

Synchronous Editing via Web Services

Combining Heterogeneous Client and Server Technologies

Jon A Preston and Sushil K Prasad

Department of Computer Science
Georgia State University
Atlanta, GA USA

jon.preston@acm.org and sprasad@gsu.edu

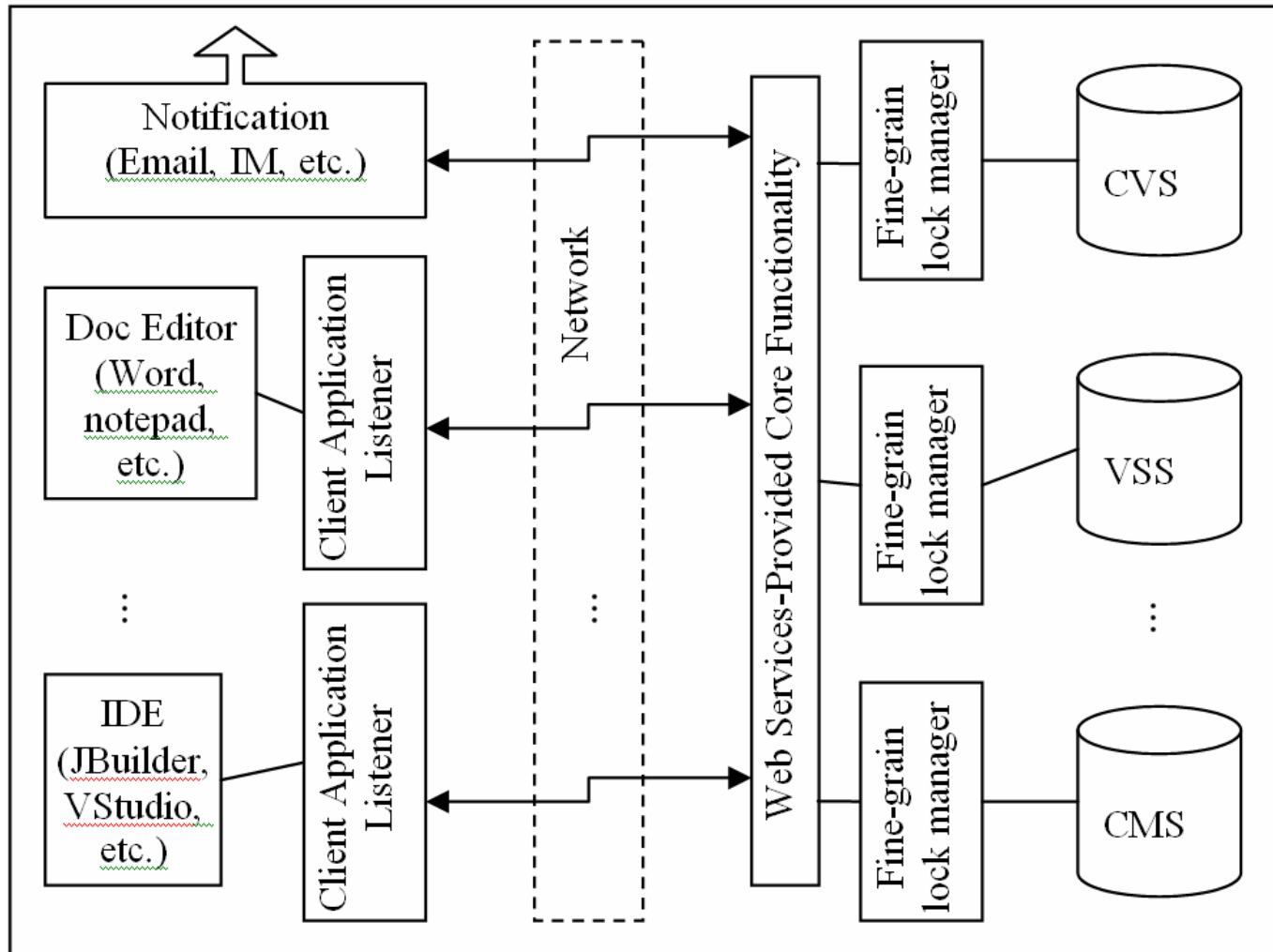
Motivation

- Combine heterogeneous editing tools
- Combine heterogeneous document repositories
- Utilize Web-services to standardize API
- Generalized collaborative editing system
- Increase concurrency while minimizing communication costs

Architecture

- Connects to existing, heterogeneous client applications
 - MS Word, OpenOffice, JavaBeans, MS Studio, etc.
- Connects to existing, heterogeneous server document repositories
 - VSS, CVS, RCS, etc.
- Synchronous and asynchronous change notification
 - Email, IM, etc.

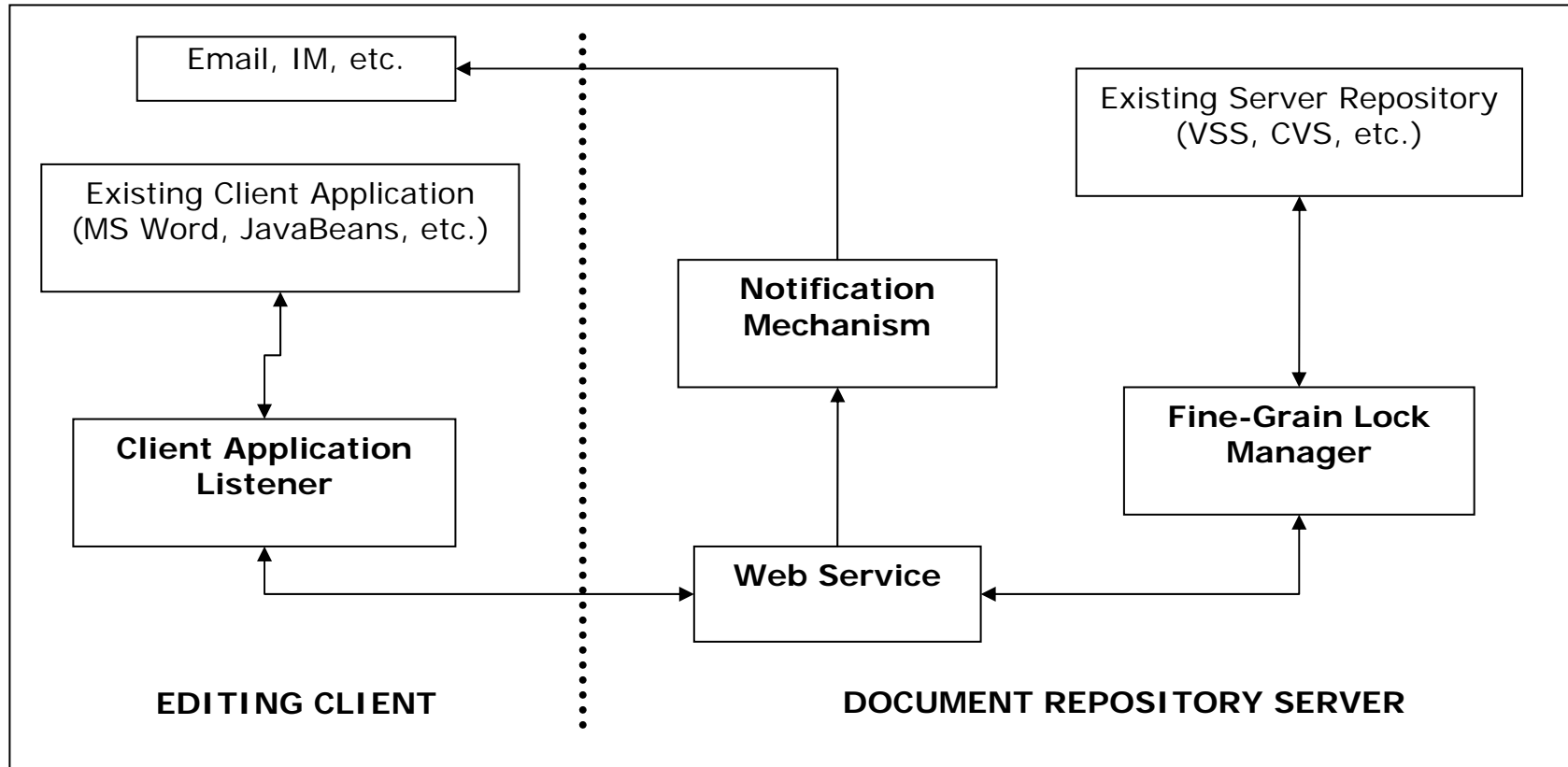
Achieving Heterogeneity



Components

- Client Application Listener
 - Hooks to existing editing software
 - Caches changes
 - Sends changes to Web Service
- Web Service
 - Publishes API – check-in, check-out, subscribe
 - Manages subscriptions and connected users
- Fine-Grain Lock Manager
 - Connects to existing document repository (CMS)
 - Intercepts check-in and check-out messages
 - Acts as check-in, check-out proxy
 - Adds fine-grain locking
- Notification Mechanism
 - Communicates to Email, IM, etc. to notify users of changes based upon their subscription preferences

Open-system Architecture for Distributed Repositories



Client Application Listener

The **Client Application Listener** component listens to change events that occur within the application (edits to the document) and cache changes when possible, sending these changes to the server coordinating the CES. This component also receives update notifications from the server and sends the changes to the existing client application (editing software), thus maintaining consistency among all users' copies of the shared document.

Web Service

The **Web Service** component provides an API for traditional CMS systems (check-in and check-out, etc.) as well as an API for managing changes among the users that are collaborating together (insert, delete, move, etc.). This component also provides an API by which users can subscribe to receive synchronous and asynchronous notification when a document has been changed.

Fine-Grain Lock Manager

The **Fine-Grain Lock Manager** component acts as a proxy that checks-out and checks-in documents from the existing server repository (such as RCS, CVS, VSS, etc.). This component receives check-in and check-out events from the Web Service component and processes and executes these requests via the existing server repository. This component provides the ability to manage artifacts at a finer granularity (viewing an artifact as a collection of sub-artifacts); as an example, a user can edit page one of a shared artifact at the same time another user is editing page two. This component tracks who is currently working on each artifact in the server repository and is thus able to “push” these changes to the necessary clients.

Notification Mechanism

The **Notification Mechanism** component is responsible for passing on any events for which the user has requested notification (document change, check-out, etc.) to the users' preferred email, IM, cell phone text messaging, etc. Clients may subscribe for notification when changes are made (even if they are not currently editing the document); thus the system supports synchronous and asynchronous collaboration.

Fine-Grain Lock Manager

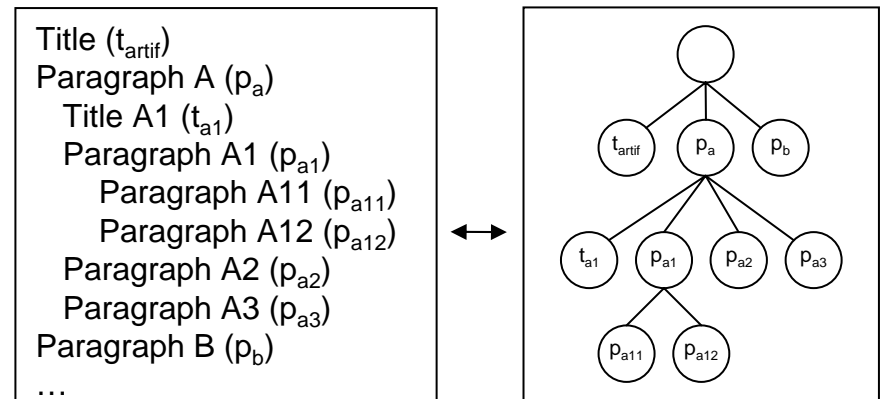
- Situated between the Web Service component and the existing CMS
- Intercepts check-in and check-out requests
- Maintains state of who has which section of each document
- Checks-in and checks-out via proxy
- CMS is unchanged
- Adds ability to do fine-grain locking

Dynamic Locking Algorithm

- Tree representation of document
- Lock largest sub-tree possible
- Demote lock if request creates contention
- Promote lock if contention is removed
- Deadlock-free
- Can integrate OT to share larger subsections if desired

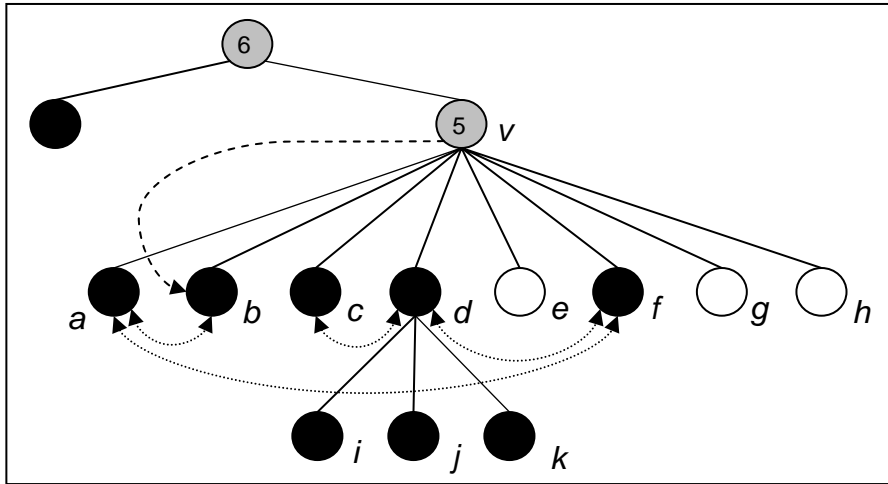
N-ary Document Tree

- Model the document as an n-ary tree
- Sections and subsections
- Users can “own” a portion of the document without blocking other users from editing other portions of the document



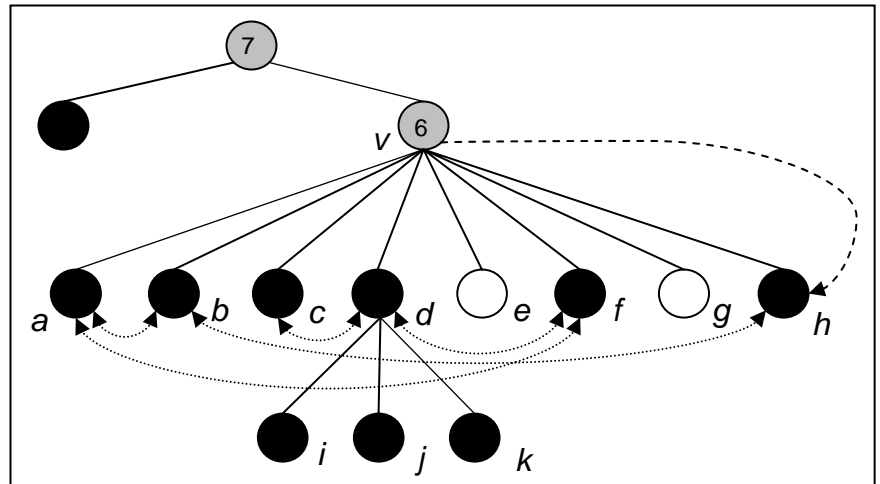
ObtainLock

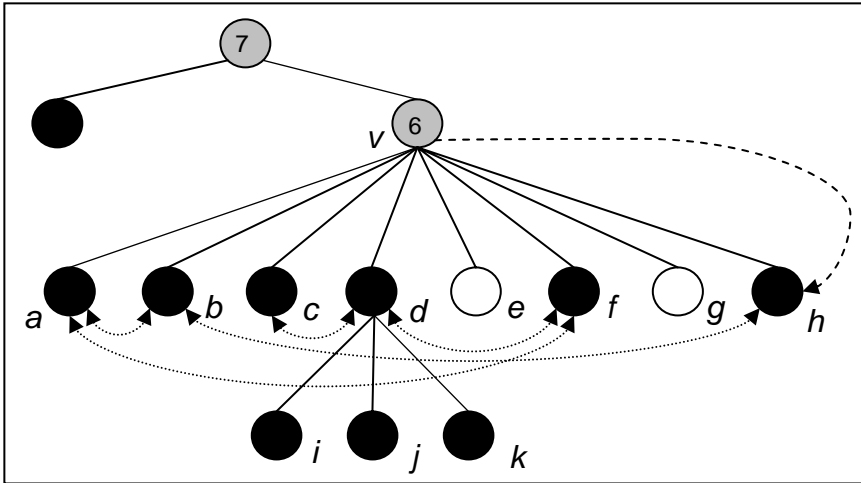
- Traverse top to bottom
- Increase “grey count” as you descend
- Keep descending until you reach a resolution node
 - Obtain an available node
 - Demote another user to resolve conflict
 - Deny request (or adopt OT)



ObtainLock(u_1, h)

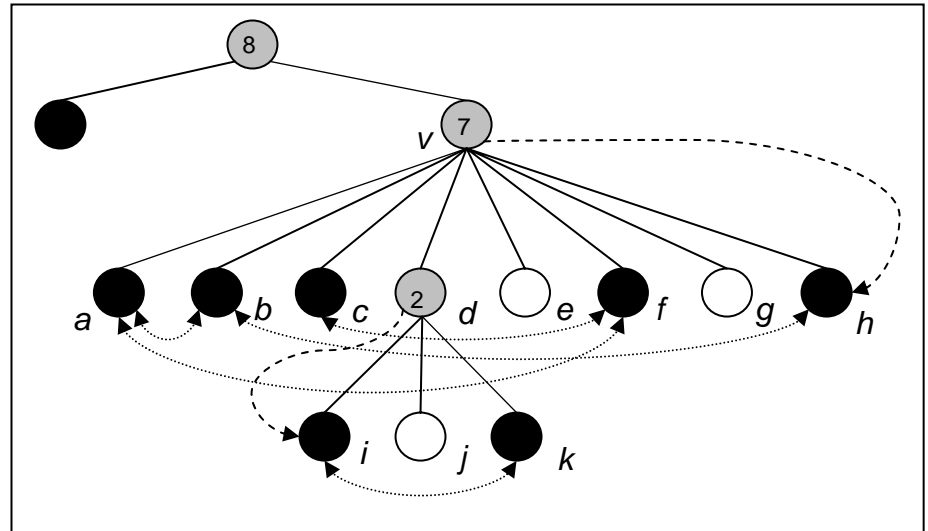
No Demotion





ObtainLock(u_2, k)

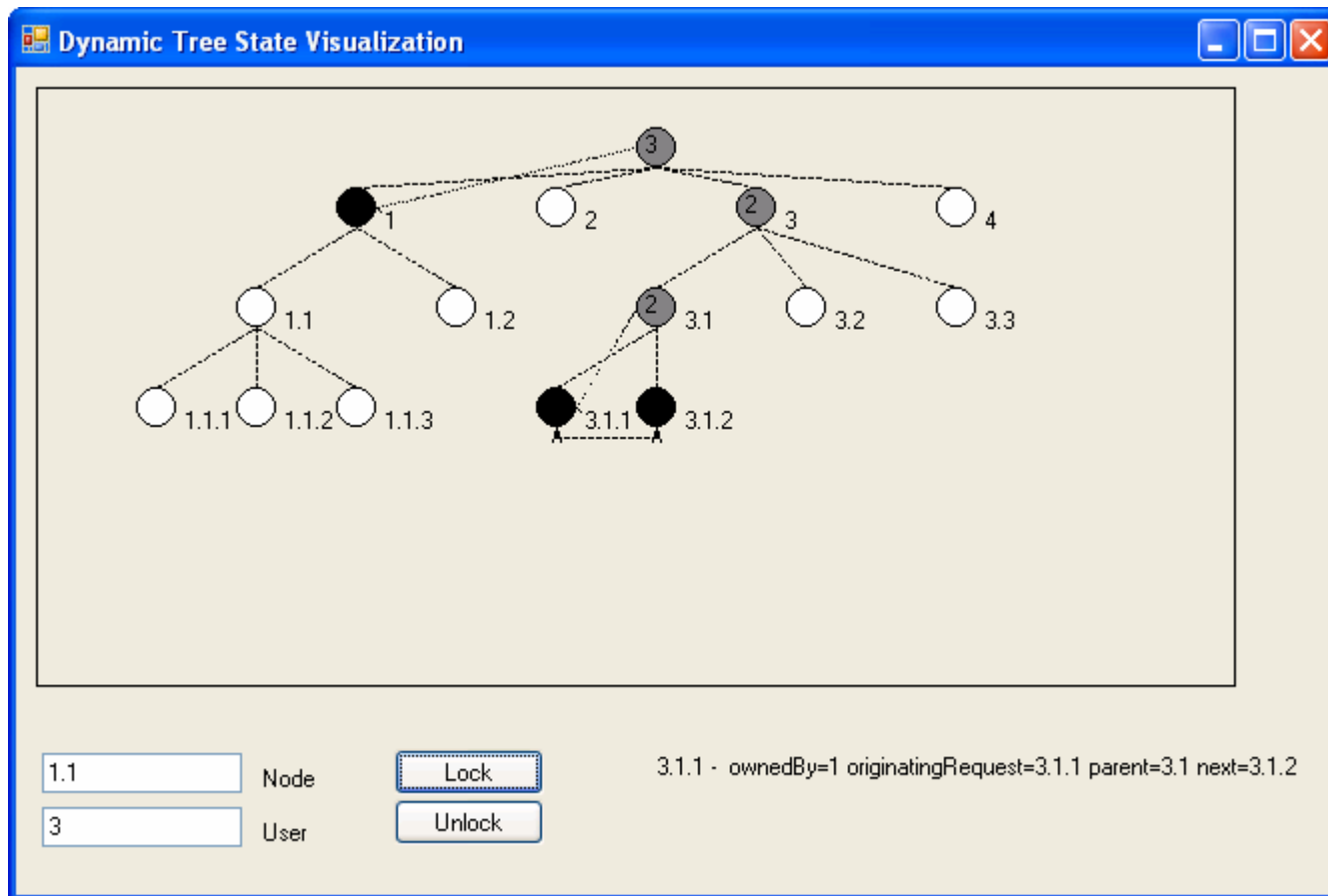
Demotion of u_1 from node d to node i



RemoveLock

- Traverse top to bottom
- Decrease “grey count” as you descend
- Keep descending until you reach a resolution node
 - Release owned node
 - Promote another user if conflict is now removed (“grey count” decreased to 1)

Visualizing the Dynamic Locks



Client Editor

The screenshot shows the 'Client Text Editor' application window. The main text area contains the following content:

Providing a Dynamic RTCES Client

Section 1
In this demo we discuss how an application can connect with a server to download a document from
We also show how locks can be maintained dynamically as users move about the document.

Section 2
A demo of the system would be here

Section 2.1
Architecture of the system

Section 2.2
Components of the system

Section 2.2.1
Client components

Section 2.2.2
Server components

Section 2.2.3
Communication/notification components

Section 2.3
Analysis of architecture

Section 3
Simulation design.
Simulation results

The right-hand pane shows a tree view of the document structure:

- Document
 - (0) Providing a Dynamic RTCES Client
 - (0) Section 1
 - (1) In this demo we discuss
 - (1) We also show how locks
 - (0) Section 2
 - (1) A demo of the system would be here
 - (1) Section 2.1
 - (2) Architecture of the system
 - (1) Section 2.2
 - (2) Components of the system
 - (2) Section 2.2.1
 - (3) Client components
 - (2) Section 2.2.2
 - (3) Server components
 - (2) Section 2.2.3
 - (3) Communication/notification components
 - (1) Section 2.3
 - (2) Analysis of architecture
 - (0) Section 3
 - (1) Simulation design.
 - (1) Simulation results
 - (0) Section 4
 - (1) Conclusions
 - (0) Section 5
 - (1) References
 - (0) Section 6
 - (1) Appendix 1
 - (1) Appendix 2

Parsing Word Documents

