Reefknot: Client Server Architecture

Kirrily "Skud" Robert

e-smith, inc.

1. Introduction

This paper outlines a possible architecture for Reefknot calendaring and scheduling clients and servers.

2. Explanation of examples

The examples given in this paper are based on the accompanying diagram, `client-server-arch.dia`, in dia format.

This diagram depicts a situation which can be described as follows:

- Foo Company (foo.com) is an organisation of moderate size with some network infrastructure. It has its own mail server, and its own iCalendar server for the use of its staff, two of whom are A and B.
- Bar Company (bar.com) is similar to Foo Company. Its staff includes C and D.
- Baz.net is an ISP used by individuals and small businesses. Although it provides a mail server for its customers, it does not provide an iCalendar server.
- Users A, B, C, D and E use an iCalendar client (presumably any client capable of handling RFC 2445 objects). User F has no iCalendar client, just email.

This paper will attempt to explain what happens in a number of calendaring and scheduling situations situations involving these people and organisations. The situations we will look at include attempts to schedule events where:

- A invites B (within the same company with an iCal server)
- A invites C (different companies, both have iCal servers)
- A invites E (E has no iCal server)
- E invites A (E has no iCal server)
- A invites F (F has no iCal client or server)

3. Assumptions

It is assumed that each client regularly "syncs" with its local server, if a server is being used.

Handling scheduling conflicts is outside the scope of this document

It is assumed that all users' email clients (MUAs) are configured to open attachments of type text/calendar using their preferred iCalendar client (except for F, who doesn't have one).

4. Worked examples

4.1. A invites B

1. A's client software is configured to use foo.com's iCalendar server
2. A uses her client to create an event, list attendees, etc

3. A hits "OK" after setting up the event
4. A’s client records the event in its local database
5. A’s client generates an iCalendar object and sends it to foo.com’s iCalendar server
6. The event is stored in foo.com’s iCalendar server
7. When B next syncs, he receives notification of the event
8. B accepts (or declines) the invitation and clicks OK
9. If B accepts, the event is stored in the client’s local database
10. B’s client connects to foo.com’s iCalendar server and sends the response
11. The response is stored in foo.com’s iCalendar server
12. When A next syncs, she receives the response.
13. Final state: A and B both have the event in their local clients’ databases, and the foo.com iCalendar server is also aware of the event

4.2. A invites C

1. Steps 1-5 as for the previous example
2. The event is stored in foo.com’s iCalendar server under A’s name
3. However, foo.com’s iCalendar server recognises that c@bar.com is not a local user
4. foo.com’s iCalendar server generates an email with a text/calendar attachment and sends it to c@bar.com
5. C receives an email inviting her to an event with A
6. When C opens the attachment, it is loaded using C’s preferred iCalendar client
7. C accepts (or declines) the invitation and clicks OK
8. If C accepts, the event is stored in the client’s local database
9. C’s client connects to bar.com’s iCalendar server and sends the response

10. The event is recorded in bar.com's iCalendar server under C's name
11. The server recognises that a@foo.com is not a local user and generates an email with a text/calendar attachment, then sends it
12. A receives an email with the response
13. When A opens the attachment, it comes up in A's iCalendar client
14. A looks it over, clicks "OK, and it's recorded in A's client's local database
15. When A next syncs, the results are uploaded to foo.com's iCalendar server
16. Final state: A and C both have the event in their local clients' databases, and both the foo.com and bar.com iCalendar servers are aware of the event

4.3. A invites E

1. As above, up to the point that E accepts and clicks OK
2. Since E has no iCalendar server, E's iCalendar client directly generates an email with a text/calendar attachment and sends it back to a@foo.com via baz.net's mail server
3. A receives the email, and continues as for the previous example
4. Final state: A and E have the event in their local client database, and it also exists on foo.com's server

4.4. E invites A

1. E generates the event, which is sent to A via email
2. A acknowledges the invitation, clicks OK, and it goes up to foo.com's iCalendar server

3. foo.com’s iCalendar server realises that e@baz.net is not local and emails the response
4. E receives the response and clicks OK to save it locally
5. Final state: as for previous example

4.5. A invites F

1. As for "A invites E" up until the point that F receives the email
2. F has no client, so merely reads the textual part of the message
3. F responds manually by email to A
4. A reads the email, and notes the response manually in her iCalendar client
5. The results are sync’d up to foo.com’s server in due course
6. Final state: A has the event in her local client database and it exists in foo.com’s server. However, F has the event written down on a bit of paper or something.

5. Complications

5.1. Private events

If an event is marked as being private, it shouldn’t be stored in the iCalendar server, only in the local client’s database (?). Alternatively, it could be stored in the server but flagged as private. This might be a user-configurable option, eg: "Store private events locally only" vs "Store private events on server".

5.2. Direct connections between organisations' iCalendar servers

The situation of "A invites C" would be considerably simplified (particularly once they start negotiating free/busy times) if there were a trust relationship between foo.com and bar.com, so that the iCalendar servers could talk directly to each other.

The servers would need to identify each other securely, and there would need to be some kind of ACL system in place to configure which organisations are allowed to see or manipulate parts of an organisation's calendar. For instance, foo.com might want to allow or disallow bar.com from booking its meeting rooms.

6. Summary of required functionality

6.1. Reefknot clients

1. MUST provide a user interface for the creation of iCalendar objects
2. MUST parse iCalendar objects into a user-readable format
3. MUST be able to store a single user's calendar information in a local database
4. SHOULD be able to sync with servers
5. SHOULD be able to send email directly
6. but MUST be able to either sync or send email

6.2. Reefknot servers

1. MUST be able to store all its users' calendar information in a database

2. MUST explicitly know who its users are (no guessing based on domain name, because that's subject to weird human error and DoS attacks)
3. MUST be able to sync with clients
4. SHOULD be able to send email to external attendees
5. MAY be able to talk directly to other iCalendar servers