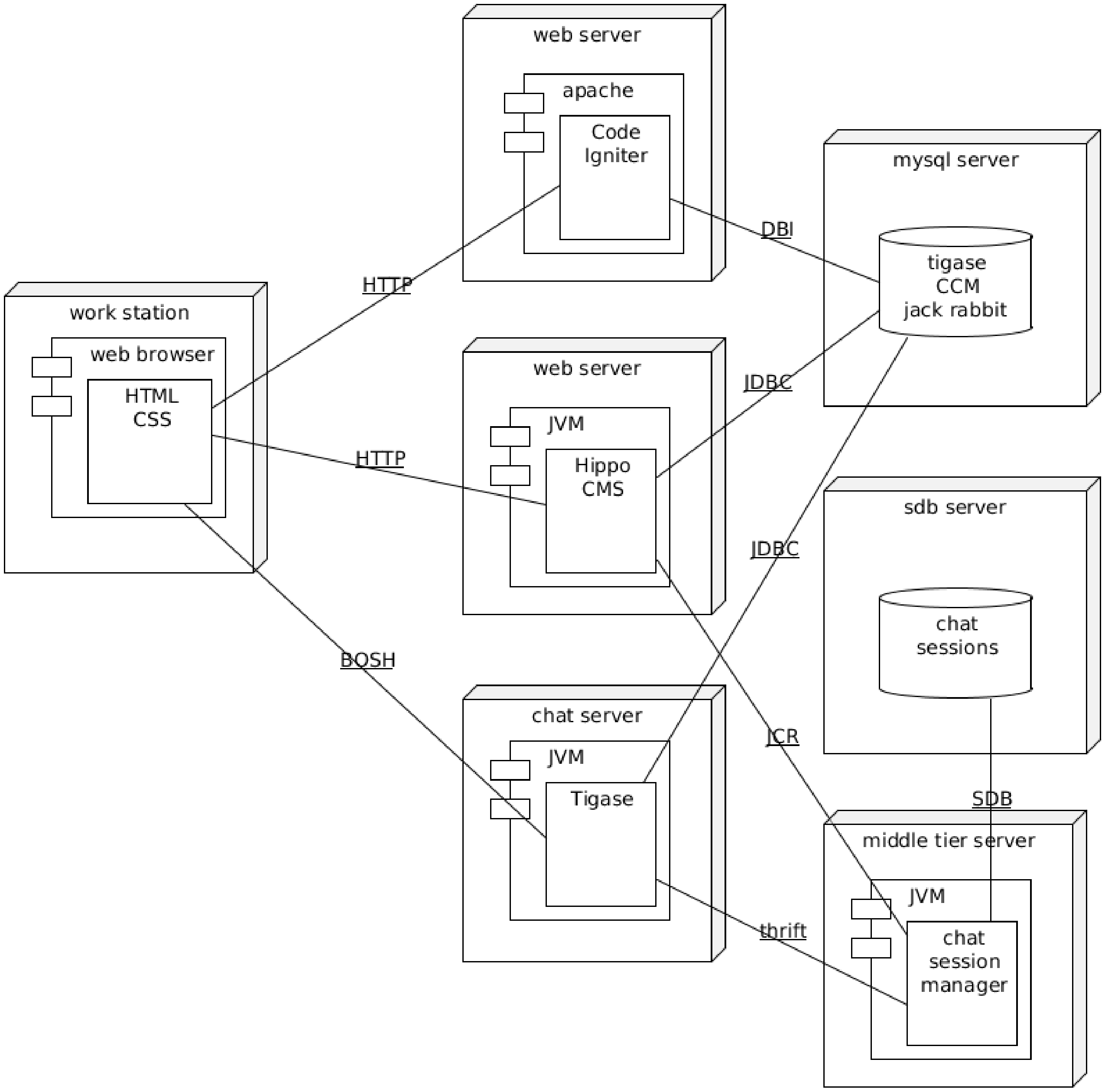
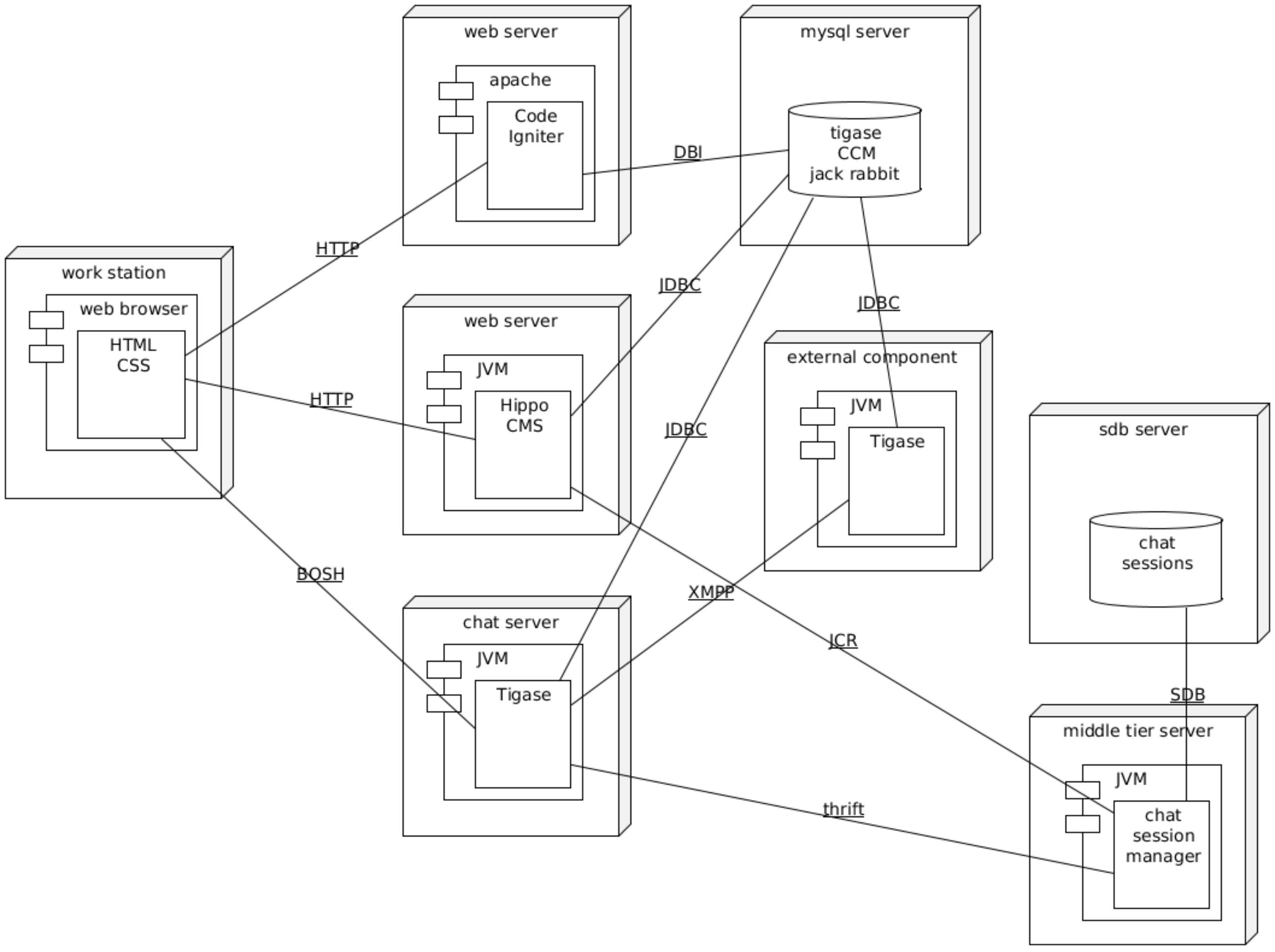
Towards a Software Architecture for Real-Time Communications

Recently, I released the second beta to an experiment called Conversational Content Management in which an actionable summary gets published by a software agent listening in on a chat based group discussion. Here is how the first beta was deployed.



- The dashboard was an old school forms based PHP application written on top of the Code Igniter framework.
- The user starts the XMPP session once they entered into a chat room.
- Strophe is the BOSH web client technology and Tigase is the XMPP server technology.
- Chat session state is managed by a middle tier component using Amazon's Simple DB as the data store.
- Once the discussion has finished, the chat reporter would publish the summary as a document in an instance of Hippo CMS which anyone could access (not just the topic participants).

The software architecture for the second beta looks similar but has been upgraded for higher scalability.



- What the web server delivers for the dashboard experience is mostly static content only. That includes a lot of javascript, CSS, and a little HTML for good measure.
- The Code Igniter app is used only for new user registration and email validation.

- The user immediately signs in to the chat server.
- The dashboard is a pure web 2.0 app with no new page loads once the user has signed in.
- All requests for data (e.g. previously created topics and invitations and creating new topics and invitations) get routed by the chat server to external component processors.
- Both chat and external component servers can be easily clustered.
- Google closure is the javascript library used to make the dashboard web 2.0 compliant.

Advantages to using BOSH instead of AJAX or Page Reloading

- Leverage XMPP authentication instead of having to build your own AJAX friendly OAuth.
- A unified approach to data interchange makes for a simpler client side application design.
- Though the client side javascript looks as simple as an AJAX call, requests and responses are actually batched in a sophisticated way yielding a more scalable application without increasing code complexity or noticeable retrieval latency.
- The commingling of chat with CRUD operations affords a more friendly user experience. Instead of forcing the user to make corrections in the form, the back end component can make its best guess then notify the user in a chat message of any changes. For the CCM Beta 2, the only thing the user has to enter is the email address of the person they are inviting. The component will make its best guess at topic name if it is missing or already in existence. The component will reschedule the meeting date and time if it has already occurred. The components lets the user know of any changes via the chat window. It also lets the user know if any errors occurred when emailing the invitee. This is more friendly than constantly telling the user what they did wrong then making them resubmit the request or abandon in frustration.

Advantages to using Java as the middle tier instead of PHP

- Java's first class, built in support for multi-threading allows you to get more work out of each server than what PHP code can achieve. This means fewer servers with a higher number of users per server.
- Tigase's clustering infrastructure for both chat and external component servers nets a highly scalable multi-processor approach that doesn't depend on the component developer having a lot of expertise in distributed computing.

Most Efficacious Scenarios for Adopting this Architecture

- When your application requires real-time communications features such as chat, presence, toasts, and other push notifications.
- When your application needs to be more scalable than a classical 3-tier architecture can provide including more graceful ways to degrade from network outages via traffic shaping.
- When a pure web 2.0 application is required and there is concern about connection throttling at the client.