A Document-Oriented Approach to the Development of Knowledge Based Systems[1]

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

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

Abstract. ADDS (Approach to Document-based Development of Software) is an approach to the development of applications based on a *document-oriented paradigm*. According to this paradigm, applications are described by means of *documents* that are marked up using descriptive domain-specific markup languages. Afterwards, applications are produced processing these marked up documents. Formulation of domain-specific markup languages in ADDS is a dynamic and eminently pragmatic activity since these languages evolve in accordance with the authoring needs of the main actors that participate in the development process (i.e. *domain experts* and *developers*). OADDS (Operationalization in ADDS) is a processing model that promotes the construction of modular language processors and their incremental evolution. Thus, OADDS is specifically designed to cope with the evolutionary nature of the domain-specific markup languages encouraged by ADDS. ADDS and OADDS have successfully been applied to the development of applications in knowledge-intensive domains (i.e. transport networks and educational hypermedias). This paper also describes the advantages (incremental development and maintenance improvement) that this approach supposes for the development of knowledge-based systems.

1 Introduction

The development of applications in general, and of Knowledge-based Systems (KBSs) in particular, can be considered as a linguistic activity arising from the collaboration between the clients that have a problem to be solved, the domain experts with the knowledge required to solve that problem, the developers building the application, and the final users. Indeed, looking for ways to facilitate the communication between all the actors in this process is essential in order to guarantee a successful development. This is particularly true in the development of KBSs, where communication between clients, knowledge engineers and developers has always been considered to be critical.

[1] The Spanish Commitee of Science and Technology (TIC2000-0737-C03-01, TIC2001-1462 and TIC2002-04067-C03-02) has supported this work.

This paper describes our approach for application development, which we called *document-oriented* paradigm, and its specialization in the development of KBSs. According to this paradigm, the building of an application begins by describing the application using one or more documents, marking up these documents using a domain-specific markup language, and finally, producing the application using a suitable processor for this language. Thus, the development of a KBS using this paradigm implies the provision of a document written in a natural language subset. This document contains the knowledge that is going to be managed by the system. Then, tags and attributes are pragmatically added to this document in accordance with a previously defined markup language. These tags and attributes make the data and knowledge structures relevant to the inference engine explicit. Finally, the inference engine for the KBS is conceived as a processor driven by the markup.

The ADDS approach (Approach to Document-based Development of Software) is an implementation of the document-oriented paradigm where the document types and the languages used to markup them up evolve incrementally according to the needs of domain experts and developers. OADDS (Operationalization in ADDS) is a processing model for ADDS documents that introduces mechanisms used in the production of modular processors to adapt to the evolutionary nature of languages in ADDS.

The rest of this paper is structured as follows. Section 2 describes the development of KBSs according to the document-oriented paradigm. Section 3 describes ADDS, the approach that implements this paradigm. Section 4 describes OADDS, the operationalization model of ADDS. Section 5 describes some related work. Finally, section 6 presents the conclusions and gives some ideas for future work.

2 The Development of KBSs using the Document-oriented Paradigm

Documents play an important role in human communication. Therefore, the adoption of a *document-oriented paradigm* for the development of applications must be seen as a plausible alternative that could alleviate the communication problems arising among the different actors engaged in the software development process.

The development and maintenance of KBSs is particularly sensible to these communication problems. Indeed, the knowledge acquisition problem is a critical aspect of this type of system. The model-based approaches that arose during the nineties (see [16] for a survey) conceive the solution to this problem as the explicit formulation of a *knowledge model* capable of identifying and structuring the different types of knowledge required to solve a problem, together with the roles played by these types of knowledge in the reasoning process. Nevertheless, these approaches usually distinguish between the model and its subsequent implementation. This means that for the participants either an initially complete model is provided (and this is not realistic even for little toy domains), or they must cope with the maintenance problems derived from the translation of model changes into the implementation. This scenario is similar to that arising in the domain of educational applications [5].

The document-oriented paradigm gives a pragmatic solution to the maintenance problem in the construction of model-based KBSs. Indeed, according to this paradigm:

- The model leads to different domain-specific languages for describing the different types of knowledge. Actually, these languages are suitable subsets of the natural language similar to those used by the domain experts (see Fig.1a). This similarity facilitates the experts' elicitation of knowledge as documents in natural language.
- Then, using descriptive domain-specific markup languages the structure of this knowledge is made explicit. Thus, documents marked using these languages are prepared for their automatic processing (see Fig.1b). This initial markup can be performed by the developers. But because of the simplicity and legibility of the descriptive markup, domain experts can *understand* these documents, and they can directly modify the knowledge described in them (the *contents* of such documents), and with the help or supervision of the developers, they could even extend the language by adding new tags (either directly or using a specific edition tool).

(a)

```
Jam Problem in Incorporation to M30.
The speed registered by the sensor S1 is low,
the speed registered by S2 also is low,
but the speed registered by S3 is normal.
```

(b)

```
<pattern>
  <name>Jam Problem in Incorporation to M30.<name>
  <body>
<and><measure>The <type>speed</type> registered by the sensor <sensor>S1</sensor> is <value>low</value></measure>,
<measure>the <type>speed</type> registered by <sensor>S2</sensor> also is <value>low</value></measure>,
but <measure>the <type>speed</type> registered by <sensor>S3</sensor> is <value>normal</value></measure>.</and>
  <body>
</pattern>
```

Fig. 1. (a) Knowledge about an anomalous situation pattern in a traffic network, (b) markup of the knowledge expressed in (a).

- The developers, in turn, can include additional *operational contents* oriented to make the final processing of the knowledge possible. Examples of this situation are problem–solving methods written in some suitable formal language or knowledge transformations given as document transformations.
- Finally, the implementation of the KBS is obtained building a suitable processor for the markup language used in the pragmatic markup of the document.

With the document-oriented paradigm, the *implementation–model* duality disappears, collapsing into documents where the knowledge provided by the experts and other additional knowledge mix together. Tags, attributes and structure are added by the developers and domain experts to make document processing possible. In addition, the use of descriptive markup facilitates the incremental evolution of the languages and documents. These domain-specific markup languages are not static, unmovable entities, but they can evolve according to changes in the needs of experts and/or developers, or when new markup needs are discovered as a consequence of model evolution.

Fig. 2 sketches the structure of a KBS according to the document-oriented paradigm. Such a structure is a generalization of the one arising in the arena of electronic document processing based on descriptive markup technologies [6].

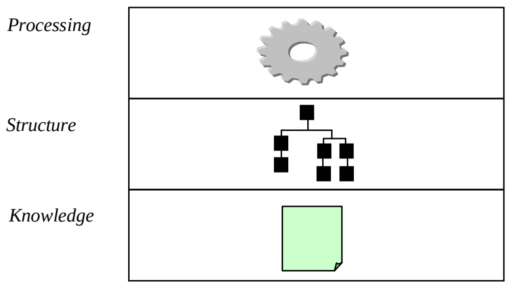

Fig. 2. Organization of a KBS according to the document-oriented paradigm.

The next sections analyze how to adapt the pragmatic nature of the document-oriented paradigm, either in the formulation of markup languages, or in the construction of the processors for such languages.

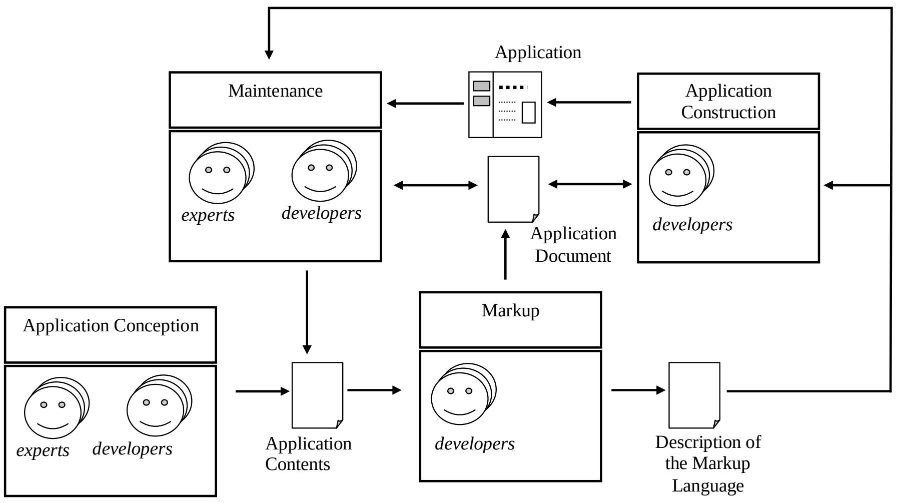

Fig. 3. Participants, activities and products in ADDS.

3 The ADDS Approach

ADDS [13] is an implementation of the document-oriented paradigm that is mainly driven by the authoring needs of the people involved in the process of documenting the applications (domain experts and developers). Fig. 3 sketches the different products, participants and activities involved in ADDS. The next subsections detail each of these aspects.

3.1 Participants

ADDS distinguishes two types of participants in the application development:
- *Domain experts*. They are responsible for the provision and maintenance of the application contents. In the KBS domain, they are the experts that provide the different types of knowledge that will finally be included in the system.
- *Developers*. They are responsible for building the final application. In the KBS domain, the range of developers includes, from knowledge engineers that design the knowledge models, to programmers developing the inference engine and other software needed to produce the executable application.

According to the document-oriented paradigm, the interaction between these participants is mediated by the final document to be produced (*application document*). For instance, during the initial stages of the development, developers interact with domain experts to decide the type and the form of the contents to be included in the application documentation. In addition, developers mark up these contents and assist the domain experts during the maintenance of the marked up document.

3.2 Products

According to ADDS, application construction involves the following types of products:
- The *contents* integrated in the application document.
- The *application document* produced by marking up such contents with tags and attributes.
- The *description of the specific markup language* used for the markup process of such contents.
- The final *application* produced by processing the application document.

3.3 Activities

ADDS identifies the following activities in the application production:
- *Initial application conception*. In this activity, the domain experts, assisted by the developers, conceive and produce an initial description of the application to be built. In the KBS construction, the developers help the experts to informally define the set of documents required to describe all the knowledge needed by the system.
- *Markup*. In this activity, the developers decide how to mark up the application contents documents to obtain the application document. As a result, an explicit description of the markup language (given by an schema or DTD) is produced, together with the application document marked up with this language. Note that because the iterative nature of ADDS, the markup language can evolve to accommodate newly identified markup needs.

- *Application construction*. In this activity, the developers produce the application from the application document. During this activity, developers can add new operational contents to the document and mark up such contents. Finally, they produce the application following the OADDS model introduced in the next section.
- *Maintenance*. In this activity, the experts, assisted by the developers, perform suitable modifications on the marked contents. These modifications are driven by the evaluation of the application produced at previous stages. The domain experts can even master the markup language, thus being able to use this language to add new markup. This situation is promoted by the use of descriptive markup, focused on content structure, and providing domain significant tag names and attributes. In addition, during this activity the need for introducing new contents may arise (in the case of a KBS, to introduce new knowledge as a consequence of an evolution in the model). Therefore, this will mean an evolution in the markup language used. This evolution will be done by developers and approved by domain experts.

4 The OADDS Operationalization Model

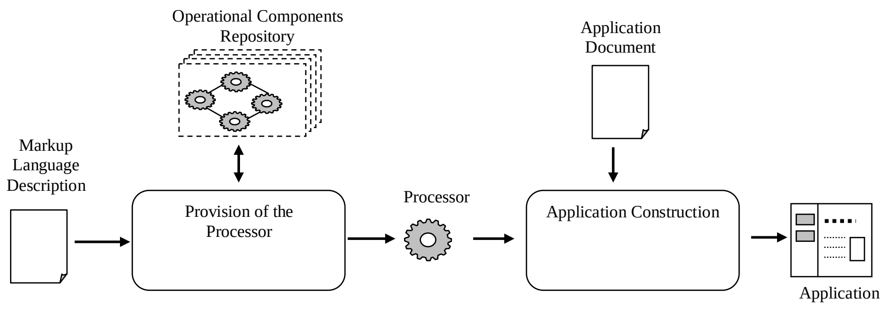
Operational Components Repository
Application Document
Markup Language Description
Provision of the Processor
Processor
Application Construction
Application

Fig. 4. Products and activities in OADDS.

OADDS is the operationalization model used in ADDS to produce applications from marked documents. OADDS is based on the classical techniques of construction of language processors based on syntax-directed translation [1]. Thus, OADDS conceives operationalization as the processing of the application document with an appropriate language processor, that is, a processor specifically built for the markup language used to mark up the application document. But, because of the evolutionary nature of the markup languages used in ADDS, OADDS establishes mechanisms to obtain *modular* processors from components. These components can be extended and combined according to the markup language evolution. Despite being independent from specific implementation technologies, OADDS is naturally implemented as an object-oriented framework. Finally, the document-oriented paradigm itself can be applied in the construction of OADDS processors. So, it is possible to describe processors as a collection of marked documents. The contents of these documents will be the code associated with the basic *semantic actions* required during the processing of the application documents, while the markup will establish how to combine these actions to produce the final processor (in this point of view, OADDS extends and combines similar approaches used in languages such as YACC and XSL [19]). Thus, the complete documentation of an application could include not only the document describing it, but also the documentation of the processor used to process the application document.

Fig. 4 sketches the different products and activities involved in OADDS. All these activities are carried out by the developers (their presence is omitted). The following subsections detail each one of the aspects depicted in Fig. 4.

4.1 Products

In addition to the application document and the description of the language used to mark up this document, OADDS introduces the processor of this language, together with a repository of *operational components* that facilitates the modular construction and the evolution of this processor.

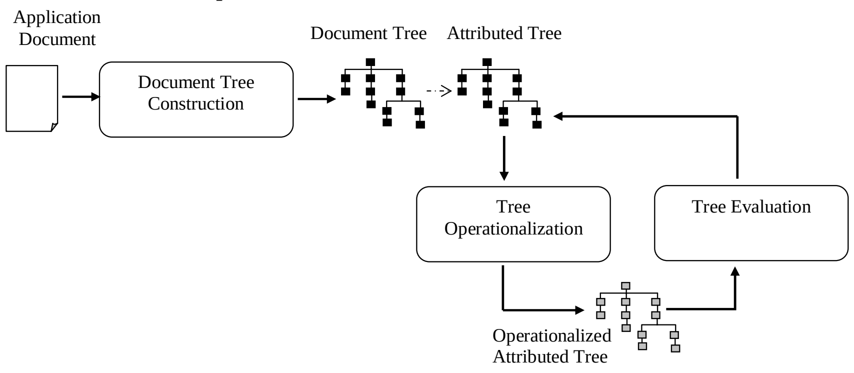

Fig. 5. Outline of the information flow in OADDS processors.

Fig. 5 sketches the information flow (that can be implemented in a modular way using the appropriated operational components) inside the OADDS processors. The key object of such processing is the *attributed tree*. Each node of this tree is associated with a set of attributes, each one having a value. The processing starts building the tree representing the application document. The construction of this tree can be carried out using any of the usual parsing frameworks for structured documents [3]. Next, the processing proceeds with the iteration of the attributed tree over a *tree operationalization* stage, followed by a *tree evaluation* stage. During the tree operationalization step, each node in the tree is decorated with (i) a *controller*, which is a procedure determining the evaluation order for the neighbours of the node, and (ii) an *initializer*, an *advancer* and a *finalizer*, which are the procedures organizing the local processing of this node. During the evaluation stage, the procedures decorating the tree are applied in the right order. Basically, this stage consists of a tree traversal commanded by the controllers. In this traversal, the processing procedures are applied in the right order. The modularity of the model is obtained thanks to the possibility for extending these procedures. The extensions will be devoted to propagating new attribute values in the tree, and to interrupting the evaluation when errors or other abnormal conditions are discovered. These adaptation and extension capabilities are essential in order to simplify the development and maintenance of complex systems, such as KBSs.

4.2 Activities

OADDS introduces the following two activities into the application development:
- *Provision of the processor*. In this activity the processor used to execute the application documented by the application document is provided. Usually such a processor has been previously constructed for a similar application, so it will be reused on the new application document. In case the new application document uses new markup structures, the old processor will be adequately extended by adding new operational components for dealing with the new structures and with the extensions of existing ones. Only at the initial stages of the development of a new type of applications will the implementation of a new processor from scratch be mandatory, and, even in this case, the provision of operational components that can be reused in the construction of new processors will pay off in the long run
- *Application construction*. The application arises as the result of processing the application document with its processor.

5 Related Work

Descriptive markup languages were introduced as a convenience for the processing of electronic documents [6]. HyTime [9], an SGML [6] extension devised to deal with the design and construction of hypermedia applications, demonstrated that in some domains, these kinds of languages could be used for describing applications in terms of documents that, in turn, could be processed for building the final application. Moreover, proposals like DSSSL [8] proved that this document-oriented paradigm could be used not only for the applications, but also for describing the processors used to produce the applications. XML [19] and its related technologies have generalized the use of descriptive markup languages as a standard way for information interchange between applications and for many other uses. Indeed, there are several proposals for applying markup languages to the KBSs domain (see [2]). Note that most of these approaches conceive markup languages as *static* entities. ADDS takes a more pragmatic position because markup languages are considered as *dynamic* objects that evolve when the contents or the markup needs of these contents change. OADDS gives an operational solution to this dynamic nature of the languages, encouraging the construction of modular processors from components that can be extended and adapted according to markup language evolution.

ADDS shares many features with the approach to software development based on *Domain-Specific Languages* (DSLs [17]). The main difference is that, while these kinds of languages are, in essence, specific purpose programming languages, ADDS follows a document-oriented paradigm, more suitable for content intensive applications, such as KBSs, where there is a clear distinction between contents and the languages used to structure such contents.

Modular language processor construction has been popularized by the functional programming community, where the main approach is based on *monads* and *monads transformers* [7], although proposals in the object-oriented paradigm (based on the use of *mixins* [4]), and in the attribute grammar approach to the construction of language processors can also be found [18]. OADDS semantic modularity mechanisms are inspired by these proposals, and also resemble the extension mechanisms of methods in CLOS [15]. Indeed, the extensions of initializers, advancers and finalizers are similar to the definition of *before*, *around* and *after* methods in CLOS. In this sense controllers are analogous to primary methods.

ADDS generalizes the methods for the construction of educational applications for foreign language text compression presented in [5]. ADDS also generalizes the approach for the generation of hypermedia prototypes from XML documents describing the hypermedia contents and navigation presented in [10]. Work in [12][13][14] shows the evolution of ADDS. Initially, in [12][14] this approach was called DTC (structured Documents, document Transformations and software Components). The use of this approach for the construction of applications in the transport networks domain (more precisely, subway networks) is described in [12]. Work in [11] explores its use in the educational hypermedia domain.

6 Conclusions and Future Work

This paper outlines the development of KBSs using a document-oriented paradigm. According to this paradigm, knowledge is initially described using documents formulated in the same language used by the domain experts: a subset of the natural language. Afterwards, domain-specific descriptive markup languages are used to make the structure of the knowledge described in these documents explicit. This makes its automatic processing possible. The ADDS approach, together with the OADDS operationalization model, provides for an implementation of this paradigm. The pragmatic nature of ADDS supposes the evolutionary nature of these markup languages, as a response to the dynamic process of determining all the knowledge needed by KBSs. The modularity and extensibility of the inference engines promoted by OADDS simplifies the maintenance and the updating of the final application. Moreover, it also simplifies the development of application families, because, once all the basic components are made available, it is very simple to produce new related applications.

As future work it seems interesting to perform a more systematic study about the markup process applied to knowledge documentation and the cooperation between domain experts and developers in this process. Also, a study of the viability of knowledge acquisition tools based on ADDS / OADDS is needed. These tools will facilitate the edition and the markup processes of knowledge documents.

References

1. Aho, A. Sethi, R. Ullman, J. D. Compilers: Principles, Techniques and Tools. Adisson-

Wesley. 1986.

2. Antonoiou,G.; van Harmelen,F. Web Ontology Language: OWL. In Staab,S.; Studer,R. Handbook on Ontologies in Information Systems. Springer-Verlag. 2003
3. Birbeck,M et al. Professional XML 2nd Edition. WROX Press. 2001.
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. 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
6. Goldfard, C. F. The SGML Handbook. Oxford University Press. 1990
7. Hudak,P. Domain-Specific Languages. Handbook of Programming Languages V. III: Little Languages. And Tools. Macmillan Tech. Publishing. 1998
8. International Standards Organization. Document Style Semantics and Specification Language (DSSSL). ISO/IEC 10179. 1996.
9. International Standards Organization. Hypermedia/Time-based Structuring Language (HyTime) – 2d Edition. ISO/IEC 10744 . 1997.
10. 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
11. Navarro,A.; Sierra, JL.; Fernández-Manjón, B.; Fernández-Valmayor, A. XML-based Integration of Hypermedia Design and Component-Based Techniques in the Production of Educational Applications. In M. Ortega and J. Bravo (Eds). Computers and Education in the 21st Century. Kluwer Publisher. 2000.
12. 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
13. 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. Third International Conference on Web Engineering ICWE 2003. Oviedo. July 14-18. 2003.
14. 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.
15. Steele JR, G.L. Common LISP: The Language (Second Edition). Digital Press. 1990.
16. Studer, R.; Fensel, D.;Decker,S.;Benjamins,V.R: Knowledge Engineering: Survey and Future Directions. In: F. Puppe (ed.): Knowledge-based Systems: Survey and Future Directions. Lecture Notes in Artificial Intelligence (LNAI), vol. 1570, Springer-Verlag, 1999.
17. Van Deursen, A. Klint, P.Visser, J. Domain-Specific Languages: An Annotated Bibliography. ACM SIGPLAN Notices. 35(6). 2000.
18. 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
19. www.w3.org/TR