Introduction

I have been interested in decentralised social software for about 7 years. I have been pleased in the last couple of years have seen a flurry of interest in FSW technology. Whilst decentralisation of existing closed source data silos is a worthy goal, opening up the data and algorithms used to process them offers a lot more potential than simply adding privacy to existing social websites. RDF/XML has matured as a standard, but is lacking widespread adoption in part due to the operators of data silos economic incentive to prevent widespread use of the data therein. FSW developers face the reverse situation, in which non-adoption of semantic web technologies is a competitive disadvantage. I look forward to decentralised systems which allow not only machine readable data but also algorithms are free to flow between users.

Evolution

Since its original incarnation as a system of interlinked static pages, WWW has undergone various evolutions such as CGI scripts, RSS feeds and web services. While helpful, these have nevertheless stopped short of allowing a deep level of cooperation and systems integration which might be compared to a multi-user operating system.

My vision is that, rather than just giving one other static data, users should have the option of entrusting one another with some or all of the resources available to them, such as disk space, CPU cycles, installed software and even cryptographic keys. Such extensive collaboration would obviously require agreement on cryptographic standards to establish identity, but that alone would not be sufficient to allow extensive collaboration, since a network of diverse computing environments would have trouble interoperating without a common language. As a minimum, agreement on the format of requests is needed to allow users to exactly articulate their requests. I propose a method designed to do this whilst making few assumptions about how their friends might choose to respond to them or what software or hardware they might use to do so.

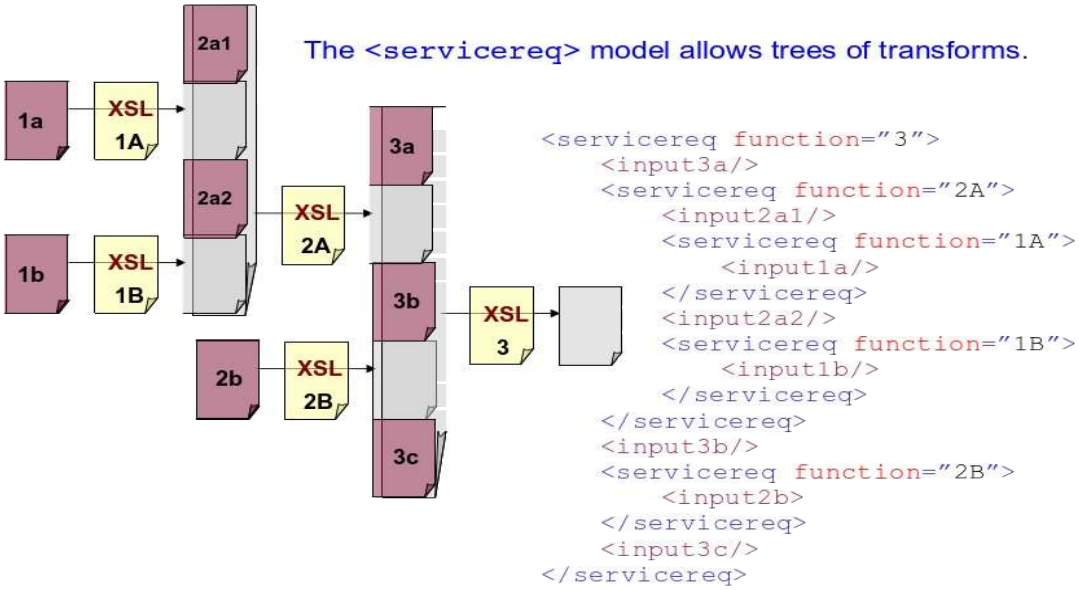
XML Request Processing Model

Each request is expressed as an XML element with a the *function* attribute identifying the desired functionality as a URI, optional extra attributes for fine tuning and optional contents constituting arguments. One simple example is

```
<f2f:servicereq function="http://friend2friend.net/modules/demo/hello-world"/>
```

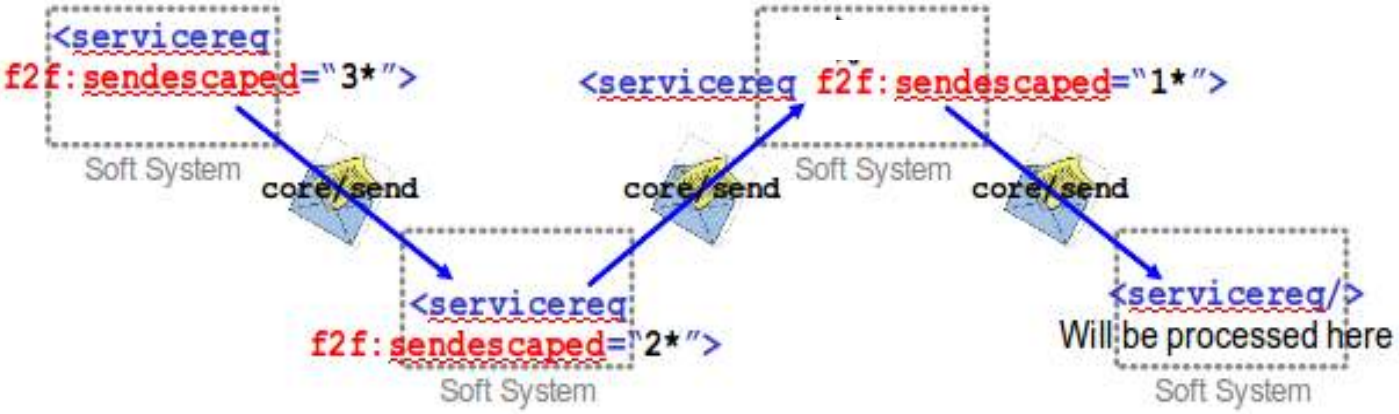
Meaning should accrue to particular URIs via a semantic web style process of decentralised consensus, so while individual XML-VMs remain free to interpret such requests how they see fit, in practice, requests will likely end up running either identical or equivalent services, albeit perhaps on different data.

Requests may be nested, resulting in XML pipelines:



A set of attributes (called 'processing directives') are available to fine tune the request or tweak its output. These were found invaluable to fit services together, and are mainly xpath based modifications such as adding attributes or sorting or discarding some of its content.

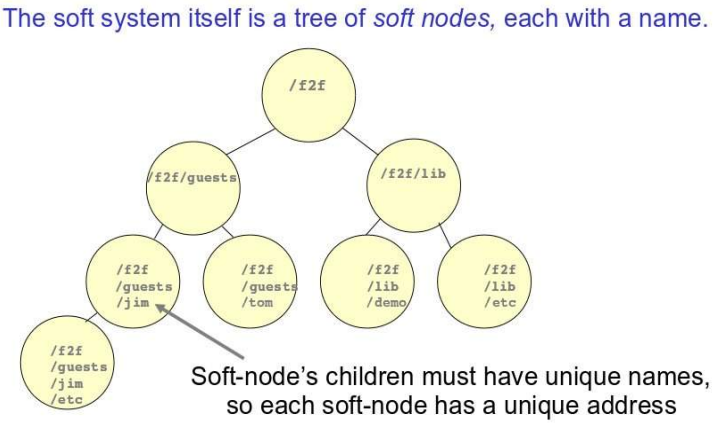
By default, requests are processed in document order, but children are processed before their parents, though this can be modified by the *f2f:escaped* processing directive, which can also allow requests to go unprocessed. The *f2f:sendescaped* processing directive has a similar effect, but is decremented when sent between different soft-systems, allowing requests to take effect on remote XML-VMs:



XML Virtual Machine

The XML virtual machine (termed a 'soft-system') is a tool to simplify the coordination of multiple computers using instructions expressed in XML. Since its input and output are XML, it uses XML as its native data type. The XML-VM is implemented as a set of independently addressable units ('soft-nodes'), each of which may have its own XML data stores and custom software, and green threads which interpret requests, perhaps contacting other soft-nodes or soft-systems and spawning child threads there in the process.

Soft-nodes are arranged as a tree, and communicate by passing XML messages up and down. All contact with the outside world occurs via the soft-system's root node.



These XML messages are accessible by filters between nodes and before and after execution. Whilst most programmers need not know about them, filters present a set of hooks available for logging, intercepting and virtualizing system operations.

The XML-VM has a set of core functions for tasks such as configuration and interfacing with the file system, so that the XML scripting language can be used to automate system configuration.

Modules

Core functionality can be augmented by coding of modules. Module definition (c.f. WSDL) files define the functionality provided by the module, e.g. declaring permissions, locating XML datastores, module scripts and XSDs. Programming of complex modules has been tried and found difficult. The Unix philosophy of programming small pieces and slotting them together is much more suitable.

XML Scripting Language

Module scripts are written in an extension of XSLT1, which adds the core functions that allow transforms to have side effects, such as communication and writing to disk. The XML scripting language was designed to abstract away details of hardware and physical location of machines but also of the structure of soft-systems, allowing the programmer to focus on the required semantics. Modules could be coded in languages other than XSLT but would not be automatically ported, so could not be automatically transferred between soft-system to another.

Security Considerations

Each F2F server has one administrator soft-system, which has the highest level of privilege. Other users have various restrictions such as limits on access to the file system and F2F server internals. A filter at the root node of the soft-systems translates between externally used keys and internally used ids. The ids form a minimal base on which to build a more nuanced system; at the base level, access control is by id and services have either a blacklist or a whitelist. Message filters could form the basis of a range of security measures, such as checking for malicious scripts, asking for user intervention or informing other soft systems of suspicious activity.

The server is highly configurable, and whilst great flexibility and openness are possible, it could also be provided in a locked down default state which denied friends the option of doing anything more interesting than sending status messages or tagging one another in photos. The philosophy of Friend2Friend has been to make simple bottom layers and keep the design as clean as possible so that refinements can be built on top of the core system.

A system in which users may be trusted to automatically upgrade their friends' soft-systems offers a great deal of facility and a commensurate level of insecurity. I would however disagree that this is a worse situation that more traditional systems in which a relatively large number of users rely on the integrity of a relatively small number of generally closed source providers of security products.

Current Implementations

An implementation in is available from http://friend2friend.net/docs/installation/. This requires PHP 5.2 (not 5.3!). It is has a JS interface which is sufficient for simple testing and basic scripting. As well as comprehensive log files, the system throws a soft error and accompanying call stack if it encounters an exception.

The code is not suitable for extensive production use, as PHP's XML/XSL handling is subtly broken and unlikely ever to be definitively fixed:(. It also lacks a system of secure communication.

Documentation

- Several presentations are available from http://altruists.org/ff
- A wiki is at http://wiki.friend2friend.net/