Exploring Primo:
A developer's perspective

A quick tour about Primo development

What is Primo (simplified)

- Cloud-based authoring, submission, and proofing tool with collaborative editing.
- Target audience: Authors who need to publish papers with a publisher.
- Author-centric: Empowers authors to control their publishing process.
    - Authors can see and tweak the output, minimizing the difference between the submitted and published versions.

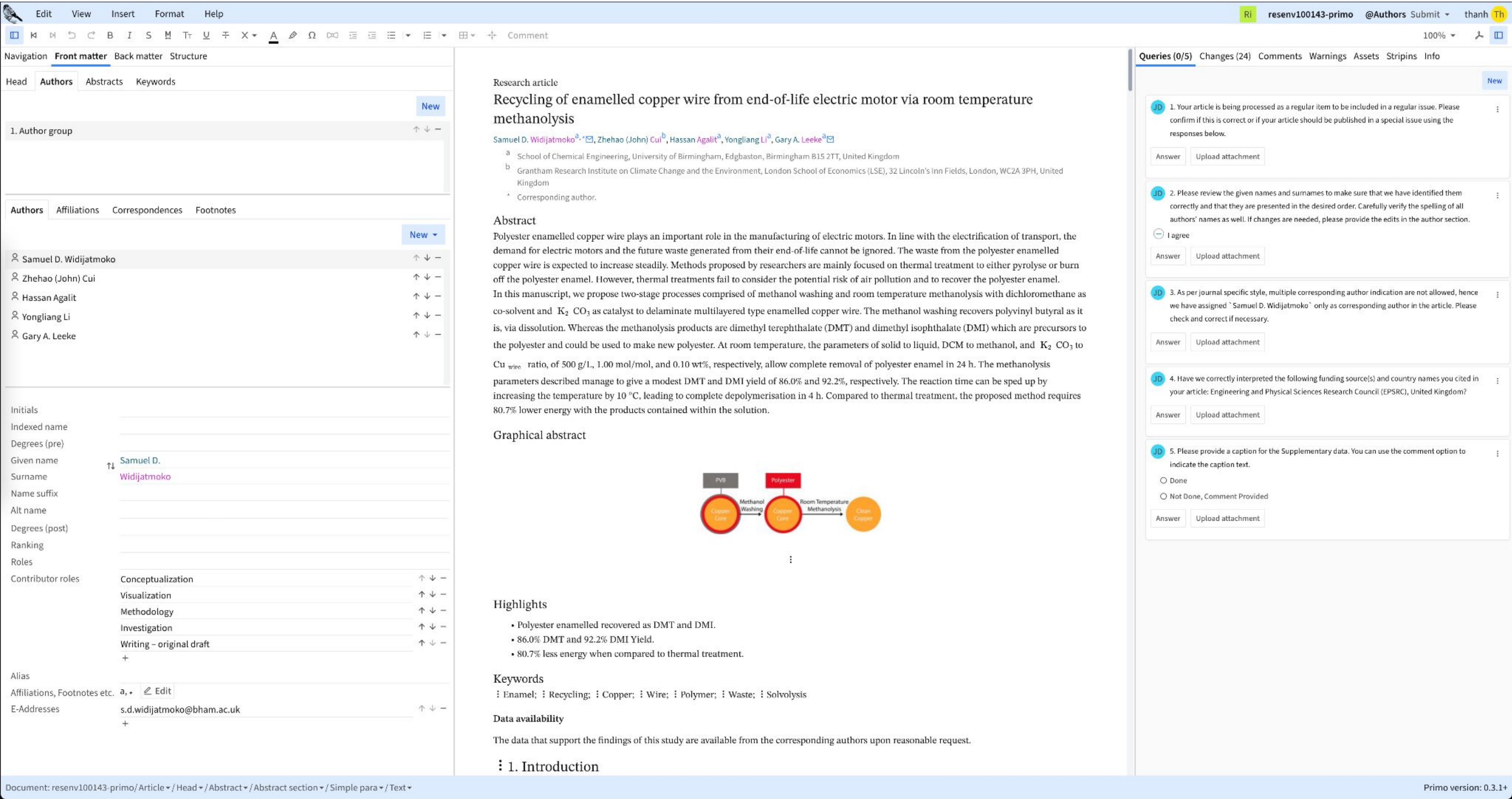
What is Primo (simplified)

- XML-based workflows:
  - Required by publishers.
  - Often disliked by authors.
  - Made simple by Primo.
- High-quality PDF output with math
  - Difficult for XML-based workflows.
  - Easy with Primo using TeX backend.

It sounds good. But what is it **really**?

- Like Google Docs, but for academic publishing.
- Input: XML-based datasets (XML + figures, etc.).
- Editing: WYSIWYG and structural (via menu/form/dialog) editing.
- Output: XML, PDF (by TeX).
- Makes communication between authors and publishers efficient.

Structural editing tools   Editor   Proofing tools



Editor PDF

Edit View Insert Format Help | Ri resenv100143-primo @Authors Submit thanh Th

Comment | 100%

Options | Synchronize scrolling | 3 of 14 | Automatic Zoom

Research article

Recycling of enamelled copper wire from end-of-life electric motor via room temperature methanolysis

Samuel D. Widijatmoko[a,*], Zhehao (John) Cui[b], Hassan Agalit[a], Yongliang Li[a], Gary A. Leeke[a]

[a] School of Chemical Engineering, University of Birmingham, Edgbaston, Birmingham B15 2TT, United Kingdom

[b] Grantham Research Institute on Climate Change and the Environment, London School of Economics (LSE), 32 Lincoln's Inn Fields, London, WC2A 3PH, United Kingdom

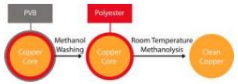
[*] Corresponding author.

Abstract

Polyester enamelled copper wire plays an important role in the manufacturing of electric motors. In line with the electrification of transport, the demand for electric motors and the future waste generated from their end-of-life cannot be ignored. The waste from the polyester enamelled copper wire is expected to increase steadily. Methods proposed by researchers are mainly focused on thermal treatment to either pyrolyse or burn off the polyester enamel. However, thermal treatments fail to consider the potential risk of air pollution and to recover the polyester enamel.

In this manuscript, we propose two-stage processes comprised of methanol washing and room temperature methanolysis with dichloromethane as co-solvent and K_2CO_3 as catalyst to delaminate multilayered type enamelled copper wire. The methanol washing recovers polyvinyl butyral as it is, via dissolution. Whereas the methanolysis products are dimethyl terephthalate (DMT) and dimethyl isophthalate (DMI) which are precursors to the polyester and could be used to make new polyester. At room temperature, the parameters of solid to liquid, DCM to methanol, and K_2CO_3 to Cu_{wire} ratio, of 500 g/L, 1.00 mol/mol, and 0.10 wt%, respectively, allow complete removal of polyester enamel in 24 h. The methanolysis parameters described manage to give a modest DMT and DMI yield of 86.0% and 92.2%, respectively. The reaction time can be sped up by increasing the temperature by 10 °C, leading to complete depolymerisation in 4 h. Compared to thermal treatment, the proposed method requires 80.7% lower energy with the products contained within the solution.

Graphical abstract



Highlights

- Polyester enamelled recovered as DMT and DMI.
- 86.0% DMT and 92.2% DMI Yield.
- 80.7% less energy when compared to thermal treatment.

Keywords

Enamel; Recycling; Copper; Wire; Polymer; Waste; Solvolysis

Data availability

The data that support the findings of this study are available from the corresponding authors upon reasonable request.

1. Introduction

RESENV: 100143

ARTICLE IN PRESS

Resources, Environment and Sustainability xxx (xxxx) xxx

Contents lists available at ScienceDirect

Resources, Environment and Sustainability

journal homepage: www.elsevier.com/locate/resenv

Research article

Recycling of enamelled copper wire from end-of-life electric motor via room temperature methanolysis

Samuel D. Widijatmoko[a,*], Zhehao (John) Cui[b], Hassan Agalit[a], Yongliang Li[a], Gary A. Leeke[a]

[a] School of Chemical Engineering, University of Birmingham, Edgbaston, Birmingham B15 2TT, United Kingdom

[b] Grantham Research Institute on Climate Change and the Environment, London School of Economics (LSE), 32 Lincoln's Inn Fields, London, WC2A 3PH, United Kingdom

ARTICLE INFO

Keywords:
Enamel
Recycling
Copper
Wire
Polymer
Waste
Solvolysis

ABSTRACT

Polyester enamelled copper wire plays an important role in the manufacturing of electric motors. In line with the electrification of transport, the demand for electric motors and the future waste generated from their end-of-life cannot be ignored. The waste from the polyester enamelled copper wire is expected to increase steadily. Methods proposed by researchers are mainly focused on thermal treatment to either pyrolyse or burn off the polyester enamel. However, thermal treatments fail to consider the potential risk of air pollution and to recover the polyester enamel.

In this manuscript, we propose two-stage processes comprised of methanol washing and room temperature methanolysis with dichloromethane as co-solvent and K_2CO_3 as catalyst to delaminate multilayered type enamelled copper wire. The methanol washing recovers polyvinyl butyral as it is, via dissolution. Whereas the methanolysis products are dimethyl terephthalate (DMT) and dimethyl isophthalate (DMI) which are precursors to the polyester and could be used to make new polyester. At room temperature, the parameters of solid to liquid, DCM to methanol, and K_2CO_3 to Cu_{wire} ratio, of 500 g/L, 1.00 mol/mol, and 0.10 wt%, respectively, allow complete removal of polyester enamel in 24 h. The methanolysis parameters described manage to give a modest DMT and DMI yield of 86.0% and 92.2%, respectively. The reaction time can be sped up by increasing the temperature by 10 °C, leading to complete depolymerisation in 4 h. Compared to thermal treatment, the proposed method requires 80.7% lower energy with the products contained within the solution.

1. Introduction

The research of end-of-life EVs can be classified into three main broad topics of battery, traction motor, and power electronics (Elwert et al., 2015). Researchers are mainly targeting batteries as this is where the main recycling incentives lie (Nurdiawati and Agrawal, 2022). In contrast, only a few studies are focussing on electric motors and power electronics (Elwert et al., 2015). Hence, there is the need for more research aimed at electric motors and power electronics to enable close-loop EV recycling.

Tight windings in electric vehicles' motors, transformers, and any other types of electromagnetic equipment require enamelled copper wires as their main component. The enamel is a multi-layer of either a single type or a combination of polymeric insulators such as polyester (Liu et al., 2020), polyester imide (Petitgas et al., 2011), polyurethane (Ueda, 1989), polyimide (Petitgas et al., 2011) and poly(amide-imides) (Murray, 2008), with a typical thickness of 2 μm for each layer (Petitgas et al., 2011) up to a total thickness of 20 μm (Haque et al., 2014).

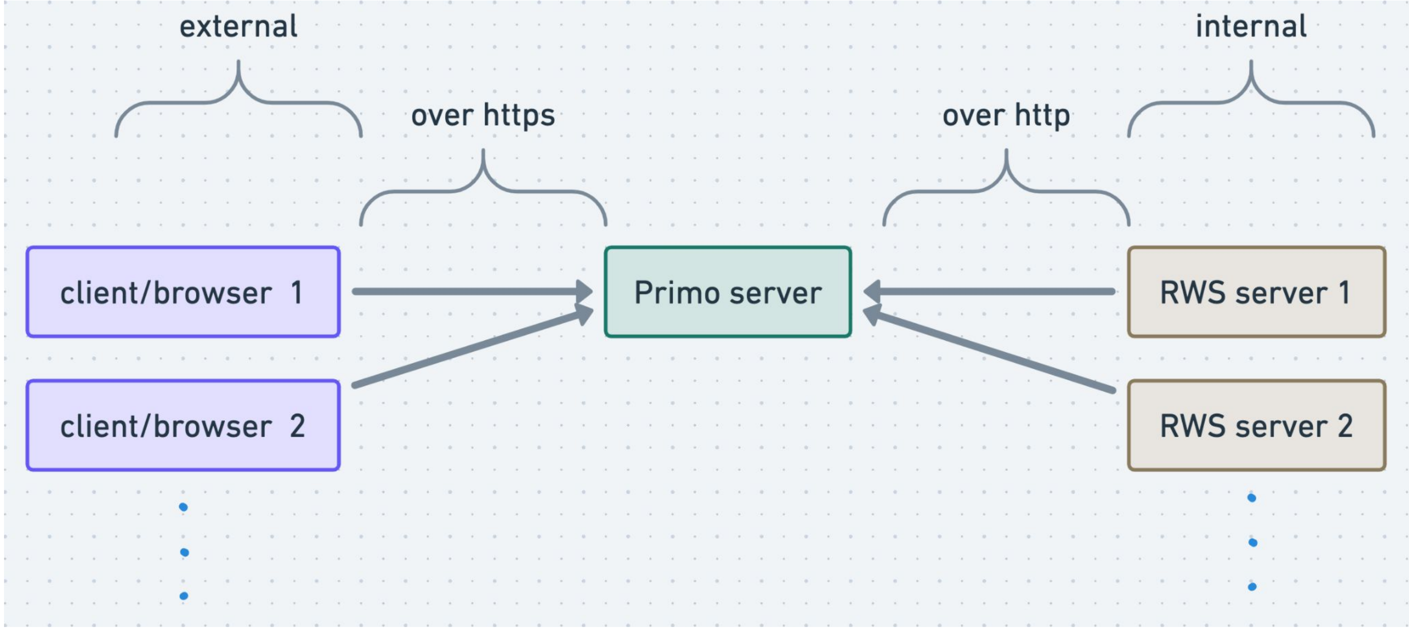
Having excellent processability, adhesive properties, good flexibility, and a simple process of synthesis, polyester enamels have been the most widely used for over 50 years. In 2008, the production of polyester enamelled copper wire was reported to be at least 50,000 tonnes a year or equivalent to 15% of the total enamelled wire in the world (Bhanu, 2008). Currently, it is estimated that the end-of-life polyester enamelled copper wire has reached millions of tonnes (Ma et al., 2023). Findings by Ma et al. (2023) showed that waste enamelled copper wire may contain 3.0 wt% polyester which translates to at least 30,000 tonnes of polyester needing to be recycled for every million tonne of waste enamelled copper wire. Recycling 30,000 tonnes of polyester from waste enamelled copper wire, taking into account the embodied carbon of manufacturing polyester which is 32 kg CO_2 eq/kg (Kilgore, 2023), saves at least 960,000 tonnes of CO_2 eq. Due to the enamel being closely attached to the copper wire and difficult to remove, recent studies to recycle waste enamelled copper wire only focus on thermal treatment. The thermal treatment proposed involves high temperature or pyrolysis under an inert atmosphere with temperatures ranging from 500 °C to 900 °C (Li et al., 2023, Liu et al., 2020) or directly applying a voltage to heat the copper wire above the decomposition temperature

[*] Corresponding author.
E-mail addresses: s.d.widijatmoko@bham.ac.uk (S.D. Widijatmoko), g.a.leeke@bham.ac.uk (G.A. Leeke).

UNCORRECTED PROOF

Document: resenv100143-primo / Article / Head / Abstract / Section title / Text | Primo version: 0.3.1+

Overview of runtime parties



Runtime overview (cont.)

- Users connect to Primo server from Internet
- RWS servers connect to Primo server from internal network
- RWS servers keep asking Primo server for “work”
- “Work” means resource-demanding tasks:
    - PDF compilation (using XeTeX)
    - MathML alt. image creation (using XeTeX)
    - XML validation (using a validation tool)
- RWS servers can be easily added to scale

- Language: Scala 2, Scala.js (transpiles Scala to JavaScript)
- IDE: IntelliJ IDEA
- VCS: Git
- Build tool: sbt
- Libraries (Java, Scala):
  - DB: SQLite
  - HTTP: Undertow
  - DOM: scalajs-dom
  - SFTP: apache Mina sshd/sftp server
  - Text index: Lucene

Technology (cont.)

- Front-end:
    - VDL: our own framework for developing UI components
    - CSS: written in Scala, translate to pre-css then to css via Tailwind CSS
    - Fonts: STIX (body text), Source Sans VF3 (UI), Courier Prime (fixed width)
    - Icons: RemixIcons plus self-made icons using Inkscape

Codebase size

- Scala source code: ~3000 files, ~6MB total
- Git commits: ~6000 commits currently
- Almost everything is written in Scala

- Primo codes consist of many modules
- Most modules have 3 submodules:
  - **js**: codes for Scala.js only
  - **jvm**: codes for JVM only
  - **shared**: codes for both
- Scala.js translates codes in **shared** and **js** to JavaScript, so codes in **shared** can be used for both JS and JVM

Some important modules

- **admin**: administration UI
- **demo**: demo & test of UI components
- **drive**: file management UI, similar to Google Drive
- **editor**: editor UI
- **gallery**: gallery of UI components
- **logon**: UI for login/signup

Example: the **admin** module will have this file structure

```
admin/
  js/src/main/scala/io/trivic/primo/admin
    // code compiled to JS and run on client
  jvm/src/main/scala/io/trivic/primo/admin
     // code that run on server
  shared/src/main/scala/io/trivic/primo/admin
    // code that run on both client and server
```

Document model - Structure

- Document = Tree of nodes with unique IDs
- Nodes can reference each other, so it’s actually a graph
- But core structure is basically a tree

Document model - Node types

- Node with multiple dynamic sub-nodes (`NodeWithSubNodes`)
- Node with a single dynamic sub-node (`NodeWithSubNode`)
- Node with fixed, pre-created child nodes (`NodeWithChildren`)
- Special node holding a text-string (`TextPart`)

- Nodes have attributes called facets
- Facet value can only be set/removed
- No sub-structure of facet value
- Facet types can be non-primitive but lack sub-structure

Document model - Node hierarchy

- Node types: Derived from base types (called traits in Scala)
- Nodes follow a specific DTD, derived from a DTD used by a large publisher.
- This DTD is very complete and covers everything that we need
- Initially we tried to create a generic set of nodes to match any DTDs
- Found it infeasible due to specific and extensive functionality required

Document model - Node hierarchy (cont.)

- Porting to another DTD needs specific nodes, view classes, and functionalities.
- Core collaborative editing remains unaffected

Document model - Observation and Binding

- The document model supports observation
- Observer can be registered for direct sub-node changes (insertions, deletions), for facet changes; The observer is specific (we know what changed)
- "Deep-change" observation also possible, for any change below a node no matter how deep; The observer is generic (something changed)
- Observers connect the model to UI view classes
- UI updates with model changes

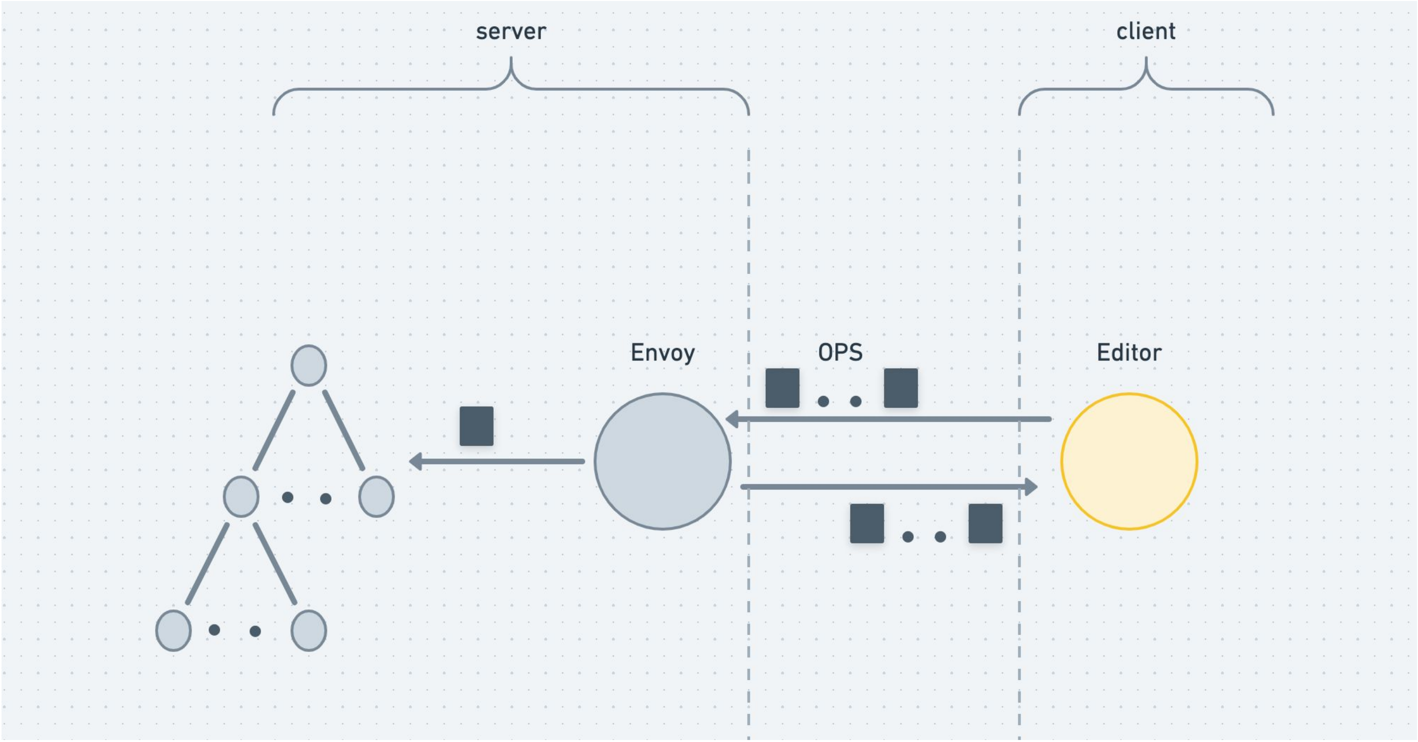
Collaboration – Operations

- Changes made to document are represented as operations (Insert, Delete, Move, SetFacet)
- Operations are sent to/from server asynchronously
- Clients keep fetching operations from server even when no edits take place

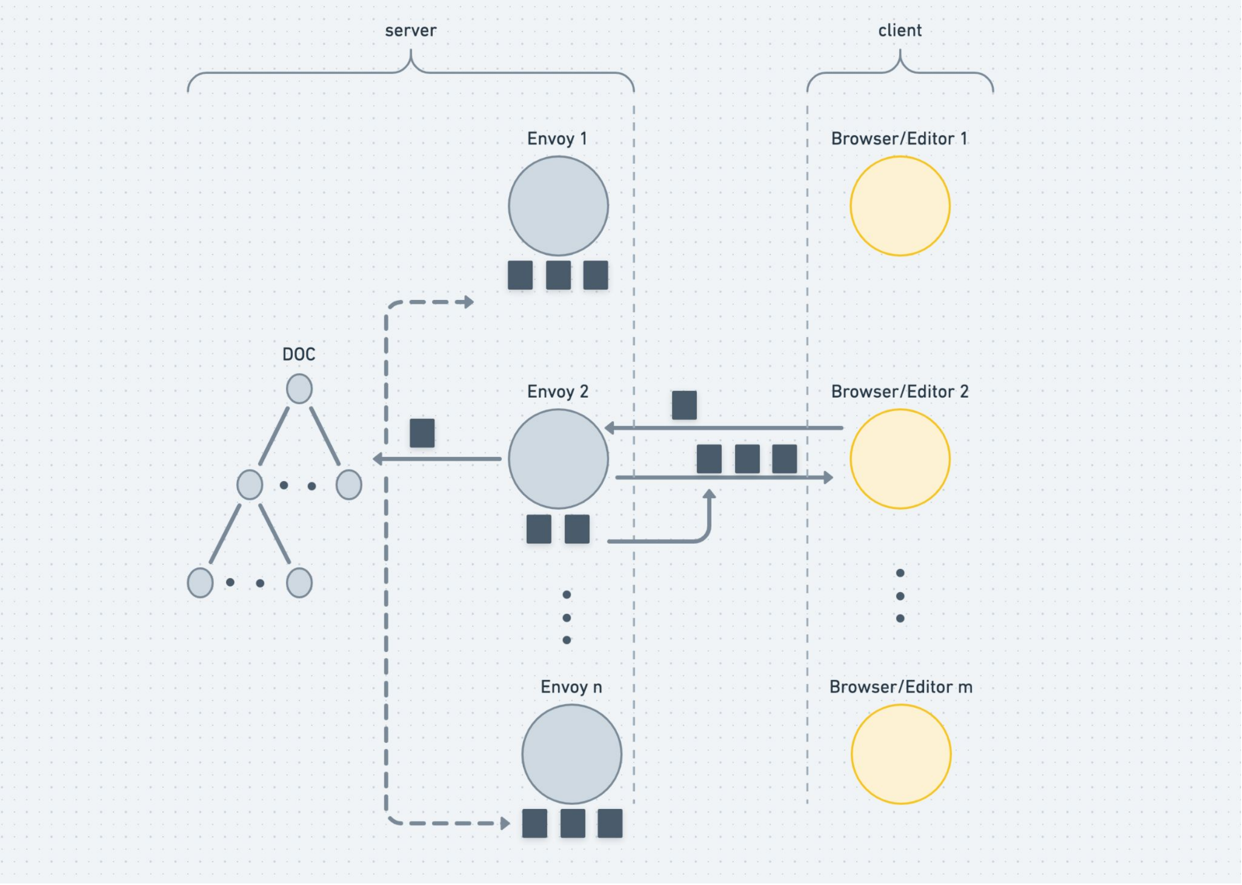
Collaboration – Operations

Operations in Scala code (simplified):

```
case class Insert(targetId, offset, cargo)

case class Delete(sourceId, range, cargo)

case class Move(sourceId, range, targetId, offset, cargo)

case class SetFacet(nodeId, facet, change)
```


server
client
Envoy
OPS
Editor

Collaboration – Operations



- When multiple users edit the same document simultaneously, there might be conflicting changes that must be resolved

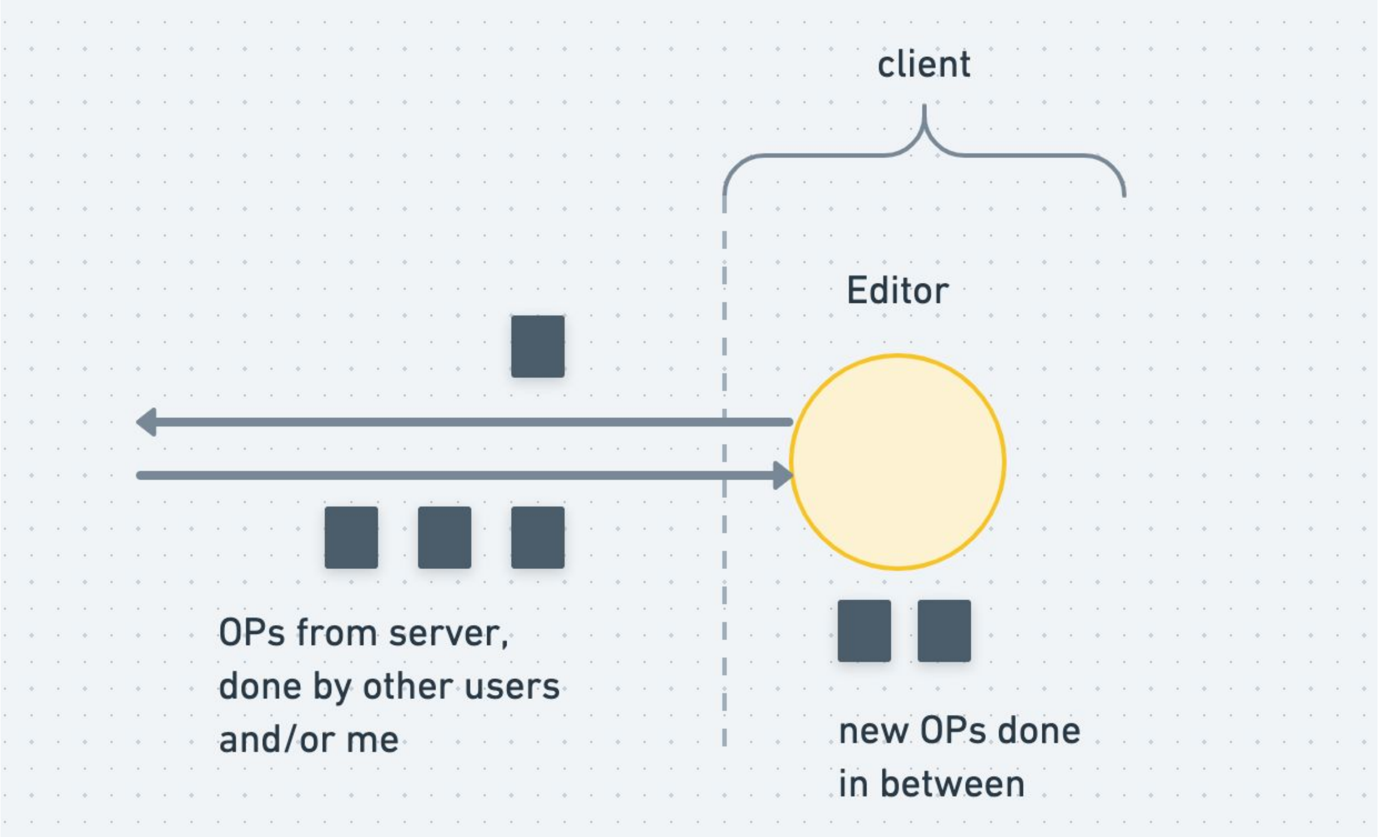
- When a client A fetches the operations (done by other clients, e.g. client B), it must apply those ops to the document model

- At this time, there might be some new operations made on client A that have been already applied to document model (on client A) and updated the document view

Collaboration (cont.)

- so we have incoming changes from the server (already accepted), and we have our own changes, some of them just sent/recv to/from the server, and some of them not yet sent to the server, but all those are applied to the document model, and reflected in the view (view was updated via observers)

- there are different ways to solve this problem; ours is to use a technique called TriLayer(s)


client
Editor
OPs from server,
done by other users
and/or me
new OPs done
in between

Collaboration – TriLayer (cont.)

- Document model tree is internally "triplicated"
- TriLayer: Enumeration of 3 layers
  - L0: Main layer, UI observes and updates the document view
  - L1: Server layer, holds latest server version of the document
  - L2: Temporary layer, used for new changes on the client

Collaboration – Rebase changes

How TriLayer works (simplified):

- Changes fetched from the server are applied to layer L1
- Layer L1 is transferred to layer L2
- New changes on client (not sent to server yet) are rebased against the latest server changes, then applied to layer L2
- Finally, layer L2 is transferred to the main layer L0, and the document view is updated (via observers)

Questions & Answers