI>http://www.w3.org</URI>
<text>WWW Consortium</text>
<date_validation>2006-06-14</date_validation>
<description>
  <parag>The World Wide Web Consortium (W3C) develops interoperable technologies (specifications,
    guidelines, software, and tools) to lead the Web to its full potential.</parag>
  <parag>W3C is a forum for information, commerce, communication, and collective understanding.</parag>
</description>
</lien>
<lien>
  <URI>http://www.epfl.ch</URI>
  <text>École Polytechnique Fédérale de Lausanne</text>
  <date_validation>2006-06-14</date_validation>
  <description>
    <parag>Technical school in Lausanne, Switzerland</parag>
  </description>
</lien>
</liste>

```

On constate que, si un utilisateur veut éditer la description de premier lien, il est inutile de bloquer l'accès au document entier à tous les autres utilisateurs, mais seulement bloquer la partie important, c'est-à-dire, l'élément <description> du premier lien. De même, si l'utilisateur choisit de seulement modifier le deuxième paragraphe de la description du premier lien, donc lorsqu'il se décide de travailler avec une granulation plus fine, il est inutile de bloquer toute la description.

On arrive donc à un système avec niveau de couplage lâche, qui peut utiliser les protocoles standards, comme par exemple le HTTP, et qui présente de nombreux avantages des éditeurs en temps-réel. De plus, les nouvelles approches comme Ajax [Ajax 05], qui utilisent la possibilité d'envoyer des XMLHttpRequest depuis des pages Web, arrivent à un niveau d'interaction avec l'utilisateur très élevé. On envoyant des requêtes à un service (une page php, un servlet ou un Web service) qui traite les paramètres, accède aux données sur le serveur et puis envoie une réponse bien spécifique pour cette requête, on peut à la fois réduire la bande passante ainsi qu'améliorer les fonctionnalités d'un interface d'édition Web. Un exemple ce qui est possible avec Ajax peut être trouvé sur les pages de Google, qui sont très actif dans ce domaine. Exemple: <http://www.google.com/webhp?complete=1&hl=en>

Puisque le document est soumis aux contraintes et restrictions d'un XML Schema, il est facile de valider le nouveau contenu, après édition par un des utilisateurs, en utilisant la validation d'un document XML par un XML Schema. Cette méthode teste efficacement si la structure et le contenu d'un document XML correspond à sa description dans le Schema et avertit par un message d'erreur en cas de violation. Lors d'un essai de mis-à-jour, en cas de violation des restrictions du Schema, le message d'erreur est

présenté à l'utilisateur (comme par exemple un texte qui dépasse la taille maximale de 300 caractères dans la définition de notre description des liens dans l'exemple) et la mise-à-jour est refusé. L'utilisateur doit modifier le texte et ré-exécuter le processus de mise-à-jour (et donc le processus de validation) pour faire les changements au document commun.

4.4 Restrictions et contraintes d'intégrité

4.4.1 Restrictions imposables par XML Schema

Les spécifications de XML 1.0 (2ième édition) décrivent deux types de contraintes sur les documents XML: la « well-formedness » et la « validity ». Les contraintes de « well-formedness » sont celles imposées par la définition du XML (comme par exemple la bonne utilisation de < et > et un nesting correct), alors que les contraintes de validité sont des contraintes additionnelles sur la structure du document par une DTD.

Le W3C reconnaît quatre sections différentes sur la validation, qui sont les suivantes: Schema Component Constraint, Schema Representation Constraint, Validation Rules, Schema Information Set Contribution (pour plus d'information, veuillez visiter la partie correspondante dans les spécifications de XML Schema: <http://www.w3.org/TR/xmlschema-1/#concepts-schemaConstraints>).

Les possibilités de restreindre la structure et le contenu dans XML Schema sont bien nombreuses. Au niveau le plus bas, XML Schema nous permet de définir la structure d'un document XML en utilisant les opérateurs de séquence (xs:all, xs:sequence et xs:choice) pour définir le nombre d'occurrence des sous-éléments. La possibilité de donner un type à un document (avec des types déjà défini dans le namespace de xml schema, comme xs:string ou xs:date) nous permet de valider le contenu d'un élément d'une manière efficace. De plus, on peut définir un nombre de facettes ou des expressions régulières qui imposent des contraintes supplémentaire sur le contenu d'un élément. Finalement, la validation d'intégrité, donc l'utilisation et la définition exactes de link entre les éléments (par exemple xs:id ou xs:idref), est la dernière contrainte qui est testé par XML Schema.

4.4.2 Contraintes d'intégrité

Une contrainte d'intégrité, qui a ses origines dans le domaine des bases de données

relatives, est « un invariant du champ d'application qui concerne n'importe lequel de ses états ou n'importe lequel de ses changements d'états » (M. Leonard: structures des bases de données, chapitre 2.1). C'est une règle qui définit la logique interne d'un document et on peut en distinguer plusieurs types:

a) Contraintes de structure:

Les contraintes de structure définissent quelles éléments sont permis à l'intérieur d'autres éléments. Ceci est facilement exprimable avec XML Schema et ne nécessite aucune technique additionnelle à part la définition du XML Schema.

b) Contraintes de contenu:

Les contraintes de contenu imposent des restrictions sur le contenu d'un élément, comme par exemple la taille, le type ou la sémantique. De nouveau, ceci est possible avec XML Schema: la possibilité de typage d'un élément, par exemple avec un type de base ou avec un type défini manuellement, ainsi que les facettes, les expressions régulières et les listes de valeurs sont des fonctionnalités puissantes pour définir les contraintes de contenu.

c) Contraintes d'édition

Les contraintes d'édition sont des contraintes introduites spécifiquement pour l'édition collaborative et définissent le grade d'interaction de chacun des utilisateurs avec les différents éléments. Si on suppose qu'un système d'authentification permet d'attribuer à chaque utilisateur une ID unique et de plus, l'utilisateur peut faire partie d'un groupe de personnes, alors les accès se laissent facilement exprimer avec du XML, dont la structure est défini dans un XML Schema défini pour l'application d'édition collaborative.

```

<access>
  <allowed>
    <groups>
      <group>Group01</group>
      <group>Group02</group>
    </groups>
    <individuals>
      <individual>User01</individual>
      <individual>User02</individual>
      <individual>User03</individual>
    </individuals>
  </allowed>
  <denied>
    <individuals>
      <individual>User04</individual>
    </individuals>
  </denied>
</access>

```

Ce petit code de XML montre comment on arrive facilement à définir des accès à des éléments XML de notre document commun. Une application devrait tester pour l'existence d'un tel élément « access », qui est un élément optionnel à chaque élément du XML Schema. Si un tel élément est défini, les règles d'accès de celui doivent être appliqués, sinon les règles d'accès les plus proches, en remontant dans l'arborescence, doivent être appliqués. Il faut remarquer que dans l'exemple, la partie « denied » permet de bloquer certains utilisateurs, même si le groupe auquel ils appartiennent se trouve dans « allowed/groups ». Finalement, on peut imaginer un système plus complexe, qui permet de définir pour chaque utilisateur, pour chaque élément, les différentes actions permises. Les actions peuvent être du type « read », « write » et « delete » mais aussi des actions plus spécifiques au domaine de l'édition collaborative comme le droit de mettre un verrou sur un élément (et donc bloquer l'édition par tout autre utilisateur).

d) Contraintes logiques

Les contraintes logiques sont les plus durs à exprimer. Ils portent sur le contenu de certains éléments et peuvent exprimer des contraintes temporelles, géographiques, structurelles (ex: l'élément « a » doit figurer dans la liste défini par l'élément « b ») ou autres. Comme le domaine d'application est tellement grand, il est difficile de trouver une méthode qui marche pour chaque contrainte. Pour exprimer les contraintes temporelles, on pourrait imaginer une solution qui utilise les contraintes de timing et de synchronisation de SMIL 2.0 (qui utilise la logique temporelle [TEMP]). De même pour les contraintes géographiques, une solution utilisant les contraintes spatio-temporel [MADS] est envisageable mais très difficile à réaliser.

Un autre type de contraintes logiques est celui qui suit une expression booléenne classique. Ceci est exprimable en XML si on rajoute une DTD qui contient les règles pour les opérateurs booléens (les opérateurs unaires et binaires). Une fois cette DTD définie, on peut construire un arbre qui contient l'expression booléenne qui doit être évalué par une application. Un système se basant sur `xs:ID` et `xs:IDREF` peut être utilisé pour préciser quels éléments sont à utiliser dans cette expression.

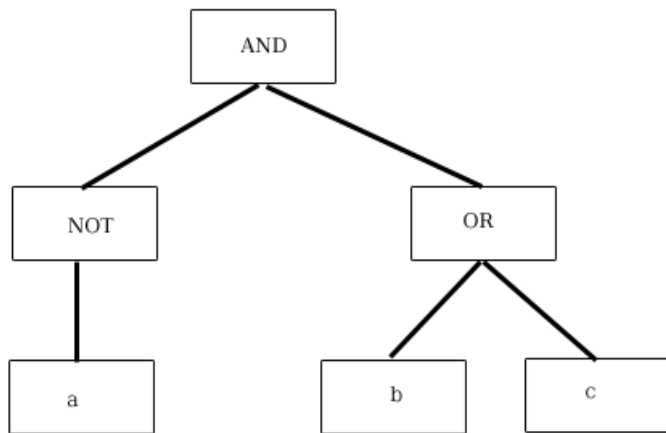


Illustration 5: Arbre d'une expression booléenne

Ceci est un exemple d'un arbre d'une expression booléenne qui peut être évalué avec une simple grammaire récursive (de la même manière comme on évalue les expressions logiques en compilation). Les éléments `a`, `b` et `c` doivent être définis plus haut en utilisant les `xs:IDREF` des éléments qui contiennent les valeurs dans le document XML.

4.5 GroupEdit, un éditeur collaboratif online

Dans ce chapitre, j'aimerais présenter GroupEdit, un éditeur collaboratif online imaginaire. Bien qu'il n'existe pas, c'est le genre d'éditeur qu'on peut implémenter grâce à la technologie de XML, XML Schéma et AJAX (Asynchronous Javascript, XML, CSS, et XHTML). L'éditeur est constitué de deux parties:

- a) la partie serveur qui est programmé en PHP et qui s'occupe de gérer les documents XML qui sont stockés sur le serveur. Comme pour chaque application, où plusieurs utilisateurs veulent accéder en parallèle à des informations, on nécessite un système qui garantit les propriétés ACID (Atomicity, Consistency, Isolation et Durability) et pour cela on utilise une base de données XML. (on pourrait utiliser une base de données relationnelle mais ceci est un problème qui s'adresse au laboratoire de bases de données).

b) la partie client qui doit permettre plusieurs tâches: la gestion des utilisateurs, la création d'un nouveau XML Schéma, la création d'un nouveau document XML en utilisant un XML Schéma existant, l'édition du document XML et finalement la définition d'une XSLT pour visualiser le document XML d'une façon adéquate. De plus, le client contient une infrastructure AJAX qui permet la communication avec le serveur et l'évaluation des informations XML reçues.

4.5.1 Gestion des utilisateurs

La gestion des utilisateurs est une tâche facile et ils existent déjà plusieurs systèmes très performants pour la gestion et l'authentification des utilisateurs sur une page Web (comme par exemple dans le projet PEAR: <http://pear.php.net>). Le système permet de définir des groupes d'utilisateurs, des utilisateurs et l'attribution des utilisateurs dans des groupes.

4.5.2 Création d'un XML Schéma

La création d'un XML Schéma de nos jours est assez simple si on utilise une application comme Altova XMLSpy, pourtant ceci peut être une tâche compliqué à réaliser dans une page Web. C'est pourquoi l'application permet de faire un upload d'un XML Schéma qui a été construit offline et s'occupe seulement de la validation de ce XML Schéma. Pour les personnes qui ne disposent pas d'une telle application, le client offre un éditeur qui contient les règles de grammaire des XML Schéma. En utilisant AJAX, le client peut demander au serveur les différentes règles et proposer la version nécessaire avec un système d'auto-complete. Dès que l'utilisateur entre les caractères « xs », le client montre dans une petite fenêtre toutes les possibilités et l'utilisateur peut choisir son choix avec la souris ou le clavier. De même si on est en train de spécifier les attributs: le client affiche dans une petite fenêtre tous les attributs possibles et affiche en rouge ceux qui sont obligatoires. En cliquant sur un élément défini, une petite fenêtre à côté de l'éditeur permet de définir les différents paramètres comme le type, les facettes etc.

Une petite toolbar au-dessus de l'éditeur permet de facilement tester la validation du XML Schéma ou de le sauvegarder sur le serveur.

4.5.3 Création d'un nouveau document XML

Pour permettre une création efficace, cette étape peut être réalisée en peu de temps: dans une même fenêtre, le client affiche le champ pour entrer le nom du document XML, les XML Schéma disponible pour la création du document et les groupes/utilisateurs possibles. Une fois les choix effectués, le document est créé sur le serveur et les accès au document sont spécifiés dans l'élément root du document comme expliqué dans 4.4.2 partie « c ».

4.5.4 Édition du document XML

Ceci est bien sûr la partie la plus importante et contient le plus de fonctionnalités. Pour ne pas trop distraire l'utilisateur, la plupart des fonctionnalités sont cachées initialement et un simple click de la souris permet de les afficher en utilisant DHTML. Pour ne pas trop surcharger le client, il est même possible de ne pas charger toutes les fonctionnalités cachées et de les charger en cas de besoin avec AJAX, qui les rajoute dynamiquement au DOM de la page Web.

La plus grande partie de la fenêtre est bien sûr occupée par l'éditeur du document. Lorsque l'utilisateur décide de vouloir éditer un document, le client envoie une XMLHttpRequest au serveur et celui-ci renvoie seulement le root élément. L'utilisateur peut ensuite choisir l'élément root (il ne sélectionne pas l'élément mais click sur une icône qui se situe à côté de l'élément) et une nouvelle XMLHttpRequest est envoyée au serveur mais cette fois-ci elle contient le xpath de l'élément choisi. Le serveur renvoie les fils de cet élément qui sont rajoutés au DOM du document sur la page Web et donc affichés pour l'utilisateur. On arrive de cette façon à un niveau d'interaction qui sont jusqu'à maintenant seulement possible dans les applications desktop classiques.

Si l'utilisateur décide de rajouter des éléments au document, un simple click sur un des éléments déjà existant suffit pour faire afficher une liste de tous les sous-éléments permis par le XML Schéma (bien sûr ceci se fait de nouveau avec une requête XMLHttpRequest et Ajax). Il peut ensuite choisir l'élément qu'il veut rajouter et dynamiquement ceci est changé sur le serveur et dans le client (après validation par le serveur).

Lorsque l'utilisateur veut changer le contenu d'un élément, après avoir choisi l'élément à éditer, un simple click sur « Édition » permet d'ouvrir une nouvelle fenêtre qui

est cette fois-ci un éditeur texte classique. Une fois l'édition fini, le nouveau contenu est envoyé au serveur où il est validé, comme expliqué dans le chapitre 4.3. Comme les types défini par XML Schéma sont dans la plupart du cas du CDATA, cet éditeur peut être très basique car il n'a pas besoin de gérer toutes les parties concernant le formatage du texte entré.

A toutes ces possibilités d'édition se rajoutent maintenant les éléments de « collaboration ». Si l'utilisateur le désire, une petite fenêtre affiche les utilisateurs qui travaillent actuellement sur le document avec une couleur unique qui les identifie. Comme le document est affiché sous forme d'arbre, il est possible de colorier les parties qui sont verrouillées par d'autres utilisateurs avec la couleur respective sans trop distraire l'utilisateur. Seulement l'élément sera colorié et tous les sous-éléments ainsi que le contenu des sous-éléments ne seront plus affichés. Lorsque l'édition est fini, la couleur disparaît et l'utilisateur peut faire afficher les éléments changés et donc voire les modifications.

Finalement, comme un document XML n'a pas de charme visuel, le client dispose d'un système de CSS efficace qui permet d'afficher l'arborescence de façon agréable.

4.5.5 Visualisation du document XML

Le plus grand problème avec une approche XML est que le résultat est « seulement » un document XML. Il ne contient aucune instruction pour le formatage des informations et ne peut pas être qualifié de « final » car il nécessite des traitements supplémentaires pour être distribué. Pour cela, le client permet de définir un document XSLT qui s'occupe de transformer le document XML en document HTML. Comme cette transformation n'est pas simple, l'utilisateur a le choix de télécharger le document XML et de faire les transformations dans une autre application. La création du document XSLT a les mêmes fonctionnalités que déjà les autres éditeurs et ceux-ci sont de nouveau réalisés avec AJAX.

4.6 Conclusion

Avec avoir vu les solution actuelles pour une édition collaborative et l'approche avec des documents structurés, il est possible de comparer les différentes solutions de nouveau:

	<i>Avantages</i>	<i>Désavantages</i>
Niveau de couplage fort	<ul style="list-style-type: none">- permet de définir les unités de partage- conscience des autres utilisateurs- plus de possibilités de collaboration	<ul style="list-style-type: none">- protocoles propriétaires- risque de distraction- utilisateur doit s'habituer à un nouveau éditeur
Niveau de couplage lâche	<ul style="list-style-type: none">- protocoles standards (HTTP, FTP)- éditeur personnel	<ul style="list-style-type: none">- unité de partage est le document entier- possibilité d'incohérence entre les mises à jour- moyens d'interaction très limités
Approche XML	<ul style="list-style-type: none">- grande liberté pour la structuration du document- permet de définir les unités de partage- conscience des autres utilisateurs- protocole standard (HTTP par exemple)- éditeur simple car seulement texte- distraction limité à un minimum	<ul style="list-style-type: none">- nécessite de définir un document XSLT pour la mise en page- utilisation des technologies XML trop avancée pour certaines personnes

Table 2: Avantages et désavantages des différentes approches

On voit clairement que cette approche permet d'avoir les avantages des deux approches avec le niveau de couplage en réduisant à un minimum les désavantages. Cependant, cette approche a plusieurs désavantages, notamment parce qu'elle est très compliqué à un utiliser. Même si un utilisateur définit seulement trois paragraphes en XML Schéma, même avec l'éditeur le plus performant, il doit pourtant apprendre de manipuler un nouveau éditeur. Une fois le XML Schéma défini, l'affichage du document en arbre peut poser un problème à des utilisateurs peu expérimenté ce qui rend l'édition pas intuitive. Finalement le plus grand désavantage est que cette approche produit un document XML. Bien que ce document est dans un format qui permet d'innombrables méthodes pour le transformer, il est pourtant nécessaire de rajouter des transformations pour avoir un résultat sous forme de pdf ou de html. Comme cette transformation est en plus pas facile, la définition de celle-ci est une tâche qui nécessite un expert ou une personne avec beaucoup d'expérience.

5. Possibilités additionnelles

L'approche avec des documents structurés présente non seulement des avantages pour l'édition de document mais nous offre de plus certains avantages qui sont spécifiques à la structure des document XML. Une telle possibilité est le workflow, une approche pour organiser des processus que j'aimerais expliquer maintenant et ensuite expliquer comment on peut l'intégrer dans le document XML.

5.1 Workflow

Le workflow désigne le mouvement de documents ou de tâches à travers un processus. Un tel processus est un graphe directionnel où chaque noeud du graphe correspond à une tâche. Le graphe entier définit la structure des différentes tâches, l'ordre dans lequel ils doivent être exécuté, les personnes qui doivent réaliser une tâche, la synchronisation entre les différentes tâches et les informations qui circulent dans le graphe.

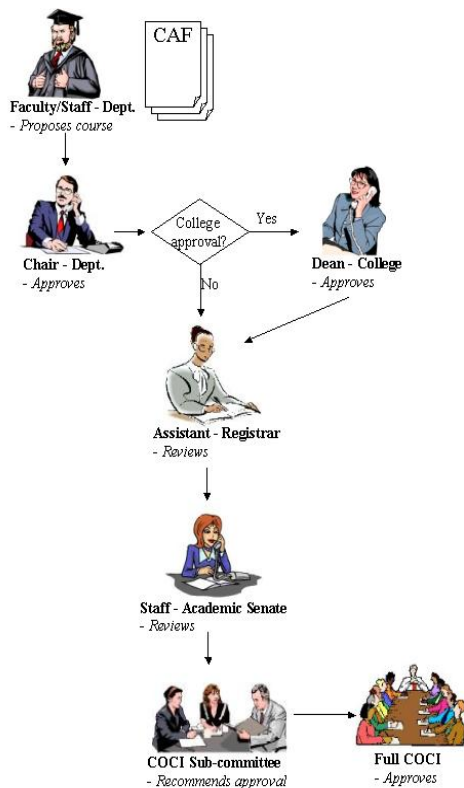


Illustration 6: Représentation d'un graphe d'un workflow

Comme l'illustre l'image de gauche, un graphe d'un workflow définit des documents qui traversent le graphe et qui sont transformés dans chaque noeud par une autre personne. Un type spécial de noeud permet l'exécution parallèle de tâches, la synchronisation de tâches où la validation du document en utilisant des règles bien définies.

Comme ceci est un domaine de recherche très actif, des nombreuses solutions pour la formalisation d'un telle graphe ont déjà été proposées, notamment par IBM, qui ont proposé le WSFL (Web Services Flow Language), et finalement le BPEL (Business Process Execution Language). Ce langage est basé sur XML et suit un but bien précis, celui

de « programming in the large ».

Programming in the large generally refers to the high-level [state transition](#) interactions of a process—BPEL refers to this concept as an Abstract Process. A BPEL Abstract Process represents a set of publicly observable behaviors in a standardized fashion. An Abstract Process includes information such as when to wait for [messages](#), when to send messages, when to compensate for failed transactions, etc. Programming in the small, in contrast, deals with short-lived programmatic behavior, often executed as a single transaction and involving access to local logic and resources such as [files](#), [databases](#), etc. BPEL's development came out of the notion that programming in the large and programming in the small required different types of language.

Comme cette approche peut être aussi appliquée à une édition collaborative, je veux essayer d'expliquer comment ceci peut être réalisé.

5.2 Workflow définition dans un document XML

Afin de pouvoir utiliser la notion de workflow, il est nécessaire de séparer le document XML en deux parties: la première contient la définition du workflow et la deuxième contient le document XML. Il est possible, en utilisant BPEL ou en définissant une DTD pour exprimer les graphes d'un workflow en XML, de définir le graphe que le document doit parcourir de sa création jusqu'à son état final.

On a vu que l'édition d'un document XML nécessite une phase très importante de réflexion sur la structure du document. A cette étape se rajoute une nouvelle, notamment celle où on réfléchit sur différentes tâches nécessaires à l'édition de ce document. Ensuite, il est nécessaire de définir le graphe du workflow en utilisant les possibilités à disposition.

Cette approche peut être subdivisée en plusieurs étapes:

a) Trouver les étapes importantes à l'édition de ce document. Si on prend l'exemple d'un rapport pour un travail de semestre, on peut imaginer que la dernière étape d'un tel graphe est la validation du document par le professeur responsable avec deux sorties possibles: si le professeur accepte le document, le document va dans l'état final (ou l'état de sorti du graphe), sinon le document doit aller dans un état d'édition qui appartient à l'étudiant.

b) Trouver les personnes importantes: Certaines parties peuvent être effectuées par tout le monde alors que certaines tâches sont liées à une personne spéciale. Dans l'exemple de notre rapport pour le travail de semestre, le professeur est une telle personne. Il doit être spécifié dans le système.

c) Définir les conditions temporelles et logiques: les graphes peuvent contenir des étapes spéciales qui contiennent des conditions logiques ou des conditions temporelles. Dans l'exemple du rapport pour le projet de semestre, on peut imaginer qu'après édition du rapport, celui est envoyé au professeur et à son assistant. Le professeur a le droit de valider le rapport alors que l'assistant a seulement le droit de le refuser. Donc, ceci représente deux étapes différentes qui sont pourtant relié par une condition logique. Il est nécessaire d'introduire une étape logique et exprimer la condition par une expression booléenne.

d) Rajouter les personnes aux étapes: une fois que la structure du graphe est défini, les personnes sont rajoutés aux étapes qu'ils ont le droit d'effectuer.

e) Définition des permissions pour chaque étape: il est nécessaire de définir les permissions qu'une personne a lorsqu'elle effectue une étape. Il se peut qu'elle a seulement droit de modifier une partie du document et ceci doit être exprimé.

Une fois que ce graphe est défini, il est rajouté à la définition du document XML dans une partie qui lui est attribué. De plus, une instance de ce graphe doit être créée qui contient l'état du graphe, comme par exemple où se trouve le document actuellement ou quelles sont les valeurs de sorties d'un état. Si un tel graphe est défini pour un document, l'édition devient structuré: il n'est plus possible d'éditer n'importe quelle partie du document si on le désire mais on doit attendre jusqu'à ce que le document ait atteint un état dans lequel on peut faire l'édition. Une telle approche permet d'éviter des conflits lors d'un travail de groupe qui peuvent provenir de malentendus ou de manque de communication entre les membres du groupes.

6. Fonctionnalités nécessaires

Maintenant qu'on a vu quelques approches pour l'édition collaborative, on peut se demander quelles sont en fait les fonctionnalités qu'un tel système doit offrir pour être utilisable. Alors qu'un simple serveur ftp permet déjà une forme très basique d'édition collaborative, il présente cependant aucune autre fonctionnalité. J'aimerais ici présenter les fonctionnalités nécessaires d'un système avancé et non d'un simple système comme l'email, qui possède aussi des possibilités de collaboration:

a) Protocole de communication: chose élémentaire, un protocole de communication permet de transférer des messages et des fichiers à travers l'internet. On a vu qu'il est possible de construire une application solide en utilisant des protocoles standard ou en définissant un protocole propriétaire et je suis d'avis que ceci n'a aucune influence sur le succès et d'une telle application.

b) Éditeur de texte: ceci est un choix compliqué. Est-ce qu'il est nécessaire d'intégrer un propre éditeur ou est-ce qu'il suffit d'appeler l'éditeur standard défini par un utilisateur. Cela dépend beaucoup de l'application. Alors qu'une application avec un protocole propriétaire peut rajouter des instructions cachées au code pour permettre plusieurs fonctionnalités, une application qui ne transfère que du texte n'a pas besoin des astuces supplémentaires. Avec le grand nombre de choix qu'on a pour améliorer l'édition collaborative, je crois qu'à partir d'un certain niveau de fonctionnalité, il est nécessaire d'intégrer un propre éditeur texte pour faciliter à un maximum le travail de l'utilisateur.

c) Choix des unités de partage: Afin de permettre une édition rapide sans trop de conflit entre les utilisateurs, il est indispensable de pouvoir choisir une unité de partage plus fine qu'un document entier. L'édition d'un seul paragraphe (ou même un seul mot) ne devrait pas empêcher les autres utilisateurs à travailler convenablement.

d) Garantir la consistance du document: Avec un nombre de collaborateurs croissant, il est nécessaire que le système garantisse la consistance du document. Les personnes qui ont déjà utilisé CVS connaissent les problèmes: deux utilisateurs mettent à jour en même temps un document, résultant dans une différence du document sur le serveur et une intervention manuelle est nécessaire. L'utilisation d'un processus de verrouillage, si le système permet des niveaux de partage fins, est une bonne idée.

e) Conscience des autres utilisateurs: comme on l'a pu constater dans l'expérience avec les étudiants, il est assez facile de distraire les collaborateurs si le niveau de conscience est trop élevé. Pourtant, je suis d'avis qu'un certain niveau de conscience est nécessaire: savoir qui est en train de travailler sur le document et savoir ce qu'il est en train de modifier suffit déjà comme informations. Avoir la position exacte de sa souris, la position où il se trouve dans le document et même quels boutons il a cliqué est trop exagéré.

f) Possibilité de sauvegarde: Ceci peut être réalisé de deux manières. L'application peut permettre de définir des versions de documents avec sauvegarde automatique ou l'utilisateur peut télécharger le document et uploader le document à n'importe quel moment, ce qui permet de faire une sauvegarde manuelle. Pourtant il faudra introduire des niveaux de permission dans le cas où un utilisateur peut uploader une version d'un document car sinon on entre de nouveau dans le conflit avec la consistance du document.

7. Conclusion

8. Références

- [Dewan 91] P. Dewan et R. Choudhary, Flexible user interface coupling in a collaborative system. In proceedings of the ACM CHI'91, pp. 41-48 1991
- [SubEdthaEdit] SubEthaEdit, Collaborative Text Editing. Share and Enjoy
See: <http://www.codingmonkeys.de/subethaedit/index.html>
- [Pacull 94] F. Pacull, A. Sandoz and A. Schiper, Duplex: a distributed collaborative editing environment in large scale, In proceedings of the conference on Computer supported cooperative work (Chapel Hill, United States), ACM Press, pp.165-173, 1994
- [Decouchant 93] D. Decouchant, V. Quint, M. Riveill and I. Vatton, Griffon: A Cooperative, Structured, Distributed Document Editor, Bull-IMAG (Grenoble, France), R.R. 20, 1991
- [ACE] ACE, a collaborative editor.
See: <http://ace.iserver.ch/>
- [Rapport-Ace] Mark Birgler, Simon Räss, Lukas Zbinden: Ace, a collaborative editor, Report Evaluation Algorithms. Berne University of Applied Sciences, April 2005.
- [Bonjour] Bonjour, a network service solution by Apple.
See: <http://www.apple.com/macosx/features/bonjour/>
- [Writely] Writely, The Web Word Processor. Currently owned by Google
See: <http://www2.writely.com/info/WritelyOverflowWelcome.htm>
- [Horstmann 97] T. Horstmann and R. Bentley, Distributed authoring on the Web with the BSCW shared workspace system, StandardView, 5 (1), pp. 9-16, 1997
- [CVS] CVS – Concurrent Versions System.
See: <http://www.nongnu.org/cvs/>
- [Postel 85] J. Postel and J.Reynolds, File Transfer Protocol (FTP), RFC 959, October 1985
- [Fielding 99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee, Hypertext Transfer Protocol – HTTP/1.1, RFC 2616 – IETF Draft Standard, June 1999. See: <http://www.ietf.org/rfc/rfc2616.txt>
- [Goland 99] Y. Goland, E. Whitehead, A. Faizi, S. Carter and D. Jensen, HTTP Extensions for Distributed Authoring – WEBDAV, February 1999
See: RFC 2518 (<http://www.ietf.org/rfc/rfc2518.txt>)
- [Mitchell 95] Alex Mitchell, Ilona Posner and Ronald Baecker, Dynamic Graphics Project,

University of Toronto. Learning to Write Together Using Groupware
See: <http://www.dgp.toronto.edu/CMRG/Projects/CWPublications/CHI95.html>

- [Aspects] Aspects was developed by Group Technologies Inc.
- [SASSE] The SASSE Collaborative Editor, Baecker, R.M., Nastos, D., Posner, I.R., and Mawby, K.L. "The User- Centred Iterative Design of Collaborative Writing Software." *Proceedings of INTERCHI '93*, pp. 399-405, 541.
- [W3C] World Wide Web Consortium
See: <http://www.w3.org/>
- [Sire 01] Stéphane Sire, Yassine Rekik, Christine Vanoirbeek. A flexible collaborative authoring process based on document fragments and contracts. Proceedings 4th International Conference on the Electronic Document, October 24-26, 2001, Toulouse, France
- [Ajax 05] Jesse James Garrett. Ajax: A New Approach to Web Applications, February 2005
See: <http://www.adaptivepath.com/publications/essays/archives/000385.php>
- [TEMP] See: http://en.wikipedia.org/wiki/Temporal_logic
- [MADS] Christine Parent, Stefano Spaccapietra, cooperation with Esteban Zimanyi. MADS: Modeling of Application Data with Spatio-temporal features
See: <http://lbdwww.epfl.ch/e/research/mads/>