

Software Engineering and the Internet: A Roadmap

Paolo Ciancarini
(with L.Bompani and F.Vitali)

Dipartimento di Scienze dell'Informazione
University of Bologna - Italy

Limerick
June, 2000

Outline

Designing network-aware, WWW-oriented applications

XML and related technologies

Software environments centered on active documents

Active documents are agents

Towards agent-oriented software engineering

A roadmap

Acknowledgements: most ideas exposed here were developed in discussions and joint papers with our colleagues and students in Bologna, especially Cecilia Mascolo, Davide Rossi, and Alessandro Ronchi.

Using the Internet as a programmable platform

“The computer is the network”: the convergence of computer, telecom, and multimedia technologies produces new opportunities for industry, research, and teaching.

Example 1 The mp3 format and the Napster system are challenging the music industry: the old ways of distributing, selling, storing and playing music are obsolete

Example 2 Several Universities are now offering courses for designers, managers, and art directors (eg. “Web DJ”) of WWW sites and portals: they started forming people for the “contents industry” over the Internet

The convergence, or “networking”, of Information Technology industries is pushing the development of novel appliances, applications and services where the Internet plays a major role

Software engineers are challenged to adapt themselves to the new platform, inventing new methods and tools to exploit the new computing models enabled by the Internet

Example 3 Some projects (eg. Argo-UCI, Box-UCL) are building SEE able to manage UML documents using standard WWW browsers and technologies

The Internet as a platform for groupware

We claim that the future of Software Engineering in relationship with the Internet will be especially fruitful in the field of *groupware*, as a domain offering interesting and important design problems

Groupware: multiuser, document-centric applications

- *mobility* of people, hosts, and documents
- *communication*: synchronous (“same time”) or asynchronous (“any time”)
- multi-user *interaction*: “same place” or “different places”
- *composition*: groupware is usually the result of the combination of several software technologies
- *agenthood*: in groupware applications several activities can be carried out by autonomous program
- *document-centric*: documents are complex data structures with user-specific contents, structure, and behaviours

WWW-based, agent-oriented groupware is especially challenging:

software technologies related to the WWW are fastly evolving, usually as a result of some standardization process (by organizations like W3C or OMG)
agent-oriented software engineering techniques are suitable for network-aware applications, but we need specific ontologies and tools

Mobility layers

Software technologies involving code distribution and mobility need novel, “network-aware” specification, design, and programming languages and tools

A key issue when an application includes “network-aware” (e.g. mobile) components is how to design its *architecture*, which can be decomposed in at least three different layers:

- The *physical network* layer, made mostly of *immobile hosts* and reliable and fast connections, where a mobile entity consists of a piece of hardware using a connection usually unstable (e.g. wireless) and with low bandwidth
- The *middleware network* layer, made of *abstract machines* (e.g. a JavaVM, an XWindow server, etc.), where a mobile entity consists of a whole process or a service migrating from a host to another host
- The *logical network* layer, made of application code scattered over the middleware network, where a mobile entity consists of an *agent* moving from an abstract machine to another one

Mobile entities in the WWW

The original WWW was based on 2 concepts of mobility:

HTTP servers distribute documents on demand to client browsers: this is *mobility of data* (HTML and XML are not Turing equivalent: they are just SGML dialects to specify data structures like paragraphs or tables)

A browser can “navigate” through the hypertext links, requesting HTML pages to a server, and sometimes “jumping” from a server to another server. Although URLs denote static, “physical” resources, they can be passed around: this is *mobility of references*

Remark: Mobility of reference means both that a channel name can be passed around, and that a process can detach a channel and connect to another channel

A third concept of mobile entity:

An **active document** (= contents+structure+behavior) moves across the net, from servers to browsers and back

Representing documents

We use the term “document” with a wide scope, meaning any kind of data structure which network-aware applications can exchange

(passive) document: contents + (structured) representation

Example: RTF is a language of commands that a word processor has to apply to a document to render its contents, in terms of fonts, justification, margins, etc.

HTML has been invented to write passive documents to be displayed inside WWW browsers

(HTML) document: contents + procedural markup

Document processing applications use two different approaches to represent a document

a) a proprietary, binary, *machine-readable*, code:

Examples: MS Word, Adobe PDF, SUN Java bytecode

b) an open (standard), ASCII-based, *human-readable* code:

Examples: RTF, HTML, PostScript, TeX

Remark: ASCII-based documents are “flat”: their structure either is “wired” inside the applications which first parse then can manage them, or some further code (markup) is needed to give structure to an ASCII file

Procedural vs declarative markup

Formatters (eg. TeX or PostScript) assign a rendering behaviour to documents represented as files mixing formatting commands (*mark-ups*) and text

In order to build a “page”, formatters are driven by markup commands interspersed in the document text: formatters are in fact compilers

A system such as TeX produces high quality results because layout algorithms are able to approximate the behaviour of experienced professionals

However, TeX (and HTML) markups are very procedural, and this is bad for two reasons:

- the logical structure of a document is not expressed in the markup, thus searching document abstractions (eg. all titles, represented by indented bold lines) is difficult
- the concept of style is non existent, thus changes in style require revising all markup commands

A solution was the introduction of declarative markup languages (like LaTeX), useful to declare both the logical structure of a document and the styles to be used

Example

```
\documentstyle[twocolumn]{article}
```

Declarative markup: SGML

As an a effort to develop a standard declarative markup language, IBM in 1980 proposed the Generalized Markup Language, which when adopted in 1986 by ISO became SGML (ISO/DIS 8879)

Markups in SGML is based on tagged parentheses:

```
<chapter>
Once upon a time...
...
... they all lived happily ever after.
</chapter>
```

Since total parenthesising is cumbersome, the SGML syntax allows the omission of end-tags where they are redundant

```
<list>
<item>First item, no end marker.
<item>Second item. Only the end list
marker is required
</list>
```

SGML includes a meta-language to declare new tag types, to form a Document Type Declaration (DTD)

Thus, the markup syntax can be redefined arbitrarily: what about the semantics?

An SGML example

```
<!DOCTYPE report [
  <!ELEMENT report      - - (chapter)* >
  <!ELEMENT chapter    - - (title, section*)>
  <!ELEMENT section    - - (title, para*)>
  <!ELEMENT title, para - - (#PCDATA|emph|formula)*>
  <!ELEMENT formula    - - (#PCDATA|emph|sub|sup)*>
  <!ELEMENT emph,sub,sub - - (#PCDATA)>
  <!ATTLIST
    formula type (inline|separated) inline >
]>
```

```
<report>
  <chapter> <title>Open problems</title>
    <section> <title>Perfect numbers</title>
      <para>A number is said to be <emph>perfect</emph>
        if it is the sum of its divisors. For example, 6
        is perfect because <formula>1+2+3 = 6</formula>,
        and 1, 2, and 3 are the only numbers that divide
        evenly into 6. </para>
      <para>It has been shown that all even perfect
        numbers have the form <formula type=separated>
        2<sup>p-1</sup>(2<sup>p</sup>-1)</formula> where
        p and <formula>2<sup>p</sup>-1</formula> are both
        prime. </para>
      <para>The existence of <emph>odd</emph> perfect
        numbers is an open question. </para>
    </section>
  </chapter>
</report>
```

Chapter 1

Open problems

1.1 Perfect Numbers

A number is said to be *perfect* if it is the sum of its divisors. For example, 6 is perfect because $1+2+3=6$, and 1, 2, and 3 are the only numbers that divide evenly into 6.

It has been shown that all even perfect numbers have the form

$$2^{p-1}(2^p - 1)$$

where p and $2^p - 1$ are both prime.

The existence of *odd* perfect numbers is an open question.

Short summary on SGML

SGML is a *Standard Generalized Meta-markup Language*

- Standard: it is the ISO standard 8879/1986
- Generalized: it is not meant for any specific usage, but for providing information about chunks of text
- Meta-markup: it does not provide a markup, but a syntax to define markup

A SGML document contains elements, entities, comments and processing instructions (eg. the LINK marker is used to give behavioural semantics to declarative tags)

```
<!LINK abstract p indent=10>
```

A SGML document is composed of three parts: the SGML declaration, the Document Type Declaration, and the document instance.

SGML only specifies the structural elements composing a document

SGML does not specify rendering semantics of its documents (this is a task for *stylesheets*)

SGML does not provide support for hypertext links (meant for non-interactive uses).

Towards active documents: XML

XML is a standard markup language (defined by W3C, and derived from SGML) to describe the *structure* of documents; instead, documents *behaviours* are specified using either stylesheet languages (e.g. XSL) or even programming languages (e.g. Java)

document: contents + structure + behaviours

Remark: declarative markup is either structural or semantic

Structure: A book is composed of chapters, sections, titles, notes, etc. A letter is composed of sender, addressee, salutations, body, signature, attachments, etc.

Semantics: A news item about a criminal act may specify the source of the news item itself, the description of a sequence of acts, the name of the place where the acts took place, the name and rank of the involved police officers, the stolen amount, etc.

What is a hypertext document

Definition: a *hypertext document* is defined by its contents, its structure, one or more “behaviours”, and its relationships with other documents

Intuitively, a document carries some *information* and has some *structure*: a book, a report, a letter, a program are examples of documents of different forms

When documents live inside a computer they have a *physical representation* based on some data structure

When we consider a document in abstract, it is fully defined by its contents and logical structure

The structure of a document is an instance of a (logical) *document model*, that is an ontology of abstract entities suitable to describe the document’s elements (eg. chapters, sections, paragraphs, pages, etc.)

Any document can display several *behaviours*:

- a rendering, or presentation behaviour, defines how a document is displayed on an output device, like a screen or a printer (e.g. pretty printing is a rendering behaviour for documents including C programs)

- a view, or control behaviour, defines how a user can interact with a document (e.g. using hypertext)

- a static semantics defines how a document can be analysed with respect to some verification rules

Another important operation on documents is *interchange*,

which happens when two different applications (with different internal representations) have to share the same document (have to *interoperate*)

XML and related standards

The SGML community was not satisfied by the HTML language, and proposed to create a simple dialect of SGML and a few related standards. These efforts, hosted by W3C, were joined by Microsoft, SUN, and others.

XML is a proper subset of SGML: all XML documents are SGML documents. Most SGML documents are XML documents or can be easily transformed

DTDs are accepted by XML parsers but not required: an XML parser should accept both valid documents (i.e., documents correct according to a given DTD) and well-formed documents (i.e. documents that “balance” the tags)

Therefore, document designers can still create special document structures for their needs, but casual authors do not need to first design a DTD for their purposes.

Furthermore, it is not necessary to check a document against its DTD every time, but only at creation time.

For anything else, XML is still a generalized meta-markup language where authors can *freely* define tags, attributes and entities

XML is actually a family of technologies:

- XML 1.0

- XSL

- XLink and XPointer

- XML Namespaces

XSL

Extended Stylesheet Language (XSL) is the way to attach styles to XML documents.

XSL is a tree rewriting, rule-based language. The idea is to rewrite a document meaningful to the author into another one meaningful to some display program (eg. a browser)

XSL stylesheets are composed of rules made of a pattern and an action. The action is an XML chunk that is re-written in the destination document.

XSL also dictates the set of final elements to be used. They are all and only typographical elements.

Although the re-writing capabilities of XSL are appropriate for tree-based structures representing XML documents, the set of formatting objects currently included in the XSL specification are limiting.

XLink and XPointer

HTML only provides support for simple, directional, embedded links. XLink allows to define:

- simple links (*a la* HTML)
- Multi-directional links
- External links
- Automatic links
- Document groups

XLink uses the reserved attribute `xml:link` to identify link elements.

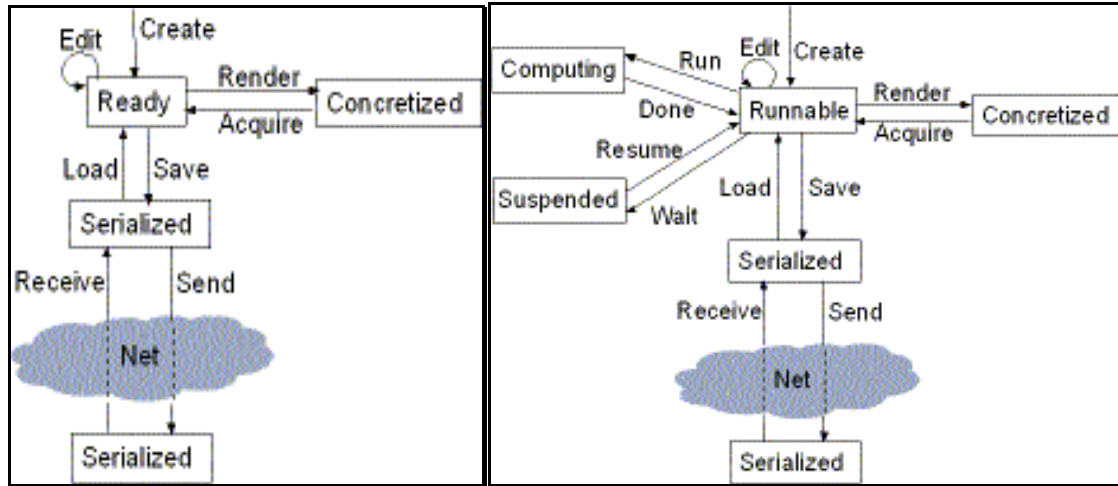
HTML refers to elements only if they have a NAME attribute. SGML (and simple XML) refer to explicit ID attributes. XPointers specify in a general way chunks of content inside an XML document. They allow to combine absolute references (eg. a named element, the root of the XML tree, etc.), relative references (eg. the second child), and text span (eg. third character).

Example: the third character of the second child called “P” of the element with ID=27 in the document `http://www.dom.com/doc.xml` becomes:

```
http://www.dom.com/doc.xml#id(27).child(3,P).string(3)
```

Technologies for active documents

Document lifecycles:



Passive document

Active document

An "active" document provides support for actions on the document, that is it can: include animations, perform computations, provide support for searching, etc.

Active-X objects, Java applets, JavaScript scripts or even complex PostScript or TeX programs can be used to build documents offering more than their content

The idea of active document consists of putting the behaviour together with the contents: using **generic markup** behaviors and actions are applied just like formatting

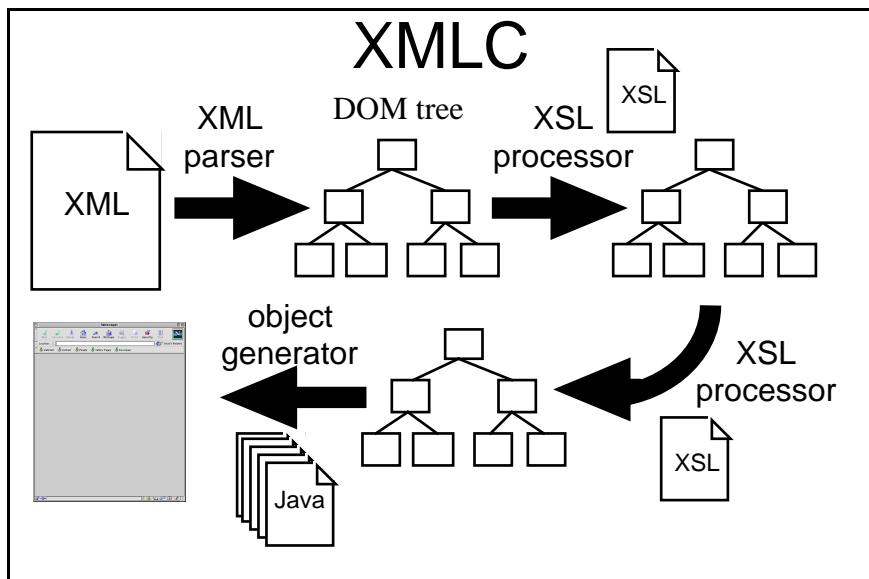
Remark: an active document is an (autonomous) entity including code and data, and moving around: it is an agent!

Displets

XSL is not up to XML in terms of generality: specialized notations are not supported

Our proposal is to create an open set of formatting objects that are loaded when needed, depending on the required behavior of a document.

Each document has a stylesheet associated that maps its elements to the available formatting objects; each formatting object is then associated to a sw module, a JavaBean, that we call *displet*, displaying the information (we exploit the dynamic linking capabilities of Java)



Some applications that we have explored:

- Special typographical elements; layout management
- Notations for software engineering documents
- Management system for hypertext UML documents
- Porting of ToolBooks inside standard browsers
- DSS for financial applications

- WorkSpaces (a workflow management system)

Declaratively active documents

We have two reference architectures for displets, that we call “server-side” and “client-side”, respectively

The *server-side* architecture is more efficient, but less flexible (a behaviour is pre-processed and “wired-in” a document, that then is sent to a browser and displayed)

The *client-side* architecture can be used to have documents performing activities, rather than just be displayed

Since we associate displets which are Java Beans to XML documents, we can ask a Bean to paint itself, or to perform any other method of its classes

Depending on the stylesheet and the Java classes, then the same document can behave in any of different ways

The code performing the activities is not part of the document (as in Active X or similar systems) but declaratively associated to the document

Example: Petri Nets

We have defined a DTD for Petri Nets, thus we can represent textually any Petri Net. Then we have defined stylesheets to display and animate a Petri Net. Then we have enriched the Petri Beans with editing capabilities, so that the original document can be modified (but not saved, if we use the full Java security model)

Using XML in software engineering

The advent of WWW made clear that documents should be considered as portable, application-independent components requiring specific models and languages

XML is rapidly becoming a reference interchange language in a variety of fields: for instance, IBM introduced XMI, that is an XML-based metamodel for UML (it is a DTD for UML documents)

OMG adopted XMI as a metamodel for UML. Thus, all future XMI-compliant UML tools (eg. Rational Rose™) will be able to be easily integrated into a coherent system

Our XMLC architecture, where JavaBeans are used as generic (formatting) objects, proves that component-based systems can be defined as collections of active documents

However, the design of WWW-based groupware needs more than just a technology for sw components: documents are not only active; they are *interactive* agents (their users play some role) and moreover they activities need to be coordinated

Sw architectures: from components to agents

Databases were the focus of application design in the '80

Components were the focus of appl. design in the '90

Agents will be the focus of application design in the '00

Until 1995 application vendors based component interoperability on platforms like CORBA and proprietary document models like Microsoft OLE

Component-based applications are usually built on top of a distributed middleware platform (that is a reference software architecture)

CORBA (Common Object Request Broker Architecture)

ActiveX/Network OLE (OLE 2/DCOM)

Lotus Notes

World Wide Web (HTTP servers/clients + CGI)

Sun Jini

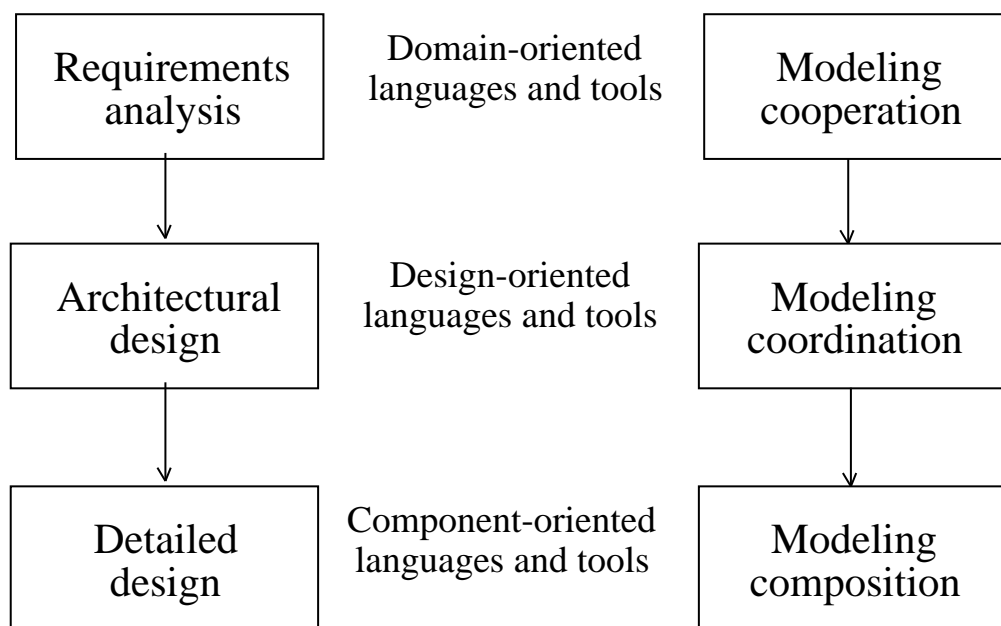
All these architectures offer some concept of *agent-hood*

Agent oriented sw engineering

Sw engineering deals with production processes and tools

“Classic” sw engineering is not adequate for the network, because it lacks of conceptual tools to deal with *interaction*

Classic vs. agent-oriented sw engineering process:



Cooperation model: the abstract description and analysis of all roles involved in a system (cf. Use cases in UML)

Coordination model: the description of a sw architecture in terms of agents and their activities (eg. UML is weak in dealing with agent-oriented architectures)

Composition model: the set of mechanisms used to implement, reuse, or activate components and assign resources (eg. “JavaBeans components representing XML documents and roaming a web of HTTP servers”)

An example: conference management over the Internet

Basic idea: submitted papers are agents (active documents)

Cooperation model: *which social laws for roles and activity workflows?* we have to decide which conference organization we prefer:

- single/multiple tracks and conference workflow;
- tyrannical, oligarchic, democratic cooperation management;
- authors of papers anonymous/known to reviewers; etc.

Coordination model: *how do we organise agents?*

we have to decide which sw architecture we offer to the agents, eg. a database-like or a MUD-like one, and how agents interact with their environment

Composition model: *how do we reuse and compose components into architectures?*

we have to detail which components we allow and how do they "connect", eg. we could ask that all papers are based on PostScript (so that some browser could manage them) or on XML/XLink-Xpointer (so that they could form an hypertext and be managed by a search engine)

A Roadmap

Software engineering researchers and practitioners have to follow closely what organizations for “network governance” like W3C or OMG develop and define as software standards

In the next five years, we believe that:

the family of XML technologies will have a strong impact on programming languages and tools

the concepts of agent-oriented software engineering will gain momentum, possibly succeeding object-oriented software engineering as the dominant design method

“Active document”-centric software architectures will substitute client-server architectures

Conclusions

Network-aware, agent-based applications are designed by integrating several technologies; from a sw engineering perspective, we need methods and tools to master the complexity of such an integration

Cooperation, coordination and composition are important concepts for agent-oriented software engineering

However, we have a long way before understanding their inter-relationship

At present we are especially interested in:

- developing and evaluating network-aware models and languages for agent-documents and related software architectures

- developing formal specification languages useful to design software architectures including mobile components

- building a Web-based software architecture able to support active documents (integrating XML and Java)

Please visit us at the site:

`www.cs.unibo.it/projects/displets`