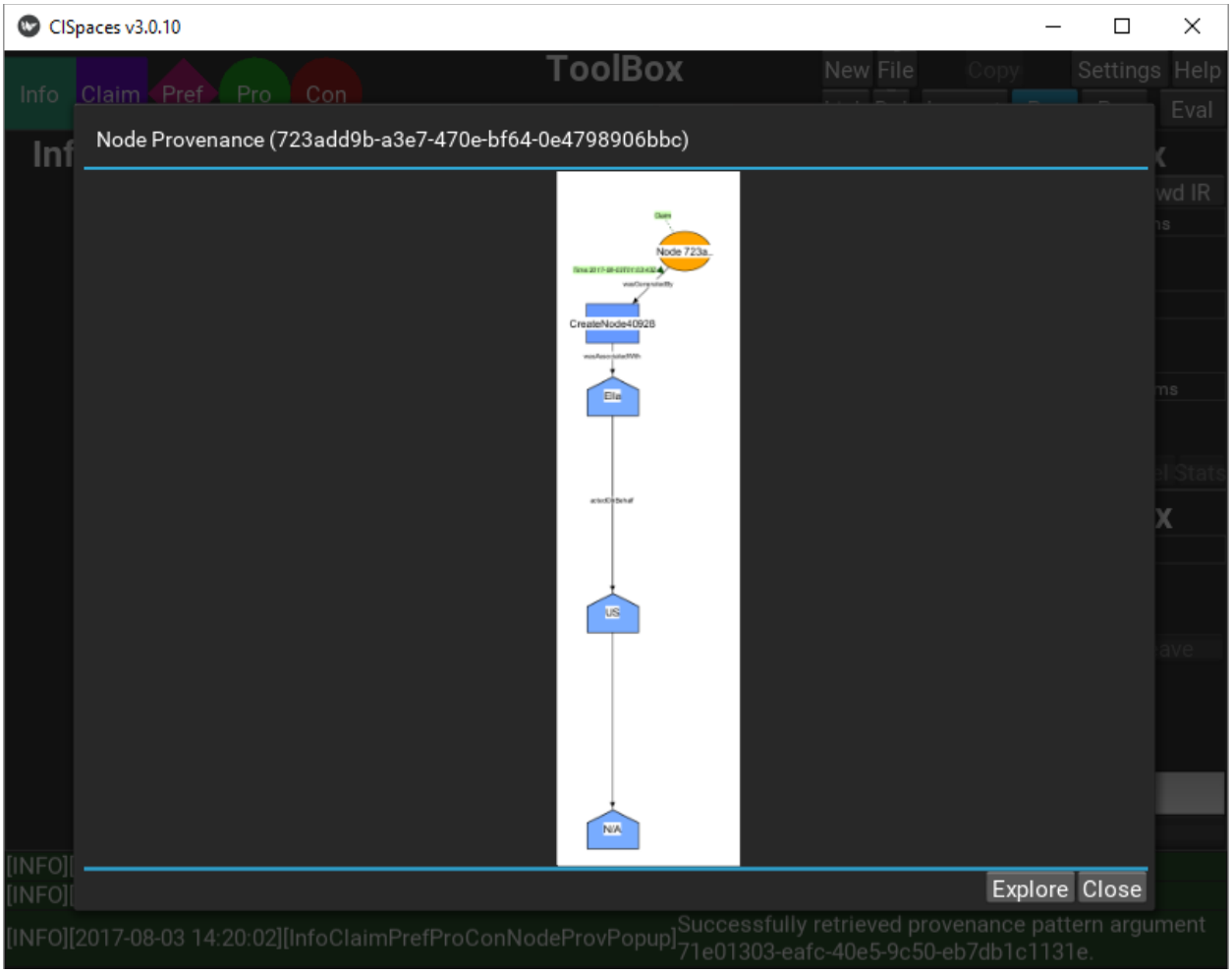
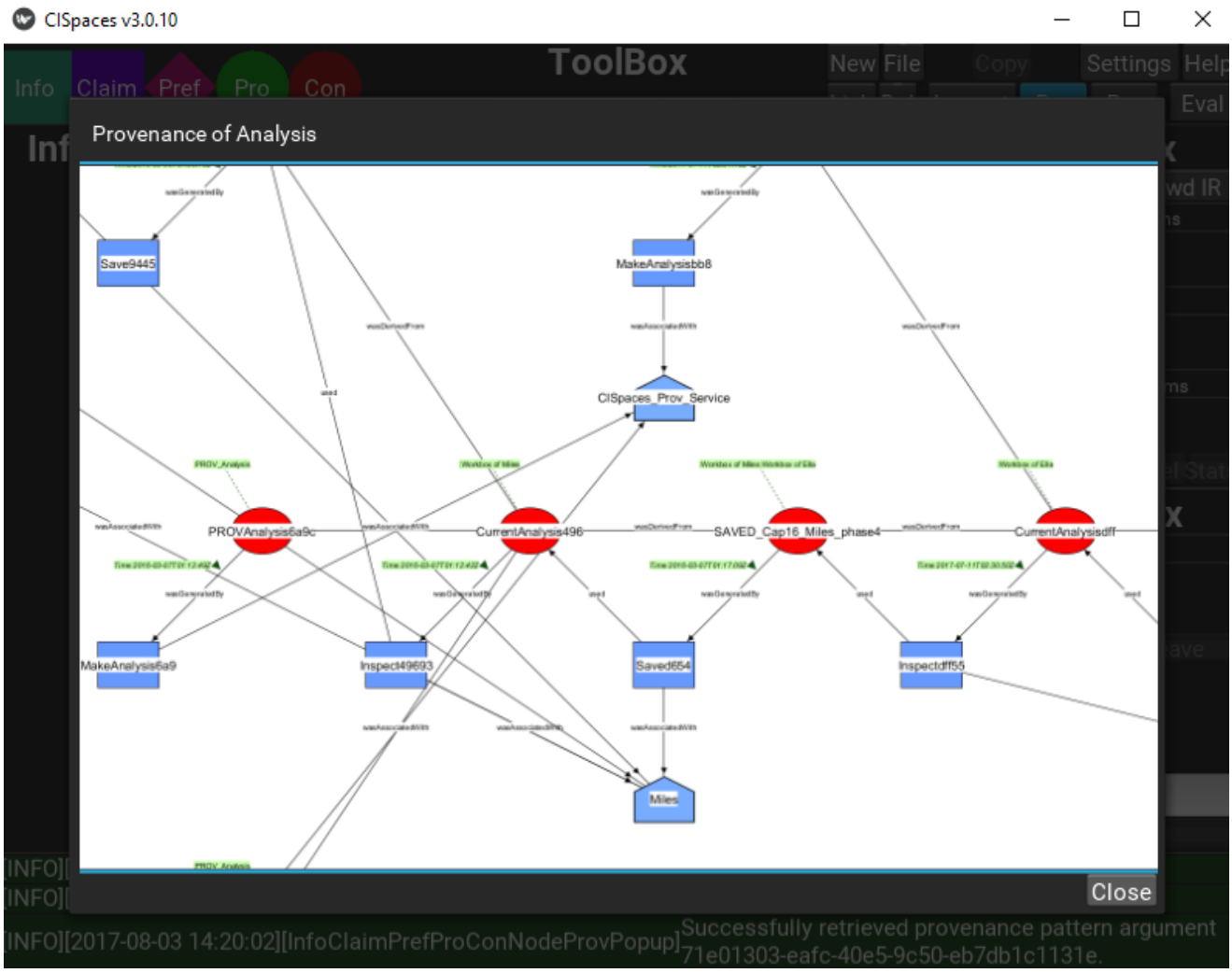
Provenance Service

Attempts to implement the W3C Provenance Standard. The primary purpose of tracing the provenance of an object or entity is often to provide contextual and circumstantial evidence for its original production or discovery, by establishing, as far as practicable, its later history, especially the sequences of its formal ownership, custody, and places of storage. This web service records the provenance of the nodes and the provenance of analyses within the application – who created them, the time of their creation, the textual information within the nodes, etc. The provenance of a node in the application looks like this:



This visualization shows how nodes and actions related to them are linked together. This service produces some sort of historical data providing a trace to what happened with nodes in the process of analysis.

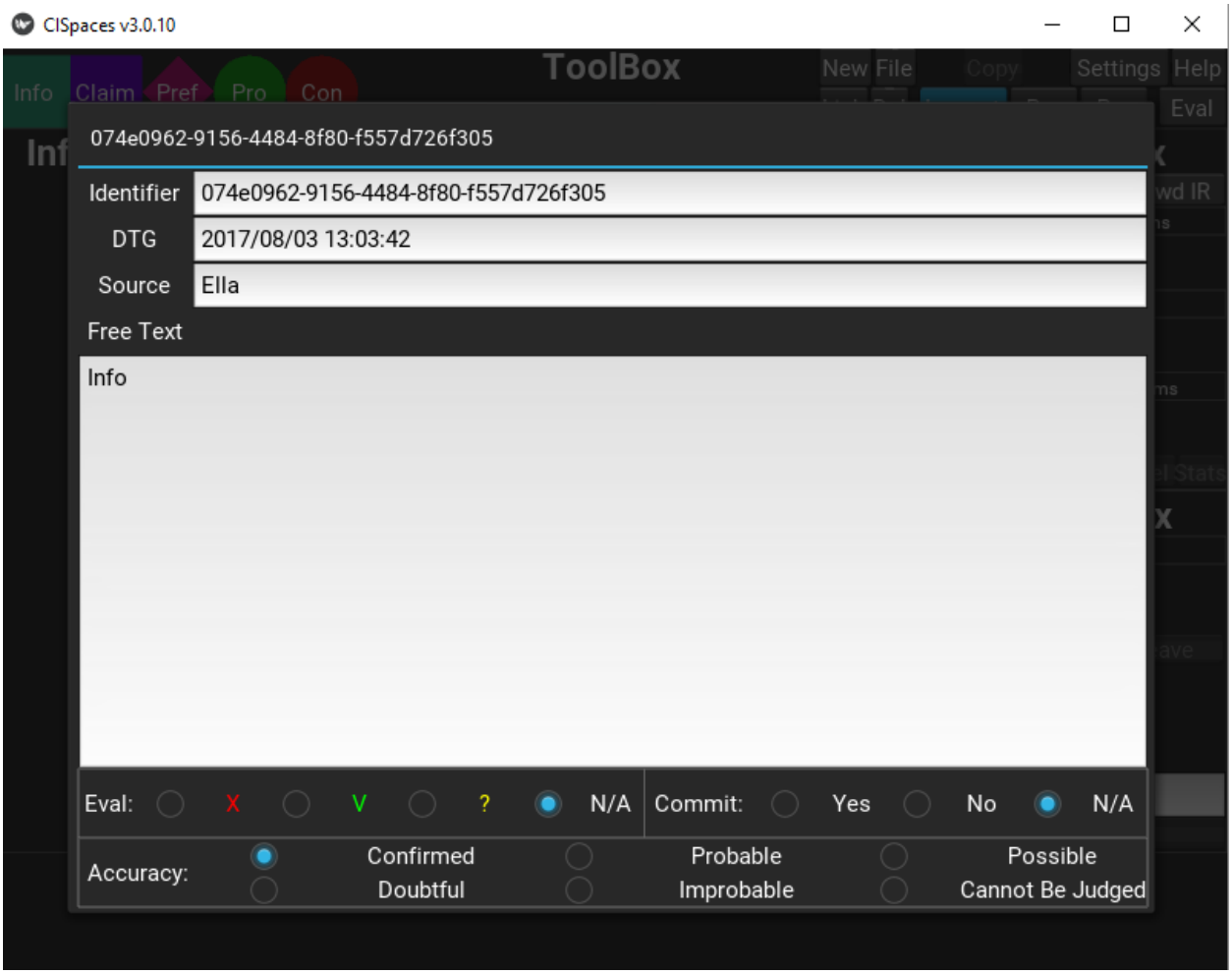
The application also displays the provenance of the whole analysis which would look something like this:



The provenance service logs every action within the current analysis and links them together in a specific, rather complicated way. The purpose of this is that every single event that happened is historically recorded and can be traced back to its original source by following the links between nodes. Everything has a unique ID which means it can be referenced from external sources via this unique ID.

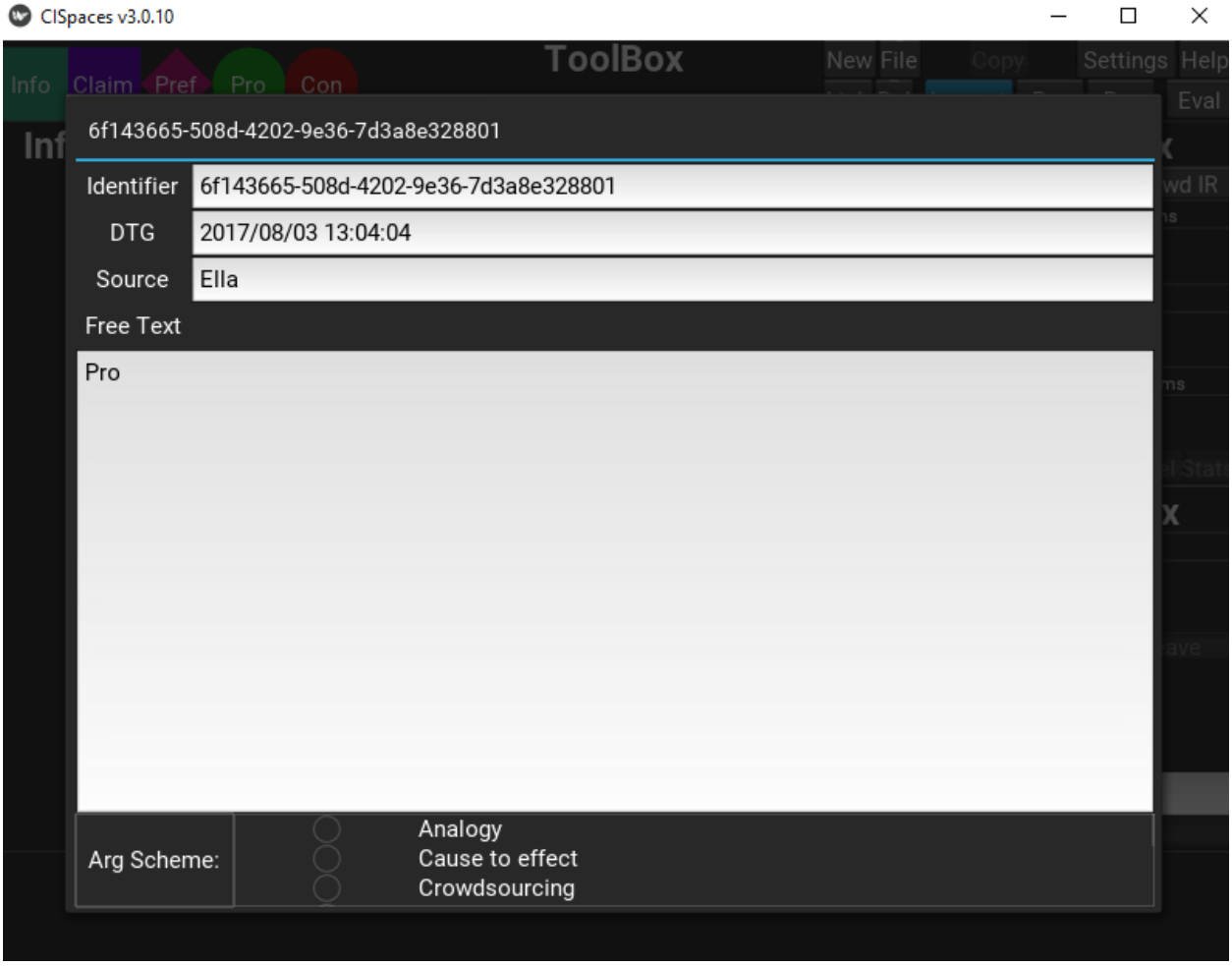
Evidential Reasoning Service (ERS)

This web service is responsible for analyzing the graphs created in the application. (These are not the graphs in the pictures above. The ones above are just graphs showing the provenance of analysis. The program allows analysts to create graphs that link hypotheses together to understand why a certain event occurred). Clicking on any Info or Claim node opens this popup:



You can see that each node has a unique identifier, a timestamp (DTG), and a source. Each Info or Claim node also carries a textual data which introduces a new piece of information to the analysis. The text in a node is NOT evaluated by the ERS! The evidential reasoning service is using the metadata about nodes such as the Accuracy of Info and Claim nodes, the argumentation scheme of the PRO or CON links between Info and Claim nodes, etc. This data is used to evaluate the analysis. Every linking node (PRO or CON) is connecting nodes based on the rules of some argumentation scheme.

Clicking on a linking node (PRO or CON) opens a similar popup to this one:



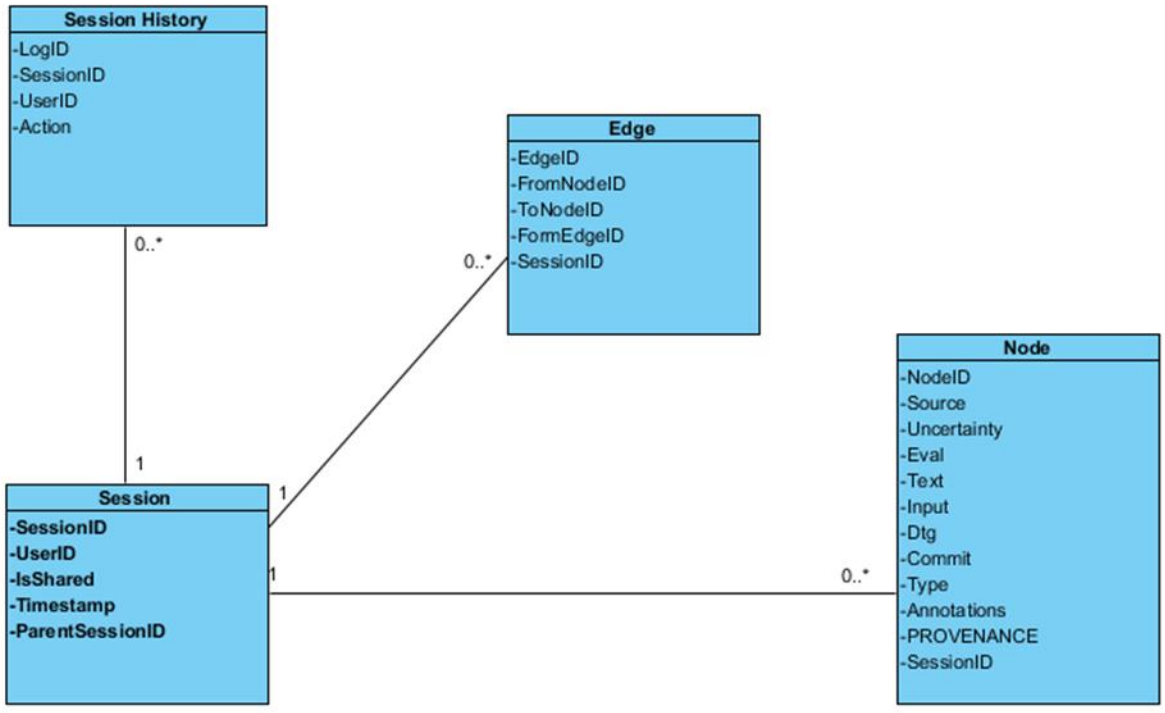
You can see that all linking nodes can have different argumentation schemes. If you link a Claim node to an Info node via a PRO link node which has the argumentation scheme “Cause to effect”, that would mean that the claim was made because the analyst knows that the information in the Info node caused the thing that was claimed to happen.

The ERS works solely based on the structure of the graph, it does not use the textual information inside the nodes. It uses what I call the metadata about nodes and how they are linked together. The ERS is built based on logical rules which use assumptions, premises, etc. This is web service requires a certain level of expertise in logical analysis. I have taken this service for granted, and have not considered it in detail.

Version Control Service (VC)

One of the core functionalities of the CISpaces allows different analyst to work collaboratively on the same analysis remotely. Naturally, this introduces the need of version control. In the previous application changes to a shared graph were happening at runtime – they used a distributed messaging service called ZeroMQ. Clients would subscribe to the service and receive all changes to the shared graph in the form of JSON strings via publish requests. These changes were happening instantly after an analyst edited the analysis. In the new application, we are looking to improve this by building a system like git. Users checkout an analysis, work on it in their personal workspace, and then commit it to the shared space. We have established a set of rules under which this collaboration will be possible.

The Version Control is the web service responsible for recording the versions of analyses in the database. It forms the backbone of the whole application and controls the management of different analysis both in a personal workspace and in shared workspaces. Every action on the interface level that changes the analysis will cause an HTTP request to be sent to this web service, which will record the changes in the database. It stores all edges and nodes, along with all the information about them necessary for the other web services.



This is the part of the DB schema responsible for the version control

Back-End Model: Database Schema Design

The main entity which enables the support of versioning is the Session. A new session is created every time a user creates a new workspace, or when someone creates a new collaborative space. A unique workspace/collaborative space has a unique session ID. A session is associated with the user ID of the user who created it in the case of a personal workspace, and in the case of a shared space it is associated with all users who are contributors within the shared space. All sessions are timestamped. The IsShared flag indicates whether a session is linked to an independent single-user workspace or a shared space.

All user actions are recorded in the Action field (text) inside the Session History table. Adding, deleting nodes, changing the text on a node, linking nodes, etc. Every action has a separate LogID and is linked to the UserID of the user who made that change.

Every session is linked to the nodes and edges in it. The Nodes table records all relevant information about nodes in a session. Similarly, all edges are recorded in the Edge table. Changing a node will be supported by UPDATE queries in the database. Deleting nodes and edges will execute a DELETE query on the respective entity.

Regarding the information about Nodes: all the fields in the Nodes table will be sent to the VC web service in a JSON format, except for the PROVENANCE and the Session ID which is recorded upon creating a node. The PROVENANCE field will be dealt with by the PROVENANCE web service. This web service will be responsible for populating the provenance of a node when it is created. For now, the annotations are stored in a string in the database. This may be changed later depending on how these annotations are used.

For now, only the primary keys are indicated. The foreign keys are not because of legacy from the previous application. If we want to enforce referential integrity, we may decide to put foreign keys in place. The Primary Keys are:

Session Table: SessionID, UserID (Can have ParentSessionID as a FK)

Session History Table: LogID (Can have SessionID, UserID as a FK)

Node Table: NodeID (Can have SessionID, UserID as a FK)

Edge Table: EdgeID (Can have SessionID, UserID as a FK)

The ParentSessionID links a session to a session from which it might have emerged. This can happen if a user chooses to ‘branch out’ from a (shared) analysis and work on their own analysis. This will create a tree structure of analyses. This user can then choose to put this new analysis back in the shared space. But what happens if three users check out the same analysis, work on it, and then all try to submit their latest version in the same shared space? See the Rules for Collaborative Analysis below.

Rules for Collaborative Analysis

There are two types of collaboration. They differ in the way updates happen at the user end.

Suppose you have joined a shared space for analysis, along with Bob and Alice. You all want to work on the same analysis, but you don't necessarily want to see all updates coming from Alice and Bob's ends occurring at runtime. Instead, you want to have a more seamless type of collaboration where you make your changes, submit them and the application handles the merging of versions between yours, Alice's and Bob's.

1. Locked mode (exclusive lock): You can choose to check out the shared analysis and put it in your personal work space by locking it. This will not allow for Bod and Alice to edit the analysis graph in any way until you choose to submit your changes or unlock the graph. This mechanism should come with a lock expiration time in case someone never unlocks the analysis.
2. Semi-collaborative (shared lock): When you check out the shared analysis, you can disallow Bob and Alice to make changes to the existing graph by changing nodes or edges, or deleting them. In other words, Bob and Alice can only add new things to the graph, but cannot modify the existing structure. The app will run a merging algorithm for yours, Bob's and Alice's versions and put them all in the shared space analysis.

The following things need to be taken into consideration at this stage of implementation:

- How to send HTTP requests from the JS Clients to the server. Each web service handles POST requests with different JSON inputs. The web services are implemented in Java and use annotations to mark the type of requests being processed. Java servlets are used to respond to the different types of requests.
- Only the part of the UI which allows users to add, edit, delete nodes has been fully developed. The rest of the components on the interface are not linked to the back-end yet. The middle layer between the front-end and the back-end does not exist yet.
- A timeout mechanism needs to be developed for the shared space. In the case where someone checks out a shared analysis by requesting an exclusive lock and never commits it back to the shared space, the other analysts will be left waiting for them to commit forever. Therefore, we need to implement a locking system which allows users to hold locks for a finite amount of time only.