On the design of the URL

Steven Pemberton
CWI

Notations can affect the way we think, and how we operate; consider as a simple example the difference between Roman Numerals and Arabic Numerals: Arabic Numerals allow us not only to more easily represent numbers, but also simplify calculations and the manipulation of numbers.

One of the innovations of the World Wide Web was the URL. In the last 30 years URLs have become a ubiquitous element of everyday life, so present that we scarcely even grant them a second thought. And yet they are a designed artefact: there is nothing natural about their structure – each part is there as part of a design.

This paper looks at the design issues behind the URL, what a URL is meant to represent, how it relates to the resources it identifies, and its relationship with representational state transfer (REST) and the protocols that REST is predicated on. We consider, with hindsight, to what extent the design could have been improved.

The URL

The start of the open internet in Europe in November 1988 was an impetus for the creation of the World Wide Web. It couldn't have been created in a more international environment: by a Brit, and a Dutch-speaking Belgian with a French surname on designated international territory straddling the Swiss and French borders. Tim Berners-Lee wrote his proposal in March 1989, the first web server was running in 1990, and announced in 1991 [tbl2].

The World Wide Web cleverly combined several existing technologies such as hypertext, SGML, and the internet to create a successful global information system. One of its innovations was the URL, allowing resources from diverse sources over the whole world to be identified for retrieval.

There are four terms commonly used: URL, URN, URI, and IRI.

– URL: Uniform Resource **Locator**, to locate and retrieve a *resource*, such as

  http://www.cwi.nl/~steven/Talks/2020/10-09-urls/

https://doi.org/10.1075/da.2020.pemberton.design
*Proceedings of Declarative Amsterdam 2020*8 and 9 October 2020

– URN: Uniform Resource **Name**, to give a name to a resource, without specifying how to retrieve it.

  `urn:isbn:0451450523`

– URI: Uniform Resource **Identifier**
  The umbrella name for URL and URN. [rfc2396]

– IRI: **Internationalised** Resource Identifier, using a wider range of Unicode characters [rfc3987]:

  `https://www.石川.日本/雅康#mimasa`

This paper will largely consider the URL.

The first URLs to be published were in Berners-Lee's announcement of the Web in 1991 [tbl1].

An early design document (where they were call UDIs, and not URLs), *Universal Document Identifiers on the Network* [osids29] (1992) explained some of the design reasoning behind parts of the notation.

The first formal definition was in [rfc1738] (1994), which was later updated in 2005 [rfc3986].

It was later internationalised in a delta specification (in other words, only changing parts of the definition) in [rfc3987] (2005), and similarly updated for IPv6 in [rfc6874] (2013).

The document *URI Design and Ownership* (2014) [rfc7320] explains how to go about designing other sorts of URI.

Resources

A URL doesn't necessarily locate a unique *document*, but uses the abstraction of a *resource*. Although the term was used before then, it was first described in [rfc2396]:

> *A resource can be anything that has identity. Familiar examples include an electronic document, an image, a service (e.g., "today's weather report for Los Angeles"), and a collection of other resources. Not all resources are network "retrievable"; e.g., human beings, corporations, and bound books in a library can also be considered resources. The resource is the conceptual mapping to an entity or set of entities, not necessarily the entity which corresponds to that mapping at any particular instance in time. Thus, a resource can remain constant even when its content – the entities to which it currently corresponds – changes over time, provided that the conceptual mapping is not changed in the process.*

As an example, a PDF of a book, and an ebook of the same book can be considered as two representations of the same resource. Furthermore, a French translation of the same book can also be considered to be a representation of that resource. Therefore a URL can refer to a group of documents that all represent the same resource; aspects of the HTTP protocol allow you to further specify which of the representations are of interest. We will consider this further when talking about the use of the Accept headers in HTTP.

The Structure of the URL (and of this paper)

A URL consists of several distinct parts, each with its own special character to demarcate either its start or end:

```
scheme: //authority /path ?query #fragment
```

the authority part is dependent on the scheme, but the structure of the authority for HTTP is

```
user:password@ host :port
```

Every single part of a URL is optional, though there are some combinations you seldom or never see in real life, such as leaving the scheme out but not the host:

```
//www.example.com/
```

Or retaining the scheme but not the host:

```
http:/file.txt
```

This optionality means that even the empty string is a valid URL (meaning "the current resource"), and does even have some uses. [rdfa]

The syntax of a URI is very general, and different parts may have different syntaxes. The URI specifications only define the characters that may be used, but in general not the syntax of those parts. Owners of other parts are:

- the scheme definition defines the syntax of the authority and path;
- the media type definition of a retrieved resource defines the syntax of a fragment.

It is somewhat unclear who owns the syntax of a query [rfc7320].

A client uses a URL to retrieve a resource from a server; the client uses all parts except the fragment to retrieve the resource while the server only sees and uses the path and the query parts. The client uses the fragment once the resource has been retrieved.

We shall now treat each part of a URL one by one, before returning to some general remarks about the use of URLs.

Scheme

```
scheme: //user:password@ host :port /path ?query #fragment
```

The scheme principally represents how to access a resource, in particular:

- the protocol to be used,
- the default port number.

Not all schemes have a formally defined protocol (nor port for that matter), for instance `mailto:` and `file:`, which depend on native properties of the client platform, and `news:` which leaves it open how the particular news message is to be retrieved, either by protocol such as `nntp`, or locally.

Some ill-behaved applications use schemes in order to identify document types for that application. This is wrong: schemes are there for defining how to retrieve resources; it is the returned mediatype of the retrieved resource that should be used for initiating an application, since then the application can be started independently of how the resource is retrieved.

The most important scheme is clearly `http`, although it was clever of Berners-Lee to incorporate other schemes in his design, since the Web immediately interoperated with other protocols when it was introduced.

An obvious observation is that there is no reason for the name of the scheme to match the name of the protocol it identifies. It can be seen as one of the great missed chances not to have used `web:` as the name of the main scheme, rather than the technical-looking `http:` [gbldgk].

The https: scheme will be considered later.

Authority

```
scheme: //user:password@ host :port /path ?query #fragment
```

The double slash announces the start of the *authority*, the name for everything up to the path, in other words, for http, `user:password`, `hostname`, and `port`.

If there is no authority part, you don't need to use the double slash; that notwithstanding, you very often see file: URLs of the form

```
file:///home/user/documents/file.txt
```

when in fact

```
file:/home/user/documents/file.txt
```

means the same thing.

Any suitable character would have done to announce the start of the authority. Berners-Lee reports [faq] that he took the double slash from the Apollo file system, where they needed a syntax to differentiate a local file

```
/folder/file
```

from a file over the network:

```
//machine/folder/file
```

As he said:

> "When I was designing the Web, I tried to use forms which people would recognize from elsewhere."

Berners-Lee has later been reported [nyt] as saying the double slash was the only thing he would change in his design because "it is unnecessary". However, since it is useful to be able to distinguish between the machine (the authority) and the path, you need to be able to distinguish between

```
http:machine/file
```

and

```
http:directory/file
```

and unless you dictate that a scheme always requires an authority (which isn't out of the question), the difference is not visible.

User and password

```
scheme: //user:password@ host :port /path ?query #fragment
```

It may seem bizarre in the context of the modern web that there is even a place in a URL for a username and password, let alone that anyone might use it: since they are both exposed in any document containing the URL, and exposed again as they are passed over the connection (there is no secure method of transmitting them in the protocol), there is frankly no security provided by the password at all.

The reason they are there is almost certainly because ftp required them to retrieve documents, and so they had to be present in the URL [osids29]. The username would almost always be *anonymous*, and so there was no security issue involved in that case.

We will say more about passwords later.

Host

```
scheme: //user:password@ host :port /path ?query #fragment
```

The syntax of the host part was already a given, having been defined for the DNS system in the mid 1980s [dns].

It can be either a hostname using DNS, or an IP address. Interestingly, a hostname can also be a number, so you may not be able to tell them apart until the very last segment:

```
192.168.178.com
```

and it is only possible at all because none of the highest-level domains consist purely of digits.

The host part of a URL originally identified a unique machine. It was only later realised that several hostnames could point to the same machine, and that a single machine could therefore host servers for more than one hostname. Unfortunately the protocol didn't support this (the protocol assumed the server knew its own identity), and so required an update to the HTTP protocol to support this properly [http11], letting the server know which domain the request was for (and therefore letting it adapt to which host it was serving the current request for).

Because originally the host *did* identify a single machine, many sites kept one machine specifically reserved as a Web server. Apparently the first two URLs published in 1991 [tbl2] were

```
http://cernvm.cern.ch/FIND
```

and

```
http://info.cern.ch/hypertext/WWW/TheProject.html
```

Notably neither server was called "www", and yet within a short period, nearly all webserver hosts were called exactly that, so that many people concluded that "www" somehow was an essential part of a URL. Again, it is frankly surprising that people didn't settle on 'web' instead of 'www', since www, with 9 syllables in English, is an acronym that is 3 times longer than the 3 syllable phrase it stands for.

Port

```
scheme: //user:password@ host :port /path ?query #fragment
```

Each protocol has a default port. For instance, http uses 80. However it doesn't have to be at 80, and this is the place you can change it. However, it is rather odd that it is here, since it is associated with the protocol, not the authority. It should really have been something like

```
http@8080://example.com/doc.html
```

or similar.

Endianism

An address in the UK looks something like:

```
Steven Pemberton
21, Sandridge Road,
St. Albans,
Herts,
UK
```

What you see is that it is almost entirely little-endian: from left to right and top to bottom the information gets more and more significant (except the 21, which is big-endian, the 2 being more significant than the 1).

On the other hand, an address in Soviet Russia looked like this:

```
РОССИЯ
  г.Москва 125252
  ул.Куусинена
21-Б
Международный Центр Научной и Технической
Информации
Чичикову П.И.
```

which is *entirely* big-endian, even down to the person's name with family name first.

We're confronted with clashes in endianism all the time [peace]. For instance, while dates are little-endian, such as 31/12/1999, (except in the USA where they are just mixed endian), times are big-endian, like 12:30:59.

Numbers

We inherited our number system from Arabic.

The interesting thing is if you look at a piece of Arabic writing, such as this from the Wikipedia page on the Second World War [ww2]:

> الحرب العالمية الثانية: هي نزاع دولي مدمر بدأ في الأول من سبتمبر 1939 في أوروبا وانتهى في الثاني من سبتمبر 1945، شاركت فيه الغالبية العظمى من دول العالم، في حلفين رئيسيين هما: قوات الحلفاء ودول المحور. وقد وضعت الدول الرئيسية كافة قدراتها العسكرية والاقتصادية والصناعية والعلمية في خدمة المجهود الحربي، وتعد الحرب العالمية الثانية من الحروب الشمولية، وأكثرها كلفة في تاريخ البشرية لاتساع بقعة الحرب وتعدد مسارح المعارك والجبهات فيها، حيث شارك فيها أكثر من 100 مليون جندي، وتسببت بمقتل ما بين 50 إلى 85 مليون شخص ما بين مدنيين وعسكريين، أي ما يعادل 2.5% من سكان العالم في تلك الفترة.

what is notable is that even though the text is read right-to-left, the numbers are still what an English-language reader would consider "the right way round".

In other words, in the original Arabic form, numerals were little-endian (which has some slight arithmetical advantages by the way), but were imported into Western languages back-to-front, and so became big-endian.

It is also interesting to note that historically, before the introduction of Arabic numerals, English speakers spoke numbers (at least under one hundred) little-endian:

- sixteen,
- four-and-twenty blackbirds baked in a pie,
- the time is five and twenty past three.

Arabic numerals started being introduced during the enlightenment, which may have affected how we spoke numbers, since in English at least we now speak numbers big-endian, with the exception of the numbers between 13 and 19 inclusive.

Shakespeare used both styles of numbering: he used little-endian numbers (four and twenty) about twice as often as big endian (twenty four), which may show that the style of speaking numbers was changing at that time (though you need to take into account the requirements of the metre).

Endianism in URLs

At the point where the authority ends, and the path begins, a URL shifts from little-endian to big-endian.

> www.example.com - little-endian
> /Talks/2020/09/slides.html - big endian.

Berners-Lee has in the past suggested [faq] that he would have preferred

> http:com/example/www/Talks/2020/10/slides.html

but this does have a problem that it is hard to locate where the authority ends, and where the path begins, and could even be ambiguous.

Path

```
scheme: //user:password@ host :port /path ?query #fragment
```

The syntax of the path suggests a file path, and indeed that's what inspired it, and was indeed so on the first implementations, and is still so with many current ones; but it isn't a requirement.

In fact, officially, the path is opaque: you cannot necessarily conclude anything from it. For instance, there is no *a priori* reason why

```
/2020/Talks/
```

and

```
/Talks/2020/
```

shouldn't refer to the same thing, and there are indeed examples of such web sites.

Type

At the end of a vast number of URLs is a file type

```
http://www.example.com/index.html
```

and

```
http://example.org/logo.gif
```

and this reflects that many implementations map a URL directly onto the filestore.

This is a great pity, because it means we lose out on a great under-used property of http: the accept headers.

Accept Headers

The HTTP protocol has a number of headers that allow you to characterise properties of the resource that you are interested in. This is called *content negotiation*, though there is in fact no actual negotiation going on. The properties are:

- language: which (natural) languages you would prefer, and which you will accept,
- document type: similarly, which you prefer, and which you are willing to accept,
- character set/encoding,
- compression types your machine can deal with.

The latter two are more of interest for machines, but the first two are definitely useful for people.

If a server used the accept-type header, then there would be more flexibility in what could be served. For instance, if a page specified an image without committing to a particular document type (in other words, specifying it as a *resource*):

```
<img src="logo"/>
```

(rather than `logo.gif`) then a client program could include an `Accept` header that specified which image types it preferred, and the site could serve the closest one that matched.

As an example, the deployment of the `png` image type was delayed for a long time partly due to the lack of wide availability of this feature. A site could have include the `png` version of the logo as well as the `gif` version, and served the `png`

version to clients that accepted it, as it became adopted, and continued to serve the gif version to older clients. As it was, sites had to wait for a critical mass of clients to accept it, before being able to deploy png images (and leaving the remaining older clients in the cold).

Similarly, if a site had a document available in several different types (HTML, PDF, epub, etc.), a client program could list the types in preference order in order to get the one best suited to the platform: for instance, an ebook reader might be able to handle PDF, but an epub version would be far better. A program could also check if a particular type was available by saying it only accepted that one type.

With Accept-Language it is similar. Many sites offer their content in a variety of languages, and allow you to click to select a different one. And yet they don't need to ask: the HTTP protocol *says* which are your preferred languages. The site google.com in particular is really bad at this: if you take your laptop to another country, their interface suddenly switches to the (or a) language of that country, even if your browser is telling them which language you want. In the international train between Amsterdam (The Netherlands) and London (UK), also passing through Belgium and France, Google serves the Swedish interface! (Probably because of the ISP being used).

One of the shortcomings of URLs is that you can't specify these features. For instance it would be useful if you could say you specifically wanted the Dutch version of a PDF document with a syntax like

```
http://example.com/specification?[type=pdf;lang=nl]
```

or that you preferred it, but were willing to accept something else:

```
http://example.com/specification?[type=pdf,*;lang=nl,en]
```

The protocol supports it; it would be easy to accomplish, and accept headers would be more likely to be adopted if users had control over it.

However, another shortcoming of many servers is that while they do use the Accept: headers to decide what to give you, they don't return a suitable response code when they fail to supply what you asked for. Typically you get a 200, as if all were well. This is nearly as bad as returning a 200 when a 404 should have been returned, and it means that even if you wanted to, you couldn't write a program to discover what a site offered.

It could be argued that there is a missing property in the accept protocol: for sites that have dated versions of resources, it would be useful to ask for the latest version of the resource before a certain date, or the earliest after a certain date. W3C itself has dated versions of resources, as does, for instance, the Internet

Archive, and news sites could offer news items published on a particular date if this were available.

Query

```
scheme: //user:password@ host :port /path ?query #fragment
```

The query allows you to send a small amount of data with the URL, rather than in the body of the HTTP request.

The major design blunder of the URL for HTML was using an ampersand to separate the values, since it is a special character in HTML, meaning that you can't just paste a URL into HTML, but have to replace every occurrence of `&` with `&`.

There was later a rear-guard action to try and replace it with semicolon, but it came too late.

Another weirdness of the HTML query is that spaces should officially be replaced with a "+", and therefore "+" should be replaced with %2B. More on this shortly.

Fragment

```
scheme: //user:password@ host :port /path ?query #fragment
```

As mentioned earlier, the fragment is for use after the resource is retrieved. Its syntax and how it is used is up to the specification for the media type of the resource returned.

Even though they are sometimes called 'fragment identifiers', and for HTML *must* be identifiers, there is no syntactic requirement in general that they be identifiers.

Berners-Lee reports [faq] that he chose the # sign, because of its similar use in postal addresses in the USA, indicating a sub-location in an address.

Special Characters

As we have seen, a URL uses a number of special characters for its syntax. They are, in order,

- : Used in three places - to signal the end of the scheme, to separate username and password, and to signal the start of the port.
- // To signal the start of the authority
- @ To signal the end of the user/password
- / To signal the start of the path
- ? To signal the start of the query
- # To signal the start of the fragment.

The general rule is that to the left of the last use of the character in the URL syntax, non-syntactic uses of the character must be encoded, and to the right, they don't have to be. For instance:

- a colon in a password has to be encoded, but not in a path.
- a question mark in a path has to be encoded, but not in a query or fragment.

One character was overlooked: this rule doesn't apply to #, which still has to be encoded in a fragment.

Encoding

The web (and URLs) were defined pre-Unicode, and initially used Latin-1, an 8-bit character encoding. This was already better than the rest of the internet, which used pure ASCII (for instance DNS is ASCII [dns]). Unfortunately, the syntax for encoding a character in a URL reflects this fact, and is an 8-bit encoding, just two hex digits: there is no terminator; as an example a space is encoded as %20. This meant that when Unicode, an encoding that doesn't fit in 8 bits, was introduced, there was no way to encode its characters; you would want to just use the codepoint of a character; unfortunately you have to know how to convert it into UTF-8, and encode those bytes.

So for

```
search?q=♥
```

while ideally you would want to write something like

```
search?q=%2665;
```

you have to write

```
search?q=%E2%99%A5
```

This is a shame.

Spaces

A URL may not contain an (unencoded) space, and this was a deliberate design decision:

> *The use of white space characters has been avoided in UDIs [an early name for URLs]: spaces are not legal characters. This was done because of the frequent introduction of extraneous white space when lines are wrapped by systems such as mail, or sheer necessity of narrow column width, and because of the inter-conversion of various forms of white space which occurs during character code conversion and the transfer of text between applications.* [osids29]

Apart from this reason, it also means that a list of URLs can be space separated in an attribute without ambiguity (used in HTML for the `archive` attribute for instance).

However, there is the weirdness that spaces everywhere but in a query should be encoded with %20, but in a query with +, therefore meaning that + has to be encoded only in a query.

HTTPS

The https scheme is a mistake, one of a number of early mistakes introduced by Netscape, such as <blink>, <img>, and <frame>.

The URLs

```
http://www.example.com/doc.html
```

and

```
https://www.example.com/doc.html
```

locate the same resource in every way. The only difference is in *details* of the protocol being used to retrieve the resource.

In other words, whether the connection is encrypted or not should be part of the protocol negotiation; we as users shouldn't have to care about, nor be confronted with, the negotiation between the machines about how they are going to transport the data.

Now we are being forced to change all our documents to use https links. That shouldn't have been necessary.

Passwords

The password situation on the web is an unnecessary disaster.

Basic authentication in the URL must be ignored of course.

The way it is done now is that every site has its own password system (and rules for passwords), and requires every site builder to understand the finer points of passwords (which they don't), and protect the passwords sufficiently (which they don't).

The combination is a royal security mess.

So let us briefly talk about public key cryptography.

Public Key Cryptography

In such a system, everyone has two matched keys:

- one is **public**: anyone can get a copy;
- one is **private**: only you have it.

'Keys' are very large numbers, 300 digits or more. You 'lock' messages with these keys, which in reality means the messages are scrambled using those large numbers.

- You can lock a message with either key, making it unreadable;
- if you lock with one key, **only the other key can open it**.

So if I lock a message with my **private** key (so that it can only be opened with my public key), and send it to you, you can get a copy of my **public** key, and if it opens the message, you know it was really from me. No more spam or phishing! Similarly, if I lock a message with your **public** key (so it can only be opened with your private key), and send it to you, I know that only you can open it to read it.

Combining those two things, I lock a message, once with **my** private key, and a second time with **your** public key, and send it to you, you know it's really from me, and only you can read it.

We can use the same process instead of passwords when logging in to websites. This could just be added to the protocol.

Registering with a site

You want to create an account at an online shop. You click on the "Create an account" button.

The site asks you to for a username, which you provide, and click OK. The site checks the user name is free, and returns a message to your browser that you

want to register. The browser asks if you really do want to register, and if you click yes, the browser and the site exchange public keys.

Logging in to a website

You fill in your username, and click on Log in.

The site looks up your username, generates a one-time password, locks it with your public key, and its own private key, and returns it to the browser. Your browser knows it is really from the site (and not someone pretending). The browser asks if you really want to log in with that username. If you say yes, it unlocks the password and sends it back, this time locked with your private key and the site's public key.

The site therefore knows it is really you and lets you in, without you having had to type in a password!

Conclusion

URLs were a great innovation, and have shown themselves to be useful, and have proven to be durable. While they exhibit some design faults, that is easy to say in hindsight. Those faults are lessons we can take on board for future designs of notations.

References

[dns] P. Mockapetris, Domain Names – Concepts and Facilities, *IETF*, 1983 https://tools.ietf.org/html/rfc882. https://doi.org/10.17487/rfc0882

[faq] Tim Berners-Lee, Frequently Asked Questions, *W3C*, 1996–, https://www.w3.org/People/Berners-Lee/FAQ.html#etc

[gbldgk] Steven Pemberton, Gbldgk, SIGCHI Bulletin, Vol.27 No.2, April 1995, https://www.cwi.nl/~steven/vandf/1995.2-Gbldgk.html

[http11] R. Fielding et al., Hypertext Transfer Protocol – HTTP/1.1, *W3C/IETF 2616*, 1999, https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.23. https://doi.org/10.17487/rfc2616

[nyt] Steve Lohr, The Web's Inventor Regrets One Small Thing, *NY Times*, 2009, https://bits.blogs.nytimes.com/2009/10/12/the-webs-inventor-regrets-one-small-thing/

[osids29] Tim Berners-Lee, Jean-François Groff, *Robert Cailliau, Universal Document Identifiers on the Network (OSI-DS 29)* (1992), OSI-DS 29, https://www.w3.org/Protocols/old/osi-ds-29-00.txt

[peace] Danny Cohen, On Holy Wars and a Plea for Peace, 1980, *IETF*, https://www.ietf.org/rfc/ien/ien137.txt

[rdfa] Steven Pemberton, RDFa for HTML Authors, *W3C*, 2009, https://www.w3.org/MarkUp/2009/rdfa-for-html-authors
[rfc1738] T. Berners-Lee, L. Masinter, M. McCahill Uniform Resource Locators (URL), *RFC 1738* (1994), https://tools.ietf.org/html/rfc1738. https://doi.org/10.17487/rfc1738
[rfc2396] T. Berners-Lee, R. Fielding, L. Masinter, Uniform Resource Identifiers (URI): Generic Syntax, 1998, https://tools.ietf.org/html/rfc2396, – merges rfc 1736 and rfc1737
[rfc3986] T. Berners-Lee, R. Fielding, L. Masinter, Uniform Resource Identifier (URI): Generic Syntax, *RFC 3986* (2005), https://tools.ietf.org/html/rfc3986. https://doi.org/10.17487/rfc3986
[rfc3987] M. Duerst, M. Suignard, Internationalized Resource Identifiers (IRIs), *RFC 3987* (2005), https://tools.ietf.org/html/rfc3987. https://doi.org/10.17487/rfc3987
[rfc6874] B. Carpenter, S. Cheshire, R. Hinden, Representing IPv6 Zone Identifiers in Address Literals and Uniform Resource Identifiers, *RFC 6874* (2013)
[rfc7320] M. Nottingham, URI Design and Ownership, *RFC 7320* (2014), https://tools.ietf.org/html/rfc7320. https://doi.org/10.17487/rfc7320
[tbl1] Tim Berners-Lee, WorldWideWeb wide-area hypertext app available, comp.sys.next.announce, 1991, https://groups.google.com/g/comp.sys.next.announce/c/avWAjISncfw
[tbl2] Tim Berners-Lee, Qualifiers on Hypertext links..., *alt.hypertext*, 6 Aug 1991, https://www.w3.org/People/Berners-Lee/1991/08/art-6484.txt
[ww2], , https://ar.wikipedia.org/wiki/الحرب_العالمية_الثانية

Biographical notes

Steven Pemberton is a researcher affiliated with CWI Amsterdam, the Dutch national research centre for mathematics and informatics. His research is in interaction, and how the underlying software architecture can support users. He co-designed the ABC programming language that formed the basis for Python. Involved with the Web from the beginning, he organised two workshops at the first Web Conference in 1994. For the best part of a decade he chaired the W3C HTML working group, and has co-authored many web standards, including HTML, XHTML, CSS, XForms and RDFa. He now chairs the XForms group at W3C.

https://orcid.org/0000-0002-8784-3814