Document-Oriented Software Construction based on Domain-Specific Markup Languages

José Luis Sierra, Baltasar Fernández-Manjón, Alfredo Fernández-Valmayor, Antonio Navarro

Dpto. Sistemas Informáticos y Programación
Fac. Informática. Universidad Complutense de Madrid.
28040,Madrid (Spain)
{jlsierra,balta,alfredo,anavarro}@sip.ucm.es

Abstract

In this paper we present ADDS (Approach to Document-oriented Development of Software), our solution to software construction based on Domain-Specific Languages (DSLs). DSLs in ADDS are formulated as descriptive Domain-Specific Markup Languages (DSMLs) that are used for marking up the documents that describe the relevant aspects of the applications (e.g. data and some aspects of the behavior). Final running applications are obtained by the processing of these documents with suitable processors. ADDS promotes the incremental development of DSMLs and their processors, so they can evolve according to the authoring needs of the different participants in the development process (domain experts and developers). The incremental nature of ADDS is eased by its document orientation. Thus ADDS palliates the high costs of formulation, operationalization and maintenance of DSLs exhibited by other approaches

Keywords: *Development Approach, Domain-Specific Markup Languages, Maintenance, Evolution, XML*

1. Introduction

The benefit of using *Domain-Specific Languages* (DSLs) for the development of applications in a given domain has been largely recognized [6][17][18]. According to [18], a DSL is a *programming language or executable specification language that offers, through appropriated notations and abstractions ,expressive power focused on, and usually restricted to, a particular problem domain*. Hence, DSLs improve productivity because they can be directly used by domain experts. However, the high costs of the formulation, operationalization (i.e the development of a suitable interpreter/compiler for the language) and maintenance of DSLs are identified as shortcomings of this approach. A DSL formulation implies an in-depth analysis of the application domain, and strong usability considerations regarding the language's end users (i.e. domain experts). Furthermore, the complexity of the DSL's operationalization process must be addressed. Lastly, during the development of applications, new aspects, not initially covered by the DSL, could be discovered. Therefore the costs of the DSL maintenance must be also considered.

This paper describes ADDS (*Approach to Document-oriented Development of Software*), our approach to the development of software applications based on DSLs. ADDS promotes the description of relevant aspects of an application by means of documents. These documents are marked up with appropriate descriptive *Domain-Specific Markup Languages* (DSMLs), and the final applications are built and executed by processing these documents with suitable processors for these DSMLs. ADDS promotes an incremental formulation and operationalization of the DSMLs to solve the previously mentioned shortcomings. Thus, our approach is driven by the markup needs discovered during the development of the applications. The incremental formulation of DSMLs in ADDS is enabled by the use of standard markup metalanguages (e.g. SGML [5] or XML [19]) and their associated declarative grammar-based formalisms (e.g. SGML/XML DTDs or other schema languages [8]). Likewise, the incremental development of their processors is eased by the adoption of modular language processors techniques [3][6][7].

ADDS has been formulated and refined for several years [11][14][15][16]. The main contribution of this paper is the description of ADDS using a two-level approach. In addition, we give a clearer distinction between the different perspectives of ADDS. Finally, we introduce a new technique for the incremental

provision of DSMLs, and formulate a new simpler and practical operationalization model.

The paper is organized as follows. Section 2 gives a general technology-independent description of ADDS. Section 3 outlines a specific implementation of the approach. Finally, section 4 presents some conclusions and lines of future work. We will use the domain of the applications for *route searching in subway networks* as a case study throughout the paper to illustrate the different aspects of the approach.

2. The ADDS Approach

There are some software development areas where not only the information processed by applications, but also a substantial part of their behaviors, are usually described by documents with well established structures. This claim is based on our previous experiences in the development of content – intensive educational and hypermedia applications [4][9] and knowledge – based systems [13]. In all these domains we have successfully used documents marked with DSMLs to improve the development and maintenance of applications. The ADDS approach systematizes and generalizes these experiences, laying out the foundations for a *document – oriented paradigm* for application construction. This section describes the approach in a technological- and implementation-independent way. This description is subsequently refined by choosing specific technologies to obtain different ADDS *implementations*, such as that described in the next section. Subsection 2.1 introduces the activities and products involved in ADDS. Subsection 2.2 describes the sequencing of these activities. Lastly, subsection 2.3 presents the main actors in the development of applications according to ADDS.

2.1. Activities and Products

Fig. 1 introduces the activities and products that comprise the ADDS approach. This subsection discusses all these aspects.

The main aim of the *DSML provision* activity is to obtain the *application DSML* that will be used for marking up the documents that describe the application. For instance, in the subway example the application DSML will allow the markup of the different aspects of the subway network (i.e. its structures and dynamics), and also the markup of the relevant variability of the user interface (e.g. a reference to an image representing the subway map, and the coordinates of the stations in this image). This application DSML will be described declaratively,

using a suitable, implementation – dependent, grammatical formalism (e.g. the implementation described in section 3 uses a formalism based on XML DTDs). Furthermore, the DSMLs formulated during this activity are stored in a *repository of DSMLs*, so they can be used in the incremental definition of DSMLs and reused in the formulation of new DSMLs. Thus, in the mid-term this repository will decrease the cost of the activity.
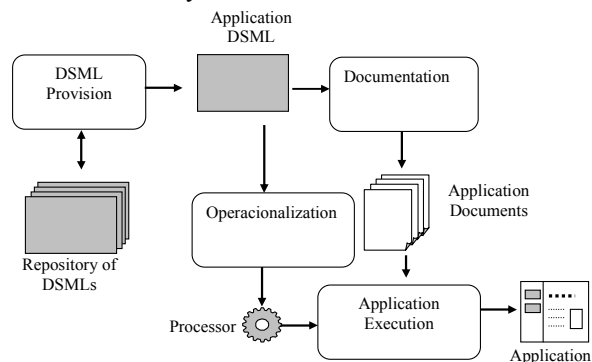

Fig. 1. Activities and products in ADDS.


```
<Subway>
  <Network>
    <Structure>
      <Stations>
        <Station id="TURINGAVE">Turing Ave.</Station>
        <Station id="KNUTHST">Knuth St.</Station>
        ...
      </Stations>
      <Lines>
        <Line id="BLUE">
        ...
    </Structure>
    <Dynamics>
      <Speeds>
        <Speed line="BLUE" value="50"/>
        ...
    </Dynamics>
  </Network>
  <UserInterface>
    <Title>Route searching in subway networks</Title>
    <ExitButton>Exit</ExitButton>
    <ResetButton>Reset Application</ResetButton>
    ...
    <Map loc="toysubwayEN.jpg"/>
    ...
    <Coordinates>
      <Coordinate station="TURINGAVE" x="65" y="157"/>
    ...
    </Coordinates>
  </UserInterface>
</Subway>
```
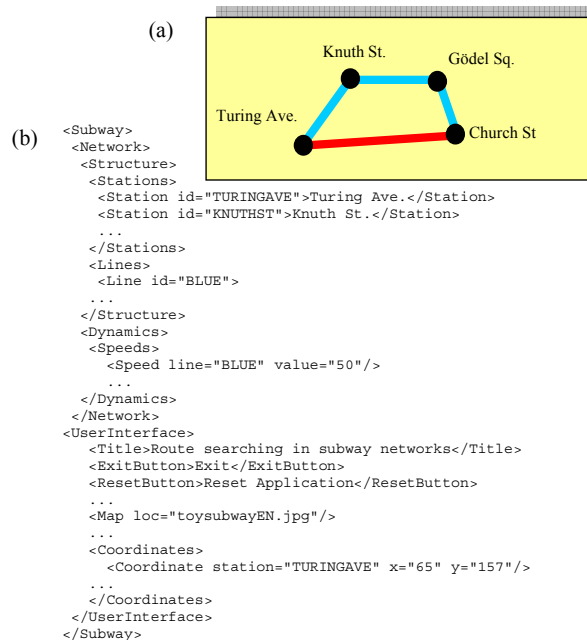
Fig. 2. (a) A miniature subway network, (b) part of the document for the route searching application in (a).

Once a suitable DSML is available, the applications can be described by means of marked *application documents* conforming the DSML. This process is carried out during the *Documentation* activity. In the subway case study, applications can be described by a single document containing the description of the subway network and the description of the user interface's variability. Fig. 2(b) drafts an example of

application document for the application associated with the fictitious miniature network in Fig. 2(a). The markup follows XML conventions, although this is implementation – dependent.
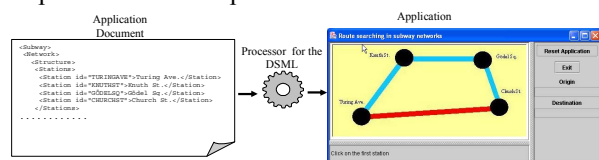

Fig. 3. Final running applications are obtained from application documents using processors for the application DSML.

The production of running applications from the application documents is carried out using suitable *processors* for the application DSMLs. The construction of these processors is the aim of the *Operationalization* activity. This activity produces a suitable processor for the DSML that is used to process the application documents during *Application Execution* activity (Fig. 3). Notice that the implementations of the *Operationalization* activity must cope with the incremental formulation of DSMLs, thus introducing mechanisms to appropriately extend the processors as the DSMLs evolve. These mechanisms can be based on standard techniques for the development of modular language processors [3][6][7].

2.2. Sequencing of the Activities

The diagram in Fig. 4 shows the sequencing of ADDS activities. This diagram reveals the iterative – incremental nature of the approach. ADDS indeed introduces two distinguished loops in the development process: the *production* loop, related to application development and maintenance, and the *evolution* loop, related to the DSML evolution and its appropriateness for marking up the documents of the application.

During the *production loop* the application document is processed to build and execute the application. Then this application is evaluated, and consequently, some modifications and/or improvements in the application could appear. Usually, these changes will only affect the application documents[1]. So this loop can be characterized by the production and modification of application documents, and by the construction and testing of the documented applications. For instance, in the subway example, a preliminary subset of the subway network can be initially documented, in order to provide a first

---

[1] Eventually the processor might also need to be adapted to correct some bug and/or misunderstanding of the operational meaning required for the DSML, although these cases will be typically less frequent than changes in the document.

working prototype of the final application. Next, this documentation can be completed to tackle the overall network, and, then, in a third iteration the variability of the user interface can be fine-tuned. New maintenance iterations can arise during application exploitation when the network changes (for instance, due to the addition of a new station or a new line).
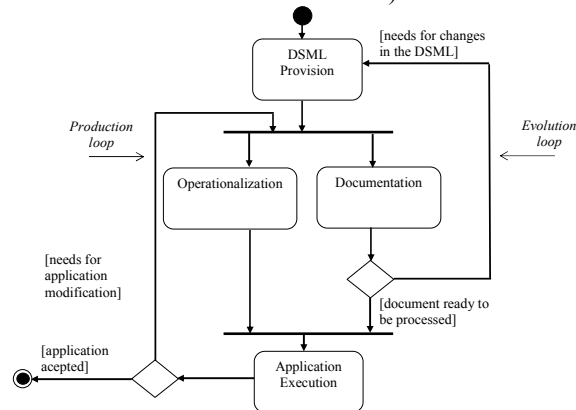

Fig. 4. Sequencing of activities in ADDS.

The evolution loop arises during the *Documentation* activity, when new markup needs are identified. Such needs can be due to a refinement of the structure of some application document, or the incorporation of new aspects into these documents to address new requirements. In this case, the usual production loop is abandoned, and the *DSML provision* activity is performed again with the aim of extending the DSML to contemplate the new markup needs. Hence it can be said that the DSML *evolves*. The evolution of the DSML is indeed mirrored at the operational level by the evolution of the corresponding processor. Finally, the usual production loop is entered again. In the subway example, the DSML can evolve to include new structural elements in the networks (e.g. corridors) together with their associated dynamics. Another example of evolution is the inclusion of different user interface styles (e.g. evolution from a simple console-based user interface to a graphic one).

2.3. Actors

ADDS distinguishes between two main actors in the development of applications: *domain experts* and *developers* (Fig. 5). The *domain experts* are the experts on the different aspects of the application's problem domain (*domain aspects*). For example, in our case study, these domain aspects will correspond to the subway network structure and dynamics, so domain experts could be the network organizers of the subway companies. In turn, the *developers* are experts in computer science whose main responsibilities are the

formal definition of the application DSML, using appropriate grammar formalisms and the construction of the processor for this DSML.
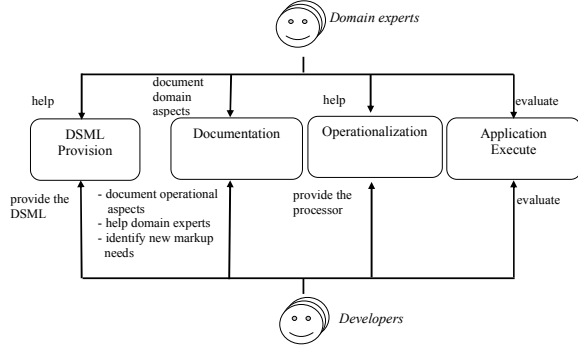

Fig. 5. Actors in ADDS and their roles in the different activities.

During the *Documentation* activity, domain experts and developers collaborate in the application description by creating and marking up the application documents. In addition, these documents can contain other *operational aspects* not derivable from those domain aspects. For instance, in our subway example, these aspects are the variability of the user interface. These operational aspects can be initially documented and marked up by the developers, but due to the readability of descriptive markup languages, these can be subsequently understood and modified by the domain experts. We have successfully used this document-mediated communication between domain experts and developers to enhance the development and maintenance of educational applications [4], and also rapid prototyping in the hypermedia domain [9].

3. Implementing ADDS

The effective use of ADDS supposes the definition of the different activities and products in terms of specific protocols, procedures and technologies, thus leading to *implementations* of the approach. This section describes briefly ADDS^{LM,OADDS}, an ADDS implementation focused on *DSML provision* and *Operationalization* activities.

3.1. DSML Provision

In ADDS^{LM,OADDS} the incremental provision of DSMLs is accomplished as an appropriate combination of *linguistic modules*, each one characterizing a part of the final DSML (hence the ^{LM} superscript). These modules are declarative, grammar-based characterizations of parts of the final DSML. The resulting DSML is also declaratively described by a document grammar. This grammar is obtained by

following a *grammar production specification*. The aim of this specification is to resolve conflicts between linguistic modules (e.g. name conflicts) and to adapt the concrete markup vocabulary to different contexts (e.g. a specification can set up the names of the tags in English, whilst another can do the same in Spanish). Notice that, by providing alternative production specifications, it is possible to get different profiles of the same DSML. In our subway example, the DSML will indeed include linguistic modules for marking up the subway networks, as well as modules for marking up the relevant aspects of the user interface. All these modules will be combined to produce a suitable grammar for this DSML following an appropriate production specification.

```
(a)    Module: Lines
       Grammar:
       <!ELEMENT Lines (Header?,LinesTitle?,LinksTitle?,(Line)+)>
       <!ELEMENT Header (#PCDATA)>
       <!ELEMENT LinesTitle (#PCDATA)>
       <!ELEMENT LinksTitle (#PCDATA)>
       <!ELEMENT Line (Name,Link+)>
       <!ELEMENT Name (#PCDATA)>
       <!ELEMENT Link EMPTY>
       <!ATTLIST Link origin IDREF  #REQUIRED destination
                                              IDREF #REQUIRED
(b)    ...          length NMTOKEN #REQUIRED>
       Module: Lines
         Map: Lines  → Líneas
              Header → Encabezado
              LinesTitle → TítuloLíneas
              LinksTitle → TítuloTramos
       ...
```

Fig. 6. (a) An example of a linguistic module, (b) part of a grammar production specification.

In this implementation, we have completely based the definition of our DSMLs on the XML markup metalanguage [19], and we have used XML DTDs for describing both the grammatical aspects of the linguistic modules and the final document grammars. Although XML DTDs are simpler than other schema languages [8] we have found several advantages in their use. On one hand, they are an integral part of the XML standard, and on the other hand, and more importantly, they are simple-to-use and more understandable mechanisms for domain experts [9]. The grammar production specifications in ADDS^{LM,OADDS} are sets of renaming rules for markup vocabularies of the linguistic modules, thus allowing for the resolution of the different name conflicts between modules' DTDs. Because name conflicts are solved at the grammatical level, the use of namespaces [19] is not necessary in this implementation. In our opinion, this facilitates the *Documentation* activity for domain experts, which is one of the main objectives of ADDS. Fig. 6(a) shows an example of a linguistic module which governs the markup of the lines of a subway network. Fig. 6(b), in turn, depicts part of a grammar production specification for the DSML in the subway example.

According to this specification, the markup vocabulary of the final grammar will be in Spanish.

3.2. Operationalization

Operationalization in ADDS^{LM,OADDS} conforms the OADDS (Operationalization in ADDS) model for the incremental development of DSMLs' processors (hence the ^{OADDS} superscript). OADDS is based on the well-known techniques of syntax-directed translation, widely used in the compiler construction domain [1], although the model also takes advantage of the descriptive nature of markup languages to promote the incremental development of processors. Early versions of the model can be found in [12][14][15]. In this paper, we briefly outline the current version, such as that expressed in [16].
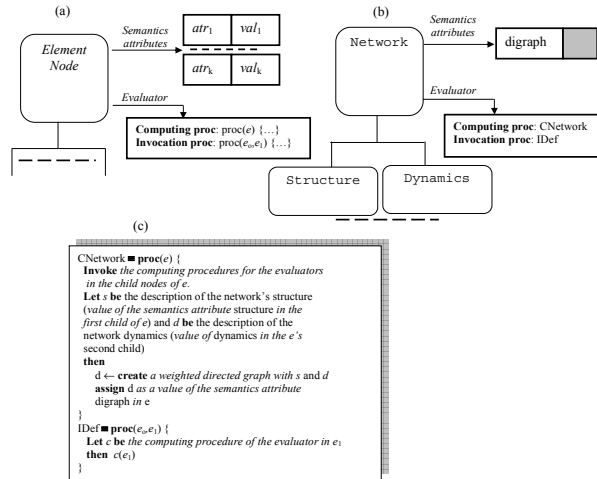

Fig. 7. (a) Operationalization of element nodes, (b) operationalization for elements of `Network` type, (b) pseudo-code for the components named in (b).

Processors in OADDS are built using appropriate, domain-dependent combinations of *operationalization modules*. These modules are used to attach suitable operational meanings with the linguistic modules for the DSMLs. Thus processors are incrementally provided as long as the DSML evolves.

The operationalization modules are used to operationalize document trees by assigning a set of *semantic attributes* and an *evaluator* to each tree node-element (Fig. 7-a-). The evaluator is used to calculate the values of the semantic attributes. Evaluators have two different procedures: a *computing procedure*, with the code required to compute the values of the semantic attributes, and an *invocation procedure*, used by this computing procedure to invoke the computing procedures owned by other evaluators in the node-element's vicinity. This structure facilitates the incremental provision of evaluators because each

constituent procedure can be independently extended (e.g. the invocation procedure can be extended to propagate the value for a new attribute without changing the corresponding computing procedure). Fig. 7-b- and Fig. 7-c- outline the operationalization of the elements of `Network` type, used for marking up the documentation of the subway networks in our example application domain.

Once a suitable set of operationalization modules for a given DSML is available, the processor for this DSML is constructed providing a little, domain–dependent, program that uses these modules to execute the applications from the DSML-conforming documents. Typically this program is based on three types of basic processes: the *construction of document trees* from marked documents, the *operationalization of document trees* using suitable operationalization modules, and the *evaluation* of operationalized trees. These processes are provided by predefined components, so operationalization in OADDS is reduced to the incremental provision of operationalization modules and to the construction of processors by providing minimal glue programs.

Being a conceptual model, OADDS is independent of any specific implementation technology. Nevertheless, OADDS can be easily implemented as an object-oriented framework. The main advantage of this implementation is that it promotes its integration with widely used object-oriented frameworks for document processing [2]. The details of this OADDS implementation can be found in [16].

4. Conclusions and Future Work

This paper describes ADDS, our document-oriented approach to software development based on DSLs. ADDS conceives DSLs as descriptive domain-specific markup languages (i.e. as DSMLs) that enable a document – oriented paradigm to application construction. Documents are a natural way to achieve communication between human beings. Consequently, the documental nature of the approach increases its acceptance in information-intensive areas of software development, where ADDS provides the feasibility of describing applications as human readable documents, understandable and editable for both domain experts and developers. The use of common markup standards (e.g. XML) also contributes to its acceptance, because the common syntax shared by DSMLs contributes to minimizing the *tower of Babel* syndrome. The use of documents marked according to standards also improves application portability. In addition, the incremental nature of the approach contributes to decreasing effort during the formulation,

operationalization and maintenance of the DSMLs, because this can be amortized through the development of applications. This incremental formulation also provides the flexibility required by the development of complex applications. DSMLs can be indeed extended when new markup needs are discovered. This also facilitates the use of the resulting DSMLs, because it avoids the inclusion of very general or sophisticated descriptive artifacts. This evolution can be managed at the operational level by adopting suitable mechanisms to ensure semantics modularity for the components used in the development of the processors, like the OADDS operationalization model.

Current work is oriented to improving ADDS and ADDS^{LM,OADDS} pragmatic applicability by using them in several projects in the domain of distributed e-learning systems. With this work we hope to achieve further refinements and improvements in our approach and in the cited implementation. In addition, we are interested in a better characterization of the authoring problems in ADDS, not only in the *Documentation* activity, but also in all the other activities of the approach. Finally, as a future work, we are considering the formulation of alternative implementations of the approach based on the use of object-oriented attributed grammars [10] both for the incremental provision and for the incremental operationalization of DSMLs.

5. Acknowledgements

6. References

[1] Aho, A., Sethi, R., Ullman, J. D. Compilers: Principles, Tech. and Tools. *Adisson-Wesley*. 1986

[2] Birbeck,M et al. Professional XML 2nd Edition. *WROX Press*. 2001

[3] Duggan,D. A Mixin-based, Semantics-based Approach to Reusing Domain-specific Programming Languages. *ECOOP 2000*. 2000

[4] 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 Conf. Building University Electronic Educational Environments*.1999

[5] Goldfarb, C. F. The SGML Handbook. Oxford University Press. 1990

[6] Hudak,P. Domain-Specific Languages. In *Handbook of Programming Languages V. III: Little Languages. And Tools*. Macmillan Tech. Publishing. 1998

[7] Kastens,U., Waite,W.M. Modularity and Reusability in Attribute Grammars. *Tech. Report CU-CS-613-92.*

University of Colorado. 1992

[8] Lee,D., Chu,W.W. Comparative Analysis of Six XML Schema Languages. *ACM SIGMOD Record. 29(3)*. 2000

[9] Navarro, A., Fernández-Valmayor, A., Fernández-Manjón, B., Sierra, J.L. Conceptualization prototyping and process of hypermedia applications. *Int. Journal of Software Engineering and Knowledge Engineering*. In press

[10] Pakki,J. Attribute Grammar Paradigms – A High-Level Methodology in Language Implementation. *ACM Computing Surveys 27(2)*. 1995

[11] 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. *Int. Conf. on Software ICS 2000. 16th IFIP World Comp. Congress*. 2000

[12] Sierra, J. L., Fernández-Manjón, B., Fernández-Valmayor, A., Navarro, A. An Extensible and Modular Processing Model for Document Trees. *Extreme Markup Languages 2002*. 2002

[13] Sierra, J. L., Fernández-Manjón, B., Fernández-Valmayor, A., Navarro, A. A Document-Oriented Approach to the Development of Knowledge-Based Systems. *LNAI 2040. Springer-Verlag*. 2004

[14] Sierra, J. L., Fernández-Valmayor, A., Fernández-Manjón, B., Navarro, A. Building Applications with Domain-Specific Markup Languages: A Systematic Approach to the Development of XML-based Software. *ICWE 2003*. 2003

[15] Sierra, J. L., Fernández-Valmayor, A., Fernández-Manjón, B., Navarro, A. Operationalizing Application Descriptions with DTC: Building Applications with Generalized Markup Technologies. *SEKE'01*. 2001

[16] Sierra, J.L. Towards a Document-Oriented Paradigm to Application Development). *Ph.D. Thesis (in Spanish). Univ. Complutense de Madrid. Madrid. Spain*. 2004

[17] Thibault,S.A., Marlet,R.,Consel,C. Domain-Specific Languages: From Design to Implementation. Application to Video Device Drivers Generation. *IEEE Transactions on Software Engineering, 25(3)*. 1999

[18] Van Deursen, A., Klint, P.,Visser, J. Domain-Specific Languages: An Annotated Bibliography. *ACM SIGPLAN Notices 35(6)*. 2000

[19] www.w3.org/TR