

1

Web applications



The explosive growth of the Web is perhaps the single most important event in the history of computing technology. What started as an information infrastructure is now turning into an infrastructure encompassing both information and applications, and is becoming the backbone for the commercial deployment of the Internet.

Web Applications

12

1-1

- objects and the Web
- Web application development – tools and environments
- DeJaVU - Web applications with *hush*
- software architectures for the Web

Additional keywords and phrases: *Web Objects, XML, Java, CORBA, multimedia, software architecture*

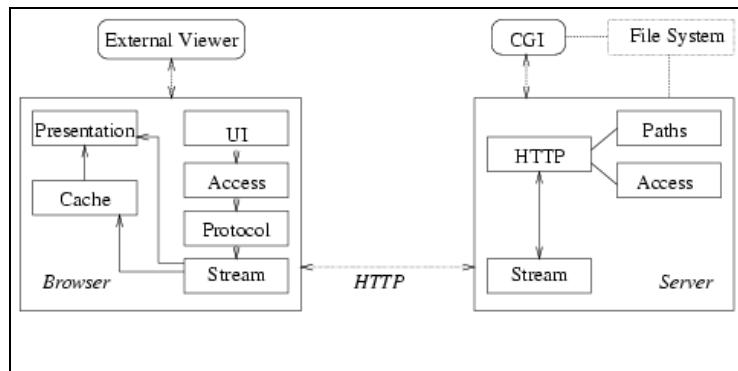
Slide 1-1: Web applications

In this chapter, we will explore how the Web affects (object-oriented) software development. First of all, we will discuss whether object orientation has any relevance for the Web and for the development of Web applications. We will look at some of the current trends and technologies, discuss the possible occurrence of the Object Web, and look at an example deploying Web technology to provide an infrastructure for distributed object computing. We will reflect on the com-

putation model underlying the Web, to explore how to program the Web to suit our needs. We will also look at the phenomenon of intelligent agents on the Web, which may aid the user in retrieving the right information and perform his/her tasks in a more convenient way. We then present some of our early research on extending the Web with multimedia functionality, carried out in the DejaVU project at the Vrije Universiteit. Concluding this chapter, and the book, we will discuss the forces that play a role in defining a suitable software architecture for (object-oriented) Web applications.

1.1 Objects and the Web

The Web originated from an initiative at CERN, nicknamed the World Wide Web (WWW), to provide an infrastructure for the exchange of information between scientists. Undoubtedly, the initiative succeeded beyond expectation. As described by [Practice], at the time there were other such initiatives. Nevertheless, the effort at CERN contained two novel ideas: the use of hypertext, to allow for easy navigation between documents, and the deployment of a client/server architecture, to separate presentation from the delivery of documents.



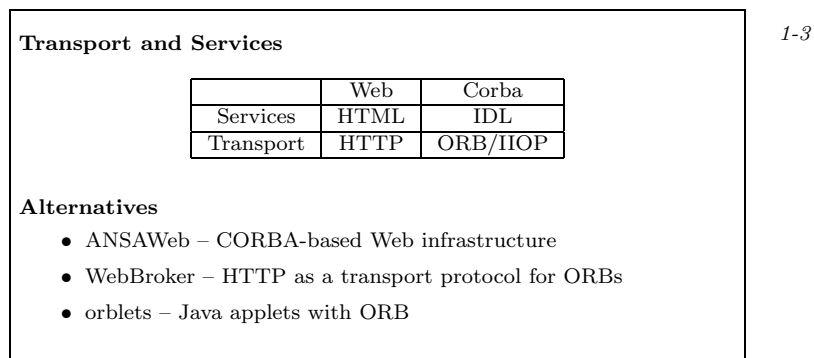
1-2

Slide 1-2: Client/server pair

Among the original requirements of the WWW was *extensibility*. As slide ?? indicates, which is an adapted rendering from a chapter about the Web in [Practice], both the browser (client) and the (HTTP) server may be extended with, respectively, additional viewers on the client side and arbitrary programs through the Common Gateway Interface (CGI) on the server side. Together with these extensions the original infrastructure, which consists of HTML (Hypertext Markup Language) as the document format and HTTP (Hypertext Transfer Protocol) for the transport of documents, proved to be sufficient for the Web to be widely adopted. In retrospect, one may wonder why the Web was not based on, for example, distributed object technology or remote procedure calls. Accepting the Web as it is, we may still ask ourselves what role objects may play in developing Web applications.

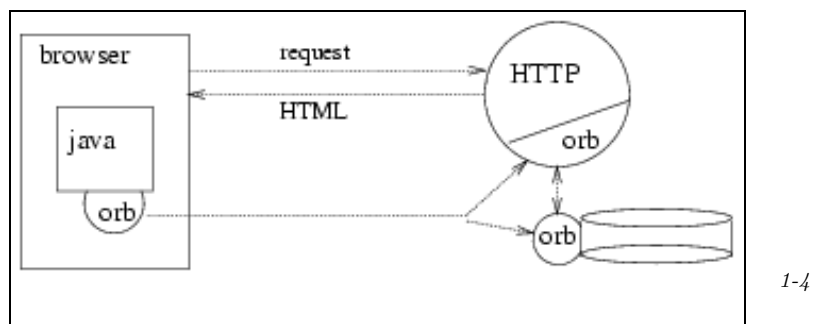
1.1.1 Trends and technologies

The Web came as a surprise, both to the hypertext community and to the distributed systems community. As a surprise because, despite its simplicity, or probably because of its simplicity, the adoption of the Web is unsurpassed, in absolute volume and growth rate. Its simplicity lies both in terms of the underlying TCP/IP-based HTTP transport protocol, and the (conceptual) functionality of the HTML hypertext format, which more or less defines the services offered by the Web.



Slide 1-3: Transport and services

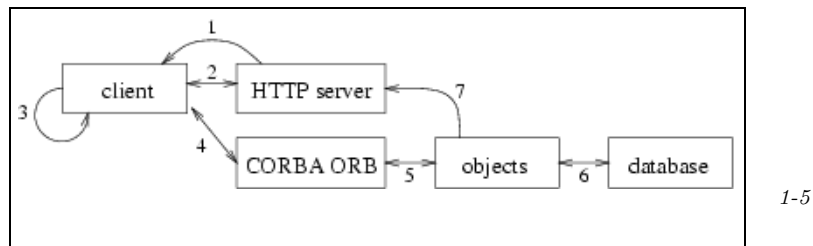
It is probably not an exaggeration to say that the entire academic community was shocked to see the sudden mass-scale adoption of a technology that was only a shallow reflection of the original conceptions of globally distributed systems and hypermedia. Not surprisingly, however, academia and other research and development institutes reacted to the Web by redirecting their research programs, in order to jump on the wagon.



Slide 1-4: Java applet with ORB

As an example, in the August/September 1996 issue of the Object Expert (Europe) the question was posed '*How to survive the Internet revolution?*'. In

answer to that question, the Web was first criticized for offering a monolithic HTML/HTTP-based structure that gave rise to many proprietary extensions. Then, as a solution, CORBA was praised as an infrastructure that allows for the creation of well-behaved extensions through the use of IDL. The most radical alternative, indeed, would be to base the Web entirely on CORBA, of which the ANSAWeb proposal is an example. A rather different route is to adopt HTTP as the transport protocol for object request brokers and turn the Web into a global infrastructure for distributed object computing, as for example suggested in the WebBroker proposal that will be discussed later. A more modest, and realistic, approach is to enhance Java applets with the capability to connect with CORBA servers, as indicated in slide ?? and slide ?. In slide ??, we see a browser with an HTML page that contains a Java applet, which may connect through an ORB directly to, for example, a database server. Alternatively, a request may pass through a CGI process to an ORB attached to the HTTP server.



Slide 1-5: Processing steps

In more detail, when we look at the processing steps, as depicted in slide ??, we may distinguish between

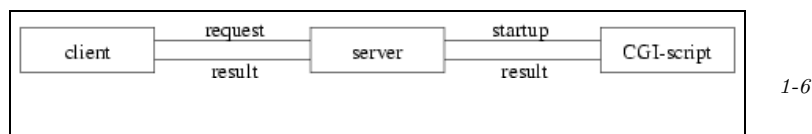
1. get the HTML page,
2. load the applet,
3. start the applet,
4. connect to a CORBA server from the applet,
5. get access to the remote objects,
6. connect optionally to a database, and
7. send output either in HTML format or directly to the applet.

Based on this setup, we may think of several alternatives and refinements, as for example the use of Java RMI or an extension of the Java ORB with full server functionality, to allow for callbacks from the objects (server) to the client (applet).

The WebBroker proposal In the scenario sketched above, Java and CORBA were used to extend the basic functionality of the Web. In a similar vein, Microsoft DCOM, as an alternative distributed object technology, might have been used to incorporate objects in the Web. The WebBroker proposal, as explained in the technical note submitted to the W3C, 11 May 1998, attempts to unify distributed object technology and the Web publishing infrastructure by providing a common *Web computing* standard based on HTTP and XML. (XML, the eXtensible Markup Language, may be considered as a lightweight version of SGML, suitable for the description of the structure and content of arbitrary documents.) The objective of the *WebBroker* is, as stated in the proposal, to have a system which is less complicated than the OMG CORBA and Microsoft COM+ distributed computing systems and which is more powerful than HTML forms and CGI. The principal advantage of the WebBroker approach is that it is Web-native. However, with the universal adoption of IIOP, which is now also the transport protocol of Java RMI, the advantage of a more efficient protocol gains more weight.

1.1.2 The Object Web – CORBA/Java versus Microsoft

No doubt, the Object Web is coming, as testified by the appearance of the *Object Web Survival Guide*, see [OWSurvival].



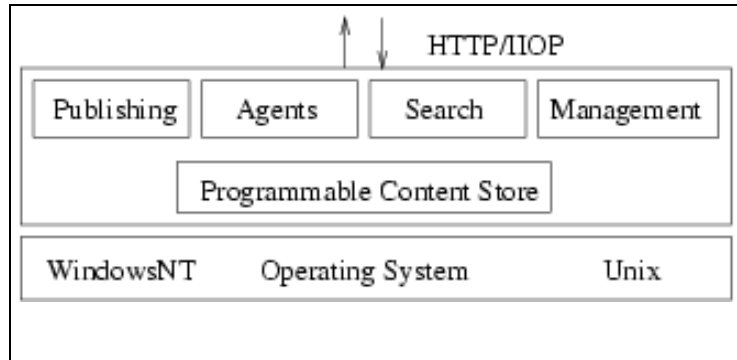
Slide 1-6: Client-Server/CGI

To state the argument for the Object Web once more, as depicted in slide ?? what we have, basically, is a client/server architecture of which the server-side may be arbitrarily extended with CGI-processes. However, CGI extensions are slow, they do not scale and, most important, they do not allow for state unless unreliable programming tricks such as cookies are used. Now, according to (the ads for) [OWSurvival], there are two camps: *Microsoft* and *Everyone Else*. We will start with the latter, which we will refer to as the *Java/CORBA Web*.

The Netscape way – Java/CORBA Web

When we consider the browser market, there are at the time of writing two major players, Netscape and Microsoft. Although Netscape is certainly not the only company selling Web servers, we will nevertheless take Netscape as representing *everyone else* to see how the Java/CORBA Web may take shape.

First of all, it must be noted that Netscape made a serious commitment to CORBA and IIOP. For example, all Java CORBA support classes are shipped with their browser. Secondly, as indicated in slide ??, we may observe that

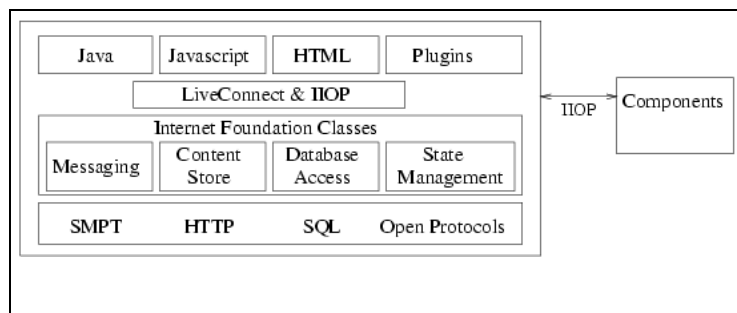


1-7

Slide 1-7: Content store

the functionality of Web servers has been significantly enhanced since the beginning days of the Web. Facilities for publishing, (intelligent) agents, search and management are now more or less standard commodities provided on top of a programmable content store, running on a variety of operating systems.

In slide ??, an architectural overview is given of one of the earlier versions of the Netscape Enterprise Server. When going from the top to the bottom, we see that content may be delivered in a variety of formats, including Java applets, Javascript, plain HTML, some legacy plugin format or any combination thereof. See section ?? for a discussion of plugins. More to the bottom, Netscape offers LiveConnect technology to allow (client) components to interact. For example, a Java applet or a plugin may be addressed from Javascript code. In addition, there is IIOP to connect to CORBA-enabled servers.



1-8

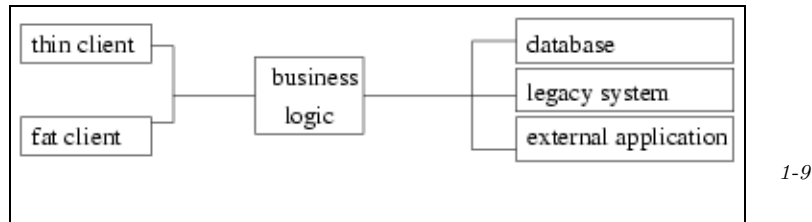
Slide 1-8: Netscape Enterprise Server

For programming server facilities, Netscape offered the Internet Foundation Classes as part of the Open Network Environment (ONE), which is based on standards such as SMTP, HTTP and SQL. However, the Internet Foundation Classes for Java have become part of the Java Foundation Classes that are delivered with Java 1.2. Server facilities include messaging, content store, database access and state management. Additional components may be provided either as

server extensions through the NSAPI, or through CORBA IIOP. For the actual creation of content and the deployment of all that technology, there is a large variety of tools from Netscape and other vendors, and plenty of documentation that may be obtained from Netscape's Web site.

The Microsoft way – DNA

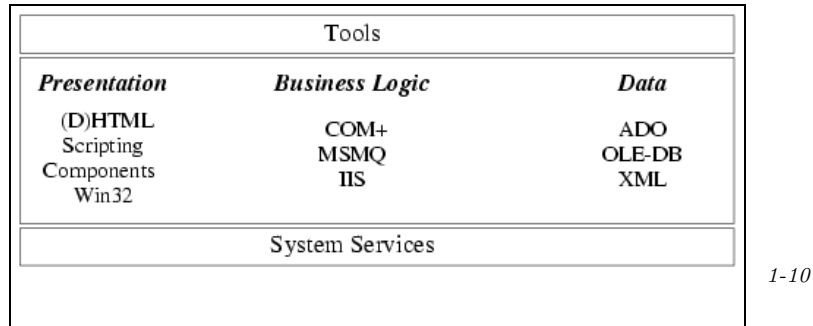
It is interesting to note that Microsoft's commitment to the Web came relatively late. Nevertheless, there is no doubt that Microsoft recognizes the importance of the Internet and the Web as the infrastructure of what it calls the *Digital Nervous System* of corporations.



Slide 1-9: Business logic

In February 1999, I had the pleasure of hearing Bill Gates speak about the *Digital Nervous System*, as a unifying concept for corporations to execute and record transactions electronically, and as a means to create corporate awareness of the actual state of business and current business goals. I found this view quite appealing, although the complexity involved in the actual archiving, search, retrieval and presentation of such material is quite immense. Ideally, as depicted in slide ??, central to any corporate information structure must be the business logic that governs the policies and information needs of the organization. At the backend of the system we may have a database, legacy systems, or external applications delivering information. For end-users, depending on the particular architecture chosen, there may be thin or fat clients giving access to the information and communication facilities.

To turn to actual technology, Microsoft's proposal to realize their vision is the Microsoft Dynamic Networking Architecture (DNA), of which the basic components are given in slide ??. In the column on the left, we have the *presentation facilities*, ranging from (dynamic) HTML to Win32 applications, going from thin to fat, indeed. In the *business logic* column, we have COM+ (which is the followup on (D)COM), the Microsoft Message Queue Server (MSMQ), and the Internet Information Server, which is a powerful server that allows for server-side scripting, Active Server Pages (ASP), and COM-based objects. For handling data, Microsoft offers the ActiveX Data Objects format (ADO), OLE-DB to connect to databases, and XML. It must be noted here that Microsoft is actively engaged in promoting XML as a data interchange standard, in cooperation with the W3C. In summary, Microsoft DNA offers Presentation Services, Application Services, Data Services



Slide 1-10: Microsoft DNA

and System Services. In addition, Microsoft offers an appealing suite of tools collected in the Visual Studio, including Visual C++, Visual Basic and Visual Interdev, for creating dynamic data-driven Web applications. Although I do not intend to make this sound like an ad, it cannot be denied that Microsoft is a serious player!

1.2 Programming the Web – a search for APIs

Leaving the Object Web for what it is, in construction obviously, we may raise the question as to what support should be provided for developing Web applications that are more finely tuned to the needs of end-users. To answer this question, or more appropriately, to gain insight into the requirements and state-of-the-art technology that was available, I organized a series of two workshops, one for the WWW5 Conference, entitled '*Programming the Web – a search for APIs*', and one for the WWW6 Conference, entitled '*Logic Programming and the Web*'. In this section we will discuss some of the issues treated in these workshops. In particular, we will reflect on the computation model underlying the Web, taking the views of Luca Cardelli presented at the WWW5 workshop as a starting point, to establish general requirements for APIs for programming the Web. Then we will look at another interesting phenomenon, intelligent agents on the Web, and discuss what would constitute a suitable framework for agent technology.

1.2.1 Models of computation

The *Programming the Web* workshop was intended to focus on concepts and requirements for high-level APIs suitable for developing Web-aware applications. The papers that were submitted, which are available on the CDROM accompanying this book, covered a wide range of interests, including computation models, applications and user requirements, software architectures and libraries, as well as heuristics and guidelines for API developers.

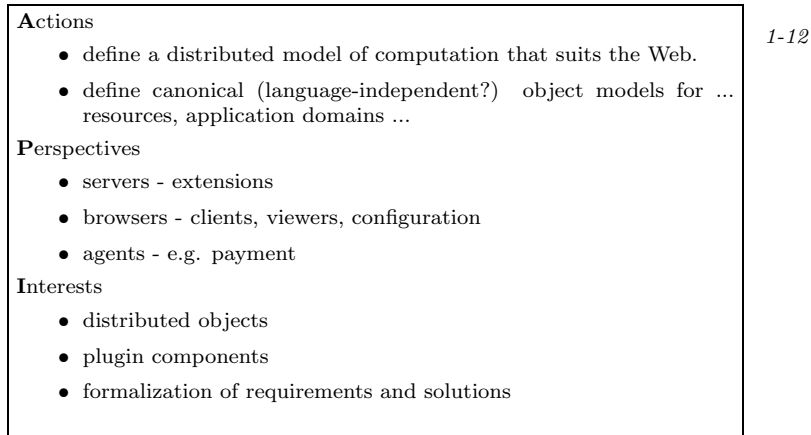
The kickoff for the workshop was given by Luca Cardelli, who raised the question 'What is the Web's model of computation?'. This question appeared

Complaints <ul style="list-style-type: none">• lack of referential integrity• undetected failures• no control over <i>quality of service</i>	1-11
Observations <ul style="list-style-type: none">• dynamic quality of services• complex interaction	
Requirements <ul style="list-style-type: none">• uniformity, openness, flexibility, orthogonality, layered	
Behavior <ul style="list-style-type: none">• reliable, configurable, monitoring, notification, thread-safe	
Answers <ul style="list-style-type: none">• object-oriented, components, virtual APIs, callbacks, plug-ins	

Slide 1-11: Requirements for APIs

to be of critical importance for understanding the requirements for APIs and for evaluating possible solutions. In summary, we may observe that there is some notion of global computation for the Web, but that computation on the Web is fraught with many obstacles, such as the lack of referential integrity (e.g. dead links), unreliable services (both in availability and quality), failures (due to servers or network congestion). What we need, in conclusion, is some (formal) model of computation that captures these properties. In addition, we need to be able to deal with such properties in our Web programs, for example we may wish to anticipate on the possible unavailability of a Web server, and provide an alternative in that case. In slide ??, an overview is given of the complaints about the functionality of the Web, observations concerning its ‘nature’, general requirements for open systems development, a wish-list of desired behavioral characteristics and potential (technological) answers.

Not surprisingly, there did not seem to be a canonical approach to the definition and development of APIs and Web applications, perhaps not in the last place because the demarcation between computation models, languages and APIs is not clear-cut. Nevertheless, as summarized in slide ??, it seemed clear that we need to define a suitable computation model as well as (abstract) object models that capture the requirements for resources and application domains (such as for example e-commerce). In addition we must distinguish between client and server perspectives, with autonomous (intelligent) agents as a possible third perspective. And, naturally, our own (technological) interests play a role as well, to the extent that it may determine possible solutions. Considering the basic needs for the development of Web-aware applications, as expressed by the workshop’s participants, which ranged over resolving URLs, billing and payment facilities, and quality of service constraints, we may observe that facilities for Web programming are nowadays as a standard provided (as extensions) by languages such as Tcl,

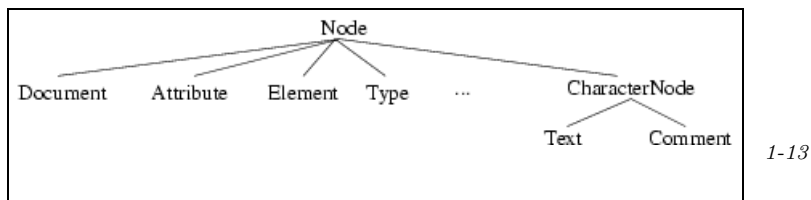


Slide 1-12: Dimensions of APIs

Perl, Python and Java. More domain-specific facilities are being developed in a CORBA context, or for frameworks such as San Francisco.

Document Object Model

Client-side scripting has been popularized by Dynamic HTML (DHTML) as originally introduced by Netscape and Microsoft. Nevertheless, scripting facilities are not standard accross the various browsers. To remedy this situation, the W3C has developed a recommendation for a *Document Object Model* (DOM), that provides a standard application programmer interface to access the structure and content of HTML and XML Web pages. The DOM allows XML and HTML pages to be treated in an object-oriented way, providing facilities for access, navigation and manipulation.



Slide 1-13: Hierarchical structure of DOM

Since XML is increasingly being used for other applications, such as Electronic Data Interchange (EDI), the DOM may in effect provide a foundation for developing Web applications. The W3C DOM Recommendation provides interfaces, described in a language and platform-independent way in IDL, for the structural components that may be used in XML and HTML documents, as indicated in slide ???. These interfaces have been refined independently for both XML and HTML,

to allow programmers to access XML and HTML-specific features. In addition to the IDL interfaces, a language-binding is specified for ECMAScript, which may serve as an example for similar bindings for Javascript and other languages, such as Java.

1.2.2 Intelligent agents

In [Negro], *intelligent agents* are characterized as autonomous, intelligent processes aiding the user in complex tasks, such as answering email, gathering information and planning activities. In practice, agents on the Internet may help in monitoring changes in Web pages, collecting information on topics of interest, or searching based on personal preferences. See [Agents], [Search]. Other types of agents, such as the *shopping agents* described in [Mobile] or even *virtual players* as those described in [VR], might become possible in the future. However, despite the range of possible examples, the notion of *intelligent agent* is not very clear. In [Survey], two definitions of *agent* are given, a soft definition, characterizing agents as autonomous processes that show some intelligence, and a hard definition attributing agents with mentalistic properties such as belief, desire and intentions. At this stage, the hard definition is clearly no more than a metaphor, since there is no technology that actually supports it. Taking the soft definition, one could argue that it is (partly) realized by modern object technology, as embodied in Java and CORBA, omitting the intelligence that is. Whether or not adopting the agent metaphor, there is definitely a challenge of making applications more intelligent, and perhaps even more human. Cf. [WebWord] and [Maes]. To my mind, one fundamental problem that we must solve to realize this goal is to define the technology, or the combination of technologies, needed to support the anthropomorphic metaphor of agents. Given the merits of logic programming in a variety of application areas, encompassing areas such as diagnostic expert systems, natural language processing, and control systems, it seemed natural to organize a workshop called *Logic Programming and the Web* to investigate how logic programming technology might be deployed to make the Web more intelligent. Nevertheless, although the presentations at the workshop indicated that logic programming could fruitfully be applied in for example the creation of virtual worlds, e-commerce applications, and intelligent rental advisors, it did not shed any light on how to bridge the gap between the (mentalistic) agent metaphor and its software realization. In the remainder of this section we will discuss the *Web Agent Support Program* research project to delineate the requirements for a framework providing agent technology support for Web applications.

Web Agent Support Program

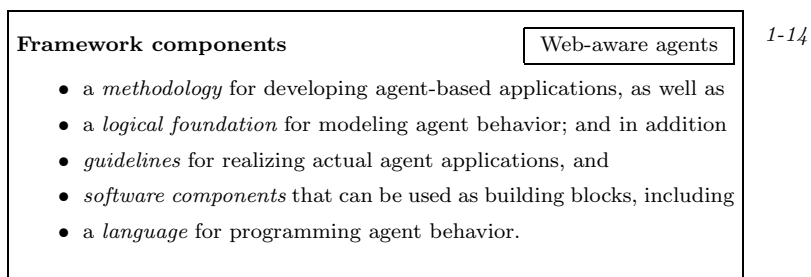
The WASP project, of which an outline is given in [WASP], concerns the development of Web Agent Support to enable average users to keep track of relevant information on the Web. The project was envisaged to result in a framework providing support for:

- intelligent navigation and information retrieval,
- information and document maintenance,
- user interfaces for Web-aware applications,
- dynamic documents with user-defined applets,
- declarative descriptions of agent behavior based on user preferences,
- declarative modeling of coordinated and cooperative behavior of software agents, and
- programming single and multi-agent systems.

As an target product for the WASP project, we envisaged developing *Pamela* (Personal Assistant for Maintaining Electronic Archives), an application combining the functional and architectural features mentioned above. In summary, our project aims at providing insight in and solutions for

- modeling the behavior of cooperating agents,
- generic means for realizing actual agents in a Web-aware context,
- architectural support for programming agent-based systems.

The aspects of our research as indicated above address the problems involved in defining and realizing the potential of the *agent* metaphor as a human–computer interface in the distributed information system domain, in particular the Web. The architectural requirements for realizing agents in a Web-aware context consist of (a) high-level support for distribution to allow for notification and the communication between agents, (b) access to the Web both in terms of server-side and client-side computation, and (c) support for information retrieval and data management.



Slide 1-14: Framework components

The WASP project is aimed to result in a framework (in its extended meaning) for the development of agent-based Web-aware applications. The components

provided by such a framework are listed in slide ???. In addition to the proper software components, the framework includes a methodology, as well as a logical foundation. Further we wish to develop guidelines for realizing actual agent applications, and our hope is to develop a language for programming agent applications, based on the language DLP, described in appendix ???.

1.3 The DejaVU experience – jamming (on) the Web

The *hush* library was originally developed to have an easy-to-use and flexible GUI library for the Software Engineering practicum at the Vrije Universiteit. New components and extensions were created by students and research assistants, including components for (Csound-based) music, video, (OpenGL-based) VRML and MIDI. Since the Web was then in its early stages, we also built a Web browser and created a number of experimental extensions to enhance the functionality of the Web with new media and communication facilities. See slide ???.

The DejaVU experience		1-15
• Applications and the Web	[Applications]	
• Bringing music to the Web	[Music]	
• Chatting on the Web	[Chatting]	
• Animating the Web	[Animate]	
• Jamming (on) the Web	[Jamming]	

Slide 1-15: The DejaVU experience

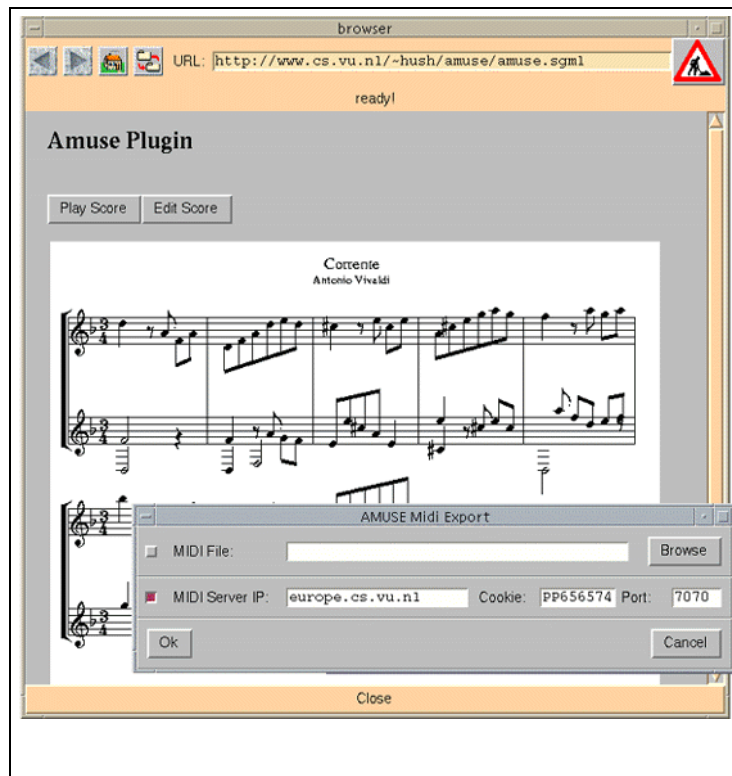
Our approach was simple but effective. First we created the components that provided the desired functionality, then we provided a script interface for these components, and finally we provided new (HTML-like) tags for the syntactic description of the new functionality. We used stylesheets to separate the syntactical description from its operational realization. These stylesheets were written in Tcl. As the Web was maturing, we did not pursue this line of research. Nevertheless, since this work still represents a valid approach, we will discuss one of my favorite extensions, an extension that allows for jamming (on) the Web.

Jamming (on) the Web

Compared to textual and graphical material, the capabilities of the Web for musical information are rather poor. The embedding of music, or sound in general, rarely goes beyond links to raw audio and MIDI files or to streamed audio connections. To display a musical work, HTML authors have to use images containing the score. All of these solutions are very low level as they basically regard music as being just sound (or a picture in the case of a score). True score files are usually a few orders of magnitude smaller, and the audio signal can be

synthesized at the client side at any appropriate sample rate. Additionally, a high-level description of music provides the browser with far more information when compared to the raw samples. In previous work we proposed to transmit musical scores (instead of the raw samples) across the Internet and to add sound synthesis functionality to Web browsers, see [Music],

and the use of generic SGML to encode structured documents, see [Animate]. In this section, we describe an experimental framework that offers many of the ingredients for true networked music support including facilities for editing, displaying and playing musical scores as well as facilities for high-level exchange of musical material and real-time collaborative work involving music and sound. Our approach is based on traditional music notation and on MIDI for playing facilities. The framework builds upon the work done in the DejaVU project at the Software Engineering section of the Vrije Universiteit, which resulted in a suite of components for developing distributed Web-aware hypermedia applications.



1-16

Slide 1-16: The score in a plugin

Scores on the Web The most ambitious markup language for the dissemination of music on the Web is probably the Standard Music Description Language, described in [SMDL]. SMDL expresses a musical work in terms of four basic

domains. The *logical domain* – the primary focus of SMDL – is, according to the standard, describable as ‘the composer’s intentions with respect to pitches, rhythms, harmonies, dynamics, tempi, articulations, accents, etc.’. The central element of the logical domain, the *cantus* element, is an abstract, one-dimensional finite coordinate space onto which musical and non-musical events can be scheduled. This allows for the inclusion of any dependent time sequences (such as automated lighting information) in a musical work. The standard uses HyTime, [HyTime], hyperlinking to specify the relations with information from the other three domains: the *gestural domain* – describing any number of particular performances (e.g. MIDI files or digital audio) of the work, the *visual domain* – describing any number of scores (a printable/displayable version) of the work, and the *analytical domain* – comprising any number of theoretical analyses or commentaries about the information in the three other domains. The addressing power of HyTime makes it possible to link directly into information expressed in other formats, including MIDI files, digital audio recordings or specific score notations, without modification. Our approach is more modest and we deploy a much simpler SGML representation, primarily geared to encode printable/displayable versions of the score (i.e. SMDL’s visual domain). However, the format used is sufficiently rich to be able to generate a playable MIDI representation as well. Information which is usually added by performers (in SMDL this is represented in the gestural domain), such as explicit interpretations of tempi, articulations and accents, are not supported in the current version.

```
<SCORE>
<TITLE>Corrente</TITLE>
<COMPOSER>Antonio Vivaldi</COMPOSER>
<STAFF>
  <MEASURE Sig="3,4" Key=F Clef=Gclef>
    <NOTE Pos="1,3" Stem=down>d6 4 0
    <REST Pos="3,6">C6 8 0
    <NOTE Pos="4,6" Stem=up>a5 8 0
    <NOTETUPLE Stem=down>
      <NOTE Pos="5,6">f5 8 0</NOTE>
      <NOTE Pos="6,6">a5 8 0</NOTE>
    </NOTETUPLE>
  </MEASURE>
  ...
</STAFF>
</SCORE>
```

To support display and editing of SGML scores on the Web, we developed the Amuse score editor as a plugin for our Web browser (see slide ??). The editor has a graphical user interface and does not require any SGML knowledge from the user.

Above is a fragment of an example score file, for which the associated style sheet with a CSS1-like syntax is shown below. Both documents can be edited by the graphical score editor plugin. Changes in the style sheet are dynamically

reflected in the display of the score. A significant enlargement of the page-width parameter, for example, will allow for more measures on a single staff, and will result in a redraw of the complete score.

```

SCORE {
  margin-left : 30;
  margin-right : 30;
  margin-top : 80;
  margin-bottom : 20;
  page-height : 1000;
  page-width : 920;
}
TITLE {
  title-align : Center;
  title-font : -*Times-Bold-R-Normal-*240-*;
}
COMPOSER {
  composer-align : Center;
  composer-font : -*Times-*R-Normal-*180-*;
}

```

1-17

Slide 1-17: An associated style sheet

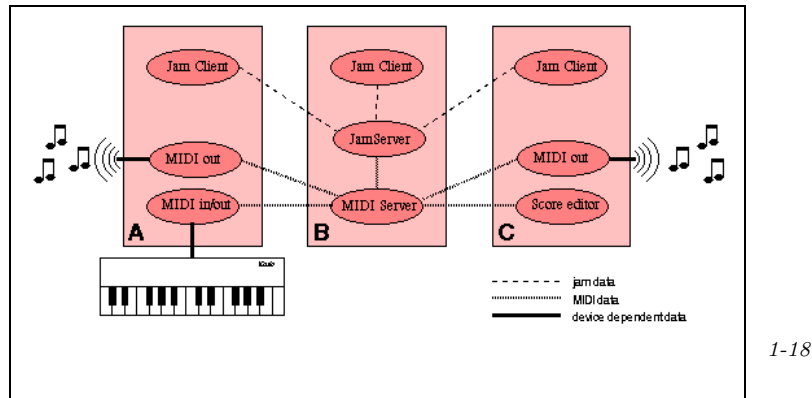
Playing on the Web The playback facilities of our framework are centered around the *MIDI server*. After registering as a MIDI client, the score editor is able to send the generated MIDI version of the score to the separate MIDI server. The MIDI server builds upon a socket-level client/server library and a class library that provides the basic functionality for MIDI devices, MIDI clients and the MIDI server. Note that the audio device is usually an exclusive resource, and by connecting to a single MIDI server, several client applications can have simultaneous access to a single MIDI output device. The functionality of the MIDI server comprises:

- registering and unregistering MIDI devices,
- routing MIDI data between clients and MIDI devices, and
- administration and security checks.

When a MIDI device is registered, a *cookie* is given out that may be used by a client to request the server to set up a virtual connection with that device. The cookie also prohibits unauthorized clients from accessing a MIDI output device.

Collective improvisation We developed the keyboard applet, depicted in slide ??, as an alternative input device to be able to send ‘live’ MIDI data to our server.

Since multiple applications can have access to the MIDI server, a user can have a score edit session running, and simultaneously be playing a keyboard applet. To engage in a jam session, the keyboard applet connects to the *JamServer* instead of the MIDI server. The *JamServer* acts as the central point of a jam session, keeping track of all clients engaged in the session. To start a jam session,



Slide 1-18: The jam server

all jam clients connect to a single *JamServer* and send it their MIDI data. The *JamServer* is connected to one or more MIDI servers, as depicted in slide ??.

By having the *JamServer* separate from the MIDI server itself, the latter is relieved from the burden of jam session management. Every connected MIDI device will receive all the MIDI data submitted by the jam clients. This data is relayed to these devices by the MIDI server(s), through the virtual MIDI data stream that is created when registering as a jam client. In slide ??

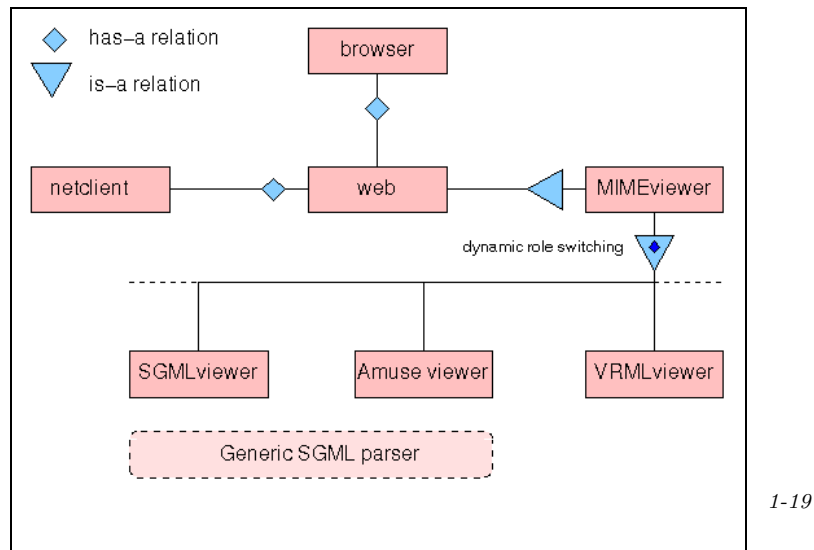
we see three jam clients connected to a single *JamServer* (on machine B). The MIDI server is running on the same machine as the *JamServer*. Both the clients on machine A and C have registered a MIDI-out device (a software sound synthesis MIDI program developed for Solaris)

with the MIDI server on B. The user on A has additionally registered a MIDI-in device (the keyboard). Using the keyboard, the user on A can contribute to the jamming. The score editor on C is directly connected to the MIDI server and is not engaged in the jam session. The MIDI server will redirect MIDI requests from the score editor only to the MIDI device on C.

Measurements To give an indication of the speed and response times of our system, we have used a special jam client, *jamping*, that measures the average delay between sending a MIDI message to the *JamServer* and receiving the same message on a connected MIDI device. For a 486DX2-66 PC with Linux with one client and both servers local, this resulted in a round-trip-delay time of 5.5 milliseconds. A similar setup on a Sparc-5 with Solaris resulted in 2.6 milliseconds. A similar configuration with the *JamServer* on a LAN gave 3.5 milliseconds

average round-trip-delay time. Nevertheless, with a server in Amsterdam and a client in Sweden, we obtained an average round-trip-delay time of 87 milliseconds, with a peak of 1.6 seconds. Clearly, the length and variability of round-trip-delay times may be a prohibiting factor for jamming on a global scale.

Architecture of the Web components The software described so far was developed for our SGML-based Web browser as an extension to the *hush* class library, [Animate].



Slide 1-19: Web components

In slide ?? an overview is given of the basic Web-related components of the *hush* library. The browser provides the top-level user interface for all Web components, including a viewer, a scrollbar, navigation buttons (back, forward, home, reload) and an entry box to enter URLs. The netclient, web and MIMEviewer components form the conceptual base of our approach of connecting to the Web:

- *viewer* – a widget for the inline display of several MIME types, such as HTML, VRML and Amuse score formats.
- *web* – an extension of the *MIMEviewer* with history and caching.
- *netclient* – the interface to the Internet, supporting several protocols.

The MIMEviewer component provides an abstract interface to viewers for several MIME types. The web widget only knows about the (abstract) MIMEviewer class while the actual functionality is implemented in several concrete viewer classes, one per MIME type. Specific viewers for new MIME types can be

plugged dynamically into the MIMEviewer object. When the MIMEviewer gets the instruction to display a document of a certain MIME type, it changes its role and becomes a viewer for that particular MIME type. This dynamic role-switching idiom is discussed in more detail in chapter 2. As a result, the addition of new viewers can be done without changing the web widget. The netclient component builds the bridge between the local web widget and the World Wide Web by providing an abstract and uniform interface to network (file) access and transport protocols. In the realization of the netclient components we have employed the dynamic role-switching idiom in the same way as in the implementation of the MIMEviewer components. The web object creates a MIMEviewer object and tells which role it should play (e.g. SGML, Amuse or VRMLviewer). This role can be changed during the lifetime of a single MIMEviewer object by calling a method to change its role. A browser typically uses only one single MIMEviewer object that changes its role according to the type of data that should be displayed. The SGMLviewer is the default viewer, it displays generic SGML documents by using style sheets for each document type. By default, a style sheet for HTML is used. Since our generic SGMLviewer is better suited to textual documents and does not offer editing support, we developed a separate viewer/editor to process our Amuse/SGML score files. Since the MIMEviewer provides no network functionality at all, it generates events whenever it needs to retrieve data pointed to by a URL. Such events are generated as a response to user interaction (e.g. clicking an anchor) or to fetch inline data during the parsing process. These events are typically handled by the web component which plays a central role in our approach because it combines the functionality of the MIMEviewer and the netclient components. Additionally, the web component adds a history and caching mechanism to the MIMEviewer. The web component's behavior is similar to the standard widgets of the *hush* framework, and can be conveniently used as a part of an application's GUI. Because the web widget has both a C++ class interface and a script interface, it is easy to create, or extend, applications with Web functionality.

1.4 Software architectures revisited

The Web is, at the time of writing, still in flux. Yet it is becoming more and more the standard infrastructure on which applications are built. A recurring question is, '*how do we build Web applications?*'. There is no definite answer to this question. There is no body of solutions that may serve to indicate proven practice. But there is, definitely, a convergence towards *objectifying*, or *object-orienting* as it is called in [OOWeb], the Web and its applications. Anyway, the following quote, taken from [Practice], p. 10, says it all.

It is a brave architect who, in today's environment, does not develop, or at least consider, an object-oriented design.

Clearly, the architecture of the technological infrastructure of the Web, as well as the architecture of Web applications, may benefit from an object-oriented

approach. Nevertheless, knowing this, we still do not know how to build actual Web applications.

Architectural software styles	<div>class of architectures</div>	1-20
<ul style="list-style-type: none"> • <i>component types</i> – process, event, repository • <i>runtime relations</i> – topology • <i>semantic constraints</i> – immutability • <i>communication and coordination</i> – connectors 		

Slide 1-20: Architectural software styles

From the perspective of software architectures, we may ask ourselves what architectural style, or for that matter which mix of architectural styles, we may deploy for building such applications. As a reminder, an architectural style, which characterizes a class of software architectures, consists of a description of the types of components used (processes, events, repositories), the (runtime) relations between these components (for example the network topology), possible semantic constraints (such as the immutability of particular components) and properties concerning communication and cooperation (such as the connectors or protocols used).

Themes and variations	<div>technological constraints</div>	1-21
<ul style="list-style-type: none"> • OO – simple call and return • CORBA – independent components • WWW – data centered • events – independent components • logic – virtual machine architecture 		

Slide 1-21: Themes and variations

A rather simple-minded categorization of architectural styles, reflecting obvious technological constraints, is given in slide ???. Each of the styles is characterized by a single phrase capturing a central feature of the style. For example, an OO approach may be characterized by the fact that it embodies a simple call and return mechanism, which, by the way, gets its power from the fact that it concerns methods or, in C++ jargon, virtual functions. Events have proven to be an excellent means to obtain a high degree of independence between components. And logic, as has been argued in section ??, may be used to promote a clear separation between knowledge-level and system-level aspects of a system, by embedding a (virtual) logic machine. The categorization is, however,

simple-minded because, as we may observe in retrospect, most of the applications discussed contain elements of at least a couple of the styles mentioned. So, instead of discussing one style, we need to consider a mix of styles, and determine what mix of styles may be effectively used to create the applications we have in mind.

The architecture of the Web

To return to the Web, why is the notion of software architecture important?

As indicated in slide ??, for one, the Web is still growing at a rapid rate, and it is becoming increasingly important economically. So, we are faced with the problem of managing this growth, and, much sooner than we could have expected, with the problem of maintaining the applications that populate the Web.

Architectural issues

- managing growth, maintaining installed base
- enhanced functionality – *synchronized multimedia*
- improved technological infrastructure – *HTTP-NG*

1-22

Slide 1-22: Architectural issues

Secondly, the Web is continuously enhanced with new functionality, including, for example, *synchronized multimedia* as proposed in the SMIL standard, see [SMIL]. Consequently, with respect to the technological infrastructure, we need to be able to accommodate new requirements, such as *quality of service*, needed for the timely delivery of multimedia material. And thirdly, many attempts are being undertaken to improve the quality of the infrastructure itself, as exemplified by the HTTP-NG effort, which aims at higher speeds and a state-full communication protocol, see [HTTP-NG]. As clearly stated by the Web's principal architect, Tim Berners-Lee, graceful extensibility has always been one of the primary goals in developing the architecture for the Web. In this respect, the Web differs significantly from other distributed technologies, such as CORBA, which does not allow for non-compliant extensions. In contrast, the Web does to a great degree allow for non-compliant extensions simply by ignoring them, until they become a standard. The challenge, then, from an architectural point of view, is to come up with better standards and better technologies without sacrificing the extensibility allowed by non-strict technologies such as HTML and HTTP.

Plugin architectures

To conclude this chapter, I would like to discuss briefly an extension mechanism that has proved to be invaluable for developing Web applications, the *plugin architecture*. Plugin architectures are becoming more and more popular, for 'ordinary' tools such as Adobe Photoshop and Macromedia Director. In a Web context, the most notable examples are Netscape Navigator and Microsoft Internet Explorer,

which both provide a facility to extend the browser with new functionality that is available in a dynamically loadable library.

<i>Client NPP/Callbacks</i>	<i>Browser NPN/Calls</i>
Instantiation and Destruction	Version Info
Stream Notification	Stream Creation and Destruction
Reading and Writing Streams	StreamAsFile
LiveConnect	

Plugin architectures are realized by using callbacks, in the same way as in object-oriented frameworks. Above, an overview is given of the callback functions required by the Netscape plugin architecture. These functions must be implemented by the (plugin) client, so that the browser can recognize and activate the plugin. Such callbacks encompass instantiation and destruction functions, notification when a stream is ready, functions for reading and writing streams, and the *Live Connect* functions, which enable the (plugin) client to communicate with Javascript functions and Java applets that are currently active. The browser, in return, provides convenience functions to obtain version information, to create or destroy streams or to store the contents of a stream in a temporary file. It should be noted that the actual API for the creation of (Netscape) plugins is not object-oriented, although a partial class library is available to create plugins in an object-oriented manner.

Nevertheless, ignoring details, plugin architectures indicate what may become the dominant paradigm of the future, framework-like environments that are extensible by components following a clearly defined pattern or protocol; that is to say, components created according to the principles of object-oriented software development.

Summary

This chapter discussed the relevance of object-oriented technology to the development of Web-applications.

Objects and the Web <div>1</div> <ul style="list-style-type: none"> • <i>trends and technologies</i> – client/server + extensions • <i>ObjectWeb</i> – CORBA/Java vs Microsoft 	1-23
---	------

Slide 1-23: Section 12.1: Objects and the Web

In section 1, we looked at trends and technologies, in particular the ongoing creation of the *ObjectWeb*, which is essentially an ongoing war between Microsoft and the rest of the world.

In section 2, we discussed the model of computation underlying the Web. We looked at the requirements we may have for APIs, and we explored the notion of intelligent agents on the Web,

Programming the Web – a search for APIs

2

1-24

- models of computation
- intelligent agents

Slide 1-24: Section 12.2: Programming the Web – a search for APIs

The DejaVU experience – jamming (on) the Web

3

1-25

- *animating the Web* – an SGML-based approach
- *bringing music to the Web* – data formats + client-side plugin
- *jamming (on) the Web* – additional communication servers

Slide 1-25: Section 12.3: The DejaVU experience – jamming (on) the Web

In section 3, some of the research efforts carried out in the DejaVU project were presented. In particular, we looked at an SGML-based approach to extend the Web with new media and communication facilities.

Architecture revisited

4

1-26

- OO – simple call and return
- CORBA – independent components
- WWW – data centered
- events – independent components
- logic – virtual machine architecture

Slide 1-26: Section 12.4: Architecture revisited

Finally, in section 4, we discussed some remaining architectural issues. We concluded that many of the applications discussed in this book draw from a mixture of technologies and architectural styles.

Further reading

For information concerning the Web, have a look at <http://www.w3c.org> which give a detailed account on the history of the Web and many other issues. For an exposition of the issues and technologies that play a role in the battle for the ObjectWeb, consult [OWSurvival]. A good introduction to agents and its associated technology is given in [Survey]. For architectural issues, again, I recommend [Practice].

Questions*1-27*

1. Describe the architecture of the Web. Explain the relevance of objects for the Web.
2. Sketch the Microsoft approach to the ObjectWeb. Discuss its pros and cons.
3. In what ways can Java and CORBA be deployed in Web applications?
4. Indicate how the computation model underlying the Web deviates from the computation models underlying, respectively, object systems and client-server systems.
5. What requirements can you think of for libraries or frameworks for developing Web applications?
6. Discuss the Document Object Model.
7. What are the requirements for a framework supporting intelligent agents?
8. Explain the issues that arise in extending the Web with additional media functionality. What solutions can you think of? Can you give an example?

Slide 1-27: Questions