The Five-of-Eight Fallacies of Context-Aware Prototyping

Mika Raento

Helsinki Institute for Information Technology
Department of Computer Science, University of Helsinki
mraento@cs.helsinki.fi

Introduction

Peter Deutsch summarized experiences in building real distributed systems into the *Eight Fallacies of Distributed Computing* [1]:

> Essentially everyone, when they first build a distributed application, makes the following eight assumptions. All prove to be false in the long run and all cause big trouble and painful learning experiences.
> 1. The network is reliable
> 2. Latency is zero
> 3. Bandwidth is infinite
> 4. The network is secure
> 5. Topology doesn't change
> 6. There is one administrator
> 7. Transport cost is zero
> 8. The network is homogeneous

When building real systems we often start with a *prototype* (this method was immortalized by Frederick Brooks [2] as "Plan to throw one away; you will, anyhow"). The idea is that the prototype will be much easier to construct, often by ignoring at least some of the eight issues above, while giving close estimates on some other properties of the system, such as performance requirements, usability or applicable technologies.

Context-aware computing [3] is still in its infancy. I would claim that basically all systems we are building today that we call context-aware are prototypes. Since they are prototypes, we assume that we too can ignore the eight issues above. But that assumption proves to be false. Context-aware systems are meant to be used throughout people's everyday lives. That means they are not the classical prototypes: used under the supervision of developers, and in settings limited both in space and time. In many ways they must be much nearer real systems than prototypes of other programs.

This paper looks at each of the eight fallacies in the light of building context-aware prototypes. We claim that we can ignore three of them, mostly with the reasoning that they haven't hit us yet. The remaining five are discussed in detail, relying on experience from building the ContextPhone context-aware prototyping platform for smartphones [4].

1 The network is reliable

In the prototype lab you can make the network fairly reliable, and more importantly you, the developer, are there to aid when the network isn't. This isn't the case when you give users a device to take home with them. The reliability should be seen as an end-to-end quality [5] of the system, including components built by you or provided by the platforms used.

In ContextPhone we built code that recovered from routine network failures—dropped connections, unresponsive servers and crashes of components. What we didn't do was to take into account how the failure of the network affected the user: media published would take unpredictable time to arrive on the server, the phone might unexplicably try to connect to the network all the time, and information about others in the presence service [6] was not up-to-date without the user necessarily knowing that. We failed to keep the user informed of the consequences of the unreliability, to show the seams of the technology [7] by assuming that faults would be too rare to bother the user about.

2 Latency is zero

Latency can be ignored in the laboratory. Users can be talked to while they are performing actions. Latency-dependant operations like clock-synchronization can be done manually, or hidden by the prototype software. Real context-aware systems are supposed to work with varying networks (Bluetooth, Wi-Fi, GPRS and various and sundry ad-hoc networks) with varying and often large latency.

Uncharacteristically, we use an example from another project, Nokia's Multi-User Publishing Environment (MUPE) [8]. MUPE is meant to be a rapid-prototyping platform for social, mobile and context-aware games and collaborations. In our experience at HIIT, it fails in large categories of these, since it assumes that users can wait for server round-trips with most actions. This proves not to be the case in the field.

With ContextPhone we have assumed that a message takes such a short time to arrive that the message's timeliness can be measured looking at the time the message was sent. To prove this false in a GPRS-connected device is left as an exercise to the reader.

3 ~~Bandwidth is infinite~~

Luckily, even lab-testing on real devices tends to show actual bandwith and I think the fallacy is not committed very often.

4 The network is secure

The network in the laboratory can often be made "secure enough", or the data limited to a safe subset. The prototype doesn't have to implement encryption, signing or secure logins. This is lucky, since the devices and platforms we use, like Symbian [9] don't provide these primitives for our applications either.

When we hand out a context-aware device to a user to be used in their private lives, the context data becomes a liability. Location, social encounters, phone calls and activities are personal data in the meaning of both data protection legislation [10] and perception of the users. Suddenly we must provide strong encryption to be fair and legal. We must confess that ContextPhone still lacks the necessary primitives, and that we have had to convince our users that the risk is small enough in the hope they don't ever make a fuss about it.

5 Topology doesn't change

You set up your test-and-prototype network and show the prototype to users. There is no need to change the setup while the prototype runs. If you should need another topology (say a server crashes) you recode the necessary bits. Since all of the devices running the software are near, this is not a great effort.

We give ContextPhone-based applications to users for 1–3 months at a time. If a server crashes, we can't just reload a new version of the software on the phones. We currently have, hah, a prototype of remote settings, but this needs a connection to a server, ending in a nice catch-22. A context-aware prototype needs the ability to either automatically reconfigure itself when changes in the network happen, or multiple independent ways of remote reconfiguration. Both take a non-trivial amount of time to implement.

6 ~~There is one administrator~~

The organizational setting of the prototyping process doesn't change as much with context-aware systems as other aspects. We can still assume, that there is one entity responsible for running the system.

7 Transport cost is zero

You can provide your own network in the lab. It might not be completely free, but you can often amortize it in a way that the amount transmitted by the prototype doesn't seem to increase the cost. You can trade amount of data transmitted against more complex implementation (caching, distributed data, compression, rate-limiting etc.).

We actually implemented rate-limiting in ContextPhone. It just wasn't enough. In our last field-study we gave the users a version of the presence service that allowed them to give a free-text description of their current situation. Presto: free instant messaging. Even with the rate-limit the users filled their 100 megabytes/month subscriptions and went above that by another 50 megs. This cost us several hundred euros per user, and since we reimbursed them for the sum later, a significant cash-flow issue to the users as well.

8 ~~The network is homogeneous~~

We can assume the network to be as homogenous as we want by only handing out devices of the kind(s) we want. That the users might want them to work with other devices as well is another story...

Conclusion

Context-aware system prototypes have surpisingly much in common with what could be termed *real* systems in other settings: they are used without supervision, by naive users, over extended periods of time and with varying connectivity. These mean that Peter Deutsches fallacies are relevant. A prototyping platform for context-aware applications needs to take them into account, which we have not fully done.

We have claimed that only five of the eight fallacies are actually relevant. We are worried that this is not true.

References

1. Deutsch, P.: The eight fallacies of distributed computing. online (2004). `http://today.java.net/jag/Fallacies.html`
2. Brooks, F.P.: The Mythical Man-Month: Essays on Software Engineering. Addison-Wesley Longman, Inc., Reading, Massachusetts, USA (1975)
3. Dey, A., Salber, D., Abowd, G.: A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Human Computer Interaction **16** (2001) 97–166
4. Raento, M., Oulasvirta, A., Petit, R., Toivonen, H.: Contextphone, a prototyping platform for context-aware mobile applications. IEEE Pervasive Computing **4** (2005)
5. Saltzer, J.H., Reed, D.P., Clark, D.D.: End-to-end arguments in system design. ACM Trans. Comput. Syst. **2** (1984) 277–288
6. Oulasvirta, A., Raento, M., Tiitta, S.: Contextcontacts: Re-designing smartphone's contact book to support mobile awareness and collaboration. In: Proceedings of the 7th International Conference on Human Computer Interaction with Mobile Devices and Services, MOBILEHCI'05. (2005)
7. Chalmers, M., Galani, A.: Seamful interweaving: Heterogeneity in the theory and design of interactive systems. In: Proceedings of the 2004 conference on Designing Interactive Systems (DIS) 2004, New York, USA (2004) 243–252
8. Suomela, R., Räsänen, E., Koivisto, A., Mattila, J.: Open-Source Game Development with the Multi-user Publishing Environment (MUPE) Application Platform. In Rauterberg, M., ed.: Entertainment Computing ICEC 2004: Third International Conference, Eindhoven, The Netherlands, September 1-3, 2004. Proceedings. (2004) 308–320
9. Harrison, R.: Symbian OS C++ for Mobile Phones. John Wiley & Sons Ltd, West Sussex, England (2003)
10. The European Commission: Directive 95/46/EC on the protection of individuals with regard to the processing of personal data and on the free movement of such data. Official Journal of the European Communities (1995) 31–50