Address Books

Sebastian J. Bronner <waschlt@sbronner.com>
April 22, 2010 – Revision 1

Concept for a Relational Address Book

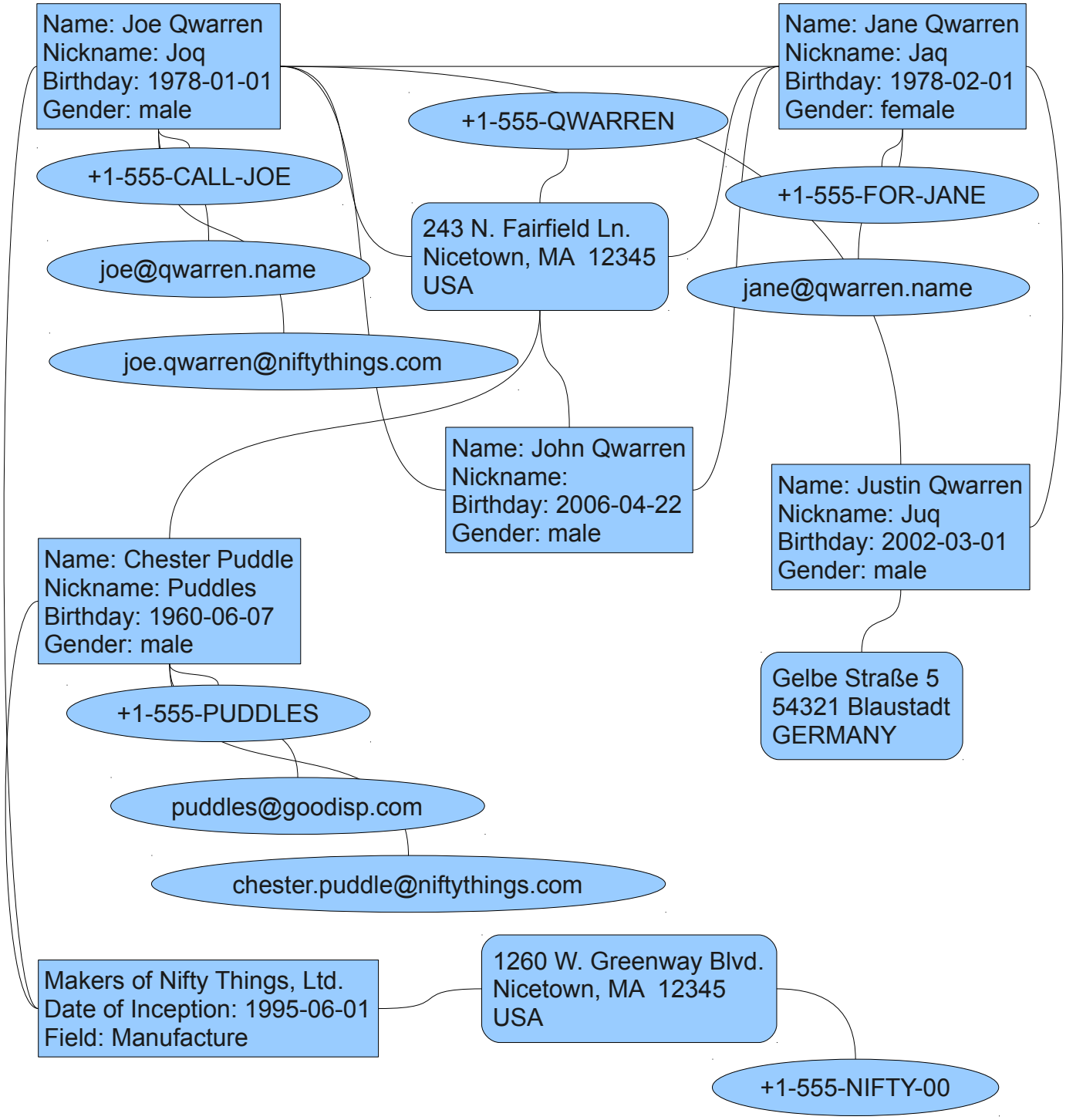
Address Books are about people. And about how to stay and get in touch with them. That should be simple. But the data should also be consistent. To realize both goals, the address database has to be complex. Here is the rationalization:

> Joe is married to Jane. They have a three-year-old child named John. Even though Joe and Jane share one address, together with John, they have different cell-phone numbers and e-mail addresses. So, you have a problem.
>
> If you make one entry for all three, you have to put notes next to the information that only pertains to one of them (i.e.: cell-phone Jane: +1-555-FOR-JANE). Try doing that with almost any existing address-book software, and you'll start going nuts pretty soon. It gets worse: sync your address book with your cell phone, then tell it to call Jane. Your cell phone, of course also has Joe's cell-phone number stored (cell-phone Joe: +1-555-CALL-JOE). How should your cell phone know that you want to call Jane's number, not Joe's? It's a nightmare.
>
> So, instead of making one entry for all three, you decide to make three entries, so that the situation always remains clear. Joe gets his own phone number entry, his e-mail address, and his postal address. Jane gets her own phone number entry, her e-mail address, and her postal address. But wait, it's exactly the same as Joe's. Never-mind, you tell yourself, gotta do it this way. So, you go on to type in John's entry. He's only three, so he doesn't have a cell-phone or e-mail account yet. But you type in the same address as his parents'. Alright, by the time you've got all your contacts in, you've been pretty busy. And now you're happy that everything is in order, your cell-phone doesn't get confused and call Joe when you want to call Jane, and so on.
>
> A month later, you get an e-mail from Joe. They're moving, and he wanted to let you know his new address. So, you open up Joe's contact in your address book, and update the address.
>
> As you're falling asleep, you realize that if Joe's moving, that means Jane's moving, too. Alright, you resolve to remember to update her contact in the morning, before you go to work. The next morning, while you're taking a shower, it hits you: you still have to update her contact information and re-synchronize your cell-phone before going off to work. So, you rush your shower and do just that.
>
> All's finally well. Three months later, you get a reminder from your address book to send John a post-card for his fourth birthday. With a single click, you call up his contact information and write his address on an envelope nearby that you will put the postcard in once you've finished it. The next day, the postcard is finished, you take it to the post office and mail it.

This story is supposed to illustrate the difficulty todays address book programs have with data management simplicity and data consistency. If you paid close attention, you'll realize that John's post-

card went to the old address. Thankfully, the post office offers forwarding service, so it'll reach him eventually, but that's not what you wanted. It is easy to imagine a scenario, where the consequences were more dire. Going for a visit, say. The solution to the problem presented here is the following:

Contact information is stored in a relational database containing the following objects: people, companies, addresses, telephone numbers, e-mail addresses, web sites, job information, and other, freely-definable, objects. The database links these objects using a whole slew of relationships. And this is where it gets complex. So, to make this understandable, the following diagram is an example of a possible database.



This database has 19 objects with 23 relationships (if I didn't lose count). They are (1) husband – wife, (2-3) father – son (twice), (4-5) mother – son (twice), (6-13) person – detail (includes any one-item data, like phone numbers and e-mail addresses) (eight times), (14-18) person – address

(five times), (19-20) address – detail (twice), (21-22) person – workplace (twice), and (23) company – address.

This for five people, one work place, and a little bit of detail. If this seems like it's getting too complicated for what it's worth, just remember the hassles you have keeping your contact data up to date and consistent the way things are now.

Notes on Implementation

In principle, this database can be set up with any relational database around, like MySQL or BerkeleyDB. If you're writing a program that will use the database through a binary API, BerkeleyDB will probably prove fastest. However, if your goal ist to remain flexible, the MySQL database will allow you to attach various frontends, including manual query easily.

I haven't given much thought to it, but perhaps, someone can see a benefit to using non-relational databases, like ZEO or Cassandra.

Any front-end implemented to manage this data has not only to display and modify the data in the objects, it must also be able to modify the relationships. For example, when John growes up and moves out of his parent's house, his relationship to the existing address has to be removed, and a new address created for him. Unless, of course, he moves in with his brother Justin in Germany, in which case only the relationship has to be reassigned.

The trick of the matter is that the user interface has to make it clear to the user what he's doing. So, when John moves away from his parent's house, it should be possible to open the contact data for John and just change his address information. That means that the user interface must make the user aware of the other people related to that object. This can be done either by asking whether the other people are also moving, or it can be done by displaying the people related to that address directly on the form. I, personally, would prefer the latter, because it gives you a better overview. The former option is kind of like magic.

Possibilities for Enhancement

The way I've presented the database, data can be kept up to date exactly how I outlined in the introductory part of this essay. However, it may be desirable to be able to see a history of changes as well. I am not covering this option here, because that will just compound one complexity with another. And I'll be happy if complexity of the solution presented here is ever implemented.

I'll just note a possible method for providing history as a starting point for such endeavors: Instead of seeing the relationships as permanent, and the objects as variable (meaning that everyone living together is always related to the same object, and when they move, the content in the object is changed), the relationships should be seen as variable and the objects as permanent. That means that an address never changes. When a person moves, a new object is created, and then the relationship is moved from the old address object to the new one. The history can be maintained by keeping a linked list of relationships with timestamps for the address, and other relevant objects. The parallelism inherent in telephone numbers and e-mail addresses (you tend to have several at once) will make the linked list somewhat cumbersome. Maybe there is a data structure that is better suited. But the idea is clear: keep the old links around with timestamps.

The Big Picture, the Way I See It

So, the question I ask myself for this section is: How do I want to access my data?

Well, the answer is many-fold. I want to access my contacts from anywhere using LDAP. That way any mail client, any LDAP-enabled device can spit out what I want to know with a minimum of configuration. I also want to manage my contacts using a conventional-type program designed for

just this purpose (a contact manager, like evolution, just better). And I want to be able to call up my contacts using a web-interface.

That means that very diverse applications need access to the same data store. That also means that a MySQL database is most probably the data store of choice. My home PC will access the database online, but also keep a synchronized copy for work while disconnected from the internet. That is especially important for my laptop. Of course, I need to be able to call my ISP when my internet connection goes down, so it's important for my home PC as well. Then, I'll need an good LDAP scheme, possible with an intermediate connector (data conversion script) so that the relational data can be processed by the hierarchical LDAP (ugh). And finally, this one should be easiest, PHP access to the database.

Other big pictures will have other requirements as to how to put it all together. I'd be interested to know of them.