Vrije Universiteit Brussel
Faculteit van de Wetenschappen
Departement Informatica
Academiejaar 1999-2000

Valideren van User Input in Web Applicaties

Plasqui Jan

Promotor: Prof. Dr. O. De Troyer

Eindverhandeling ingediend met doel
het behalen van de graad van
Licentiaat in de Toegepaste Informatica

Samenvatting

Het *World Wide Web* kent de laatste jaren een sterke groei.
Niet alleen het aantal websites is sterk toegenomen, er worden ook steeds nieuwe internet-technologieën ontwikkeld.

Moderne websites laten toe dat bezoekers het uitzicht en/of de inhoud van de website kunnen personaliseren door gebruikersprofielen op te slaan.
De overgrote meerderheid van de bedrijven willen nu ook gebruik maken van *e-commerce.* Banken geven hun klanten de mogelijkheid om aan internet bankieren te doen: via het web kan het rekeningssaldo opgevraagd worden of kunnen overschrijvingen gemaakt worden. Winkels bieden de mogelijkheid om via het web producten te bestellen en te betalen, sommige winkels verkopen uitsluitend via het web en met veel succes.
Andere bedrijven trachten dan weer mensen naar hun website te lokken door allerlei gratis diensten zoals e-mail of het zenden van SMS berichten aan te bieden.

In de bovenstaande voorbeelden moeten er gegevens die door de gebruiker zijn ingevoerd d.m.v. de webbrowser naar een server gestuurd worden. De server stuurt daarna weer gegevens terug naar de browser. Gebruikers kunnen echter foute en/of onvolledige informatie ingevoerd hebben. Daarom moeten we ons de volgende vragen stellen:

1. *hoe* kunnen we nagaan of de gebruiker van een web applicatie correcte data ingevoerd heeft
2. *wanneer* gaan we dit nagaan

Gegevens invoer van een gebruiker van de web applicatie kan ofwel nagekeken worden op het moment van invoeren (in de browser) of zodra het naar de server gestuurd is. In het laatste geval is er echter een extra verbinding naar de server nodig. Het valideren van user input van web applicaties langs de server kant kan vrij complex worden.

In deze thesis zal uitgelegd worden hoe web applicaties nu input valideren en zullen andere oplossingen bestudeerd worden die de overlast van data verkeer en processing op de server verminderen. Er zal een beschrijving van mogelijke beperkingen (constraints) op user input gegeven worden en voor elke constraint zal besproken worden hoe deze geverifiëerd kan worden.

Vrije Universiteit Brussel
Faculty of Science
Department of Computer Science
Academic year 1999-2000

Validating User Input in Web Applications

Plasqui Jan

Promoter: Prof. Dr. O. De Troyer

Dissertation submitted in view
of obtaining the degree of
Licentiate in Applied Computer Science

Abstract

The *World Wide Web* has known an enormous growth in the last few years.
Not only the number of websites has increased but also new web technologies were produced.

Websites now allow users to make a personalized view of the information they carry by keeping user profiles.
The majority of companies nowadays want to use *e-commerce*. Banks offer their clients the possibility for web banking; shops let their customers order and pay via the web, some shops only sell online and with great success.
Other companies try to attract people to their web site by offering some free service like e-mail or the possibility to send SMS messages.

In all of the above examples users need to enter information that needs to flow from the user's browser to a server and back. But users may enter incorrect or incomplete information. The questions that arise from these new types of web applications are:
1. *how* can we check if a user has entered correct information and
2. *when* are we going to check this.

User input can be checked upon entering (in the browser) or when it arrives on the server but this requires an extra connection to the server. For larger applications, checking information residing at the server may complicate the validation of user input.

In this thesis I explain how user input is currently verified and investigate other solutions that will reduce the load of traffic and processing on the web servers. Also a detailed description of different kinds of constraints on user input will be given. For each kind of constraint it will be discussed how it can be checked.

Acknowledgements

I would like to express my gratitude to my promoter, Prof. Dr. Olga De Troyer, for her guidance and encouragement during the accomplishment of this thesis.

I would also like to thank my parents for giving me the opportunity to study.

Furthermore I want to thank my sister and my friends for their help and support during my studies.

Table of contents

List of figures

1. Introduction

1.1 History and evolution of the World Wide Web

The roots of the Internet go back to the 1960's and the height of the cold war. The U.S. military, in preparation for a possible nuclear war, sought a means to ensure communications in the event of an enemy missile attack. The communications network needed to withstand large-scale destruction and at the same time deliver uninterrupted service.

The main problem with the existing system was that a direct hit on a central point of control would disable the entire network. The RAND Corporation came up with the idea of building a network without a central point of control. In this way, the system would not be vulnerable to a direct hit on a single location.

To accommodate this requirement, a network was devised that allowed data to flow around destroyed components. A special communication standard, called *TCP/IP*, (Transmission Control Protocol/Internet Protocol), was designed to direct the flow of data between computers on the network and around possibly damaged sections of the network. Thus, TCP/IP increased the survivability and reliability of the network.

In 1969, a group of Department of Defense researchers working for the Advanced Research Projects Agency linked computers at UCLA, Stanford Research Institute, the University of Utah, and the University of California at Santa Barbara to create the network. This non-centralized networked was called ARPANET (Advanced Research Projects Agency Network).

At first, the military researchers used ARPANET to discuss government projects, sending electronic messages (email) across the network. However, these researchers soon discovered that email was a very convenient way to discuss topics far outside even the most liberal interpretations of research-related actives. They created email programs that automatically sent the same message to everyone on a list. Email lists, or list servers, enabled entire groups of like-minded researchers to share their interests.

The original APRANET community grew from 4 institutions in 1969 to more than 50 universities and military agencies by 1972. The ability of APRANET users to interact and share the latest information was driving ever-greater use of the network. Non-military scientists were pressuring for access to the network too but ARPANET's acceptable use policy prohibited those outside the military establishment from using it. So, in 1983, ARPANET split into two networks; one to handle all scientific traffic and another, MILNET, to carry just military information.

These large networks, along with many small local networks, were woven and interconnected into a network of networks. Early on in the 1980's, this collection of networks was called the APRA Internet, but it eventually became known as just the Internet.

In the 1986, the National Science Foundation established the NSFNET to link supercomputers at high speed. NSFNET became the backbone of the Internet, offering transmission speeds of a million-bits-per-second rate. The acceptable use policy was further expanded to include almost everything except commercial activity.

The origins of the World Wide Web came about in March 1989, when Tim Berners-Lee of the European Particle Physics Laboratory (known as CERN, a collective of European high-energy physics researchers) proposed a project to be used as a means of sharing research colleagues in the organization. The project was envisioned to include a system of networked hypertext documents to be transmitted among members of the high-energy physics community. By the end of 1990, the first piece of Web software was developed with the ability to view, edit, and send hypertext documents to colleagues via the Internet. The Web was born.

In 1991, the U.S. Congress passed the High Performance Computing Act to establish the National Research and Education Network (NREN). The goals of NREN were to experiment and establish high-speed, high-capacity research, education networks, and to not only allow commercial activity on the Internet but to find ways to encourage it

Although the commercial restrictions on the Net where effectively removed by the 1991 act, it remained primarily a tool for researchers and academics because of the complexity in using it for communications. However, in June of 1993 Marc Andreessen, and other researchers at the National Center for Supercomputing Applications (NCSA), released a graphical Web Browser - Mosaic 1.0 for X Windows. It was soon followed by a version that would run on Microsoft Windows (the dominant desktop operating system).

In 1994, Andreessen left NCSA to form a new corporation with Jim Clark. The company created a much easier-to-use and faster Web browser called Netscape Navigator. The general public went wild over Navigator, making Netscape Communications the fastest growing software company in history. Business, sensing the opportunity, began a mad rush to establish a "presence" on the Web.

Before long, one couldn't listen to a radio, watch TV, read a magazine, or glance at a newspaper without being inundated with information about the growth and potential commercial opportunities of the World Wide Web. People and companies were signing up by the millions, with Internet service providers, to gain access to the Web. They wanted to participate in this new global communication feast. The Web soon became the dominant service, with the exception of email, on the Internet.

1.2 Web applications

Some believe a web application is anything that uses Java, others consider a web application anything that uses a web server. The general consensus is somewhere in between.

[C 99] defines a web application as a web system (web server, network, HTTP, browser) in which user input (navigation and data input) affects the state of the business.

I will follow this definition of web applications as programs that run on servers and that interface with the users via web browsers through e.g. forms or java applets. A web system is used as the front end of a web application.

The architecture of web applications is quite similar to the architecture of a regular client/server system. The biggest advantage of web applications is the ease of deployment. It is sufficient to set up a web server and installing the server side applications needed. The client can then simply use the web application via his web browser and doesn't need extra software. Depending on how the web application was built, the user might need a specific browser and/or installed plug-ins to properly use the application.

The application domain of such applications is somewhat limited because of the *connectionless* nature of the Internet.
IP or *Internet Protocol* is called connectionless because it resembles the Postal Service more than it does the telephone system. When a node using IP wishes to send a message to another such node, it simply sends the packet, properly addressed, analogous to mailing a letter or sending a telegram. (IP packets are also called datagrams) The telephone system, on the other hand, creates a *connection* between two users that is maintained for the duration of the information exchange. Unlike the Postal Service, however, the services of IP can be used to create a connection-oriented operation mode, but this is the job of higher-level protocols and applications (such as TCP, File Transfer Protocol (FTP), Telnet, and others). In IP's connectionless design, every packet is treated completely independently from all others.

HTTP (HyperText Transfer Protocol) is the client-server TCP/IP protocol used on the Web for the exchange of HTML documents. It is designed for robustness and fault tolerance instead of maximum communication throughput. All HTTP transactions go by the same general method. Each client request and server response has three parts: the request or response line, a header section, and the entity body. To retrieve a HTML document, the client sends a document request by specifying an HTTP command, e.g. the GET method. The client sends header information: browser information and mime-types of acceptable files. Multipurpose Internet Mail Extensions (MIME) is a specification for formatting non-ASCII messages for transmission over the Internet. MIME types enable the browser to display or output files that are not in HTML format. After sending the header information, the client can send additional data to the server. This is used for sending variables to CGI's using the POST method. The server responds to the client's request by first sending a status line, followed by a header containing

information on the server and on the requested document. The transaction is finished after sending the requested data or a generated error message if the document was not available. Clients can also request a *keep-alive* connection to the server to avoid repeated connections for fetching a single page containing inline images etc.

The consequence of connectionless data transmission is that servers cannot remember state. If a user sends information to the server in several steps (e.g. by means of a number of forms which are sent one by one to the server), this means that each time the server receives a form, it doesn't know what the user has filled-in in the previous forms. To overcome this lack of "memory" a number of techniques are currently used.

When a registration form requires too much data to fit on a single HTML form, it can be split into multiple forms. This requires multiple connections to the server and the input from previous forms needs to be remembered.

The following techniques are available to keep state in stateless connections:

- Cookies can be used to store previously entered data and control information in the users' browser.
- Hidden data can be included in later forms presented to the user.
- Data can be pasted as arguments in URLs.
- HTTP Sessions is a service for web servers that allows keeping track of users.

Common Web applications are:

- *Online stores* that allow users to buy items through the Internet. Such an application keeps track of what en how much items a user selects to buy and handles some form of payment.
- *Email service* that allows users to edit a message, send it to another person and let the user manage his mailbox. This is different from an online store because the state, i.e. the mailbox is always kept on the server in the form of a database. An online store will not store an order into the database each time the user selects an extra item but it will wait until the whole order and payment details are confirmed.
- *File storage service* that allows simple file management. Files can be uploaded to the server, downloaded and shared with other users of the system.

- *Discussion forums* allow users to post messages and reply to others.
- *Newsgroups* where users can read the messages from a selected newsgroup, have a hierarchical view of the threads and respond to any message.
- Sending *E-Cards*, a small application that lets a user customize the look of an electronic postcard and have it delivered to a specified e-mail address.
- …

1.3 Description of the problem

In e.g. database applications, it is common practice to check the data for correctness as soon as possible. In classical applications this is done often as soon as the user enters a value in a input box and confirms his input by either clicking on a validate button or when focussing on another input box. Validating user input as fast as possible has several advantages: it enhances the usability of the application by giving direct feedback as soon as the user performs an action. In case of an illegal action, appropriate error messages concerning the faulty action can be generated. Giving direct feedback minimizes the user's memory load. Human short-term memory has a limited capacity of about +/- 7 chunks of information, which will fade away after 2 to 3 seconds. Giving clear feedback to the user of what is already done, of the result of actions and on what he still has to do reduces the use of short-term memory.

When working with web applications, the application (the user's browser) will always have to make at least one connection to the server i.e. to send the data.
However, if the user enters incorrect or incomplete data, the server will send the form back to the user for correction. The user's browser will then again send the data to the server until all of the input is correct. All this means extra server connections and an increased workload for the server. A suitable mechanism should be provided to generate appropriate error messages in case of multiple errors.

Since many web applications are also database applications, it is necessary to maintain database integrity. Incorrect user input can invalidate the data present in a database and this should be avoided by validating the entire user input before inserting any data in the database.

1.4 Current ways to validate user input in web applications

As an example we take a small application to register people for some service via the web. Such a service could e.g. be an Internet mail service.
To register himself, a new user has to enter a lot of personal data like his name, address, occupation and general interests. All this information will be used later on for sending the user commercial messages that are more likely to interest him.

The least efficient way to get the input from the user is by asking him to fill in 1 field and have him send it to the server to check if the input is acceptable, and then present a page with the next field to continue or to retry the previous field in case of an error.
This method uses a lot of server connections and is also not appealing to the user from an interface point of view.

A better and commonly used way of getting user input is by grouping logically related input fields into one form. Then the user can enter all required information on one form and send all this information at once to the server, thus making fewer server connections when submitting the data. When splitting up a form into several smaller forms, the size of those new forms can introduce new problems. If the form size is too small, e.g. only a few input elements on a form and thus possibly followed by a fairly large number of these forms, the user will have to make too many connections to the server which will be frustrating, especially if that user is equipped with only a slow (telephone) modem connection. If, on the other hand, the forms contain too many input elements and dependencies among these input elements, the user can be asked to fill in all the fields again after making only one mistake.

In both cases, the input validating process becomes a little more complex. When the server notices the form contains invalid input it can either ask the user to re-fill the form completely or it can accumulate all faulty fields and generate a new form for those. In this last approach, the user can see what fields were wrong and need to be corrected.

Both methods for re-entering faulty data are a form of delayed validation because the entered data is not immediately checked for correctness, as it is mostly done in classical computer applications.

2. Types of errors in user input

Generally, there are two ways a user can give input to a web application. One would be performing an action on some state object by clicking a button to get the desired effect. An example of such an action could be clicking on a *'send mail'* button in an e-mail application.
Another way of user input consists of a user filling in information fields, checking options and/or selecting values from a given list. An example of such user input could be a user entering some personal details and his credit card information to purchase some items via the web. Both kinds of user input are susceptible to errors.

Errors can occur in many forms:

- Incomplete input when a user forgets to fill in a required piece of information needed by the web application

- Inconsistent input when a user enters contradicting information, this can happen when there are dependencies between some of the required input fields

- Incorrect input when a user enters logically wrong data in a form, an example could be a user who mixes up the field to enter his first name with the field for his last name. Incorrect input will be further explained by comparing it to database consistency errors.

- False input when a user intentionally enters false information; the information he gives is complete, consistent and correct but is not true. This is common practice when users sign up for 'free[1]' services to stay anonymous or to avoid *spam* (unwanted advertisement).

- Object/State errors when a user requests an action, possibly on an object, which is invalid under the current state of the application. An example could be a user trying to send an e-mail without specifying a recipient.

2.1 Incorrect input and database consistency

Many web applications are also database applications. Therefore I will give a short introduction to databases and explain the problem of storing invalid information in a database and mechanisms available to validate data before insertion. Finally, the similarities between database consistency constraints and incorrect user input in web applications will be given.

2.1.1 Database terminology

[1] Many companies provide free services like e-mail or webhosting to non-commercial users. The user doesn't have to pay for these services but instead he either receives some form of personalised advertisement or automatically promotes the company to others.

A *database* can be seen as a shared collection of logically related data (and a description of these data). This means that not only the entities and the attributes but also the logical relationships between entities are to be presented in the database ([V 99]).

In a relational database, the data is organized in tables (also called relations), columns represent attributes of a certain type and rows (or tuples) represent an entity for that relation.
A *base relation* is a named relation, corresponding to an entity in the conceptual schema, whose tuples are physically stored in the database.

A *null* represents a value for an attribute that is currently unknown or is not applicable for the tuple. Nulls are a way to deal with incomplete or exceptional data. However, a null is not the same as a zero numeric value or a text string filled with spaces because those are values and a null represents the absence of a value.

An attribute or a set of attributes that uniquely identifies a tuple within a relation is called a *superkey*. A superkey such that no proper subset is a superkey in the relation is a *candidate key*. The candidate key that is selected to identify tuples uniquely within the relation becomes the *primary key*, the other candidate keys are called alternate keys.
Duplicate values are not allowed for the primary key.

A *foreign key* is an attribute or set of attributes within a relation that matches the candidate key of some (possibly the same) relation.

2.1.2 Data integrity

Data integrity is closely associated with data security or database security, and to properly safeguard a system both sets of controls are essential ([CBS 96]). Data integrity or validity and consistency of stored data means that for example a table that is designed to store employee names actually contains strings referring to existing names. The database can also make sure that only the names of employees that currently work with the company are stored in that table. Data security is the protection of the database from unauthorized users. This is a very important aspect in databases underneath a web application because the database might contain confidential and critical business information or information that falls under the users right of privacy and is also not to be seen by others.
If security controls exist without any integrity controls, the reliability and validity of the data rely entirely on the authorized users' correct use of the database. If, on the other hand those integrity controls exist, data is guaranteed to be consistent.
There are several aspects in considering how to ensure a database maintains data integrity.
Although data integrity may be preserved, it does not absolutely guarantee that the data is correct. It is very difficult to check data correctness automatically, unless

there is a simultaneous checking mechanism such as a dual data entry system[2]. However such a dual data entry system would not be feasible in a web application as a user will have to enter personal data. Having a user to enter the same data twice would not only annoy him and keep him from wanting to use the application but it can't prevent a user to (sometimes intentionally) enter incorrect data twice.
To maintain data integrity, it is necessary to have appropriate constraints on the manipulation of the data, particularly on insert, update and delete. In many instances, these constraints may be stored in the DBMS (DataBase Management System). Ideally, the DBMS would facilitate the handling of all required controls, but to what extent this is possible depends upon the DBMS being used.

There are 3 types of constraints: referential constraints, domain constraints and enterprise constraints.

2.1.3 Referential constraints

A first requirement is entity integrity. This constraint does not allow null values in attributes of a primary key in a base relation. If one would allow nulls to appear in a primary key the result would be loss of information and/or rendering the remaining information meaningless.

If a foreign key exists in a relation, either the foreign key value must match a candidate key value of some tuple in its home relation, or the foreign key value must be wholly null.
This constraint states that if a value exists as a foreign key in a relation, then it must match the value of an existing primary key in another relation, or else be wholly null.

A number of problems can arise when enforcing referential integrity.

Consider the next tables as a small example.

`PersonLocation`

PersonID	Street	AreaCode
55486	Walenhoekstraat	1910
12345	Nieuwstraat	1024
56789	Parkstraat	1024

`City`

AreaCode	City
1024	Brussels
1910	Kampenhout
1830	Machelen

A tuple in the table PersonLocation has a foreign key AreaCode that refers to a tuple in the table City.

If we for example delete the tuple [1024, Brussels] from the table City, a problem arises in the last two tuples in the table PersonLocation where the AreaCodes no longer reference an existing value in City.
The same would happen if we would decide to use another area code for Brussels.

[2] Administrative database applications often require that the same form is entered twice by different users of the system. This allows error detection by comparing both inputs.

Suppose Kampenhout would get another area code and Brussels get the code 1910 (which is not very likely to happen in reality). This will make every reference in the table PersonLocation that originally referred to Kampenhout now point to Brussels.

Database systems allow the designer to specify how the DBMS should ensure referential integrity when a primary key is updated or an entire tuple is deleted.

- The primary key value does not occur anywhere as a foreign key:
  - Allow the operation to take place
- The primary key value does occur elsewhere as a foreign key:
  - Do not allow the operation to take place.
  - Allow the operation to take place, and also set the foreign key occurrences either to null or to a default value if one has been specified.
  - Allow the operation to take place, and also
    - in the case of update propagate the changed value to the foreign key occurrences, and where the foreign key forms part of the primary key of this relation, propagate the changes where this primary key is also a foreign key another relation.
    - in the case of delete propagate the deletion, that is, delete the tuples that have a foreign key value matching the primary key, and where the primary key of these deleted tuples is also a foreign key in another relation.
  - Enter into dialog with the user.

[CBS 96]

Referential constraints can also be implemented in the application using the database if the DBMS used doesn't support this.

2.1.4 Domain constraints

A domain constraint is the set of allowable values for one or more attributes.

A first level of domain constraints is the native data types supported by the DBMS. When defining relations, a data type has to be provided for every

attribute. The domain of the native data type is determined by the characteristics of the system.
A character data type will be able to assume all the available characters present in an implementation-defined character set, such as ASCII. The legitimate operations that can be carried out on these values may include comparison but exclude operations such as addition and subtraction, which usually belong to numeric data types.

Another kind of domain constraints is a set of allowable values for an attribute. Many database systems allow enumerating all valid attribute values when creating the relation.
If a database permits the creation of named domains, it would be possible to define for example a domain *Gender*, which allows either *M* or *F*. This domain name could then be used to specify the type of an attribute in a relation.

Depending on the capabilities of the DBMS used, domain constraints may need to be implemented in the application itself.

2.1.5 Enterprise constraints

Enterprise constraints are additional rules specified by the users or database administrator of a database. Other names for enterprise constraints are application constraints, business rules and assertions.

Enterprise constraints can be implemented with the SQL *Assert* statement, although not all database systems support this yet. An assertion is a named constraint that may relate to the content of individual rows of a table, to the entire contents of a table, or to a state required to exist among a number of tables.

Cardinal constraints, constraints that indicate how many tuples a relation can hold, can also be seen as a form of enterprise constraints.

2.1.6 Applying database constraints on user input

- Incomplete information can conflict with referential constraints because attributes from a (primary) key may be missing. If we would allow the DBMS to store a key with missing attributes (nulls), the tuple can become impossible to be referred to. Tuples can lose the property of being uniquely identified by their primary key if this key contains nulls.

- Inconsistent information can conflict with referential constraints when a tuple is inserted that contains an illegal or missing foreign key value.

- Incorrect input can conflict with domain constraints when the entered data is considered to be of a different or incompatible data type of the attribute’s domain as specified in the table definition. When user input is of the correct

data type but falls beyond the boundaries of allowed values for the attribute, domain constraints will be violated as well. For example, entering some alphabetic characters for a numeric data field or entering an unrealistic numerical value for a person's age can, depending on the DBMS used, result in an error.
Incorrect input can also conflict with integrity constraints for the same reasons as inconsistent input.
Enterprise constraints can also be broken; entering some value of the correct domain does not mean it is valid under the enterprise constraints present for that attribute. For example, not every sequence of digits is a valid bank account number.

2.2 Errors concerning state of an object

Let's take as example a web application that allows users to store electronic documents on a server and associate a short summary and keywords with each document. All this information is stored in a database.
Users can then via a web interface perform searches on keywords to retrieve the matching documents. Authorized users can edit a document's associated information or delete a document and it's data.

Various problems can then occur:

- A user opens an edit window on a document; while the first user is editing the document's information, another user deletes this document. When the first user then wants to update the document, his web application is in an invalid state without him knowing, resulting in an error.

- Web applications that interface to the user via web pages generated by the server are run in the user's browser. Most browsers keep the most recently viewed pages in their cache for faster retrieval later on. In such web applications, the user can go back in time by clicking the *back button* and thus viewing the state of the application before the most recent action. Then again, the user can perform some actions that might not be compatible with the real state of the object but which are not shown because of the cached information.

3. Technology available to validate user input in web applications

3.1 Keeping state in a stateless connection

As mentioned earlier in 1.2, the Internet Protocol is *stateless*, this means that when a user makes a connection to e.g. a server, the server does not remember information about previous connections from that user.

This stateless property is a problem for a lot of web applications. The most typical example is a online store that lets users browse between the available items and lets them add items to some virtual shopping basket and in the end produces a list of all selected items with a price total, followed by some choice of payment. The issue here is how to have the server remember what items a particular user selected and to accumulate all of them until the user is satisfied and goes to the payment section. As shopping items are most likely to be spread over different pages, multiple server connections are needed in the selecting phase.

These state problems are not related to validating user input in web applications, it is just a problem that needs to be solved for many web applications.

Several techniques will be discussed here: cookies, hidden data in forms, data in URLs and HTTP sessions. All these techniques are used to keep state on the server; it is always possible to keep state at the client using a scripting language as e.g. *JavaScript.*

3.1.1 Cookies

Cookies are a general mechanism that server side connections (e.g. CGI scripts) can use to store and retrieve information from the client [Netscape]. Cookies are kept on the user's PC.
Besides the value(s) to be stored, cookies can contain extra information like a description of the range of URLs for which the value is valid. All future HTTP requests made by the client, which fall in that range, will result in a send of the current value of the cookie from the client to the server. A time constraint stating how long the cookie data will remain valid can also be included. Cookies can be configured to be sent to the server only if the connection between client and server is secure. Currently this means that secure cookies will only be sent to HTTPS (HTTP over SSL) servers.

SSL (Secure Sockets Layer) is a program layer created by Netscape for managing the security of message transmissions in a network. Netscape's idea is that the programming for keeping messages confidential needs to be contained in a program layer between the application (e.g. Web browser) and the Internet's TCP/IP (Transmission Control Protocol/Internet Protocol) layers.

TCP/IP is a two-layered program. The higher layer, Transmission Control Protocol, manages the assembling of a message or file into smaller packets that

are transmitted over the Internet and received by a TCP layer that reassembles the packets into the original message. The lower layer, Internet Protocol, handles the address part of each packet so that it gets to the right destination. Netscape's SSL uses the public-and-private key encryption system from RSA, which also includes the use of a digital certificate.

The SSL Handshake Protocol consists of two phases: server authentication and an optional client authentication [RSA]. In the first phase, the server, in response to a client's request, sends its certificate and its cipher preferences. The client then generates a master key, which it encrypts with the server's public key, and transmits the encrypted master key to the server. The server recovers the master key and authenticates itself to the client by returning a message authenticated with the master key. Subsequent data is encrypted and authenticated with keys derived from this master key. In the optional second phase, the server sends a challenge to the client. The client authenticates itself to the server by returning the client's digital signature on the challenge, as well as its public-key certificate.

A short cookie example in PHP:

the PHP statement

```
setcookie ("FileUploadLogin", "loginok", time()+300, "/");
```

stores the following values at the client:

```
wendy.vub.ac.be FALSE / FALSE 60538966  FileUploadLogin  loginok
```

(note that the domain attribute *wendy.vub.ac.be* was automatically set)

3.1.2 Hidden data in forms

Another solution consists of copying all the input from a previous form as hidden data in the next form. That way no information is kept on the server until the last form has been submitted. The copy process needs to be done by some server script that either generates the HTML form or inserts the data into an existing HTML page before sending it to the user.

Data can be stored into a form using the `<INPUT TYPE="HIDDEN" ...>` tag [see also 3.2].
The advantage of this approach is that it is very easy to implement and requires almost nothing extra from the server.
On the other hand, the user can view the generated source code of the HTML form and see the 'hidden' variable name - value pairs.
It is also possible to save the source code, edit the 'hidden' values and use the newly modified form to connect to the server. Normally, if the input on every form is validated before going to the next one, the server can assume that all the hidden data is correct but these data can be manipulated and can make the data of

the database inconsistent. (A user might alter a foreign key value to a non-existing one, given that the DBMS doesn't check this when inserting the data.)

Hidden fields should not be used to store sensitive data such as credit card numbers or other private data because at each transaction this data will be sent and can be intercepted by others.

To be safe that the user didn't modify the source code before posting hidden data to the server, the location of the CGI can be 'hidden', or made non-trivial making hard on the user to submit the data. The best way to avoid all this to happen would be some mechanism to make sure the page is unaltered and submitted from it's home domain.

3.1.3 Data in URLs

A Universal Resource Locator (URL) can carry a query to be performed on the object it is referring to. Name=Value pairs can be added for usage in server-side scripts.

Example:

```
http://wendy.vub.ac.be/~jplasqui/IIS/download.php3?id=7&dl=1
```

Here the `id=7` would tell the *download.php3* script what file should be downloaded.

With this approach, users will always see the variables passed to the server and they can very easily alter them. Server-side scripts that rely on data stored/transferred by urls need to check whether these data are valid and complete, i.e. all the necessary variables are present.

3.1.4 HTTP sessions

Another way of keeping state is storing the user's information on the server. If a server script can uniquely identify each user, it can keep state of those users.

A session either stores data on the server only or it can use one of the above techniques to store some data at the client in combination with data storage at the server.

In the first case a CGI script could use the client's IP address to uniquely identify him (although multiple PCs can surf with the same IP behind a firewall). In Internet Protocol Version 4, an IP address is a 32-bit number that identifies each sender or receiver of information that is sent in packets across the Internet.
One could use the REMOTE_ADDR environment variable to obtain the client's IP address. By definition of the CGI specification, at least the REMOTE_ADDR variable should be set by the server prior to execution of the CGI program. For security reasons, a user session should expire after a specified duration.

There is one drawback to this approach:
Take a user with a simple and unreliable telephone connection who has browsed e.g. some online shopping website and selected some items for purchase. If the user's connection gets interrupted and he first has to reconnect to his ISP (Internet Service Provider) he will most likely be assigned another IP address than he previously used to store all his session information on the server. This will result in the shopping website not recognizing the user anymore and he has to start all over again.

Another approach to sessions is to generate some unique session id on the server for each user that connects. This session id is then stored and sent back to the server every time the user performs some action on the website. All other state information is kept at the server.

Unlike the previous case, if a user gets disconnected and gets a different IP address he can continue browsing the website without losing any previous state information because his session id is still stored in his browser.

3.2 Interfaces used in web applications

Web applications are applications where the interface is executed in the browser. Currently this restricts the interface to either HTML forms or Java applets.

HTML forms are most commonly used because they are easy to implement and don’t require the usage of another programming language. This also saves the user from the overhead of first downloading the byte code or script for the interface and then interpret it on his browser.

The following constructs are available in HTML forms:

- Buttons: Start execution of specified action
- Checkbox: Select a value *On* or *Off*
- File: Let the user select a file for upload to the server
- Hidden: This is not a input element, a hidden field serves for carrying extra non- visible information in a form
- Image: This is an image that acts like a button
- Password: This is a textbox that masks user input with ‘*’
- Radio button: This allows to user to select one of several given values by checking a small box near that value
- Reset: A button that restores all the input to its default values
- Submit: Invokes the action specified in the form tag
- Text: Allows the user to enter a single line of text
- Select: Allows the user to select one or more values from a pull-down menu
- Textarea: Lets the user enter multiple lines of text

HTML syntax:

```
<FORM
    ACTION="serverURL"
    ENCTYPE="encodingType"
    METHOD="GET"|"POST"
    NAME="formName"
    ONRESET="JScode"
    ONSUBMIT="JScode"
    TARGET="windowName"
>


            <INPUT TYPE="BUTTON"
               NAME="buttonName"
               VALUE="buttonText"
               ONCLICK="JScode"
            >

            <INPUT TYPE="CHECKBOX"
               CHECKED
               NAME="name"
               ONCLICK="JScode"
               VALUE="checkboxValue"
            >


            <INPUT TYPE="FILE"
               NAME="name"
               VALUE="filename"
            >

            <INPUT TYPE="HIDDEN"
               NAME="name"
               VALUE="value"
            >


            <INPUT TYPE="IMAGE"
               ALIGN="LEFT"|"RIGHT"|"TOP"|"ABSMIDDLE"|"ABSBOTTOM"|
               "TEXTTOP"|"MIDDLE"|"BASELINE"|"BOTTOM"
               NAME="name"
               SRC="location"
            >


            <INPUT TYPE="PASSWORD"
               MAXLENGTH="maxChar"
               NAME="name"
               ONSELECT="JScode"
               SIZE="charLength"
               VALUE="textValue"
            >


            <INPUT TYPE="RADIO"
               CHECKED
               NAME="name"
               ONCLICK="JScode"
               VALUE="buttonValue"
            >


            <INPUT TYPE="RESET"
               NAME="name"
               ONCLICK="JScode"
               VALUE="label"
            >


            <INPUT TYPE="SUBMIT"
               NAME="name"
               VALUE="label"
            >
```

```
            <INPUT TYPE="TEXT"
               MAXLENGTH="maxChars"
               NAME="name"
               ONBLUR="Scode"
               ONCHANGE="JScode"
               ONFOCUS="Scode"
               ONSELECT="JScode"
               SIZE="lengthChars"
```

All form elements have attributes to embed *JavaScript* code.

The following is an example of how such a HTML form interface looks like.

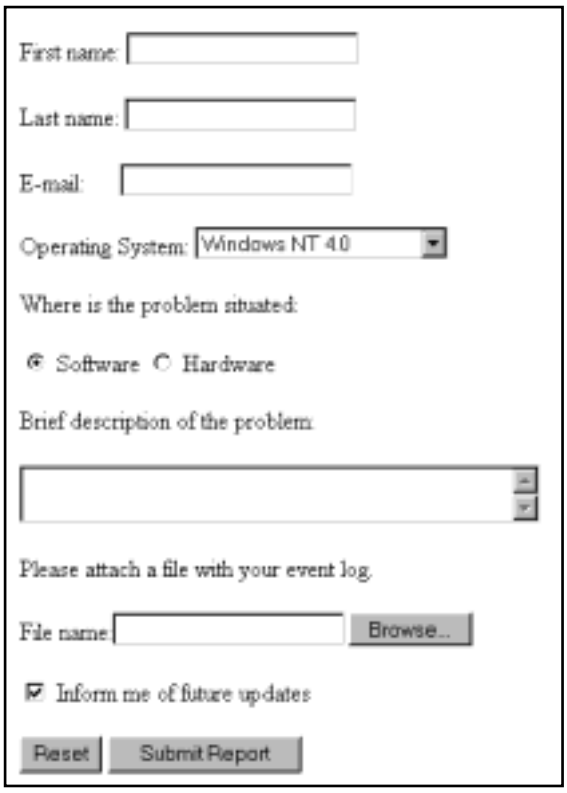

Fig. 1, HTML form interface

The usage of java applets allows for a more complex interface that can have the look and feel of regular applications for windows-like operating systems.

3.3 Server-side scripting versus client-side scripting

Before going into detail about scripting on either client or sever side, a definition of a scripting language should be given.
Following [O 98], scripting languages (or *glue languages*[3]) are weakly typed or untyped, have little or no provision for complex data structures, and programs in them (scripts) are interpreted. Scripts need to interact either with other programs (often as glue) or with a set of functions provided by the interpreter, as with the file system functions provided in a UNIX shell. An MSDOS batch file is a typical example of a script.
In general, script languages are easier and faster to code in than the more structured and compiled languages such as C and C++ and are ideal for programs of very limited capability or that can reuse and tie together existing compiled programs. However, a script takes longer to run than a compiled program since each instruction is being handled by another program first (requiring additional instructions) rather than directly by the basic instruction processor.

3.3.1 Server-side scripting

[3] A script can call various programs, using the output from one program as input for the next. This gives the impression of gluing applications together to handle a specific problem.

A server-side script is a program that is run on the server. These scripts are used to generate dynamic web pages or implement small services like sending e-mails or sending SMS (Short Message Service) to a GSM network.

The Common Gateway Interface (CGI) is a standard for interfacing external applications with information servers, such as HTTP or Web servers. A plain HTML document that the Web daemon retrieves is static, which means it exists in a constant state, a text file that doesn't change. A CGI program, on the other hand, is executed in real-time, so that it can output dynamic information.
Dynamic information can come from a database that is regularly updated or can be based upon input from the user via textboxes, buttons or links.

CGI programs are not limited to generating HTML text but can also transfer output from other programs or libraries as image or sound fragments on a web page. CGI programs can be programmed in any language. If one chooses a language like C or C++, the program needs to be compiled and will be executed when the page is requested.

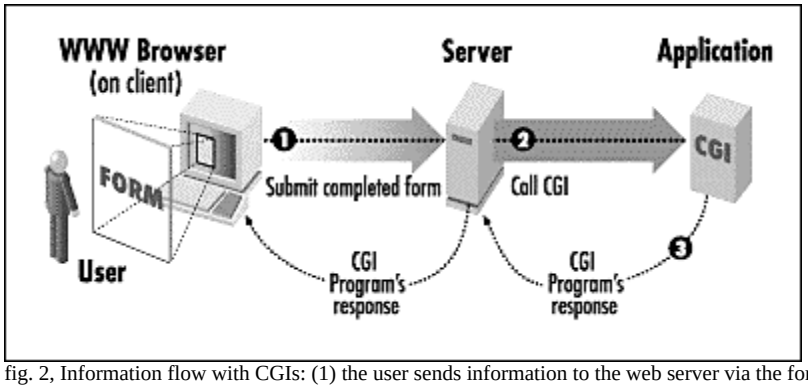

fig. 2, Information flow with CGIs: (1) the user sends information to the web server via the form; (2) the web server calls the appropriate CGI program; (3) the CGI generates a response page which is sent to the user by the web server, picture taken from Webmaster in a Nutshell, Stephen Spainhour & Robert Eckstein, O'Reilly & Associates

CGI scripts can be divided into three major categories:

- Scripted pages

  All the web pages available to the client browser are stored as scripted files on the server. Such a scripted file is usually a mix of HTML and some other scripting language. When a page is requested, the web server first has an engine (that recognizes the file) process it. The engine will generate an HTML formatted data stream by executing/interpreting the embedded scripting statements. This data stream is then sent back to the client browser as if it was a regular HTML page.
  Examples of scripted pages are: Active Server Pages, Cold Fusion, Hypertext PreProcessor (PHP), Server Side Include (SSI)

- Compiled pages

  Technologies like Netscape's NSAPI (Netscape Server Application Programming Interface) or Microsoft's ISAPI (Internet Server API), allow programmers to extend the web server's functionality by creating loadable binary modules to add or replace elements such as authentication, authorization, error logging, or content generation.
  This means that when a user requests a page, the web server loads and executes a binary component. This component, as with scripted pages, can use values from form fields or parameters and access server side resources to generate the HTML stream.
  The main advantages of compiled pages in respect to CGIs are faster response and better performance because compiled code executes faster than interpreted code. There is less server overhead with compiled pages because the component loads in the server's process space and does not spawn additional processes, as is the case with CGIs.

- Hybrid of scripted and compiled pages

  Using this approach, scripted pages are compiled once they are requested. The compiled version is then used for all later requests for that page. The compiled page will remain the same until the content of the page has changed. After an update, the page will be recompiled. This combination of scripted and compiled pages benefits from both the flexibility of scripted pages and the efficiency of compiled pages.
  Java Server Pages uses this technology.

3.3.2 Client-side scripting

Client-side scripts are pieces of code that are interpreted in the client's browser. These scripts are usually embedded into the page's HTML source. Client-side scripts are used to enhance the browser capabilities, create extra navigation possibilities, or simply make the web page more fancy. Apart from visual enhancements, client-side scripting can be used for form validation before submitting data to the server. This will be examined in detail in chapters 4 and 5.

3.4 Overview of server-side scripting/programming languages

3.4.1 PHP

PHP, the Hypertext PreProcessor, is an interpreted programming language designed for generating dynamic web pages.

As with most scripted languages designed for the web, PHP statements can be interleaved with regular HTML code. The PHP interpreter can distinguish both by the enclosing `<? ?>` or `<PHP ?>` tags. The PHP engine can access a wide variety of libraries to access almost any database system, use other server resources, parse XML data, generate gif images and so on [PHP3]. Up to version 3, PHP was open source and free. PHP_ wasn't suited for big, high-traffic websites due to performance problems.

The improved version 4 of PHP, also called ZEND, is split into several separate modules, which are not all freeware. The *Zend Engine* is the basic scripting engine that powers PHP. The *Zend Optimizer* uses multi-pass code optimizations to double the running speed of PHP applications and thus reducing the CPU of the server. The *Zend Cache* is a script-caching module that stores an intermediate coded version of a PHP application in the Web server's memory. A compiled version of the PHP script is stored in the server's persistent cache registry, avoiding redundant compilations. This saves server processing time and disk accesses. The *Zend Compiler* allows compilation of PHP scripts before distributing them. By saving the code in a closed Zend Intermediate Code format, source code is protected from copyright violations, enabling companies to develop exclusive commercial PHP applications [ZEND]. Other benefits besides the enormous performance gain include e.g. native session support and real platform and web server independence.

Thus depending on the components used, PHP scripts act like scripted pages or compiled pages.

3.4.2 ColdFusion

ColdFusion is a product developped by Allaire Corporation for creating web applications and database driven websites. ColdFusion has a server engine that processess CFML pages. The Cold Fusion Markup Language is a tag-based scripting language.

CFML example from the CF manual to print rows from a database:

```
<HTML>
<HEAD>
    <TITLE>Employee List</TITLE>
</HEAD>
<BODY>
<H1>Employee List</H1>

<CFQUERY NAME="EmpList" DATASOURCE="CompanyInfo">
    SELECT FirstName, LastName, Salary, Contract
    FROM Employees
</CFQUERY>

<CFOUTPUT Query="EmpList">
    #FirstName#, #LastName#, #Salary#, #Contract# <BR>
</CFOUTPUT>

<CFOUTPUT>
    The query returned #EmpList.RecordCount# records.
</CFOUTPUT>

</BODY>
</HTML>
```

CFML offers three major advantages over other server-side scripting languages [CF]. It tightly integrates with HTML and XML, making the process of developing Web applications easier and faster. CFML provides a high level of encapsulation for complex processes, eliminating the need for excessive scripting and increasing developer productivity. Finally, CFML is easy to extend with new ColdFusion Extensions (CFX) and Java objects, which serve as reusable components.

The ColdFusion server runs as a multithreaded process with native support for load balancing and can work with almost any web server. Other features of the server include database connection caching, just-in-time compilation and automatic failure recovery.

Since ColdFusion is a commercial software package, it supports a lot of other technologies as wel to reach a broad range of customers. ColdFusion can therefore work with ActiveX components, which are in fact Component Object Models (COM), Microsoft's core object technology for building software out of encapsulated functionality in reusable blocks.
ColdFusion also provides a generic interface to CORBA ORBs. The Common Object Resource Broker Architecture (CORBA) is an industry-standard system for allowing distributed systems to invoke methods on other servers across platforms and application architectures. CORBA works by using Object Request Brokers (ORBs) that translate method calls from each application. The Interface Definition Language (IDL) provides a language-neutral way to describe a

CORBA object and the services it provides. The IDL code then needs to be compiled into Java or C++ code stubs and skeletons. The client uses the stubs, and the server skeletons provide the framework that needs to be filled in with the code for the service the object is to provide.
ColdFusion applications can call Java objects, Servlets and JavaBeans, leaving alternative technology choices open for specific problems.

3.4.3 ASP

Microsoft's Active Server Pages (ASP) is a server-side scripting technology that can be used to create dynamic and interactive Web applications. ASP works as a filter and a redirector for IIS, Microsoft's Internet Information Server, intercepting incoming client requests with the *.asp* extension and caching information as needed to gain server performance. Active Server Pages are actually a series of dynamic link libraries or DLLs that have to be installed on the web server, which has to run on Windows NT machines.
The big difference between ASP and other server side scripting tools is that ASP supports different languages. The default ASP programming language is Microsoft's VBScript but also JScript and PerlScript are supported. The first line of ASP in a document states which language will be used further on. As with many scripting languages, ASP statements are embedded into HTML and distinguished from HTML by enclosing them with an ASP specific tag (<% %>).
ASP has database support and makes full use of Microsoft's COM technology.

The following example taken from Microsoft's Active Server Pages tutorial demonstrates how to use an ODBC connection to a Microsoft Excel data file and to generate a HTML table containing the data records:

```
<%@Language=VBScript %>
<html>
<head>
<title> Displaying An Excel Spreadsheet in an Web Page </title>
</head>
<body bgcolor="#FFFFFF" text="#000000" >
<h1>ASP Table of Contents</h1>
<%
'Creates an instance of an Active Server Component
 Set oConn = Server.CreateObject("ADODB.Connection")
'Connects to the Excel driver and the Excel spreadsheet
'in the directory where the spreadsheet was saved
strConn = "Driver={Microsoft Excel Driver (*.xls)};
DBQ=C:\Inetpub\Wwwroot\Tutorial\ASPTOC.xls;"
'Opens the connection to the data store
 oConn.Open strConn
'Selects the records from the Excel spreadsheet
strCmd = "SELECT * from `ASPTOC`"
Set oRS = Server.CreateObject("ADODB.Recordset")
'Opens the recordset
oRS.Open strCmd, oConn
'Prints the cells and rows in the table
Response.Write "<table border=1><tr><td>"
'Gets records in spreadsheet as a string and prints them in the table
Response.Write oRS.GetString (, , "</tr><td>", "</td></tr><tr><td>", NBSPACE)
%>
</body>
</html>
```

3.4.4 Java servlets

A java servlet can be seen as a java applet running on the server side. These are just small programs loaded by the web server to handle client requests, just like CGIs. This fairly young technology of server side applets has several advantages over CGI programs [JSVL] :

Servlets can run on any platform without the need for recompiling or rewriting the servlet. This platform independence supports the Java philosophy of ‘write once, run anywhere’.
Servlets are written in Java, a robust, well-designed and fully object-oriented language. Specialized Java libraries, development tools and database drivers are available and growing in number all the time, and servlets can utilize any of them. This brings a high level of extensibility to servlet developers. Developers don’t have to worry about the inner workings of the server either. Form data, server headers, cookies, ... are all handled by the servlet's underlying classes.
The most important advantage of servlets is their performance. A CGI is executed at each request, this includes process creation, setting up a database connection if needed and/or use other server resources and finally closing the process. Servlets on the other hand are loaded once when they are called and then they stay resident in memory for fast execution. Database connection also remain allocated, this results in one connection for a servlet against maybe several hundred or thousand connections for a CGI. All this makes up for Java’s poor performance when compared to compiled languages. Java servlets can handle multiple requests of the same servlet concurrently.
A nice feature of servlets is their possibility of chaining. Output from one servlet can be provided as input for another and so on. This allows reuse of components to perform some standard tasks as formatting output in a desired way for the user.
Java servlets can off course easily communicate with java applets.

Clearly the most common use of Java Servlets will be to function as part of middle tiers in enterprise networks, connecting to SQL databases via JDBC, Java’s facilities for database communications. This three-tier architecture is a special type of client/server architecture consisting of three well-defined and separate processes, each running on a different platform:

1. The user interface, which runs on the user's computer (the client browser).
2. The functional modules that actually process data (the servlet).
3. A database management system that stores the data required by the middle tier.

The added modularity makes it easier to modify or replace one tier without affecting the others. Separating the application functions from the database functions makes it easier to implement load balancing as can be done with servlets.
Servlets are called from HTML pages just as any regular CGI. In order to use them, the server needs to be compatible with the Java Servlet API. Some web

servers like the World Wide Web Consortium's free JigSaw Web Server has native servlet support, for others plug-ins like the Java Servlet Developers Kit can be installed to be able to run and manage java servlets.

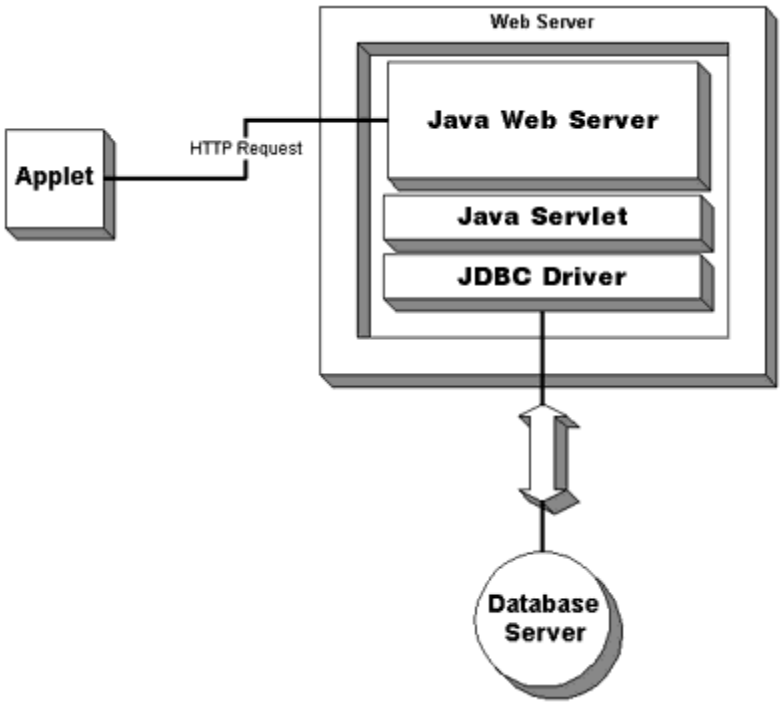

fig. 3, Java servlets in middle tiers, [JWS]

3.4.5 JavaScript

JavaScript was originally developed by Netscape as a programming language that could be embedded in web pages to extend the functionality of the Netscape Navigator browser [see 3.5.1]. It's original name LiveScript was changed to profit from the popularity of the object-oriented programming language Java from Sun Microsystems.

Server-side JavaScript extends the core JavaScript language (predefined objects and functions) by supplying objects relevant to running JavaScript on a server. Sever side functionality includes connecting to a relational database, providing continuity of information from one invocation to another of the application, performing file manipulations on the server, sharing information across users of an application and communicating with other applications through LiveConnect and Java.
Server side JavaScript is also embedded in HTML pages, which can also contain client side statements to be executed in the client browser.

Unlike the interpreted client side JavaScript statements, server side JavaScript pages are compiled into bytecode executable files. Multiple HTML pages containing embedded server side JavaScript can be compiled into a single binary

executable. These application executables are run by a web server that contains the JavaScript runtime engine. The runtime engine uses the application executable to look up the requested source page and dynamically generates the HTML to be sent back to the user.

Figure 4 depicts the global view of server side JavaScript. The Java virtual machine is not only for use with JavaScript; any Java application running on the server uses this virtual machine. It has been extended to allow JavaScript applications to access Java classes using JavaScript's LiveConnect features. The server side JavaScript runtime library offers basic functions like session management and so on. The application manager is built on top of the system and is needed to install the JavaScript applications and to provide access to them for the users.

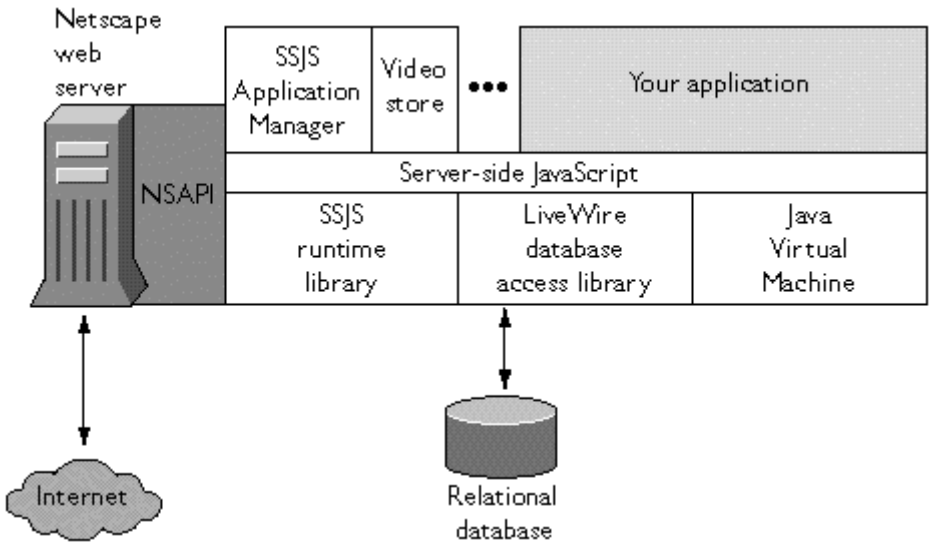

Fig 4, Server side JavaScript in the Netscape server environment, taken from Server Side JavaScript Guide, 1999 Netscape Communications Corp.

3.4.6 SSI

Server Side Includes (SSI) are directives which are embedded into HTML documents to execute other programs or output such data as environment variables and file statistics (e.g. web counter).

SSI works by inserting HTML comment tags containing a function call as is shown in this simple example:

```
<HTML>

<BODY>

The document is titled:

<!--#echo var="DOCUMENT_NAME"- ->,

and was last modified on

<!--#echo var="LAST_MODIFIED"-->
```

In order to avoid the server having to parse every HTML document at request, documents containing SSI directives carry a different file extension which is often *.shtml*.

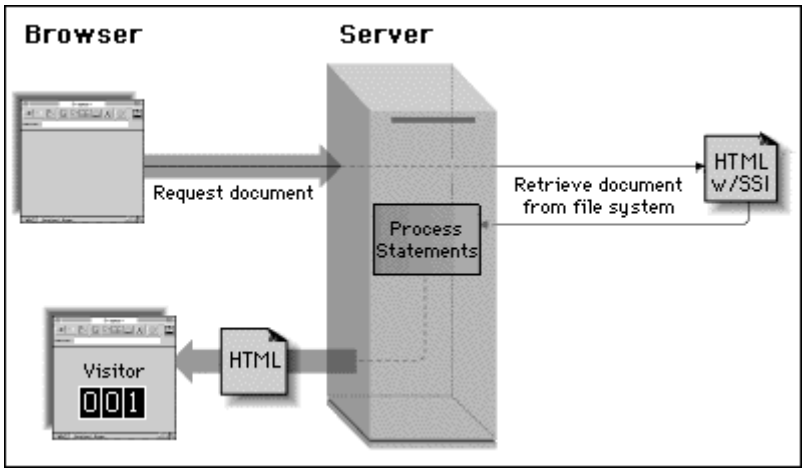

Fig. 5, Processing SSI configured pages, taken from CGI Programming on the World Wide Web, Shishir Gundavaram

Besides printing system variables, SSI also allows to execute system commands or CGIs.

Example:

```
This page has been accessed <!--#exec cgi="/cgi-bin/counter.pl"--> times.
```

Care should be taken when processing shtml documents as they can execute *any* command and can cause damage to the underlying system.

SSI is only useful to bring little dynamics to static web pages without having to write a CGI for it. SSI is not to be used in dynamic web pages or web applications as far much better technologies are available.

3.4.7 Perl

The Practical Extraction and Report Language (PERL) Perl is a scripting language that combines powerful text-manipulation functions with features and purposes of many command languages. Perl is quite popular for programming World Wide Web electronic forms and also serves as glue and gateway between systems as is the main purpose of a scripting language.

Perl example taken from [PERL]: the example prints a simple HTML form, after submitting some values the script will report those values back to the user. Generating a form and handling the form is often done in the same script; the script starts with some check whether there was any input and reacts accordingly.

```
#!/usr/local/bin/perl
# Send error messages to the user, not system log
open(STDERR,'<&STDOUT');  $| = 1
require "cgi-lib.pl";  # Get external subroutines
print &PrintHeader;

$script = $ENV{'SCRIPT_NAME'};
$webserver = $ENV{'SERVER_NAME'};

if (! &ReadParse(*input)) { &showform }
else { &readentries }
exit;

sub showform {
# If there is no input data, show the blank form

print <<EOF;
<HTML><HEAD>
<TITLE>Form Example, Part 1</TITLE>
</HEAD><BODY>
<H1>Web Form Example</H1>
<P>(From http://$webserver$script)
<FORM METHOD="POST"
ACTION=$script>
<PRE>
Enter your ID Number:  <INPUT NAME=idnum>
Enter your Name:       <INPUT NAME=name>
Select favorite Color: <SELECT NAME=color>
<OPTION>red<OPTION>green<OPTION>blue
</SELECT>
</PRE>
To submit the query, press this button:
<INPUT TYPE=submit VALUE="Submit Request">
</FORM>
</BODY></HTML>
EOF
} # End of sub showform #

sub readentries {
# Input data was detected.  Echo back to form user.

print <<EOF;
<HTML><HEAD>
<TITLE>Form Example, Part 2</TITLE>
</HEAD><BODY>
<H1>Results of Form</H1>
<P>(From http://$webserver$script)
<P>Your ID Number is $input{'idnum'}, your name is $input{'name'},
and your favorite color is $input{'color'}.
<HR>
[<A HREF=$script>Try again</A>]
EOF
} # end of sub readentries #
```

3.5.1 JavaScript

At the client side, JavaScript is an interpreted, object-based, event-driven programming language embedded into HTML pages.
Although JavaScript is different from other OOP languages (such as Java) in its approach to objects, it is nonetheless object based. No distinction between types of objects is made. Being object-based, JavaScript has a built-in object hierarchy reflecting the building blocks of a page. All properties can be accessed in an object-oriented way.

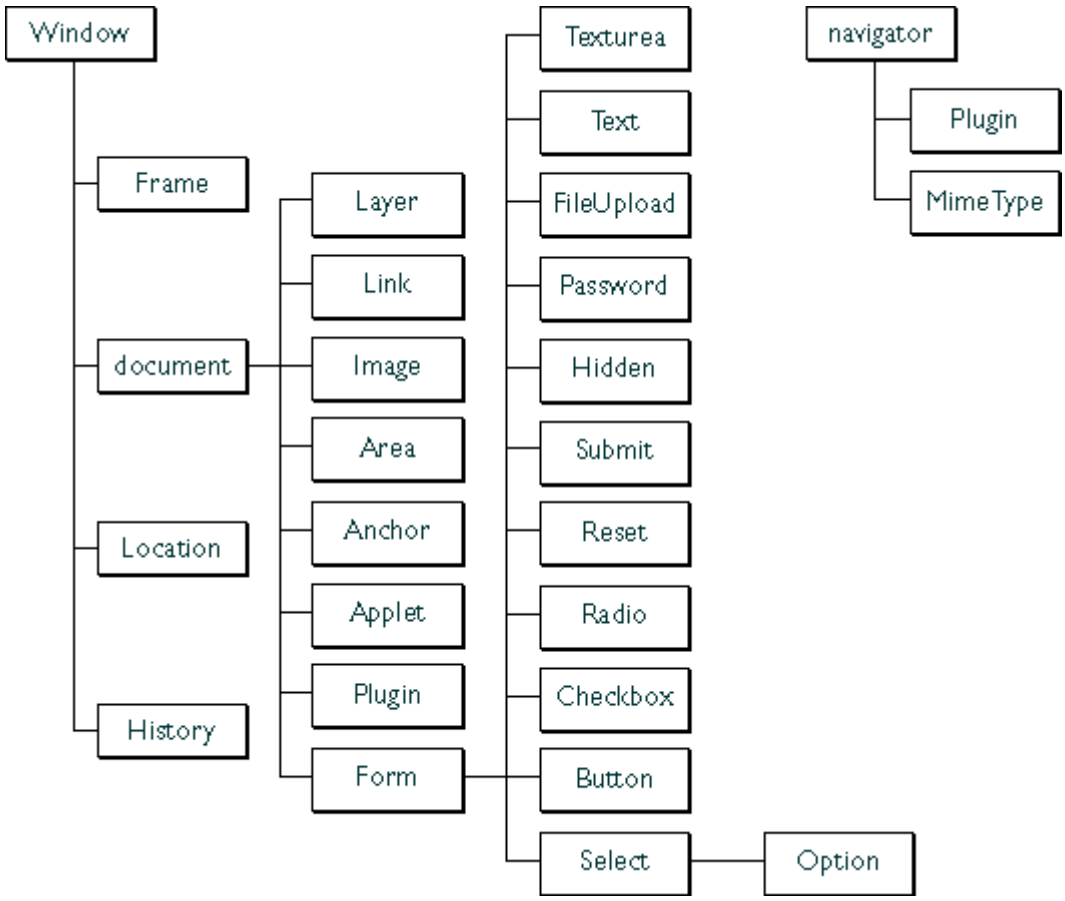

Fig. 6, The Netscape Navigator Object Hierarchy, Copyright 1997 Netscape Communications Corporation

Inheritance is achieved through the prototype mechanism and properties and methods can be added to any object dynamically. JavaScript is loose typed or untyped, variables don't need to be declared but they can be. Bindings are dynamic since they are only checked at runtime.
Functions can be called on various events in a page. This can be while the page is loading in the browser or when the mouse pointer is moving away from an item.
JavaScript has a security mechanism that prevents writing or retrieving files from the users hard disk. The only files that can be read are the cookies stored in the user's browser.
JavaScript can be used to implement small applications to be run at the client but is mostly used to make animations happen at certain events or to create a dynamic interface for a web page. Furthermore, JavaScript is well suited to perform input

validation in HTML forms before actually submitting the data to a server. Given the object hierarchy as depicted in fig. 6, JavaScript can access user input from all the form elements for testing.

Using LiveConnect, JavaScript can communicate with Java Applets. Using this idea, a page containing some form interface can both contain a Java Applet that implements complex validation functions which are then called with the variable of what the user entered in some form field via a JavaScript statement.

The next example demonstrates the communication between JavaScript and Java: The Java applet `SerialValidator` contains a method `public boolean validateSerial(String s)` which is called in the same object like way from within a JavaScript function.

```
<HTML>

<HEAD>
<TITLE>Serial validation</TITLE>
</HEAD>

<BODY>


<APPLET CODE="SerialValidator.class" WIDTH="0" HEIGHT="0" NAME="Validator">
</APPLET>

<SCRIPT LANGUAGE="JavaScript">

function validateSerial(form) {

      if(document.Validator.ValidateSerial(form.serial.value)) {

              return(true);

      } else {

              form.serial.focus();
              window.alert("Invalid serial. Please try again.");
              return(false);
      }
}

</SCRIPT>

<FORM
      NAME="serialInput"
      ACTION="test.php3"
      METHOD="POST"
      OnSubmit="return validateSerial(document.serialInput);"
>

<P>Enter serial number:
<INPUT TYPE="text" SIZE="10" NAME="serial">
<P><INPUT TYPE="submit" NAME="Submit" VALUE="Submit">

</FORM>

</BODY>
</HTML>
```

JavaScript is supported in almost all current graphical browsers. Netscape has two libraries available with JavaScript functions designed to validate different forms of user input in forms. These functions include basic type checking, common

input elements validation for zip codes, email addresses, dates and credit card numbers.
However, it is not safe to rely on its working too much because users always have the option to turn off JavaScript interpretation in their browsers. This can be overcome by only showing the form if JavaScript is enabled. This can either be done by 'writing' the form with JavaScript statements or by first checking the browser object model for a JavaScript-enabled flag.
Another issue with JavaScript is that users can edit the HTML file they are browsing and simply remove the call to the validation function. This can be done in the example above by removing the *OnSubmit* attribute from the form tag.

3.5.2 Jscript

Jscript is Microsoft's equivalent of Netscape's JavaScript.
On the surface, the JavaScript supported by both companies is identical. They provide the same conditional control statements, have defined objects such as *window* or *document*, and can be used directly in HTML documents. They both support events based on user actions and support functions in a similar manner. However, just as there were incompatibilities between proprietary HTML tags in the Browser War[4], differences exist between both JavaScripts. These incompatibilities are documented and can be solved but require more programming efforts.

3.5.3 VBScript

Visual Basic Scripting Edition, also known as VBScript, enables authors to create scripts using a subset of the Microsoft Visual Basic language. VBScript is implemented as a fast, portable interpreter for use in Web browsers and applications that use ActiveX controls, Java applets, and OLE Automation servers. As VBScript is a proprietary language, it is only supported by Microsoft browsers. VBScript can be used for form validation. [H 96]

3.5.4 Java

Java is an object-oriented programming language developed by Sun Microsystems. Java is a platform independent language; java source files are first compiled into class files containing byte code. A Java Virtual Machine on the host computer then interprets these class files. Java has an extensive library of classes and methods for distributed processing, creating graphical user interfaces, image processing etc.
The portability of Java classes and the fact that Java applications can be built as a Java Applets for usage on the Web has created a certain hype around Java thus making it very popular. Java is used on both client and server; on the client side applets are used to enhance the interface of web pages. A Java Applet is a Java

[4] The strong competition between Netscape and Microsoft to provide the dominant web browser to the public is often referred to as the 'Browser War'.

program that can be executed in the user's browser. Applets are mostly used to create a dynamic and animated interface to a web site but they can also be used to deliver small applications that entirely run in the client browser. These applications vary from calculators and text validators to games.
Java applets use the so-called Sandbox model for security; applets cannot access the user's file system to do any damage there. Java classes can be linked to a page as an applet to provide extra functionality to other scripting languages such as JavaScript (see 3.5.1).
Packages such as the Abstract Window Toolkit (AWT) and Swing provide classes to build graphical user interfaces. A web page could load an applet with a more sophisticated interface than HTML forms currently offers and use that applet to immediately validate user input like in a classical application and in the end post the necessary data to the CGI or servlet running on the server.
Most current browsers support Java, however users have control on whether or not to allow Java applets to run in their browser.

3.5.5 Netscape plugins

A plug-in is a separate code module that behaves as though it is part of the Netscape Communicator browser[5]. The browser can be extended via the Plugin API with a wide range of interactive and multimedia capabilities, and that handle one or more data (MIME) types. Other browsers also support this Plugin API. The main purpose of plug-ins is providing inline viewers for types of data not supported by the browsers. A popular example is Macromedia's Shockwave Flash plug-in to display interactive animations in a web page.
Using Netscape's LiveConnect technology, plug-ins can communicate with Java. Plug-ins are written in C++ and can benefit of existing classes and libraries. Plug-ins are not portable as their execution is operating system (and browser) dependent.

3.6 XML and related standards

3.6.1 From SGML to XML

The Standard Generalized Markup Language (SGML) is an international standard for the definition of hardware-independent, software-independent methods of representing texts in electronic form. SGML isn't a language on itself, it is a means of formally describing a markup language. A markup language is a set of markup conventions used together for encoding texts. A markup, or encoding, is any way of making an interpretation of a text explicit. Spacing can be thought of as a markup to distinguish the beginning from one and the end from another word.
SGML uses descriptive markup rather than procedural markup. A descriptive markup system uses markup codes which simply provide names to categorize parts of a document. Markup codes such as <p> simply identify a portion of a

[5] Netscape was the first browser creator that provided facilities for plug-in creation and documented it. Therefore I maintain the term 'Netscape plug-in' although meanwhile Microsoft also supports plug-ins.

document and assert of it that “the following item is a paragraph”. On the other hand, a procedural markup system defines what processing is to be performed at particular points in a document. The benefit of using a descriptive system is that different applications can process the same file. A database application can scan the document for certain information elements to store them in a database while ignoring other unneeded data. Another application might have to use all of the descriptions to render the document for printing.
Another feature of SGML is the use of Document Type Definitions (DTD). An e-mail message has a different structure than for example a book. Usage of DTDs allows for more intelligent processing of documents. DTDs also bring structure to a document, e.g. a book could be structured as a title followed by an author name which is then followed by a number of chapters which also comply to some structure. Besides it’s reuse, another advantage of a DTD is document validation for completeness and correctness.
Another and very important aspect of SGML is the demand of document portability from one hardware and software environment to another without loss of information. SGML provides a general purpose mechanism for string substitution, that is, a simple machine-independent way of stating that a particular string of characters in the document should be replaced by some other string when the document is processed. This is useful to encompass the differences in character sets from different systems.

Although SGML will remain the preferred data format for creating and storing enterprise-critical documents and data, it has several shortcomings in document delivery.

[XML] describes why SGML isn’t used for distributing documents on the web:

The primary problem is that SGML isn’t supported by the mainstream browser providers. The reason for this goes hand-in-hand with what makes SGML so valuable: SGML offers so many options that designing tools to support them all results in complicated software.
With only HTML support available in the browsers, organizations that want to publish their SGML documents on the Web typically use some automatic SGML-to-HTML conversion of their data. This down-conversion from SGML to HTML results in a significant loss of information, and without it, it’s nearly impossible to reconstruct the original meaning of the SGML document by only looking at the HTML file.
Another problem to Web delivery is that SGML only standardizes structure and has no support for styles. There have been a couple of attempts to establish a stylesheet standard, but each of these has received little or no vendor support. The result is that there is no widely accepted standard stylesheet format for expressing SGML information.

To overcome this shortcoming of SGML, and also keeping the problems with HTML in mind [see 3.7.2], XML, or eXtensible Markup Language was built.

3.6.2 XML

The Extensible Markup Language is a highly functional subset of SGML. The purpose of XML is to specify an SGML subset that works very well for delivering SGML information over the Web. The eXtensible Markup Language describes a class of data objects (XML documents) and partially describes the behavior of computer programs that process them.
XML documents are made up of storage units called entities, which contain either parsed (characters and/or markup) or unparsed data. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure.

A Document Type Definition (DTD) is a file (or several files used together), written in XML's Declaration Syntax, which contains a formal description of a specific type of document. It sets out what names can be used for element types, where they may occur, and how they all fit together. XML has been designed so it can be used either with or without a DTD. This implies that one can invent markup without having to define it formally. Drawback off course is the loss of automated control over the structuring of additional documents of the same type. To make this work, a DTD-less file in effect defines its own markup informally by the location of elements. But when an XML application (such as a browser) encounters a DTD-less file, it needs to be able to understand the document structure while it reads it, because it has no DTD to tell it what to expect. This is also why XML documents need to be well formed.

3.6.3 XML Schema

A DTD serves for specifying the structure of an XML file: it gives the names of the elements, attributes and entities that can be used, and how they fit together. Because DTDs are designed for use with text, they have no mechanism for defining the content of elements in terms of data types, because XML has no data types and text is just text. This is why a DTD can't be used to specify numeric ranges or to define limitations or checks on the text content, only on the markup.

The XML Schema proposal provides a means of specifying element content in terms of data types, so that document type designers can provide criteria for validating the content of elements as well as the markup itself. Schemas are written as XML files, thus avoiding the need for processing software to be able to read XML Declaration Syntax, which is different from XML Instance Syntax.

The working of XML Schema can be best demonstrated by a small example [XSCH].

```
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">

 <xsd:annotation>
  <xsd:documentation>
   Purchase order schema for Example.com.
   Copyright 2000 Example.com. All rights reserved.
  </xsd:documentation>
 </xsd:annotation>

 <xsd:element name="purchaseOrder" type="PurchaseOrderType"/>
```

The example above is the schema for some order form to purchase items from a store. All the elements in the schema have a `xsd` prefix to identify the elements and simple types as belonging to the vocabulary of the XML Schema language rather than the vocabulary of the schema author. XML Schema has several built-in Simple Types such as string, boolean, float, unsignedInt, date, time and so on. Other Simple Types can also be derived from existing built-in Simple Types. Various constraints can be added to data types: domain constraints can specify a range of allowed values, cardinal constraints determine the number of instances allowed and regular expressions can be provided to match data. In the example above, the `Sku` (product number) has to be a sequence of three digits followed by a hyphen followed by two upper-case characters.
The schema example can be instantiated as is done in the following example, also taken from [XSCH]:

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-10-20">
    <shipTo country="US">
        <name>Alice Smith</name>
        <street>123 Maple Street</street>
        <city>Mill Valley</city>
        <state>CA</state>
        <zip>90952</zip>
    </shipTo>
    <billTo country="US">
        <name>Robert Smith</name>
        <street>8 Oak Avenue</street>
        <city>Old Town</city>
        <state>PA</state>
        <zip>95819</zip>
    </billTo>
    <comment>Hurry, my lawn is going wild!</comment>
    <items>
        <item partNum="872-AA">
            <productName>Lawnmower</productName>
            <quantity>1</quantity>
            <price>148.95</price>
            <comment>Confirm this is electric</comment>
        </item>
        <item partNum="926-AA">
            <productName>Baby Monitor</productName>
            <quantity>1</quantity>
            <price>39.98</price>
            <shipDate>1999-05-21</shipDate>
        </item>
    </items>
</purchaseOrder>
```

3.7 Evolution of HTML

HTML, the HyperText Markup Language, is the language used to deliver documents over the web. HTML is in fact an application of SGML, a fixed set of tags to bring (limited) structure to web documents.

HTML is very popular. People learn how to navigate between HTML documents in no time. Creating simple web pages can be learned in jut a couple of hours and for the more complicated aspects or for those who don't want to 'code' their page, an enormous amount of web development tools exist.

The HTML tags have been extended quite often, resulting in three major HTML versions:

HTML 1.0 was the first widely used version of HTML. The main characteristics of HTML 1.0 include the means of bringing simple structure to a document. Structure elements are headers, lists of list items, paragraphs and so further. Other features are the ability to handle inline images and the use of anchors to link to other documents.

HTML 2.0 greatest improvement is the handling of user input via forms. Companies as Netscape also started to develop their own tags at that time to work with their specific browser.

The arrival of HTML 3.0 brought quite a lot of new possibilities: tables, frames, more form input types and style sheets. In this version, most of the proprietary tags created by others were adopted, although often with some modifications. Although HTML 3.0 didn't receive consensus in standards discussions, it's drafts led to the adoption of a range of new features that resulted in HTML 3.2.

HTML 4.0 extends HTML further with mechanisms for style sheets, scripting, frames, embedding objects, improved support for right to left and mixed direction text, richer tables, and enhancements to forms, offering improved accessibility for people with disabilities.

Despite the continuous evolution of HTML, which often focuses on appearance rather than structure, HTML still has several problems.

HTML 3.2 is still the mostly adopted web language, together with CGI scripts, Java applets, JavaScript and others, plus plug-ins such as Shockwave, RealPlayer and Quicktime, it provides Web authors and commercial sites with a vast amount of techniques for displaying content that is visually pleasing and possibly even informative. However, these techniques do little if anything for the representation of structured data.

Using HTML, one can't markup data in a meaningful way. HTML was originally intended to provide a simple way to markup any type of document to reflect its structure and also some stylistic aspects. But companies also have the need to exchange data. Hierarchical relationships of data values, such as that which is represented by database records and object hierarchies can't be coded in HTML. An HTML document contains structure and presentation, but conveys nothing about the meaning of the marked up document.

When using some search engine to find web pages related to the keywords entered in the query, an enormous list of 'hits' will be returned to the user. The majority of these pages don't even deal with the intended topic. Of course one can argument that the user first needs to understand the working of the search engine and learn the proper syntax to reduce the amount of hits. Currently, HTML supports the use of meta tags, extra attributes in the head of an HTML document that describe the content of the page and carry some extra information to facilitate better responses from search engines. There is no guarantee that a search engine uses those meta descriptions as they typically only index frequency of words and document titles. What is needed is a way to markup the significant portions of a document and to describe the semantics of documents so search engines can focus on the semantically related pages and drop those several hundred meaningless hits.
An example of such a search could be the retrieval of pages on books about some artist. Pages about works by that artist should be retained although both would be presented to the user using some typical search engine.
Another annoying thing can happen when a user finds a web page with useful information for him that is part of a larger collection. If all goes well, the user can find a link to a table of contents, a home page, or some other means of listing the

collection. Otherwise he can guess a URL for a page higher in hierarchy that links to all page from the collection.
But how can the user print the collection without having to manually search and print each page individually? HTML currently has no means to express the interrelationship of a set of pages so they can be processed as a group.

Although the Web's current one-way hypertext link capability has proven extremely useful, but HTML documents could gain more functionality with extended link capabilities. More flexible linking schemes have existed for many years in the publishing industry to express complex link relationships, such as links with multiple targets, multi-directional links, and automatically updated link databases. [http://www.hytime.org, http://www.tei-c.org]

A popular buzzword on the web is an *agent*. An intelligent agent, or simply an agent, is a program that gathers information or performs some other service without the user's immediate presence and on some regular schedule. Typically, an agent program searches the Internet using user specified parameters, gathers information in which the user is interested in, and finally presents it to the user on some periodic basis.
Other agents have been developed that personalize information on a Web site based on registration information and usage analysis. Other types of agents include specific site watchers that tell when the site has been updated or look for other events and analyst agents that not only gather but also organize and interpret information.
The operation of such agents could be facilitated in the same way search engines could become very efficient, that is by using semantic information from the HTML document markup.

All limitations and problems stated above lead to the development of XHTML 1.0 or the eXtensible HyperText Markup Language. XHTML is in fact the reformulation of HTML 4.0 as an application of XML. The World Wide Web Consortium (W3C) lists the following reasons why to use XHTML [XHTML]:

- XHTML documents are XML conforming. As such, they are readily viewed, edited, and validated with standard XML tools.
- XHTML documents can be written to operate as well or better than they did before in existing HTML 4-conforming user agents like web browsers as well as in new, XHTML 1.0 conforming user agents.
- XHTML documents can utilize applications (e.g. scripts and applets) that rely upon either the HTML Document Object Model or the XML Document Object Model.
- As the XHTML family evolves, documents conforming to XHTML 1.0 will be more likely to interoperate within and among various XHTML environments.

The two main benefits of XHTML over plain HTML are called extensibility and portability by the W3C.

Extensibility: XML documents are required to be well-formed. Under HTML, the addition of a new group of elements requires alteration of the entire DTD. In an XML-based DTD, all that is required is that the new set of elements be internally consistent and well-formed to be added to an existing DTD. This greatly eases the development and integration of new collections of elements.

Portability: There will be increasing use of non-desktop devices to access Internet documents. W3C estimates that by the year 2002 as much as 75% of Internet access could be carried out on these alternate platforms. In most cases these devices will not have the computing power of a desktop computer, and will not be designed to accommodate ill-formed HTML as current browsers tend to do. In fact, if these non-desktop browsers do not receive well-formed markup (HTML or XHTML), they may simply be unable to display the document.

There are some syntax differences between HTML and XHTML. XHTML has to be well-formed, in other words, tags have to be properly nested. In XHTML it is illegal to write for example `<b><I>bold & italic text</b></I>`, which was already prohibited in SGML but was tolerated in HTML. The correct syntax would be `<b><I>bold & italic text</I></b>`. In fact browsers contain quite a bit of code to interpret all these ill-formed documents but doing so on PDAs could require too much processing power.

Since XML is case-sensitive, XHTML tags all have to be lower-case. Tags `<b>` and `<B>` are thus considered to be different.

In XHTML, all non-empty elements require a closing tag. In HTML, one could write `<P> a paragraph… <P> another paragraph …` .However, in XHTML the syntax needs to be as follows: `<p> a paragraph…</p> <p> another paragraph …</p>`.

Other differences include the need for quoting attribute values, for example `<table rows=″5″>`. Also, XML does not support attribute minimization. Attribute-value pairs must be written in full, for example: `<input type=″checkbox″ checked=″checked″>` instead of `<input type=″checkbox″ checked>`. Furthermore, empty elements need to written like `<br/>` or `<hr></hr>`.

The following example taken from [XHTML] shows a very basic XHTML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/strict.dtd">
<html xmlns="http://www.w3.org/TR/xhtml1/strict" xml:lang="en" lang="en">
  <head>
    <title>Virtual Library</title>
  </head>
  <body>
    <p>Moved to <a href="http://vlib.org/">vlib.org</a>.</p>
  </body>
</html>
```

XHTML can also be used inside other XML documents [XHTML]:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- initially, the default namespace is "books" -->
<book xmlns='urn:loc.gov:books'
    xmlns:isbn='urn:ISBN:0-395-36341-6' xml:lang="en" lang="en">
  <title>Cheaper by the Dozen</title>
  <isbn:number>1568491379</isbn:number>
  <notes>
    <!-- make HTML the default namespace for a hypertext commentary -->
    <p xmlns='http://www.w3.org/1999/xhtml'>
        This is also available <a href="http://www.w3.org/">online</a>.
    </p>
  </notes>
</book>
```

XHTML can make use of other XML related technologies making it even more powerful.

I'll restrict to one academically interesting example: MathML [MATH]. The incorporation of MathML allows authors to encode both the notation which represents a mathematical object and the mathematical structure of the object itself. Moreover, authors can mix both kinds of encoding in order to specify both the presentation and content of a mathematical idea.

For example, the formula $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ is represented by the following code:

```
  <mrow>
    <mi>x</mi>
    <mo>=</mo>
    <mfrac>
      <mrow>
        <mrow>
          <mo>-</mo>
          <mi>b</mi>
        </mrow>
        <mo><mchar name="PlusMinus"/></mo>
        <msqrt>
          <mrow>
            <msup>
              <mi>b</mi>
              <mn>2</mn>
```

Browsers can be extended to be able to render the XML code to a classical math representation.

W3C has implemented the Cascading Style Sheets (CSS) language to get extensive stylistic control over the presentation of Web pages. Rather than creating complex HTML documents that use tags and attributes to control the font and other layout aspects, CSS makes stylistic control a separate process. This approach of keeping structure and presentation apart has great benefits when it comes to adapting Web content for different devices such as mobile phones, PDAs,… Different style sheets can be used to deliver the same Web page to different machines, considering their limitations and focussing on their capabilities.

The eXtensible Stylesheet Language (XSL) differs from CSS as it is a language to transform XML documents to some other form. This can be used to generate XHTML pages from existing XML documents or to generate some report by scanning for required tags and ignoring the rest.

3.8 Portable internet devices

A new boost in wireless communications devices came with the third-generation GSMs that provide Internet access. Other than GSM devices, PDAs (Personal Digital Assistants), handheld devices that combine computing, fax and networking features are becoming quite popular. Many PDAs nowadays also offer

some form of Internet access. To make this possible, a new access protocol was developed.

The Wireless Application Protocol (WAP) is an open, global specification that allows mobile users with wireless devices to easily access and interact with information and services. WAP is a communication protocol that operates on most wireless networks and is also an application environment that can be built on any operating system.
Applications for WAP devices include: customer care and provisioning, message notification and call management, e-mail, telephony value-added services and unified messaging, mapping and locator services, weather and traffic alerts, news, sports and information services, e-commerce transactions and banking services, online address books, directory services and corporate intranet applications [WAPF].

The ability to surf the web, that's to say browse WAP enabled web pages, brings another group e-business users to web applications.

WML (Wireless Markup Language) is a markup language based on XML, and is intended for use in specifying content and user interface for narrowband devices. These devices are constrained by a small display and limited user input facilities, a narrowband network connection and limited memory and computational resources.
This means that WAP devices cannot simply access the same web interface for existing web applications. Thus we can either build two different web applications, one for normal PC users and one for wireless devices, which do the same thing. On the other hand, we can also build two different interfaces for the same application, one in HTML and one in WML. This would limit the use of client side technologies even more because of the computational constraints on the wireless devices.

WML can make use of a client side scripting language called WMLScript. WMLScript is a scripting language that is very similar to JavaScript. However, rather than embedding WMLScript in the WML decks (WML uses a metaphor of a deck of cards), WML contains only references to WMLScript URLs. Another difference is that WMLScript compilation units need to be compiled into the WMLScript bytecode before it can be run on a WAP client. Such a WAP browser must contain a WMLScript Virtual Machine (VM) to run the compiled script. WMLScript makes minimal demands on memory and CPU usage and omits a number of functions that are not required from other scripting languages.

Using this WMLScript, we could also perform some level of client side validation. But because of the wide variety of handheld devices (hardware) and software, usage of client side scripting on all machines is uncertain. Therefore it is advised to perform input validation on the server.

3.9 What technology to use

When deciding what technologies to use for user input validation, several aspects need our attention.

A first thing that needs to be known is what group of users is allowed to use the web application in question. If the application is meant for the employees of a company, it can only be accessed via the company's Intranet. This gives the developers either a free choice of technologies to use as they can make sure everyone in the building gets the necessary software add-ons (plug-ins) to make the application work, or they know exactly the limitations of the systems present in the company and can choose a technology that is compatible with the existing situation.
As soon as the web application is available to everyone on the Internet, the usage of technology matters to the user. A user can have any machine (hardware) running a variety of operating systems and web browsers (software). A commercial web application cannot demand from the users to use a specific technology from some company just to be able to use the application. The user's browsing software might be incompatible with the needed plug-in, the user's computer might not have enough memory or computing power or the user can simply not have the privileges to install extra software on his machine. Besides from physical computing limitations, certain groups of users simply don't want any software from certain companies on their machines. To give a concrete example, a company that intends to make money from advertisement by providing a free web service like an e-mail account should think twice before enforcing possible users to use Microsoft Windows technology to access the service, thus ruling out Unix/Linux, Macintosh, … users.

Another and far more important issue is security with client-side technologies.
A company cannot use critical business information on the client to validate user input.
One example as mentioned earlier is the validation of a registration number that a user has bought to register a piece of software and to gain access to a support network.
If the company that produces this software includes the information on how a valid registration number is composed in a function that is available on the client browser, then hackers can easily reverse-engineer this algorithm to produce their own registration numbers to unlawfully *pay* for their software. Such pirate actions can cause great damage to the company.
There exist utilities that allow one to *de-compile* the client validation program to view some form of its source code. Take for example a small program that would compare an entered value with some other values from a database on the server (the client application would then make an extra connection to the server). It would be possible to alter the source code in such a way that possibly confidential information from the server is transferred to the client PC. The modified source code can be recompiled and run from the client to achieve this.
Such actions can cause great damage to a company because the loss of credibility and distrust from the normal users can have a great (negative) impact on the company's business.

So when a company has a web site containing a form that requires multiple fields filled out by a user and only one of those fields contains information that has to be checked on the server for security reasons, then this company would prefer to check all of the input on the server to have a uniform way of interaction with the user through feedback.

So to summarize, knowing the audience gives an idea of what technologies can be considered to use and the nature of the information to be validated should imply on which side, client or server, it is to be validated.

When considering all of the above, the only technology that is guaranteed to work on all browsers is a server-side CGI or script that validates the input and gives the necessary feedback to the user. Of course that CGI or script has to generate some standard HTML text that doesn't require any special plug-ins.

3.10 New research directions

3.10.1 Advanced XML-based forms

An idea on how to standardize user input validation: forms with built-in validation.

It is clear that incompatibilities between technologies, browsers and hardware will remain a problem if one would choose a specific technology now as 'the' standard way for handling user input. Off course JavaScript seems to be the most supported technology to cooperate with HTML forms but as discussed before one cannot rely completely on some JavaScript validation done on the client side before sending data to the server. Keeping into mind the advantages of using XML applications on the web and looking at the development of XHTML as an example of such XML technology, it seems logical to embed semantic information in forms.
If all the needed information to handle a form is already contained in the form definition, various technologies can be dynamically applied to achieve the same goal: validating user input to reduce both the server load and the number of server connections.

XML Schema is a first step to the solution as various constraints on the actual data can be represented in the schema. Several other techniques focus on the development of some kind of form to enter data.

A first XML application for representing forms is the eXtensible Forms Description Language (XFDL) [XFDL]. XFDL is an XML syntax developed for the Universal Forms Description Language (UFDL), originally designed by Unisoft Wares Inc. XFDL is currently still a draft at the W3C. The purpose of

XFDL is to solve the body of problems associated with digitally representing complex forms such as those found in business and government. XFDL is the result of developing an XML syntax for the Universal Forms Description Language (UFDL), thereby permitting the expression of powerful, complex forms in a syntax that promotes application interoperability and compliance to world-wide Internet standards. The design goals of XFDL include support for high precision layout, supporting documentation, integrated computations and input validation, multiple overlapping digital signatures, and legally binding auditable transaction records, by maintaining the whole form as a single unit such that digital signatures can capture the entire context of transactions. To use those XFDL forms, an XFDL processor is needed. It is a software program that reads, processes, and writes XFDL forms. Processing may include such tasks as GUI rendering, data extraction, or modification.

XFDL supports several basic data types extended with date, time and dollar but not as many as XML Schema does. XFDL has the means to denote whether user input is optional or mandatory, what the range of allowable values is and what the maximum field length is. Also, templates describing the valid input formats for a form field can be added and other control flags can be set to state whether input is case sensitive and so on. Custom help messages can be added to a field and are displayed to the user when his input doesn't match the specified data type. An extensive list of options to specify the layout of a form is present. XFDL also incorporates computations to enhance the functionality of the form. These are simply embedded in a `<compute>` tag.

The following is a simple example taken from [XFDL] and demonstrates a simple form. It contains three fields: the first two collect side lengths for a right triangle; the third computes the length of the hypotenuse of the right triangle with the given side lengths. An editstate of readonly is given to prevent the user from accidentally destroying the compute by entering a value for field C.

```
<?xml version="1.0"?>
<XFDL version="4.1.0">
        <bgcolor content="array">
                <ae>128</ae> <ae>128</ae> <ae>128</ae>
        </bgcolor>
        <page sid="Pythagorean_Theorem">
                <bgcolor content="array">
                        <ae>192</ae> <ae>192</ae><ae>192</ae>
                </bgcolor>
                <label>Pythagorean Theorem Form</label>
                <field sid="A">
                        <label>Enter A:</label>
                        <value>3</value>
                </field>
                <field sid="B">
                        <label>Enter B:</label>
                        <value>4</value>
```

Apart from XFDL, Xforms is another project to represent forms as an XML application ([XFORM]). Xforms are currently under development by the W3C. The design of Xforms focuses on the increasing demands for improved human-computer interaction as well as the interaction mechanisms between the browser and the server. Xforms will be designed to cleanly distinguish between *form data, logic* and *presentation* ([XFORM2]).

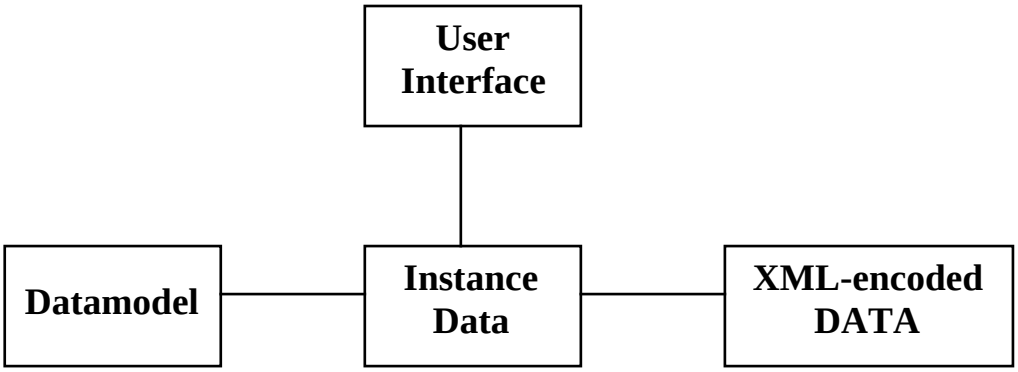

Fig. 7, Xform design layers: the Datamodel constrains the Instance Data, Instance Data is XML-encoded for exchange with the web server, the User Interface accesses the Instance Data via get/set methods in a Model-View-Control architecture

The data model allows the abstract structure of a form to be defined without explicitly specifying a user interface. Xforms will introduce a new user interface layer for richer user interaction. The device-indepedence will be limited to avoid the loss of functionality as we have now in HTML forms. Xforms are as such designed for integration with other XML applications.

The list of Xform requirements includes the following:

- Separation of purpose from presentation; Xforms fields should not be bound to a particular interface representation.
- Allow access and manipulation of forms via the XML Document Object Model by other scripting languages.
- Xforms should be able to send form data to the server as well formed XML.
- Device and application independence; usage of a form should not rely on some system specific way of input, like e.g. a double mouse click.
- Xforms will have a set of data types with basic control over it's content and achieve some form of input validation there. Functionality would be about the same as with XFDL: range constraints and patterns.
- Calculations and relations based on existing input fields should be expressed in the form.
- Saving and resuming; it should be possible for a user to save the form and then at a later time, to resume filling it out, perhaps from a different machine and even with a different user interface.
- Xforms should provide richer form controls and offer extended functionality for data acquisition.
- Other requirements proposed by the W3C include voice enhancements, paper enhancements for use with OCR (Optical Character Recognition), digital signatures, …

3.10.2 Verifying page authenticity

One problem with HTML documents and embedded scripting functions is that a user can alter the source before submitting data to the server. Even if you know that all of the users have a browser with for example JavaScript enabled, the actual call of the validation function can be removed and the web document can then be interpreted from the user's hard drive as if it was the original. Thus the server cannot rely on the client-side validation of the input received. If the server could check whether the web document containing the form was altered or not (in this closed user environment), client input could be accepted without further validation or it could simply be rejected. The user could be prompted to retry or denied further access.
In data transmissions, checksums are used to determine whether the message has been corrupted during transmission. Compression formats like RAR have a built in checksum to detect corrupt archives. This is done by calculating Cyclic Redundancy Codes (CRC) for each file and storing this CRC in the archive. Before extracting a file from the archive, the CRC is computed and compared with the original value. For file distributions in the Web (HTTP & FTP), Simple File Verification (SFV) can be used to generate a text file describing a set of files for transmission by listing their file size and a checksum. The receiver can then with this description verify whether the files he got are error-free and complete.
Applying such a technology to a web page can give the server a guarantee that nobody altered a document before retransmission to the server. A checksum can be computed on the data sent to the client and stored at the server, the client

browser can compute the same checksum of the page's content when posting data back to the server.

4. Methods for validating user input

4.1 Domain constraints

4.1.1 Numerical values

4.1.1.1 Syntax check

Numerical input can be tested on syntactical correctness by parsing it.
The standard notation for numbers can vary from country to country and because we are working on the Internet, the processing of these notations should be taken care of. Of course one could just as well give an example on the form showing how the data should be entered. But even in that case your application should be foolproof and invalid numbers should not be used in calculations, generating a run-time (data) type error.

Some examples of different notations for numbers:

123,45	123.45	123.45E10	123,-	-123.45

We suggest to take advantage of the parsing of the input number by at the same time converting numbers from the recognized valid forms and convert them to one single format that is used in the rest of the application to allow mathematical operations on it.
For numbers that are not compatible with any of the defined formats an error message is generated. Depending on how advanced the parser works, very specific messages can be generated pointing out where the error in the number is.

Some solutions:

If one doesn't need advanced checking of numbers one can:

- Use a native function of your programming language for type-checking.
- Treat the number as a string and test character by character.

For more complex parsers one can use a parser generator.
Given a grammar, these tools will produce a program that can check if input is well formed according to the specified grammar rules.

Some existing and well-known parser generators are:

- JavaCC: The Java Compiler Compiler

  JavaCC is a tool that reads a grammar specification and converts it to a Java program that can recognize matches to the grammar. In addition to the parser generator itself provides other standard capabilities related to parser generation such as tree building, actions, debugging, etc…[JavaCC]
- Lex / Yacc

  Lex (A Lexical Analyzer Generator) helps writing programs whose control flow is directed by instances of regular expressions in the input stream [LEX]. Yacc (Yet Another Compiler-Compiler) provides a general tool for describing the input to a computer program. The Yacc user specifies the structures of his

input, together with code to be invoked as each such structure is recognized [LEX].

- Flex / Bison

  Flex (A fast scanner generator) is a tool for generating scanners: programs which recognized lexical patterns in text [LEX]. Bison is a general-purpose parser generator that converts a grammar description for a context-free grammar into a, e.g. C program to parse that grammar [LEX].

4.1.1.2 Range check

Numerical values can easily be tested to check if they are within a given range.
Given a well formed input number *n* and some interval *[a, b]* where *b >= a*, then n lies within the interval if *n >= a* **and** *n <= b*.
Intervals of the form *[a, b[* [6] can also occur. In that case *n* lies within *[a, b[* if *n >= a* and *n* is only restricted by the lower bound *a*.

These range checks can either be manually programmed or a parser that takes as input a set of intervals can generate them. Operators as *union, set difference* and *intersection* can be included in the syntax to allow for more complex ranges.

4.1.2 String values

4.1.2.1 String length

When storing string values in a database one has to be sure that -depending on the database system- no empty strings are inserted or that the strings fit in the reserved field size of the database table.

Most programming languages have built-in functions for getting the string length.
A string can be compared to the empty string (“”) and if necessary replaced by a single space character (‘ ‘).

String overload can be avoided by limiting the string size at input level. This is built-in into HTML.

4.1.2.2 String syntax

If for example, a user wants to register a piece of software online and has to enter a personal serial number, the structure of the serial number can be tested.

A serial number might look like this: 12345-ABCD-678

[6] *]a, b[,]a, b]* are also possible intervals

A simple parser can check if the string is of the form [5 numbers][‘-‘][4 characters][‘-‘][3 numbers].

It is would not be safe to include the algorithm at the client side for checking the correctness of the serial number. If so, pirates can reverse-engineer the algorithm to make a serial number generator for illegal purposes.

4.1.3 Values from a set

When a user has to enter a value of which there are only limited values acceptable, there are two ways to validate this input.

One solution is to compare the input from the user in a big IF statement with all the acceptable solutions. This approach has some drawbacks. Take as an example a form that requires a user to enter the name of the country he/she lives in. First of all the user has to spell the name in exactly the same way the makers of the form did. Secondly, a lot of string comparisons are needed to validate the input.

Therefore it is better to present the user with a pull-down menu when he has to choose among a fixed number of values. This way the user cannot enter invalid data. Of course we have to rely on the user for selecting the correct value from the pull-down menu. When validating domain constraints we cannot tell if the user actually enters truthfully correct information.

4.2 Referential constraints

There are generally three ways to deal with referential constraints.

4.2.1 Referential constraints enforced by the DBMS

Many Database Managers refuse to store information if either entity integrity or referential constraints invalidate it. This would allow the programmers to completely rely on the DBMS when inserting a tuple. The only thing that should be done is to catch the error from the DBMS and to generate an understandable error message for the user using the web application.

4.2.2 Manually verify referential constraints

As implied in the previous point, many, meaning not all Database Managers check for referential correctness. The common used MySQL database server does not automatically verify referential constraints. The programmer himself can write extra database queries that first check if the foreign key of the tuple he is about to

store matches a primary key in the home relation. The programmer can then decide what action to take if the referential constraint gets violated. The drawback of this method is the loss of data independence. Data independence is an important concept in the separation of an application and the data it operates on. If the descriptions and logic for accessing the data are built into an application program, the program becomes dependent on the data. Changes to the structure of the data can require substantial alterations to the programs dependent on this data. In a DBMS, data independence is reached by incorporating the ANSI-SPARC architecture that provides both logical and physical data independence. This three-level architecture has a mapping from the *external schemas* (user views) to the *conceptual schema* (tables & relations) giving a logical data independence that refers to the immunity of external schemas to changes in the conceptual schema. A second mapping from the conceptual schema to the *internal schema* (storage structures) results in physical data independence, which in turn refers to the immunity of the conceptual schema to changes in the internal schema ([CBS 96]). The programmer now has to include elements of the table definitions in the application that describes which attributes belong to primary or foreign keys.
Subsequent changes to the database definition will most likely affect the operation of application.
If the web application allows the user to update tuples with referential constraints, strategies as listed in 2.1.3 can be applied to maintain data integrity.

4.2.3 Avoid invalid foreign keys entered by users

If the web application would make it impossible for the user to enter information in such a way that referential constraints are violated, it would eliminate the need for manually checking these if the DBMS has no facilities to do it automatically.
The best way to achieve this functionality is to generate a list of all valid entries for a foreign key. The user can then be presented with e.g. a pull-down list containing those values and thus limiting him to select a valid one. The list of all valid foreign keys can be found by querying all unique primary keys from the home relation. It is not needed to show the primary key to the user because this might be some personID number while the user is expecting to select the name of a person. The value shown in a pull-down list and the actual value sent to the server need not be the same.

4.3 Enterprise constraints

4.3.1 One cell constraints

A one cell constraint is a constraint that does not depend on other cells from either the same row, table or database. Generally, each value for an attribute dependent

on a one cell enterprise constraint, needs to satisfy some function implementing the constraint. Examples of such functions can be a test to see whether a number is prime or a function to test whether a serial number to register a commercial application is a valid one. If supported by the DBMS, these enterprise constraints can be implemented in the database itself. Otherwise, these validation functions can be implemented on both the client and the server side. Of course, care should be taken when considering what functions to implement on the client side. As mentioned in 4.1.2.2, putting a function that uses confidential information from a business should not be put on the client for everyone to view.

4.3.2 Multi cell constraints

Values for an attribute might as well be constrained in respect to values from other attributes but making it different from a domain constraint. Lets take for example a database application for administrating employee salaries. Domain constraints on the salary attribute of an employee might be an interval between the legal minimum wage and some maximum specified by the company. Extra enterprise constraints can then for example state that the salary of the manager has to be at least 35% more than the average salary of the other employees. Again, those constraints can be implemented directly into the DBMS if supported or it can be manually implemented on the server side by first making some queries and then compare it with the input before storing new data.
Such constraints can be checked on the client side as well. The server could, while generating the form interface for the user, include extra information from some queries needed to test the constraint. Once again, care should be taken on what information the user is allowed to view that way.

5 Examples

5.1 Domain constraints

5.1.1 Numerical values

5.1.1.1 Syntax check

The following example in Java uses a built-in function from the *java.lang.Integer* class that parses the string argument as a signed decimal integer.
Numbers that can be parsed by the following syntax: [-][0..9]$^{+}$ (any sequence of at least 1 digit, possibly preceded by a minus sign) are accepted.

Example in Java:

```
String intString = "123a4";

try {
int parsedInt = Integer.parseInt(intString);

} catch(NumberFormatException e) {
System.out.print("\n Error: Invalid integer " + e.getMessage() + "\n");
}
```

The next JavaScript example treats a number as a string. This is because the input for this number most likely comes from a `<INPUT Type="TEXT" …>` form element. The function *testInt* can be called from the `onClick` attribute of the form's submit button.
Input of "123A5" would result in an error message.

Fig. 8, JavaScript error message

Example in JavaScript:

```
function testInt(number) {

      s = number.length;
      stringIndex = 0;

      if(s > 0) {

              numberOk = true;

      } else {

              numberOk = false;
```

Another example demonstrates the parsing of (Belgian) telephone numbers.
Using the Java Compiler Compiler (JavaCC), one can specify a grammar and actions to be undertaken during the parsing process. Then a set of Java files will be generated that implement a parser for the grammar specified.
This grammar accepts both “012345678” and “012/34.56.78” which are not all pure numerical values. Numbers like “0123456781111” are also accepted, as the parser will first find the valid number “012345678”. This parser is not 100% correct but merely serves as an example of an alternate way to check user input.

Example in JavaCC:

```
PARSER_BEGIN(TelParser)


public class TelParser {

      public static void main(String args[]) {

              TelParser parser;
              System.out.println("Enter (Belgian) Telephone Number:");

              parser = new TelParser(System.in);

              try {
                      String theResult;
```

```
...

  |
  < #SLASH: "\\" | "/" >
  |
  < #ONE_DIGIT: ["0"-"9"] >
  |
  < # TWO_DIGITS: <ONE_DIGIT> <ONE_DIGIT> >
  |
  < # THREE_DIGITS: <ONE_DIGIT> <ONE_DIGIT> <ONE_DIGIT> >
```

5.1.1.2 Range check

A range check is very easy to implement as is shown in the next JavaScript function that verifies whether a user's age falls into the interval specified for the application needing the piece of information.

Example in JavaScript:

```
function testAge(age,minAge,maxAge) {
    if((age < minAge) || (age > maxAge)) {
            window.alert('Please enter a more realistic age.');
    }
}
```

5.1.2 String values

5.1.2.1 String length

The string length of user input can be limited at the client by specifying the maximum length for every text input box in the form. This can be easily achieved by the `MAXLENGTH` attribute. The example shows the HTML code for a text box where the user enters his login for some service and this input box is limited to accept no more than 8 characters. The Microsoft Internet Explorer browser blocks character input as soon as the maximum length is reached.

Example in HTML:

```
<INPUT TYPE="TEXT"
   MAXLENGTH="8"
   NAME="Login"
   SIZE="9"
   VALUE=""
>
```

5.1.2.2 String syntax

See example in 5.1.1.1.

5.1.3 Values from a set

HTML has facilities for creating pull-down menus from which a user can select either a single item or multiple items. It suffices to enumerate all the elements from the set and their corresponding values that will be sent to the server to fill the menu.
The following piece of HTML represents the pull-down menu from fig. 9 where a user can select his favorite music genre. “House” is marked as a default selection.

Example in HTML:

```
<SELECT NAME="MusicGenre" SIZE="1">

        <OPTION VALUE="Acid-Jazz"> Acid-Jazz
        <OPTION VALUE="Dance"> Dance
        <OPTION VALUE="Drum-n-base"> Drum-n-base
        <OPTION VALUE="Garage"> Garage
        <OPTION VALUE="House" SELECTED> House
        <OPTION VALUE="Jazz"> Jazz
        <OPTION VALUE="Lo-fi"> Lo-fi
        <OPTION VALUE="Lounge"> Lounge

</SELECT>
```

Fig. 9, Selecting values from a set

5.2 Referential constraints

5.2.1 Referential constraints enforced by the DBMS

Many database systems verify referential correctness of new data that is to be added or of existing data that is to be updated or deleted. The next example uses Microsoft Access and PHP_ to implement a small application to subscribe students for their exams. This is done by linking an exam to a student in a new table. If a student with a studentID that is not in the database is subscribed, referential integrity is violated and the Access database will give an error instead of storing the tuple. In the example, the real error message “SQL error: [Microsoft][ODBC Microsoft Access Driver] You cannot add or change a record because a related record is required in table ‘Students’., SQL state 23000 in SQLExecDirect” is suppressed and a simple “Error: Unknown student: …” is shown instead. Other database drivers allow extraction of different parts of the error message from the DBMS to construct a meaningful custom error message.

Example in PHP_

```
<?

header("Pragma: no-cache");
header("Cache-Control: no-cache, must-revalidate");

$mysqldbid = odbc_connect("test1", "", "");

if ($submit) {

// this code will be executed if the form has been sent
// ODBC error reporting is disabeled, if $status is false, referential
// integrity is violated

      $status = @odbc_exec($mysqldbid, "INSERT INTO ExamSubscriptions VALUES
```

```
…
      </SELECT>
      <P>Enter Student Role Number:
      <INPUT TYPE="TEXT" NAME="student" MAXLENGTH="5" SIZE="5">
      <P><INPUT TYPE="SUBMIT" NAME="submit" VALUE="Submit">
      </FORM>
      </BODY>
      </HTML>
```

Microsoft Access tables for the PHP example:

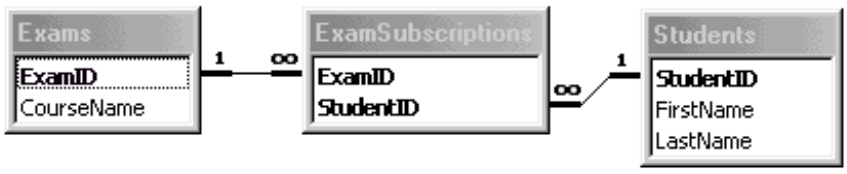

Fig. 10, MS Access tables and relations with referential constraints

5.2.2 Manually verify referential constraints

Not all database systems have facilities for enforcing referential constraints. To maintain data integrity, the server script in this example performs a query to check whether the student is in the database before adding the student-exam link. This is implemented in 5.2.3, which is a modification of the example in 5.2.1.

5.2.3 Avoid invalid foreign keys entered by users

As mentioned earlier, the user of the web application cannot enter information in such a way that storing it would invalidate referential constraints if the input were limited by e.g. pull-down menus. As the web application stores data in a database, a fixed set of input choices can either become insufficient for automatically maintaining referential correctness or it imposes a limit on the usage of the web application. Therefore the items for the pull-down menu are automatically generated every time the user requests the web page containing the form interface. The PHP script generates HTML code as shown in 5.1.3 using information fetched from a MySQL database. The MySQL tables used here have the same structure as those in the MS Access example.

Example in PHP_

```
<?

header("Pragma: no-cache");
header("Cache-Control: no-cache, must-revalidate");

$mysqldbid = mysql_connect("localhost", "jplasqui", "gnat");

mysql_select_db("forumtest", $mysqldbid);

if ($submit) {

    // this code will be executed if the form has been sent

    $foreign_key_check = mysql_num_rows(mysql_query("SELECT
    FirstName,LastName FROM Students WHERE StudentID=$student"));

    if($foreign_key_check) {

    // insertion only happens if no referential constraints are violated
    // for the student; exams are assumed to exist as they come from a
    // pull-down menu

            mysql_query("INSERT INTO ExamSubscriptions VALUES
            ($exam,$student)");

            echo("<P>Subscription OK<BR>");

    } else {

            echo("<P>Error: Unknown student: <I>$student</I><BR>");

    }

} else {

?>

    <HTML>

    <HEAD>
            <TITLE>Exam subscriptions</TITLE>
    </HEAD>

    <BODY>
```

…

…

```
<P><FORM ACTION="foreign_key.php3" METHOD="POST">

    Select course:

    <SELECT NAME="exam">

<?

    $query_result = mysql_query("SELECT ExamID,CourseName FROM Exams",
    $mysqldbid);

    // generate items for pull-down menu
```

MySQL tables for the PHP_ example:

```
CREATE TABLE Students(

    StudentID INTEGER NOT NULL PRIMARY KEY,
    FirstName VARCHAR(64) NOT NULL,
    LastName VARCHAR(128) NOT NULL

);



CREATE TABLE Exams(

    ExamID INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
    CourseName VARCHAR(128) NOT NULL

);
```

5.3 Enterprise constraints

5.3.1 One cell constraints

A one cell constraint is some form of a special domain constraint. An example of a one cell constraint is a rule used in banking to verify the well-formedness of a bank account number. A belgian account has the from of “123-4567890-02”, the last two digits are a built-in check, it should be equal to the remainder after

dividing the first ten digits by 97. The JavaScript function performs this check; it does not first test whether the account number is in the format mentioned above.

Example in JavaScript

```
function testActNbr(actNbr) {
      nbr = actNbr.substring(0,3) + actNbr.substring(4,11);
      check = actNbr.substring(12,14);

      if((nbr % 97) == check) {
              return(true);
      } else {
              window.alert("Invalid Account number ["+actNbr+"].");
      }
}
```

5.3.2 Multi cell constraints

In this example, the value of a field is constrained by other tuples for the same relation. The enterprise constraint states that a manager should at least earn 30% more than the average of the other (non-manager) employees. The server script generates the client side script with the correct test values.

MySQL table

```
CREATE TABLE Employees(
      EmployeeID INTEGER NOT NULL,
      JobStatus SET('Manager','Engineer','Junior'),
      Salary DOUBLE,
      PRIMARY KEY (EmployeeID)
);
```

```
<?
$mysqldbid = mysql_connect("localhost", "jplasqui", "gnat");
mysql_select_db("forumtest", $mysqldbid);

if($submit) {
      echo("salary $salary accepted...");
      // DataBase update here
} else {
// vars: $employee_id
?>
      <HTML>
      <HEAD>
```

```
…

        <P><FORM NAME="SalaryForm"
            ACTION="edit_salary.php3"
            METHOD="POST"
            OnSubmit="return testSalary(document.SalaryForm.salary.value);"
           >


        Enter new salary for employee
        <? echo(" $employee_id, $job_status"); ?>:

        <P><INPUT TYPE="TEXT" NAME="salary" SIZE="7">

        <P><INPUT TYPE="SUBMIT" NAME="submit" VALUE="Submit">

        </FORM>


        </BODY>
```

6 Conclusions and further work

6.1 Further work

Before coming to the conclusions, we will first briefly discuss a number of issues that are not elaborated in this thesis but are relevant for the problem and therefore could be given attention in possible further work.

6.1.1 Synchronization client/server rules

Validating user input comes down to applying business rules to it and check for conformance with the rules. If one chooses to use both server side and client side validation, problems can arise when the business rules change. If the rules on the server change before they are propagated to the client side, users will get errors or input that is now accepted by the server will still be rejected by the browser. These problems can be solved by changing both client side and server side implementations of the validation system at the same time. On the server side, this may only involve a change in the definition of a database table in contrast to altering probably several lines of code on the client. If the business rules were specified in some formal language, client side code could be generated from it. In some extent, server side validation could also be based on those business rules.

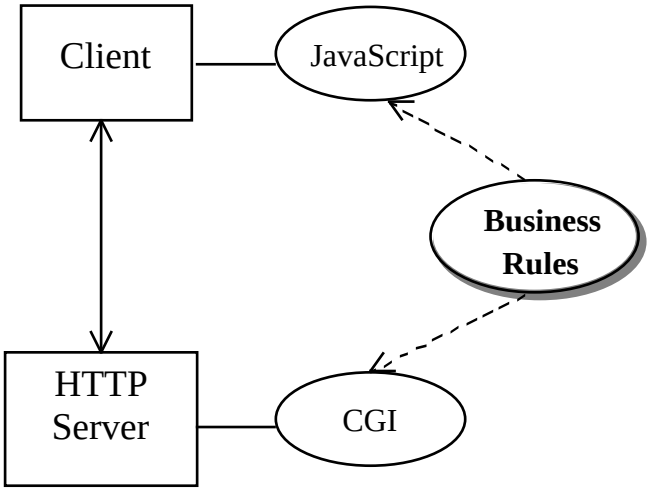

Fig. 11, Business rules

6.1.2 Web document authenticity

As listed in 3.10.2, further research is needed on mechanisms to guarantee document authenticity. His includes researching whether existing technologies already implement this, verifying which technologies can be used for it and how this extra processing impacts server performance and the way of serving documents to the users. Some documents are self-modifying, how will such a modified document be compared to the original? Also, in what way will browsers have to be enhanced to compute a checksum for a page and how can this checksum be sent to the server with current technologies (e.g. using cookies).

6.2 Conclusions

In this thesis, we have investigated the problem of validating user input in web applications. We have seen that an increasing amount of programming languages allows client and/or server side validation of user input. User input can be validated at the client but the developers have to consider possible data overload for the client when checking foreign keys or the possible risk of including sensitive information in the client browser. A suitable server side scripting language has to be chosen based on the frequency of requests for the application to run. Performance of web applications is directly related to the way they are executed on the server. Compiled scripts execute faster than scripts that need to be interpreted at every request.

Clearly, client-side validation has several benefits:

- It reduces load on the server. “Bad data” are already filtered out when input is passed to the server-based program.
- It avoids a series of connections to the server to handle faulty data. This saves bandwidth for the server.
- It reduces delays in case of user error. Validation otherwise has to be performed on the server, so data must travel from client to server, be processed, and then returned to client for valid input.
- It simplifies the server-based program.

Although there are several advantages when using client-side validation, very few web applications actually use it. Actually there are several reasons why client-side validation is almost not used today:

- There exist quite a number of different technologies that allow some form of client-side validation; however there is no guarantee that the user’s browser is compatible with the used technology.
- The above implies that a company cannot just rely on one technology. If the company rules out users that cannot or don’t want to use a certain technology, then they lose a probably large piece of their so much needed audience. On the other hand, providing a choice to the users means a greater cost for the company.
- It is still possible that certain user agents cannot use any form of client-side validation or that users disable the interpretation of some client-side scripting language or even play around with hidden data or function calls in the form. Thus the server needs to check the input or at least some of it anyhow, in order to avoid database inconsistencies or incorrect behaviour of the application.
- Not all kinds of user input can be validated at the client. Especially if the validation process involves confidential and/or critical information from the company. Validating foreign keys on the client may require a huge amount of

data that needs to be sent with the form to the client. Aside from possible privacy issues with that data, the form generating process and the traffic can cause a greater load on the server then what might be saved with client-side validation. Thus some data is better validated on the server.

- Using some kind of a CGI to validate input and send feedback to the user in valid (X)HTML is in fact the only reliable way of validating user input in a way that is compatible with all devices.

To conclude, client-side validations are good for preventing roundtrips to the server but are not yet to be trusted such that server-side validation becomes redundant.

References

Books

[CBS 96] Thomas Connolly, Carolyn Begg, Anne Strachan
Database Systems – A Practical Approach to Design, Implementation and Management

Addison-Wesley, 1996

[V 99] Prof. Dr. E. Vandijck
Databases
Dienst Uitgaven VUB

[DFAB 98] Alan Dix, Janet Finlay, Gregory Abowd, Russel Beale
Human-Computer Interaction, second edition
Prentice Hall Europe, 1998

[H 96] Scott Hillier
Inside Microsoft Visual Basic, Scripting Edition
Microsoft PRESS, 1996

Papers

[L 97] Barry M. Leiner, Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Kleinrock, Daniel C. Lynch, Jon Postel, Lawrence G. Roberts, Stephen S. Wolff
The Past and Future History of the Internet
Communications of the ACM, Vol. 40, No. 2, February 1997

[C 99] Jim Conallen
Modeling Web Application Architectures with UML
Communications of the ACM, Vol. 42, No. 10, October 1999

[O 98] John K. Ousterhout
Scripting: Higher Level Programming for the 21st Century
IEEE Computer magazine, March 1998
http://www.scriptics.com/people/john.ousterhout/scripting.html

URLs

[Netscape] Client Side State – HTTP Cookies
http://www.netscape.com/newsref/std/cookie_spec.html

[RSA] RSA Security Inc.
http://www.rsasecurity.com

[WAPF] WAP Forum
http://www.wapforum.org/

[JavaCC] JavaCC
http://www.metamata.com/JavaCC/

[LEX] The LEX & YACC Page
http://www.combo.org/lex_yacc_page/

[PHP3] PHP
http://www.php.net

[ZEND] ZEND
http://www.zend.com

[XML] XML for Managers – Evaluating SGML vs. XML from a Manager's Perspective
http://www.arbortext.com

[XHTML] XHTML 1.0
http://w3.org/TR/xhtml1/

[JSVL] William Crawford
Developing Java Servlets
http://webreview.com/

[JWS] Phil Inje Chang
Inside The Java Web Server
http://java.sun.com/

[CF] The ColdFusion Web application server and development from Allaire
http://www.allaire.com/products/coldfusion/

[PERL] Introduction to PERL
http://www.cclabs.missouri.edu/things/instruction/perl

[MATH] The Mathematical Markup Language
http://www.w3.org/TR/2000/WD-MathML2-20000328/overview.html

[XSCH] XML Schema Part 0 – Primer
http://www.w3.org/TR/xmlschema-0/

[XFDL] Extensible Forms Description Language (XFDL) 4.0
http://www.w3.org/TR/NOTE-XFDL

[XFORM] XForms requirements
http://www.w3.org/TR/xhtml-forms-req

[XFORM2] XForms 1.0 : Data model
http://www.w3.org/TR/xforms-datamodel