Journal of Universal Computer Science, vol. 8, no. 10 (2002), 881-891
submitted: 24/4/02, accepted: 15/10/02, appeared: 28/10/02 © J.UCS

Design for All as a Challenge for Hypermedia Engineering

Volker Mattick
(Chair of Programming Systems and Compiler Construction
FuLDIT Research Group
University of Dortmund, Germany
mattick@ls5.cs.uni-dortmund.de)

Abstract: Design for All is an important challenge for hypermedia engineering. We analyze this challenge and show that it is necessary to find a way of describing partially designed hypermedia documents that can then be transformed into different hypermedia applications according to user needs and call this concept "semi-documents". We sketch similarities and differences to existing formalisms and conclude that there are three areas in which functional languages can make a contribution: the development of an embedded special-purpose language for describing semi-documents, the building of generators which produce hypermedia applications from semi-documents, and the realization of support tools for the development of semi-documents.

Key Words: Hypermedia Engineering, Functional Programming, Design Techniques
Category: H.5.4, D.1.1, D.2.2

1 Introduction

Design for All is an important challenge for hypermedia engineering.

Firstly, a law has been passed in Germany a few months ago which obligates public institutions to ensure that all their newly created internet pages are barrier free, and that existing ones will be modified accordingly over the next years ("Bundesgleichstellungsgesetz für Behinderte"). While this is the first law of its kind in Europe, other countries such as Switzerland and France are preparing similar regulations. In the US, comparable anti-discrimination laws have been in existence for about six years. Discriminations, which abound in the non-virtual world, may be at least partially compensated with hypermedia, e.g. in the growing field of e-government.

Secondly, it is to be expected that an increasing number of elderly people will use the internet. While most of them are not handicapped, many have limited physical abilities in certain areas. This market segment also presents an enormous economic potential which is hardly being addressed at present.

Thirdly, an increasing number of people want to access the internet while away from optimal resources. Mobile devices often have very small screens, and are used under difficult conditions.

Hence, both social responsibility and economic considerations make it important to present hypermedia content such that it is accessible to all people under all conditions. This is the aim of *Design for All.*

The first two sections analyze this challenge. First, we summarize the meaning of Design for All for hypermedia, and its relationship to the more popular terms *barrier-free internet* and *web accessibility*. In addition, we give the reasons why we hope that Design for All has a much better chance to be realized in the "virtual world" than in the "real world" (Section 2). We then embed this paradigm in the hypermedia engineering design process (Section 3). Next, we discuss the possible uses of functional programming languages in this context (Section 4). In the following section, we conclude that we need a possibility to describe semi-designed documents, called *semi-documents* for short. We present ideas on how semi-documents can be described (Section 5.1), what a generator must do to produce hypermedia applications from a semi-document (Section 5.2) and how a development tool for semi-documents could be realized (Section 5.3). All these ideas stem from the area of functional and functional-logic languages.

2 Design for All

Design for All means the designing of products, services and systems such that they are flexible enough to be used directly, without assistive devices or modifications, by people with the widest range of abilities and circumstances [Trace Center (96)]. It is thus a design methodology that helps achieve the aim of *barrier-freeness*. In the area of hypermedia, barrier-freeness means accessibility of the produced applications. *Accessibility*, as defined by the World Wide Web Consortium (W3C), means that the content of a web page can be used by someone with a disability [W3C (99)].

In the "real world", a product is designed and produced once. You cannot save the blueprint of a bridge and build it every time someone want to cross the river, adapting it to her or his needs. A person in a wheelchair will need an elevator, whereas someone with an elevator phobia will ask for stairs. To reach the Design-for-All goals, it is often necessary to design a lot of redundant means of doing the same thing, according to the abilities of the user. Or you must find a solution which caters for several groups of users, a ramp in our example. Both options can be expensive and need an extremely foresighted design process, because once the product is finished, there is little chance to change it.

In the "virtual world" of hypermedia, it is possible to save a blueprint, that is, nothing but a specification, and produce customized applications. Customization means creating applications tailored to the abilities of the user. All information and all possibilities of navigation should be available for every user. The presentation and, at least in part, the structure should be adjusted to fit each individual situation. This is possible because hyperdocuments differ radically from traditional documents not only conceptually but also in their realization. Conceptually, traditional documents are organized in a linear fashion, whereas hyperdocuments are organized according to a non-linear link structure. Various models focus on this difference, and there are numerous tools to support the navigational design of hyperdocuments. The realization of a linear text is usually some kind of book or article which is designed once, printed out and not changeable afterwards. Hyperdocuments are typically realized as pieces of software and are therefore adaptable by nature.

The Web Accessibility Initiative (WAI) is a W3C activity which among other things seeks to ensure that the core technologies of the Web are accessible, including HTML, CSS, XML, SMIL, SVG and DOM. Moreover, WAI has compiled extensive accessibility guidelines. In principle, all these guidelines describe rules that help achieve barrier-free web pages with the HTML or XML markup languages in conjunction with stylesheets. In other words, they all address the problem at the source code level. This is a very "real-world" approach which has its advantages and disadvantages. On the positive side, this concept is easy for developers to understand and can be realized with little additional effort. However, there is the significant disadvantage that often a lot of different alternatives must be coded to achieve a satisfactory result. This can lead to large source code, presenting a problem e.g. when there is a very slow internet connection. Such problems rarely affect users on the grounds of disabilities, but they do present a problem when using mobile access devices. This means that the Design for All paradigm is not respected.

Another approach is the development of tools such as screen readers or the text-based browser Lynx in an attempt to transform the code adequately. However, this can be extremely difficult. Nearly all hypermedia applications are developed for a two-dimensional display. If they are developed in a markup language, the two-dimensionality is encoded in the application at a very early stage. Translating it for one-dimensional output devices would need linearization information. The author of a document usually has such knowledge, but it cannot be saved in the markup language. Markup languages, and Java or Flash even more so, require far more complete and more restrictive information than is desirable or necessary for the application as such.

We therefore propose to design a specification which is sufficiently concrete to generate an application, and sufficiently underdetermined to customize the output application for the widest range of users.

3 Hypermedia Engineering

Hypermedia applications are a special kind of software often mainly developed with rapid prototyping. This is frequently done with only very basic use of engineering methods, or totally ad hoc. The positive aspect of rapid prototyping is that customers can quickly get an impression of the look and feel of their application. The drawback is that this look and feel may only work on the specific customers' hardware and with their abilities. This is acceptable for a local hypermedia application, but usually hypermedia is designed for distribution over the World Wide Web and must be fit to be displayed on several different output media. This is even more important if barrier-freeness is an aim.

To realize more structured approaches, some years ago hypermedia engineering came into being as a particular field of software engineering. [Lowe, Hall (99)] single out its two major components: The *product* itself and the design *process*.

Hypertext research has gained extensive experience with more or less formal descriptions of *product models*. There are three basic kinds of product models (cf. [Lowe, Hall (99), p. 221 ff.]), the *programming language-based*, the *information-centered* and the *screen-based* models. The programming language-based approach, which uses any general-purpose programming language starting from scratch, was used in the past due to the lack of more sophisticated models, and is of little or no importance at present. For a long time, the information-centered model has dominated. The most popular product model for hyperdocuments, the "Dexter Hypertext Reference Model" [Halasz, Schwartz (90)], is information centered. Dexter or one of its modifications, e.g. [Grønbaek, Trigg (94)] or [van Ossenbruggen and Eliens (95)], describes the structure of a hyperdocument, divided into its logical structure, its linkage, and its style. A hyperdocument can import components from a "within-component layer" via an anchor mechanism and can contain a "presentation specification" stating how it is to be presented. Similar ideas are implemented in an object-oriented style in the so-called "Tower Model" [de Bra, Houben (92)], that adds a hierarchization according to which components can include other components. The markup languages, too, obviously have their roots in the information-centered paradigm, even though many designers use them in a screen-based way (cf. Section 2). *Screen-based* means that the focus is not on the logical structure of the document, enriched with some display attributes, but on the display of the document itself.

An overview of several *process models* can also be found in [Lowe, Hall (99)]. If we abstract from a concrete process model, we can distinguish two main phases, the *requirement engineering phase*, whose output is usually a requirement specification, and the *design phase* itself, whose output is usually a prototypical hypermedia application which is then iteratively improved. With such tools, a document is only described by its representation in a markup language, in other words, the source code of the application. They do not provide a special description formalism for the document, other than the graphical representation on the screen. The popularity of this approach might result from the benefits of rapid prototyping or of the wide-spread authoring tools, especially screen-based tools like Frontpage, Netscape Composer or Bluefish, which have no separation between the *description* and the *implementation* of a hyperdocument.

4 Functional Programming and Hypermedia

This section discusses the possible uses of functional programming languages in the context of Design for All in Hypermedia Engineering. Both markup languages and functional languages are usually declarative, such that they can easily be used together. Programming languages are better suited for structuring problems and building abstractions. The goal of working at the high level of structural markup, where documents are specified in terms of their logical features rather than of particular rendering procedures, is similar to the ideals of functional programming, where computations are specified in mathematical rather than machine-oriented terms. Documents described with a markup language can be seen as trees, and functional languages usually offer extensive facilities for representing and manipulating trees. Moreover, if a typed functional language is used, the type system can provide additional structure and integrity. In the last years, some interesting approaches have been presented which combine markup languages (mainly the Extensible Markup Language (XML) [W3C (00b)]) and functional languages. They follow two strategies, both based on the design of a library of combinators for the selection, generation and transformation of XML trees (cf. [Wallace, Runciman (99)]). A more detailed discussion of XML can be found in [Parsia (01)].

The first strategy consist in extending the functional language with adequate libraries and utilities. Hypermedia developers can use this extended language the same way as the original functional language. A prominent example is *HaXml* [Mertz (01)], which is a collection of utilities for the combined use of Haskell [Jones, Hughes (99)] and XML. Its basic features include a parser and a validator for XML, a separate error-correcting parser for HTML and pretty-printers for both. It contains a combinator library for generic XML document processing, including transformation, editing, and generation.

The second strategy consist in developing embedded domain-specific languages. Hypermedia developers who use such a special-purpose language may not even notice that they are using a functional language. An interesting example of this is the *Web Authoring System Haskell (WASH)* [Thiemann (01)]. It is a family of embedded domain-specific languages for programming HTML and XML applications. Each language is embedded in the functional language Haskell and is implemented as a combinator library. A similar idea underlies the modeling of basic HTML in the context of server-side web scripting in the functional-logic language Curry [Hanus (00)], [Hanus et al. (00)].

Our research group "Functional-Logic Development and Implementation Techniques" has implemented a Dexter-based hypertext reference model in the functional language ML, as part of a research project in the area of design automation. It is called the *HMD Model* [Mattick, Wirth (99)]. This prototype is not powerful enough, and the choice of ML leads to some language-inherent problems, but we believe that a revised implementation with Haskell or Curry will overcome these difficulties.

Currently we are experimenting with these different above mentioned strategies on a small but realistic application. The results will influence the strategies to choose for our further work.

5 Semi-Documents

To achieve the aims of Design for All, documents must be specified in a way that makes it possible to derive applications; but the specification must not be fully determined, in order to permit customization of the applications for the widest range of users (cf. Section 2). Typical hypermedia engineering strategies and tools coerce the designer to determine things at the "implementation level", because markup languages are used as the only description formalism (cf. Section 3).

Therefore, the design phase must be split into two phases. The first one is the design process of the semi-document. The output of this phase should not be an application but an executable specification. To achieve this, it is advisable to start with a maximal number of possibilities, which means that a lot of information is not given. In the process, this information must be interactively filled in until you reach a stage where enough details are given to produce automatically adapted hypermedia applications. So what we need is a new adequate description language, based on a hypermedia model in which things can be left variable for as long as possible. The second phase generates browsable applications from these specifications.

In principle, a document is made up of *media objects* such as blocks of text, pictures, sound files or animations. There normally exist some restrictions on the order in which these media objects should be presented, e.g. because certain pieces of information must be consumed before certain others in order to understand the document in any of its forms. We call the minimal necessary set of sequence constraints the *meaningful structure* of the document.

The specification mechanism we propose must make it possible to describe the meaningful structure of a document. We call such a specification a semi-designed document or *semi-document* for short. A semi-document describes a class of applications. A semi-document must contain a list of the basic media objects used, a meaningful structure, and a set of links. We assume that media objects are produced by media designers appropriately. That means that a picture is created with an audio description for blind users or a sound file also has a textual representation to give deaf people an idea of its content. The constraints are defined according to the requirement specification. The links are the result of a Navigational Design Phase.

These semi-documents can be stored on a server. Of course they are not browsable, at least not with current technology. When documents become semi-documents, browsers become constraint-solving generators, which dynamically produce documents from a semi-document according to a user profile and with the help of certain rules. Theoretically this is good approach, but in practice it will not work: we are not very confident that all producers of browsers will follow our theory and produce new tools which can generate applications from semi-documents.

Therefore, we propose another solution. A provider who wants to support barrier-freeness can develop semi-documents and store them locally, together with a generator. This generator can be used to produce all possible applications that comply with the chosen formal product model of hyperdocuments, respect the given constraints and contain all given media objects and links. If there are too many constraints, this class may be empty. If there are too few constraints, the class might be huge but obviously finite, as long as the list of media objects used is reasonably finite. Usually, the class will contain more than one application. So the next step is to choose applications that satisfy a given user profile. In principle, a user profile is a further set of constraints. It must be checked which of the generated implementations can actually be used. While the generation process should be automatic, in this phase interactivity with the user appears necessary (cf. Section 5.2). The chosen application can then be transferred to a web server and downloaded from there like any other hypermedia document. Variants that are needed frequently can be generated offline to speed up the delivery.

Obviously, semi-documents cannot be produced with **W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et editors. According to its name, the WYSIWYG philosophy means that you produce an artifact that every user sees in the same way as the author. This does not even work well for HTML or XML documents, because with the use of stylesheets they can appear in very different ways. So these tools are not suitable for creating semi-documents, whose appearance is even less strongly determined than that of XML documents. It is therefore necessary to develop hypermedia engineering methodologies and tools to support the design of semi-documents (cf. Section 5.3).

We do not want to conceal the fact that our approach will most probably not work for every kind of hypermedia application. It focuses on applications which present structured information, not on hypertext fiction, multimedia art or highly interactive network-based interfaces.

5.1 Describing Semi-Documents

The descriptions of semi-documents should be as human readable as possible, as abstract as possible and as concrete as necessary. As human readable as possible means either a good textual description with terms from the area of hypermedia rather than the programming-language world, or a graphical representation into which a possibly less readable textual format can be transformed and vice versa, or both. As abstract as possible means covering all invariant information of a document, including all media objects used, but avoiding to describe anything that is not strictly necessary. Finally, a formalism must be concrete enough to enable the semi-automatic and rule-based creation of valid documents that can be delivered. In short, we need a formalism to describe propositions over collections of documents. In an earlier project, located in the area of design automation, we have specified a product model for hypermedia, called the HMD Model [Mattick, Wirth (99)], that is described with algebraic specifications and with the help of Swinging Types [Padawitz (00b)]. Because of the affinities between algebraic specifications and functional languages, we have implemented a prototype in the functional language ML [Paulson (96)]. This yields an embedded domain-specific language for the domain described by our HMD Model. Because of its roots in design automation, the HMD Model contains representatives for documents at any stage of the design process. As semi-documents are not fully designed documents, we need to find the right stage in the design process to declare the representative a semi-document and save it for further processing by the generator. This is by no means a trivial task.

Of course there are similarities between the HMD model and the XML Schema definition language [W3C (01)]. Both cover the same domain, and both describe classes of executable hyperdocuments. A detailed examination of the differences and similarities is still an open issue. It is, however, clear that in order to use the XML Schema description language instead of our proprietary HMD Model, one would have to embed XML Schema in a (preferably functional) language in the way WASH handles XML or HaXml is enriched with libraries for XML.

5.2 Generating Hypermedia Applications from Semi-Documents

We need to evolve strategies and tools for transforming semi-documents into browsable documents which are valid w.r.t. a given semi-document.

In general, there are two means of ensuring that a document is valid w.r.t. a specification. Firstly, you can create a document description and then check the result, e.g. with a validator like the one included in HaXml or with some kind of model-checking approach. Secondly, you can build documents with a generator that can only produce correct documents, following the principle of "correctness by construction".

To build such a generator, the description formalism for the semi-documents must be embedded into a system of editing functions. It must be extended with strategies that help build a document description from the semi-document description in a step-by-step manner. In these strategies, constraint solving plays a major role, and this is a professed domain of functional-logic languages. In the above-mentioned design-automation project [Mattick, Wirth (99)], basics of this idea have also been formulated and prototypically realized with ML. Of course, this is also possible with a domain-specific language based on XML Schema instead of the HMD Model.

5.3 Developing Semi-Documents

"The need for a 'universal accessibility' engineering tool" has already been pointed out by [Lindenberg, Neerincx (99)]. With the concept of "semi-documents", this need becomes a concrete demand for tools and techniques to design a semi-document from a requirement specification and a navigational model. Because semi-documents cannot be graphically represented with a WYSIWYG strategy, and purely text-based development is not satisfactory, we need new approaches.

UML-like notations appear promising, and can be specialized for the hypermedia domain (e.g. [Baumeister et al. (99)]). This does not come as a great surprise, because the most important data structures of UML-like notations are trees and forests. Roughly speaking, a semi-document also can be described as a tree, together with constraints and rules. A collection of semi-documents can be connected by hyperlinks into a graph in which every node contains a semi-document. So a support tool for the development of semi-documents must essentially be a tool for developing certain trees and forests, possibly at a graphical level. UML can be combined with an algebraic representation [Padawitz (00a)]. It is therefore likely that it can be combined with a functional language as well. Functional languages are known to be very powerful tools for tree manipulation. Moreover, state-of-the-art functional languages are enriched with graphical possibilities and interactivity. An example is the object-oriented extension of Haskell named O'Haskell [Nordlander (n.d.)].

Apart from UML, a further possibility would be concept maps [XTM (01)], an ISO International Standard for device- and implementation-independent recording of information about any subject matter.

We should not forget that web accessibility and Design for All mean that not only the applications must be accessible, but also the tools needed to produce them. Therefore, the W3 Consortium has developed an "Authoring Tool Accessibility Guideline" [W3C (00a)] which presents rules for building an authoring tool. How can UML-like notations or concept maps be made accessible to blind people? We don't know. But we believe that with a well-designed description language for semi-documents, other facilities can be designed, which will make it equally convenient for visually impaired persons to create these artifacts. So the goal "as human readable as possible" is really needed, not a nice add-on.

6 Conclusion

Design for All is an important challenge for hypermedia engineering. An analysis of this challenge has shown that it can be met by describing partially designed hypermedia documents that can then be transformed into different hypermedia applications according to user needs. We have called this concept "semi-documents" and sketched its similarities and differences to existing formalisms such as the XML Schema description language. We can conclude that there are three areas in which functional or functional-logic languages can make a contribution: the development of an embedded special-purpose language for describing semi-documents, the building of generators which produce hypermedia applications from semi-documents, and the realization of support tools for the development of semi-documents.

There already exist some promising approaches at the intersection of functional programming and hypermedia development. In industry, probably under the influence of the new laws, there is a notable intersection between hypermedia development and basic principles of Design for All, which however has received little attention in research so far. We do not know of any project at the intersection of all three paradigms (functional programming, hypermedia development and Design for All). We have only had this idea very recently. We have since started work on a few small case studies, but we do not have a presentable prototype yet.

Apart from all technical considerations: Design for All is a challenge for all, not only for Hypermedia Engineering. Computer science and hypermedia research can develop tools and techniques. The goals of Design for All constitute a multidisciplinary task, in which everybody who wants to overcome the barriers of today's hypermedia reality needs to make a contribution.

References

[Baumeister et al. (99)] Hubert Baumeister, Nora Koch, Luis Mandel. Towards a UML Extension for Hypermedia Design. Proceedings of UML '99, Springer LNCS 1723 (1999), 614-629. `www.pst.informatik.uni-muenchen.de/projekte/forsoft/pubs/uml99.ps.gz`.

[de Bra, Houben (92)] Paul de Bra, Geert-Jan Houben. An Extensible Data Model for Hyperdocuments. Proceedings of the ACM Conference on Hypertext'92 (1992), 222-231.

[Grønbaek, Trigg (94)] K. Grønbaek and R. H. Trigg. Design Issues for a Dexter-Based Hypermedia System. Communicatios of the ACM, 37(2)(1994), 40-49.

[Halasz, Schwartz (90)] F. Halasz, F. Schwartz. The Dexter Hypertext Reference Model. Proceedings of the Hypertext Standardization Workshop, National Institute of Technology (NIST) (1990), 95-133.

[Hanus (00)] Michael Hanus. Server Side Web Scripting in Curry. Workshop on (Constraint) Logic Programming an Software Engineering, LPSE2000, London, 2000.

[Hanus et al. (00)] Michael Hanus et al. PAKCS: The Portland Aachen Kiel Curry System, 2000. `www.informatik.uni-kiel.de/~pakcs`.

[Jones, Hughes (99)] Simon Peyton Jones and John Hughes (eds). Haskell 98: A Non-strict, Purely Functional Language. Microsoft Research, Cambridge, Chalmers University of Technology, February 1999. www.haskell.org/definition/haskell98-report.pdf.

[Jones, Peterson (99)] Mark P. Jones and John C. Peterson. The Hugs user manual. http://cvs.haskell.org/Hugs/downloads/hugs.ps.gz.

[Lindenberg, Neerincx (99)] J. Lindenberg and M.A. Neerincx. The need for a 'universal accessibility' engineering tool. Interact '99, 1999.

[Lowe, Hall (99)] David Lowe and Wendy Hall. Hypermedia & the Web. An engineering approach. Wiley, 1999.

[Mattick, Wirth (99)] Volker Mattick and Claus-Peter Wirth. An Algebraic Dexter-Based Hypertext Reference Model. Technical report, FB Informatik, Universität Dortmund, Dec 1999. ls5.cs.uni-dortmund.de/~mattick/pub/gr719.ps.gz.

[Mertz (01)] David Mertz. The HaXml functional programming model for XML. IBM DeveloperWorks. http://www-106.ibm.com/developerworks/xml/library/x-matters14.html.

[Nordlander (n.d.)] Johan Nordlander. A Survey of O'Haskell. www.cs.chalmers.se/~nordland/ohaskell/survey.html.

[Padawitz (00a)] Peter Padawitz. Swinging UML: How to Make Class Diagrams and State Machines Amenable to Constraint Solving and Proving. Proc. UML 2000, Springer LNCS 1939 (2000), 162-177.

[Padawitz (00b)] Peter Padawitz. Swinging Types = Functions + Relations + Transition Systems. Theoretical Computer Science 243 (2000), 93-165.

[Parsia (01)] Bijan Parsia. Functional Programming and XML. xml.org, O'Reilly, 2001. http://www.xml.com/pub/a/2001/02/14/functional.html.

[Paulson (96)] Larry C. Paulson. ML for the Working Programmer (2nd edition). Cambridge University Press, 1996.

[Thiemann (01)] Michael Thiemann. A Typed Representation for HTML and XML Documents in Haskell. Under consideration for publication in J. Functional Programming. http://www.informatik.uni-freiburg.de/~thiemann/papers/modeling.ps.gz.

[Trace Center (96)] Trace Centre. Universal Design: What it is and What it isn't. Trace Centre, University of Wisconsin, USA, 1996.

[Wallace, Runciman (99)] Malcom Wallace and Colin Runciman. Haskell and XML: Generic Combinators or Type-Based Translation? International Conference on Functional Programming, Paris, 1999.

[van Ossenbruggen and Eliens (95)] J. van Ossenbruggen and A. Eliens. The Dexter Hypertext Reference Model in Object-Z. Unpublished Paper, Vrije Universiteit Amsterdam. www.cs.vu.nl/~dejavu/papers/dexter-full.ps.gz.

[W3C (99)] World Wide Web Consortium. Web Content Accessibility Guidelines 1.0. W3C Recommendation 5-May-1999. www.w3.org/TR/1999/WAI-WEBCONTENT-19990505.

[W3C (00a)] World Wide Web Consortium. Techniques for Authoring Tool Accessibility. W3C Note 4-May-2000. www.w3.org/TR/2000/NOTE-ATAG10-TECHS-20000504.

[W3C (00b)] World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation 6-October-2000. www.w3.org/TR/2000/REC-xml-20001006.

[W3C (01)] World Wide Web Consortium. XML Schema Part 0: Primer. W3C Recommendation 2-May-2001. www.w3.org/TR/2001/REC-xmlschema-0-20010502.

[XTM (01)] TopicMaps.Org Authoring Group. XML Topic Maps (XTM) 1.0. TopicMaps.Org Specification, 2001. www.topicmaps.org/xtm/1.0/xtm1-20010806.html.