Building Applications with Domain-Specific Markup Languages: A Systematic Approach to the Development of XML-Based Software[1]

José L. Sierra, Alfredo Fernández-Valmayor, Baltasar Fernández-Manjón, and Antonio Navarro

Dpto. Sistemas Informáticos y Programación. Fac. Informática. Universidad Complutense. C/ Juan del Rosal nº 8. 28040. Madrid. Spain
`{jlsierra,alfredo,balta,anavarro}@sip.ucm.es`

Abstract. This paper presents ADDS, a systematic approach to sofware development using Domain-Specific Languages (DSLs) and markup technologies. XML is used as a common descriptive framework for DSLs formulation, obtaining Domain Specific Markup Languages (DSMLs). According to ADDS, the construction of applications in a domain starts with the provision of suitable DSMLs. Then, the applications in such a domain are described by means of sets of structured documents conforming these DSMLs. Finally, the application is produced by processing this documentation according to an operationalization model called OADDS. Hence ADDS provides a systematic approach to software development based on the processing of XML documentation that can be used in a great variety of domains.

1   Introduction

XML [26] has acquired a great relevance in the construction of web applications and in many other areas of software development. XML inherits from its predecessor SGML [13] a linguistic approach in the development of applications. Indeed, developing a SGML/XML application is equivalent to using SGML/XML in devising a special-purpose markup language. Previously, this linguistic approach was successfully applied to the electronic publishing domain but, with the advent of new SGML applications and, specially XML, this initial domain was quickly broadened. Now, XML is usually considered as a standard framework for information interchange between heterogeneous systems despite its origins as a document markup (meta)language. Neverthless, the new application domains should not change the initial linguistic conception: in order to develop an XML application the focus must be put on devising a markup language for describing the informational structure of the application domain. Once this language is available, one (or several) processor(s) must be provided, depending on the task to be solved using the marked up documents.

This paper presents our approach to software development called ADDS (Approach to Document-based Development of Software). This approach is the outcome of our previous experience using markup technologies in the educational domain [6][7], in the prototyping of model-driven hypermedia applications [15], and in the development of component-based software [19]. ADDS put the stress on the linguistic potentiality of XML, instead on its data oriented features. ADDS can actually be considered as a specific case of the paradigm of software construction based on Domain-Specific Languages (DSLs) [22] that uses XML as a common descriptive framework for DSLs formulation. Because ADDS conceives DSLs as XML applications, our approach provides new insight into the systematic development of these kinds of applications.

The structure of the paper is as follows. Section 2 gives a general overview of the ADDS approach. Section 3 describes the operationalization model of ADDS (i.e how executable applications are produced from documents describing them). Finally, section 4 describes related work and section 5 outlines the conclusions and future work. To illustrate the different aspects described in the paper, a case study application domain is used: the Subway Networks Route Search (SNRS) domain. Each SNRS application allows the search for paths from origin to destination stations in the subway network of a city.

2   The ADDS Approach

The ADDS approach is outlined in Fig.1. This approach comprises the following three activities: (i) the *provision of DSMLs*, oriented to provide the Domain-Specific Markup Languages (DSMLs), (ii) the *authorship of documents*, oriented to produce the documentation describing the application, and (iii) the *operationalization* activity, oriented to produce the application from the documentation. The following subsections detail these activities.
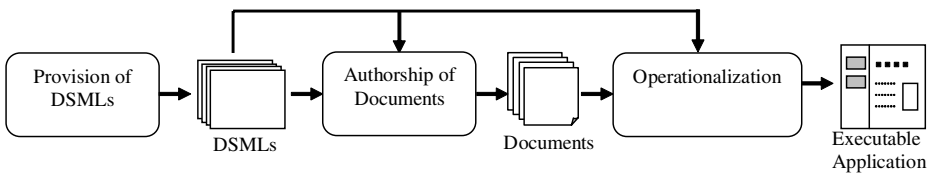

Fig. 1. Main activities and products involved in ADDS.

2.1   Provision of DSMLs

The goal of this activity is to produce DSMLs that will be used to markup the documents that describe the application and the data needed by the application in a given domain. In ADDS, each DSML comes with a set of software components which represent the primitive operations in the domain and that will be used for assembling

processors for documents conforming the DSML. There might be different components used to validate, or to edit documents, to perform domain-dependent tasks, etc.

Fig. 2 gives a top-level categorization of DSML types. There are (i) *problem* DSMLs, languages for marking up the relevant data and information about the domain of the problem to be solved by the application, and (ii) *application* DSMLs, languages for marking up documents describing high-level aspects of the application. Documents conforming problem DSMLs are called *problem domain* documents, while those conforming application DSMLs are called *application* documents.

Problem DSMLs can integrate two different sublanguages: (i) a *use-independent problem* DSML for the description of information independent of any particular use imposed by the application, and (ii) a *use-enabling problem* DSML for giving additional information not describable with the use-independent problem DSML. Moreover, to lower the cost of developing new DSMLs, more complex application DSMLs can be obtained by combining simpler ones.
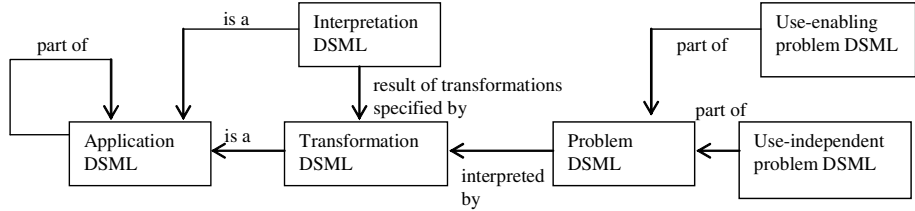

Fig. 2. An initial ADDS linguistic framework.

Table 1. DSMLs used in the SNRS domain.

| Problem DSMLs	
Use-independent problem DSMLs	– Subway Network Markup Language (SNML)
Use enabling problem DSMLs	– Subway Network Geometry and Styling Markup Language (SNGSML)
Application DSMLs	
Transformation DSMLs	– XSLT
Interpretation DSMLs	– Weighted Directed Graph Markup Language (WDGML)
	– Simple Diagram Description Markup Language (SDDML)
	– Simple Terminological Mapping Language (STML)
Other Application DSMLs	– Subway Network Route Searching Markup Language (SNRSML)

By maximizing the independence between problem and application DSMLs, we can improve the reusability of the DSMLs and of the documentation. Indeed, in some cases, application documents can be used with different problem domain documents, and problem domain documents can be used in different applications. To facilitate the relative independence between both types of languages, the application DSML must abstract the problem domain. The problem of coupling and decoupling these more specific and abstract languages used to describe the domain is solved in ADDS using *transformation* and *interpretation* DSMLs. *Transformation* DSMLs are used to specify document transformations, which, by taking problem documents as input, produce *interpretation documents* as output. Therefore, transformation DSMLs are

the link between application DSMLs and problem DSMLs. Furthermore, interpretation documents are compliant with *interpretation* DSMLs. These DSMLs are not directly used in the description of the application. Instead, they are used as the target languages for the transformation DSMLs

Table. 1 enumerates the different DSMLs used in the SNRS domain. The problem DSMLs include a use-independent problem DSML, SNML, for describing the structure and the schedules of subway networks, and a use enabling language, SNGSML, for the description of the geometry (eg. coordinates of the stations) and other stylistic aspects (eg. line colors) of subway networks. The application DSMLs include standard XSLT [26] as transformation language, and SNRSML for describing the variabilities of the application. The following interpretation DSMLs are also included: (i) WDGML for representing the weighted graph used in the search, (ii) SDDML for giving a visual representation of the subway network, and (iii) STML, used to relate the names of the stations in the SDDML diagram with the names of the nodes in the WDGML graph. These languages will be the target languages for the transformations.

2.2   Authorship of Documents

Domain experts can provide descriptions of applications as collections of marked up documents using the appropriate DSMLs.   This is done during the *authorship of documents* activity.
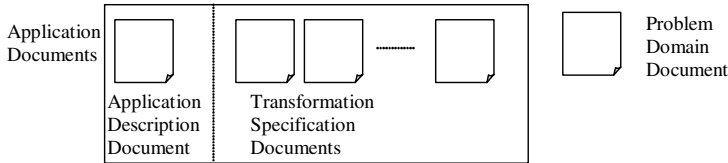
Fig. 3. Documentation associated with a typical application description in ADDS.
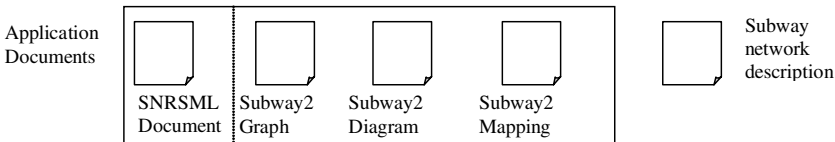
Fig. 4. Documentation associated with an SNRS application.

Fig. 3 sketches the documentation associated with a typical application description in ADDS. There is only one problem domain document. There are also several application documents: (i) an *application description* document describing the variabilities of the application, some of which are given as transformations of the problem domain information, and (ii) a set of *transformation specification* documents describing how to carry out these transformations.

Fig. 4 sketches the documentation associated with an application in the SNRS domain. The problem domain document includes a SNML description of the subway network. This also includes a SNGSML description of the geometry and style of the network. The application description document conforms to SNRSML. Finally, there are three XSLT transformations: (i) a first one (`Subway2Graph`) for generating a WDGML graph from the subway network description document, (ii) a second (`Subway2Diagram`) for generating a SDDML diagram, and (iii) a third (`Subway2Mapping`) for generating a STML mapping.

2.3   Operationalization

The operationalization activity produces an application from the marked up documents that describe it. This activity is detailed in the next section.

3   Operationalization in ADDS

The operationalization activity follows an operationalization model called OADDS (Operationalization in ADDS). It is based on the well-known techniques of syntax-directed translation [1] and attribute grammars [11][16] used in the construction of language processors. In addition, the model provides for *semantic modularity* [10]. Accordingly, processors can be built using reusable language-processor components. Semantic modularity is important in ADDS, because complex DSMLs can be obtained by combining simpler DSMLs. In this way, processors for the resulting markup document describing the whole application should arise as an appropriate combination of processors for the simpler subdocuments without changing them. The next sections provide the details.

3.1   OADDS Applications

The structure of an OADDS application is sketched in Fig. 5. According to this scketch, the application is made up by a *document tree*, a *processing graph* added to this document tree, and an *interface*.

The *document tree* is a *grove*-like representation [17] of the *top-level* document of the application. It is made of a set of objects, called *nodes*. Each node has an assigned set of *properties*. This tree includes properties for representing the markup structure (*primitive* properties), but also properties for representing other information (*semantic* properties). Each property is represented as an object that has an associated   *value*, and an associated set of *observers*. The value of a property can be either a list of nodes or an object of another type (including a node from a different document tree). Each observer is a *processing object* which can be notified when the property value changes.

The *processing graph* is a graph with properties and processing objects as nodes. The only arcs allowed are (i) from properties to processing objects, indicating that the process carried out by the processing object depends on the property value, and (ii)

from processing objects to properties, indicating that the source processing object updates the target property. Actually, the processing graph is added to the document tree registering processing objects as observers. In turn, each processing object can be asked for the properties that it updates. In terms of attribute grammars, this graph can be understood as a *dynamic* version of the *dependency graph* associated with an attributed parse tree, together with the *semantic functions* used to compute the values of the attributes.


Fig. 5. Structure of an OADDS application.

The *interface* mediates between the application and its environment. Usually, this environment will be populated by the *users* of the application (either people or other programs). In this way, the concept of *interface* in OADDS is a very abstract one. For instance, in the case of a batch application it could be a black-box, in the case of an interactive application (such as those used in the SNRS domain), it could be a graphical user interface. It could even be a SOAP [26] mediator for applications conceived as web services. In addition, the application description can also describe certain aspects of the interface. Such aspects will usually be eminently pragmatic (eg. in a GUI, label names, colors, etc.).

This structure induces a natural execution model based on the *propagation* of the properties values. Hence, computation of attribute values in the attribute grammar context has a counterpart on the behaviour induced by the network of observables/observers. Indeed, the propagation of a property value means that its observers should be notified about the availability of this value. In this way, the execution starts with the propagation of the values of the primitive properties. Next, the interface takes control. This interface can update other semantic properties in the tree in order to finish the propagation process. Finally, execution is guided by user interaction. When the user interacts with the interface, the appropriate semantic properties are updated and re-propagated. As a consequence, other semantic properties are established and their values are used to update the state of the interface.

3.2   The OADDS Model

The activities and products involved in the OADDS model are shown in Fig.6.

OADDS uses two types of *software* componentes to produce applications from structured documents: (i) (tree) *analyzers* that validate and set up the document tree to ensure that it can be properly operationalized, and (ii) *operationalizers* that enlarge the document tree by adding the processing graph. These components are provided during the *provision of software components* activity. While analyzers abstract the productions of a context-free grammar, operationalizers provide an implementation of the semantic equations associated with each one of these productions.
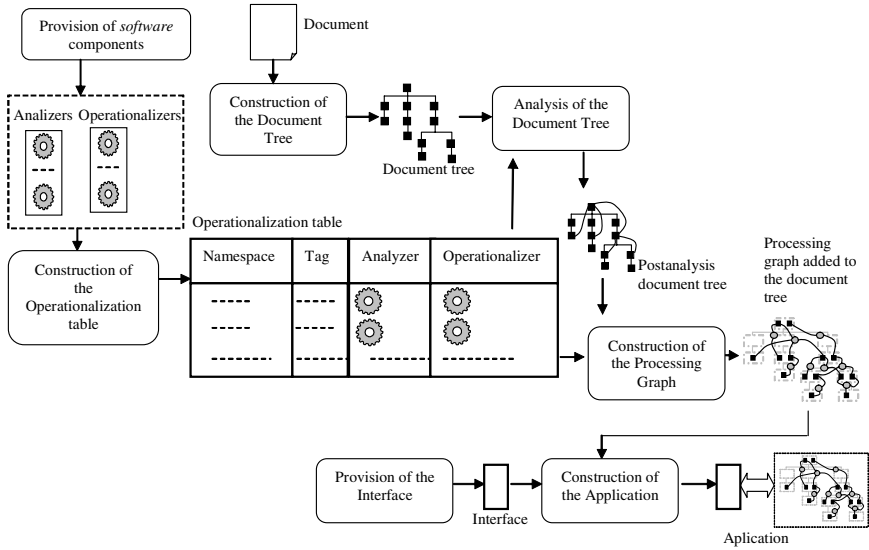
Fig. 6. Products and activities in OADDS.

The operationalization of a DSML is performed by assigning an analyzer and an operationalizer to each tag. The resulting structure is called an *operationalization table*, and it is provided during the *construction of the operationalization table* activity. Therefore, this table can be thought as an abstraction of an attribute grammar/syntax-directed translation schema. Because DSMLs can be produced by combining simpler DSMLs, the operationalization table uses XML namespaces [26] to avoid name conflicts. In addition, both the analyzer and the operationalizer associated with a given tag can be obtained by *composing* different analyzers/operationalizers. This is the main OADDS mechanism for achieving semantic modularity.

The operationalization of a particular document begins with the construction of its document tree, during the *construction of the document tree* activity. This document is subsequently analyzed during the *analysis of the document tree* activity. This activity is carried out using the analyzers referred to in the operationalization table. The tags associated with element nodes in the document tree are used to index the appropriate analyzers in the table, which are applied to the corresponding nodes. To facilitate the actuation of subsequent components, analyzers can add new properties (*postanalysis* properties) to the document tree. The resulting tree is called a *postanalysis* document tree.

The addition of the processing graph to the (postanalysis) document tree is carried out during the *construction of the processing graph* activity. Here element tags are used to index operationalizers in the operationalization table. Then, the recovered operationalizers are applied to the corresponding nodes. When applied to a node, an operationalizer adds part of the processing graph by registering processing objects as observers of node properties. Finally, the application is produced by putting together

the interface (obtained during the *provision of the interface* activity) with the document tree and the added processing graph. This is performed during the *construction of the application* activity.

Currently, we have developed an experimental Java object-oriented framework for supporting OADDS. The framework provides facilities to connect either with DOM [26] or SAX [3] compliant software for the construction of the document tree.

4   Related Work

A pioneering work in the use of SGML/XML for the description of DSMLs is [8]. In [25], some relations between markup languages and the DSL approach are highlighted. Although these works recognize the value of markup metalanguages as a vehicle to define DSLs, they largerly use SGML/XML as abstract syntax representation formalisms rather than as descriptive markup (meta)languages.

The key idea behind Jargons [14] is similar to ADDS. In Jargons, DSMLs are directly formulated, and even operationalized (using a script language), by domain experts. While we agree with this *author centered* conception of DSL development, we think that to assign language design and operationalization responsabilities to domain experts is not realistic. In ADDS, operationalization is separated from DSL provision. Therefore, operationalization can be carried out by software development experts while DSMLs can evolve according to the markup needs of domain experts. A third category of experts in ADDS can mediate between domain and software development experts, taking the responsability for maintaining the DSMLs. Semantic modularity (not comtemplated in Jargons) is essential in order to accomodate operationalization to this evolution of DSMLs.

DSLX [13] is another example of a framework to operationalize DSLMs. Although this framework enables DSL composability, it does not provide any special mechanism to ensure semantic modularity.

A well-known approach to semantic modularity in the development of DSLs is that based on *monads* and *monad transformers* [10] in the functional programming community. Because control in OADDS is imposed by the processing model itself, modularity is oriented to adaptation of the information flow between the document tree nodes, instead of control adaptations. This solution is weaker, but also more usable than monadic approaches. Another well-known approach, this time in the object-oriented arena, is based on the use of *mix-ins* [4]. Composition of analyzers/operationalizers in OADDS can be resorted into a single mix-in composition, where the execution of mix-ins do not interfere.

There are several works describing the application of syntax directed translation techniques and attribute grammars to markup languages [5][12][18].Actually, OADDS can be considered an abstraction of these approaches, where the grammatical specification is reduced to the operationalization table. Because of this, all these approaches are feasible in OADDS. In particular, the facilities provided by higher-order attribute grammars [24], where the value of an attribute can be another attributed tree, can be easily achieved. This enables OADDS to deal with interpretation documents. In addition, some usual techniques used to obtain semantic modularity in the context of attribute grammars can also be easily achieved [23].

Existing technologies for building XML-based software fit in well with our approach. Therefore, the *construction of the document tree* OADDS activity can be built on top of basic document processor technologies, such as DOM or SAX. On the other hand, data-binding proposals [2] (i.e. compilation of document grammars into object oriented representations of documents conforming these grammars) can be understood as a particular case of OADDS. In addition, Knuth's principle, which states that attribute grammars are as powerful as any other semantic specification formalism [11], can be applied here, because, at the limit, processing objects can be applied to entire document (sub)trees. This facilitates the integration of pre-existing processors (eg. standard XSLT transformers).

The work described here relies on our previous work on the DTC (structured Documents, document Transformations and software Components) approach [19][21][20]. ADDS abstracts the key idea under DTC: the use of DSMLs. In addition, the treatment of document transformations exposed here (closer to the idea of *overmarkup* exposed in [15]), where the application of transformations is a part of the application description, is more flexible than in DTC. In DTC transformations were applied for the content (problem domain) documents to obtain a pre-established set of document entities integrated into the application documents.

5   Conclusions and Future Work

ADDS is a systematic approach to sofware development founded on DSLs and markup technologies. It is based on a comprehensive linguistic view of the application development that deals with the description of the applications and the problem domains by means of marked up documents, and also with the operationalization of these descriptions. Therefore, by using the markup documents (produced according to suitable DSMLs) and a related group of software components, the executable application is obtained.

Recently we have realized that the use of descriptive markup, together with the modular nature of OADDS, enables the conception of ADDS as a pragmatic, authorship-driven, particular case of the DSL approach. Consequently, we are refining ADDS in that direction. In addition, we are working on the recursive application of ADDS to ADDS itself, either for the production of specialized editors for domain experts, and for the supporting of *meta* DSMLs to give higher-level descriptions of OADDS processors. Next step in the project is to give a precise characterization of the application domains where the ADDS particular use of the DSL approach could be competitive.

References

1.   Aho, A. Sethi, R. Ullman, J. D. Compilers: Principles, Techniques and Tools. Adisson-Wesley. 1986
2.   Birbeck,M et al. XML Data Binding. Professional XML 2nd Edition. WROX Press. 2001.
3.   Brownell, D. SAX2. O'Relly. 2002

4. Duggan, D. A Mixin-Based Semantic-Based Approach to Reusing Domain-Specific Programming Languages. 14th European Conference on Object-Oriented Programming ECOOP'2000. Cannes. France. June12–16 2000

5. Feng, A.Wakayama, A. SIMON: A Grammar-based Transformation System for Structured Documents. Electronic Publishing. 6(4). 1993

6. Fernández Manjón, B. Fernández-Valmayor, A. Improving World Wide Web educational uses promoting hypertext and standard general markup language content-based features. Education and Information Technologies, vol 2, no 3, pp. 193–206. 1997.

7. Fernández-Valmayor, A.; López Alonso, C. Sèrè A. Fernández-Manjón,B. Integrating an Interactive Learning Paradigm for Foreign Language Text Comprehension into a Flexible Hypermedia system. *IFIP WG3.2-WG3.6* Conference Building University Electronic Educational Environments. University of California Irvine, California, USA August. 4–6 1999

8. Fuchs, M. Domain Specific Languages for *ad hoc* Distributed Applications. First Conference on Domain Specific Languages. USENIX. Sta. Barbara. CA. October 17–17. 1997

9. Goldfard, C. F. The SGML Handbook. Oxford University Press. 1990

10. Hudak,P. Domain-Specific Languages. Handbook of Programming Languages V. III: Little Languages. And Tools. Macmillan Tech. Publishing. 1998

11. Knuth;D.E. Semantics of Context-free Languages. Math. Systems Theory. 2:127–145. 1968

12. Kuikka, E.Pentonnen, M. Transformation of Structured Documents with the Use of Grammars.Electronic Publishing. 6(4). 1993

13. Morrow,P. Alexander,M. Domain Specific Languages – Tools for Better Programming. PCAI Magazine. Vol 13. Issue 1. Jan/Feb 1999

14. Nakatani,L.H. Ardis,M.A. Olsen,R.G. Pontrelli,P.M. Jargons for Domain Engineering. Second Conference for Domain Specific Languages. USENIX. Austin. Texas. October 3–6. 1999

15. Navarro, A., Fernández-Manjón, B., Fernández-Valmayor, A., Sierra, J.L. Formal-Driven Conceptualization and Prototyping of Hypermedia Applications. Fundamental Approaches to Software Engineering FASE 2002. ETAPS 2002. Grenoble. France. April 8–12. 2002

16. Paakki,J. Attribute Grammar Paradigms - A High-Level Methodology in Language Implementation. ACM Computing Surveys 27(2): 196–255. 1995

17. Prescod,P. Addressing the Enterprise: Why the Web needs Groves. ISOGEN White Paper. 1999

18. Psaila, G.Crespi-Reghizzi, S. Adding Semantics to XML. Second Workshop on Attribute Grammars and their Applications. WAGA'99. Amsterdam. The Netherlands. March 26. 1999

19. Sierra, J. L. Fernández-Manjón, B. Fernández-Valmayor, A. Navarro, A. Integration of Markup Languages, Document Transformations and Software Components in the Development of Applications: the DTC Approach. International Conference on Software ICS 2000. 16th IFIP World Computer Congress. Beijing - China. August 21–25. 2000

20. Sierra, J. L. Fernández-Manjón, B. Fernández-Valmayor, A. Navarro, A. An Extensible and Modular Processing Model for Document Trees. Extreme Markup Languages 2002. Montreal. Canada. August 4–8. 2002.

21. Sierra, J. L. Fernández-Valmayor, A. Fernández-Manjón, B. Navarro, A. Operationalizing Application Descriptions with DTC: Building Applications with Generalized Markup Technologies. 13th International Conference on Software Engineering & Knowledge Engineering SEKE'01. Buenos Aires. Argentina. June 13–15. 2001.

22. Van Deursen, A. Klint, P.Visser, J. Domain-Specific Languages: An Annotated Bibliography. ACM SIGPLAN Notices. 35(6). 2000.
23. Van Wyk,E. de Moor, O. Backhouse, K. Kwiatkowski,P. Forwarding in Attribute Grammars for Modular Language Design. Compiler Construction CC 2002. ETAPS 2002. Grenoble France. April 8–12. 2002
24. Vogt, H, H. Swierstra, S, D. Kuiper, M, F. Higher-Order Attribute Grammars. Proceedings of the ACM SIGPLAN'89 Conference on Programming Language Design and Implementation. 1989
25. Wadler,P. The next 700 markup languages. Invited Talk of the Second USENIX Conference on Domain Specific Languages. USENIX. Austin. Texas. 1999
26. www.w3.org/TR