Chapter 2.10
Politeness as a Social Software Requirement

Brian Whitworth
Massey University, Auckland, New Zealand

ABSTRACT

If politeness makes society a nicer place to be, by lubricating the interaction of its human parts, then the same is important for online society. As the Internet becomes more social, software can mediate social interactions, serve as a social agent or act as a personal assistant, but to succeed in these roles it must learn a new skill - politeness. This article proposes politeness as the distinguishing mark of a new generation of community software based on the benefits of social synergy rather than technical efficiency. Conversely, selfish software is currently a widespread problem as politeness is a software design "blind spot". An informational definition of politeness as the giving of choice suggests social software should be: 1. Respectful, 2. Transparent, 3. Helpful, and 4. Personal and 5. Responsive. For the Internet to realize its social as well as technical potential, software must be not only useful and usable but also polite. [Article copies are available for purchase from InfoSci-on-Demand.com]

INTRODUCTION

Computer Agents

Software, with its ability to make choices, seems to have crossed the threshold from inert machine to interaction participant, as the term human-computer interaction (HCI) implies. Computers today are no longer just tools that respond passively to directions, but agents, assistants and, in general, online participants in their own right. Miller notes that if I accidentally hit my thumb with a hammer, I blame myself not the hammer, yet people may blame an equally mechanical computer for errors they initiate (Miller, 2004, p. 31). Computers are just as mechanical as cars, but while a car inertly reflects its driver's intentions, computers now ask questions, request information, suggest actions and give advice. Nor are computers mediating a social interaction, like email, simply passive, as the software, like a facilitator, affects the social interaction possibilities (Lessig, 1999). As computers evolve, people increasingly find them active collaborators and participators, rather than passive appliances or media. In these new social roles, as agent, assistant or facilitator, software has a new requirement – to be polite.

If software can be social it should be designed accordingly. A company would not let a socially ignorant person represent it to important clients. Yet often today's software interrupts, overwrites, nags, changes, connects, downloads and installs in ways that annoy and offend users (Cooper, 1999). While such behaviour is not illegal it is certainly impolite.

Selfish Software

The contrast to polite software is "selfish software". Like a selfish person who acts as if only he or she exists, so selfish software acts as if it were the only application on your computer. It typically runs itself at every opportunity, loading at start-up, and running continuously in the background. It feels free to interrupt you any time, to demand what it wants, or announce what it is doing, e.g. after installing new modem software it then loaded itself on every start-up, and regularly interrupted me to go online to check for updates to itself. It never found any, even after many days, so finally after yet another pointless "Searching for upgrades" message I decided to uninstall it. As in "The Apprentice" TV show, the reaction to assistants that don't do what you want is: "You're fired!"

If impolite software can drive users away, this implies a new type of software error – a social error. When a program gets into an infinite loop that "hangs" the computer the software has created an information processing error. When poor usability means a user cannot operate a computer system, the software has created a human processing error. When software socially offends and drives away users however it is a social error. In the case of a usability error, users want to use the system but don't know how to. In contrast for the case of impolite software users understand it all too well and choose to avoid it. Modern socio-technical systems cannot afford social errors as without social participation they fail. In practical terms a web site that no-one visits is as much a failure as one that crashes. Whether a system fails because the computer can't run it, the user doesn't know how to run it, or the user doesn't want to run it doesn't matter because the end effect is the same - the application doesn't run.

For example the author's new 2006 computer came with McAfee Spamkiller which when activated then deliberately (by design) overwrote my Outlook Express mail server account name and password with its own values. After checking why I could no longer receive mail I found my mail server account details were wrong, so retyped in the correct values to fix the problem and got my mail. However next time the system rebooted, McAfee rewrote over my mail account details again. The McAfee help person explained that Spamkiller was protecting me by taking control, and routing all my email through itself. To get my mail I had to go into McAfee and tell it my specific email account details. That when I did this it didn't work is less the issue than why this well known software:

a. Felt entitled to overwrite the email account details a user had typed in.
b. Could not copy my account details, which it wrote over, to create its own account.

This same software also "took charge" whenever Outlook started, forcing me to wait as it did a slow foreground check for email spam. Yet in two weeks of use, it never found any spam at all! Again, not being hostage to this software, I concluded it was selfish software and uninstalled it.