International Journal of Software Engineering
and Knowledge Engineering
Vol. 15, No. 6 (2005) 975–993

World Scientific
www.worldscientific.com

DOCUMENT-ORIENTED DEVELOPMENT OF CONTENT-INTENSIVE APPLICATIONS

JOSÉ LUIS SIERRA*, BALTASAR FERNÁNDEZ-MANJÓN†,
ALFREDO FERNÁNDEZ-VALMAYOR‡ and ANTONIO NAVARRO§

Dpto. Sistemas Informáticos y Programación, Fac. Informática,
Universidad Complutense de Madrid,
C/ Professor José García Santesmases S/N, 28040 Madrid, Spain
**jlsierra@sip.ucm.es*
†*balta@sip.ucm.es*
‡*alfredo@sip.ucm.es*
§*anavarro@sip.ucm.es*

In this paper we promote a *document-oriented* approach to the development of *content-intensive* applications (i.e., applications that critically depend on the informational contents and on the characterization of the contents' structure). This approach is the result of our experience as developers in the educational and in the hypermedia domains, as well as in the domain of knowledge-based systems. The main reason for choosing the document-oriented approach is to make it easier for domain experts to comprehend the elements that represent the main application's features. Among these elements are: the application's contents, the application's customizable properties including those of its interface, and the structure of all this information. Therefore, in our approach, these features are represented by means of a set of *application documents*, which are marked up using a suitable descriptive *Domain-Specific Markup Language* (DSML). If this goal is fully accomplished, the application itself can be automatically produced by processing those documents with a suitable *processor* for the DSML defined. The document-oriented development enhances the production and maintenance of content-intensive applications, because the applications' features are described in the form of human-readable and editable documents, understandable by domain experts and suitable for automatic processing. Nevertheless, the main drawbacks of the approach are the planning overload of the whole production process and the costs of the provision and maintenance of the DSMLs and their processors. These drawbacks can be palliated by adopting an incremental strategy for the production and maintenance of the applications and also for the definition and the operationalization of the DSMLs.

Keywords: Comprehensibility; development approach; content-intensive applications; domain-specific markup languages; maintenance; evolution.

1. Introduction

There are applications (e.g., network-traveling information systems and educationally-oriented hypermedia applications) that critically depend on the domain informational contents and on the explicit representation of the structure

of these contents. In our approach these applications are called *content-intensive*. The maintenance of these applications typically involves the addition of new contents (e.g., addition of a new stop in a transport network or new exercises and documents in an educational hypermedia) and the modification of the existing ones (e.g., temporary disabling of a network's route or changing the evaluation criteria of a student's test), as well as the customization of other relevant features (e.g., usage and new help information in their user interfaces or the adaptation of a sequence of exercises or traveling plans to the user's necessities). This type of maintenance supposes the active involvement not only of the *developers* but especially of the *domain experts* (e.g., the transport network organizers in the traveling information domain or teachers in the educational case). The fact is that domain experts are the owners and/or the authors of the contents and they have a deep knowledge of the structure of these contents and of the other aspects of the application domain. In turn, developers have good skills in computer science, but they are not necessarily proficient in that domain. Thus, while strategies in software comprehension have usually been biased to help developers to understand software artifacts from a *developer*'s point of view [1], the maintenance of content-intensive applications also requires the use of comprehension strategies oriented to helping domain experts understand the representation of the main application's features. In our opinion, this can be done by representing the content and the customizable properties of the application, as well as the structure of this information, in a formalism that can easily be understood, modified and even authored by experts.

This paper presents our *document-oriented* approach to the development of content-intensive applications. This approach is based on the comprehensibility considerations mentioned above, and also satisfies other complementary requirements (information processability and reusability). In effect, the document-oriented development of an application promotes the representation of their main features by means of a set of *application documents*. To enable their automatic processing, these application documents are marked up with an easy-to-use and easy-to-understand descriptive *Domain-specific Markup Language* (DSML), specifically defined, chosen or adapted for the type of applications at hand. In addition, the application can be produced from these documents by processing them with a *processor* specifically built for the DSML defined. Thus, maintenance is largely reduced to the authoring of familiar application documents by the domain experts. We have successfully applied these document-oriented development principles in the production and maintenance of several educational and hypermedia applications as well as of knowledge-based systems. These experiences taught us the importance of adopting incremental approaches for the definition of the DSMLs and their operationalization (i.e., the construction of their processors) as this language can evolve during the successive development iterations to deal with the new markup needs proposed by domain experts and developers.

The paper is organized as follows: Section 2 enumerates the document-oriented development principles. Section 3 describes a process model for the document-

oriented development of content-intensive applications. Section 4 describes some related work. Finally, Sec. 5 details some conclusions and future work. For simplicity, the different aspects discussed in the paper will be illustrated with the development of applications for travel recommendations in subway networks.

2. Document-Oriented Development

This section describes and gives an overview of the document-oriented development of content-intensive applications. Thus, the main difficulties in the production and maintenance of this kind of applications, which are due to a lack of efficiency in the communication between domain experts and developers, are analyzed in Sec. 2.1. Section 2.2 reveals how similar difficulties arise in the domain of the publishing of electronic documents and how these difficulties are overcome in this domain with the use of descriptive markup technologies. Finally, Sec. 2.3 shows how these solutions can be translated to the production and maintenance of content-intensive applications yielding the document-oriented development approach.

2.1. *Production and maintenance of content-intensive applications*

The production and maintenance of content-intensive applications are costly tasks. Exceptions are those applications that, due to their simplicity, can be directly produced and maintained by domain experts using suitable authoring tools without further interventions. However, when the applications grow in content complexity and sophistication of the processing and interaction, this approach is no longer valid. Therefore, the successful production and maintenance of these sophisticated applications usually need to involve both domain experts and developers. In fact, the development process of this kind of applications usually proceeds through several iterations, where experts provide the contents and developers deliver new versions of the application, which are in turn evaluated by the experts, notifying developers of their modifications and/or improvements [2, 3]. Therefore, the typical scenario for the production and maintenance of content-intensive applications looks like the one in Fig. 1. For simplicity this scenario can be exemplified in the subway case study. Here the *network organizers* (domain experts) can provide the developers with tables representing the network organization (structure, timing information, schedules, etc.), and the developers produce an application for travel recommendations in this network. Plausible modifications during maintenance will affect the encoding of the network's information as well as other customizable features (e.g., information messages in the user interface).

The efficiency of communication between domain experts and developers in scenarios such as the one depicted in Fig. 1, and thus the costs of the overall process, strongly depends on how domain experts provide the contents and notify developers of the modifications. In turn, it depends on how the contents and the other modifiable elements are encoded inside the application. In many situations these encodings are in application-dependent and processing-oriented formats, hardly com-

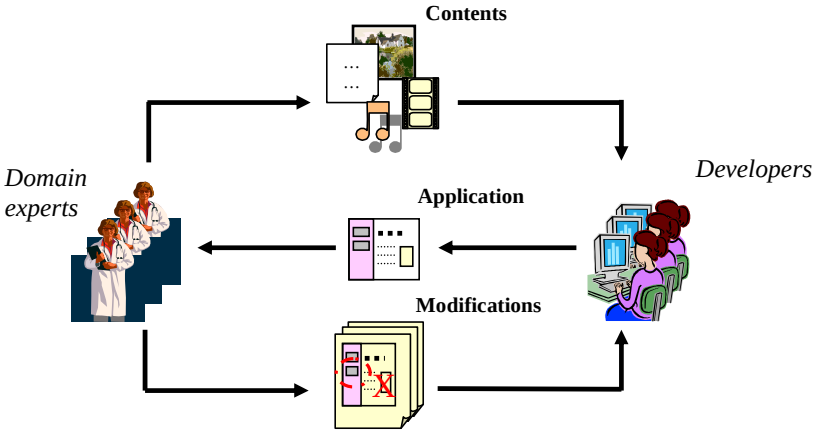

Fig. 1. Collaboration between domain experts and developers during the production and maintenance of a content-intensive application.

prehensible by domain experts. In addition, during production, developers must translate contents into internal representations, translation usually being a manual process. The consequence of the process described above is the lack of an appropriate communication channel between experts and developers to be used during the production and maintenance stages. Effectively, in applications that manage high volumes of contents and exhibit complex interactions with the user, domain experts can be in the need of annotating their modifications directly onto snapshots of the applications' screens [4]. To take into account these annotations, developers analyze them in order to correct the application.

Therefore, there is the practical need to represent the main features of the content-intensive applications in a formalism meeting the following three requirements:

- *Comprehensibility.* The formalism must be comprehensible to domain experts, who must be able to author representations using this formalism. This way, the explicit notification of modifications would be avoided, because domain experts would be able to comprehend how the different customizable attributes of the application are represented and they could directly modify these attributes.
- *Processability.* The formalism must be machine-processable to let developers automate its translation into more convenient processing-oriented formats (e.g., translation from the description of a subway network into a weighted directed graph).
- *Reusability.* The formalism must facilitate the reuse of the representations both in future evolutions of the application and in other new ones (e.g., the representation of the subway networks in the travelling recommendation application could be reused in the development of a Geographical Information System).

Our chosen representation formalism is motivated by the techniques used in modern electronic publishing outlined in the next section.

2.2. *Publishing of electronic documents*

The publishing of electronic documents exhibits strong similarities with the production and maintenance of content-intensive applications.These similarities are emphasized in Fig. 2(a), where a simplified traditional publishing scenario is depicted. According to this, the publishing process also implies two kinds of participants with very different skills: the *authors* of the originals to be published, and the *publishers*. In Fig. 2(a) an original is produced by an author. Then, this original is processed by a publisher, who delivers a first printout of it. The printout is examined by the author who indicates the corrections. On the basis of these corrections the publisher produces new corrected printouts, leading to a situation analogous to that depicted in Fig. 1. Nevertheless, in modern electronic publishing environments there are well-established procedures for speeding up this process.

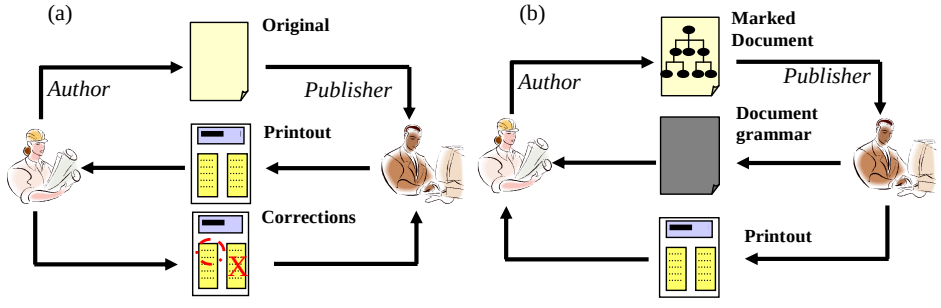

Fig. 2. (a) Collaboration between an author and a publisher during the publishing of an original, (b) improving the process with descriptive markup languages.

Descriptive markup languages are extensively used in electronic publishing to improve the publishing process [5–7]. Figure 2(b) summarizes the improved process. Here the publisher provides the author with a descriptive markup language defined in a *document grammar* (e.g. a DTD — *Document Type Definition* [7]; see Fig. 3(a) for a small example). Being descriptive, this markup is oriented to representing the logical structure of the documents instead of how these documents will be subsequently processed (Fig. 3(b)). Therefore, the author is able to comprehend this language and to use it to represent the structure of its original by marking it up. In addition, the author is able to check the correctness of the markup by validating the marked original against the document grammar. Once the original is validated, the publisher processes it to produce the printout. Because the document conforms the markup language, the defects that remain in the printouts are due to misunderstandings either in the contents or in the intentional, although

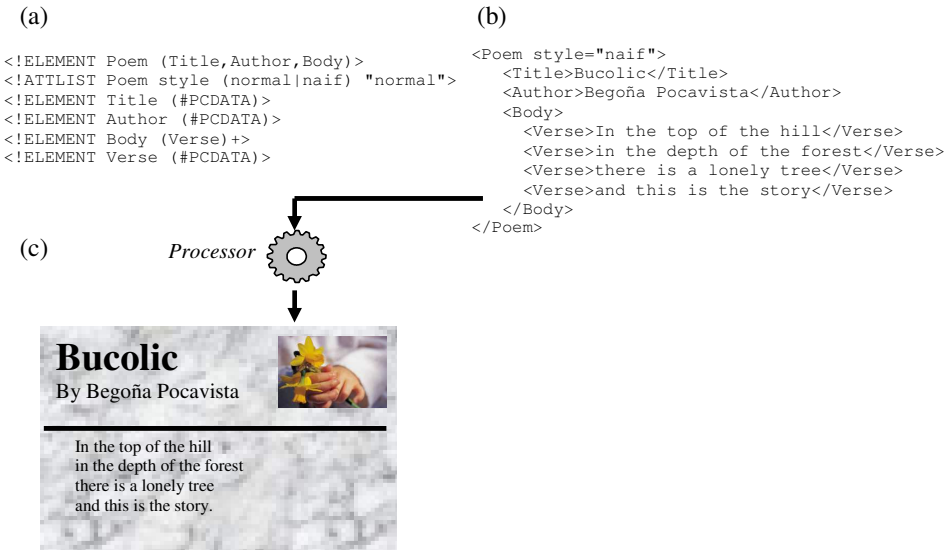

Fig. 3. (a) A simple DTD for marking up poems, (b) a document conforming the DTD in (a), (c) the document in (b) can be processed with a suitable processor to obtain printouts.

grammatically correct, use of the markup. Both types of defects can be directly solved by the author on the marked document, letting the publisher produce new printouts at no cost. Actually these printouts can be produced automatically, processing the documents with a *processor* for the markup language used (Fig. 3(c)).

Notice that the descriptive markup-based formalism complies with the comprehensibility, processability and reusability requirements when they are restricted to the publishing domain [6]. Descriptive markup is easy to understand by authors, who can use this either directly or with the help of a specialized WYSIWYG[a] editor. In addition, descriptive markup is formalized with a document grammar and therefore the processing of the marked documents can be automated. Finally, by changing the processor (i.e., by attributing a different presentation meaning to the markup language) a descriptively marked document can be used for different purposes and on different devices (e.g., a web-based presentation versus a postscript file for a printer). Thus we consider it a natural step to generalize this publishing process to the development of content-intensive applications. The resulting document-oriented development approach is detailed in the next section.

2.3. *Document-oriented development approach of content-intensive applications*

The document-oriented development of a content-intensive application promotes the use of documents to describe the contents and other customizable features of

[a]What You See Is What You Get.

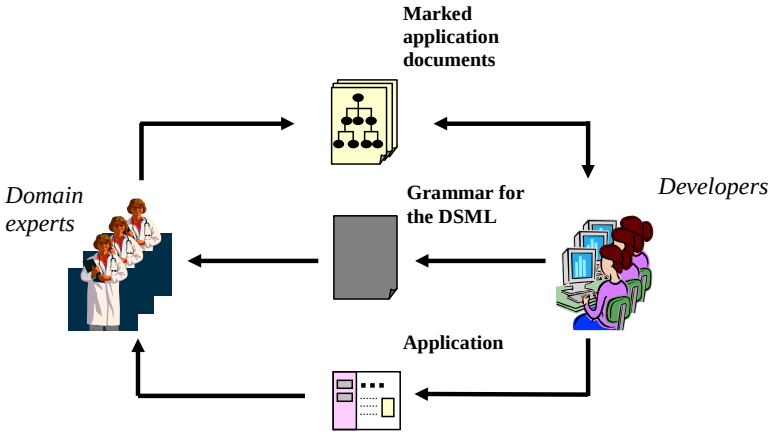

Fig. 4. Document-oriented development of content-intensive applications: a first approximation.

the application together with the structure of this information. Therefore, different facets of the application will be described by means of a set of documents that will be marked up with an application-specific descriptive DSML. Because these documents (and hence the DSML used in their markup) are specific to the application domain, they can be understood, produced and maintained by the domain experts. Moreover, the markup makes it possible for the automatic processing of these documents to rebuild the application when they change. Thus, the approach achieves a reasonable trade-off between the comprehensibility and the processability requirements. In addition, while the documentation of the customizable features is usually application-specific, descriptive markup facilitates the reusability of contents for other purposes.

In Fig. 4 a first approach to a document-oriented development scenario is depicted. This is parallel to the use of descriptive markup languages in the publishing domain given in Fig. 2(b). Here the documents are not restricted to textual prose, but they do enclose application contents and other customizable aspects of the application. As described before, these *application documents* are marked up with an *application* DSML that is specific for (and varies with) each type or family of applications. Finally, running applications are automatically produced by processing these marked documents with suitable processors like printouts are produced using the marked original documents in Fig. 2(b). Note that not only experts but also developers play an active role in the authoring of the application documents. Developers can document an initial minimal body of contents as well as other features in order to produce a first application prototype and to provide domain experts with initial document templates. In addition they also can help experts with the use of the DSML.

The document-oriented development can be exemplified in the subway case study. The main features of a traveling recommendation application for a city's

(a)

```
<!ELEMENT Subway    (Network,UserInterface)>
<!ELEMENT Network   (Structure,Dynamics)>
<!ELEMENT Structure (Stations,Lines)>
<!ELEMENT Stations  (Station)+>
<!ELEMENT Station   (#PCDATA)>
<!ATTLIST Station id ID #REQUIRED>
...
<!ELEMENT UserInterface (Title,ExitButton,
                         ResetButton,...,
                         Map,Coordinates)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT ExitButton (#PCDATA)>
...
<!ELEMENT Map EMPTY>
<!ATTLIST Map loc CDATA REQUIRED>
<!ELEMENT Coordinates (Coordinate)+>
<!ATTLIST Coordinate station IDREF #REQUIRED
                     x NMTOKEN #REQUIRED
                     y NMTOKEN #REQUIRED>
```

(b)

```
<Subway>
 <Network>
  <Structure>
   <Stations>
    <Station id="CONGOSTO">Congosto</Station>
    <Station id="VVALLECAS">Villa de Vallecas</Station>
    ...
   </Stations>
   <Lines>
   ...
   </Lines>
  </Structure>
  <Dynamics>
    ...
  </Dynamics>
 </Network>
 <UserInterface>
   <Title>Subway of Madrid</Title>
   <ExitButton>Exit</ExitButton>
   <ResetButton>Reset Application</ResetButton>
   ...
   <Map loc="madrid.jpg"/>
   ...
   <Coordinates>
     <Coordinate station="CONGOSTO" x="690" y="406"/>
   ...
   </Coordinates>
 </UserInterface>
</Subway>
```

Processor

(c)

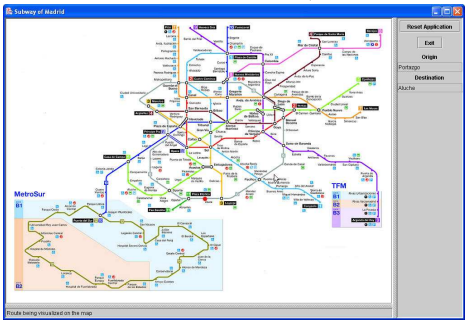

Fig. 5. (a) Fragment of the DTD for the subway's DSML, (b) fragment for the Madrid's application document, (c) customization of the travel recommendation application for the city of Madrid produced by processing the document in (b) with a suitable processor.

subway can be given in a single document containing the description of the subway network (its structural and dynamical aspects), and also the customizable properties of its user interface (e.g., names of buttons and labels, informative messages, a pointer to the image with the map and the coordinates of the stations in this image). Based on these considerations, developers can provide a DSML for marking up all these aspects. In Fig. 5(a) a fragment for the DTD of this DSML is depicted. Note that the markup vocabulary and the structure for this DSML are chosen to be understandable by the domain experts (in this case, the network organizers). Thus, with the help of developers, network organizers can author the application document. In Fig. 5(b) a fragment for the document concerning the application for the city of Madrid (Spain) is outlined. This document can be automatically processed to generate the associated travel recommendation application (Fig. 5(c)).

The document-oriented development approach meets the comprehensibility, processability and reusability requirements stated above. Nevertheless, the approach

has important initial costs, due to the need for a more explicit planning of the application development process and of the explicit formulation and operationalization of DSMLs. These costs can be amortized along the whole development process if an incremental strategy is carried out. Instead of being conceived as an immovable entity, the DSML for an application domain must be considered as a dynamic object that evolves with the application along the software development cycle, integrating the markup needs of domain experts and developers as they are discovered. This evolution not only affects the descriptive nature of the DSML, but also its operational aspects (i.e., its processor). Therefore, the success of the approach strongly depends on an appropriate management of the incremental provision of DSMLs and the incremental development of their processors. All these considerations are reflected in the process model for the document-oriented development approach proposed in the next section.

3. A Process Model for the Document-Oriented Development

This section details our process model for the document-oriented approach to the development of content-intensive applications. This process model is based on the work described in [8], its main goal being to enhance the comprehension of the elements that represent the main applications' features for the domain experts. The model also devotes special attention to the maintainability and reusability aspects, and encourages an iterative and incremental strategy for the production and maintenance of the applications and of the DSMLs and their associated processors. This is presented from three different perspectives. There is a perspective for the *activities* contemplated in the process and the *products* produced and consumed by these activities, which is presented in Sec. 3.1. Section 3.2 details the perspective of the *sequencing* of the activities, analyzing the iterative and incremental behavior of the model. Finally, in Sec. 3.3 the model is discussed from the perspective of the *actors* involved in the development and their *roles* in the different activities. Note that, this being a process model, the model does not compromise itself with concrete procedures for carrying out the activities nor with concrete technologies for getting the products. Nevertheless, sometimes we will give some hints about these aspects. More technical and in-depth expositions can be found in [8].

3.1. *Activities and products*

The activities and products involved in the document-oriented approach for the development of applications according to our process model are depicted in Fig. 6. Next we detail all these aspects and illustrate them using the subway case study.

The goal of the *DSML provision* activity is to obtain the *application DSML* that will be used to mark up the documents that describe the application's main features. This application DSML will be defined declaratively with a *document grammar* expressed in an appropriate formalism (e.g., DTDs or any other schema language [9]). This definition can be facilitated by reusing DSMLs previously defined

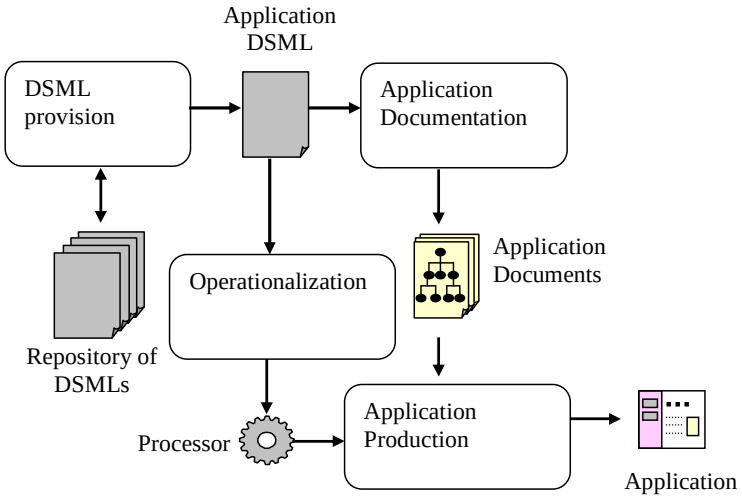

Fig. 6. Activities and products for a document-oriented development of content-intensive applications.

and stored in a *repository of DSMLs*. During this activity special attention must be paid to the comprehensibility requirement. Thus, the markup vocabulary must be close to the application domain and the markup structure must mirror the structure of the actual documentation used by the domain experts. In addition, the adhesion to a common markup metalanguage (e.g., SGML [7, 10], the *Standard Generalized Markup Language* or XML [11], the *eXtensible Markup Language*) can contribute to increasing this comprehensibility, because all the DSMLs will share a common syntax, helping domain experts avoid the assimilation of a new syntax with each new DSML. Finally, each particular DSML must be also appropriately documented to facilitate its comprehension by domain experts. In our experiences with the approach we have realized that, while SGML allows greater economy in the markup than XML, XML's simpler productions are more easily comprehended by domain experts [4]. In addition, we have successfully based the definition and the documentation of our DSMLs on SGML and XML DTDs [4, 12]. Although DTDs are simpler than other schema languages, according to our experience working with domain experts, we have realized that DTDs are simpler to use and is a more understandable formalism for these experts.

During *Documentation* the main application's features are described by means of a set of *application documents*. These documents will be marked up with the DSML provided during the *DSML Provision* activity. As presented above, this activity is facilitated by the documental nature of the representation and also by the descriptive and application-oriented nature of the DSML. Actually, domain experts are able to comprehend this representation and to author it. Although the use of markup-oriented or even DSML-oriented editors can contribute to making

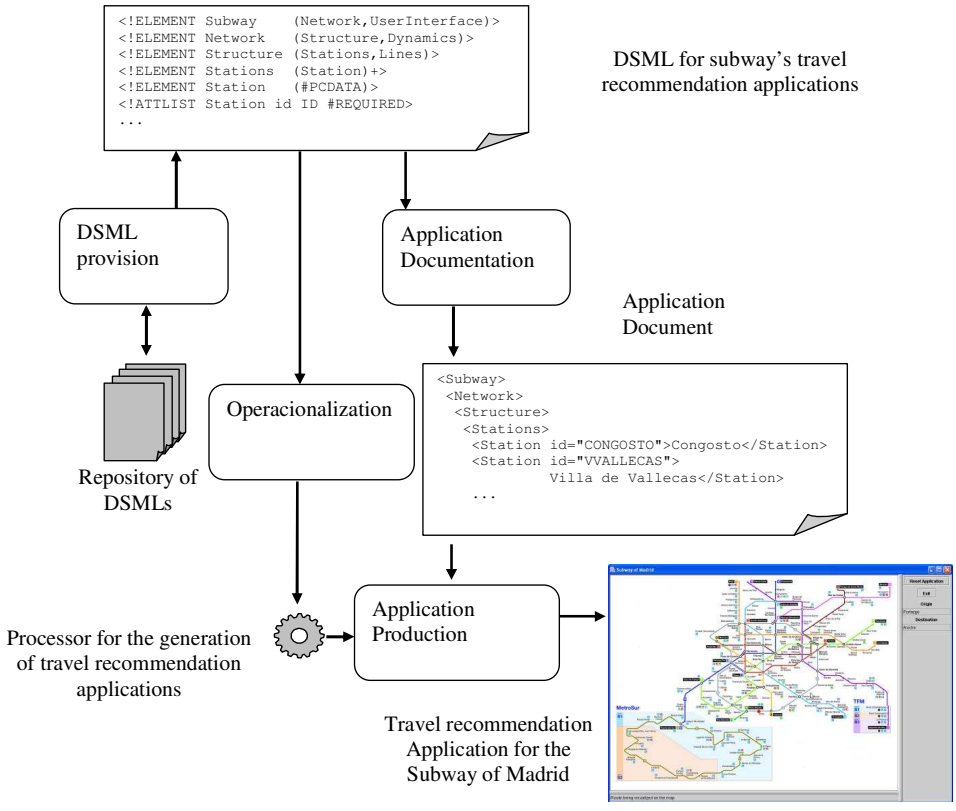

Fig. 7. Exemplification of the activities and products in the development of the travel recommendation application for the subway of Madrid (Spain).

this task easier for experts, we have realized that in some domains they are able to do it without any specific edition support [4]. This is a direct consequence of the commitment to the comprehensibility requirement.

The goal of *Operationalization* is to provide an appropriate processor for the application DSML. This processor will be used to produce the *application* from the application documents during the *Application Production* activity.

In Fig. 7, the activities and products for the development of the travel recommendation application for the subway of Madrid are illustrated. Therefore, the *DSML provision* activity yields the DSML for subway's travel recommendation applications defined by the DTD of Fig. 5(a). In this DSML several potentially reusable sublanguages can be identified (e.g., the DSML for describing subway networks, and, within this, the smaller DSMLs for describing the network's structure and dynamics). All these sublanguages can be stored in the repository of DSMLs and reused in the development of other applications in this domain. In turn, the *Documentation* activity produces an application document like the one outlined in Fig. 5(b). During the *Operationalization* activity a processor for the subway's DSML

is built, and the application itself is produced during the *Application Production* activity by processing the application document with this processor.

3.2. *Sequencing of the activities*

The development of content-intensive applications is iterative and incremental in nature, as we have shown in Sec. 2. In effect, during the application's life cycle new contents can be added, the existing contents can be refined, and the other applications' features can be fine-tuned. In these iterations new markup needs for the application documents that are not contemplated in the application DSML can be discovered. Thus, this DSML must *evolve* to accommodate these new markup needs. In turn, this evolution must also be mirrored in an evolution of its associated processor.

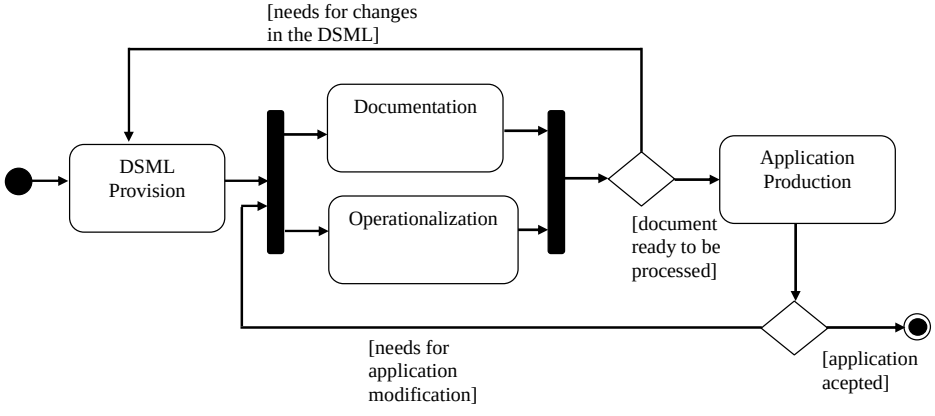

Fig. 8. Sequencing of the document-oriented development activities.

The iterative and incremental nature of the document-oriented development is made explicit in Fig. 8, where the sequencing of the activities identified in Sec. 3.1 is represented. The process starts with the provision of the application DSML. Then, the initial documentation of the main application's features and the processor for the DSML is provided, and an initial version of the application is delivered. Next the development proceeds iteratively until a satisfactory application is achieved. Iterations can typically be (Fig. 9):

- *Maintenance iterations* (Fig. 9(a)). These iterations involve the production of an application which is evaluated by domain experts. Consequently these experts can discover different aspects to be improved and/or completed. Then they proceed to refine the application documents. As a result a new enhanced application is produced. For instance, in the subway example, a preliminary subset of the subway network can be initially documented, in order to provide a first working

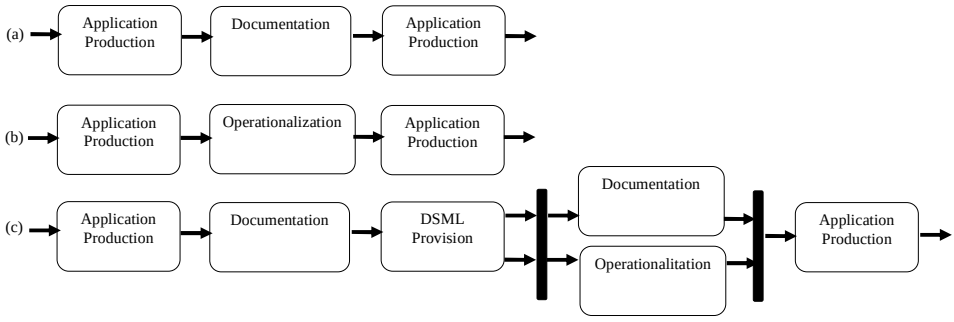

Fig. 9. (a) A maintenance iteration, (b) a corrective iteration, (c) an evolutive iteration.

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

- *Corrective iterations* (Fig. 9(b)). During application production an unexpected behavior can be discovered in the application that cannot be attributed to its documentation. This manifests a bug or an aspect to be improved in the DSML's processor that must be corrected by developers. Once the bug is fixed, the application can be produced again. For instance, when the travel recommendation application for a real-size network is produced, extremely long response times for the route calculations might be observed. These response times may demand the definition of a better search strategy that could be integrated in a corrective iteration.
- *Evolutive iterations* (Fig. 9(c)). These iterations involve evolutions of the application DSML and its associated processor. During the running of the *Documentation* activity in a usual maintenance iteration new markup needs not covered by the current DSML can be discovered. Such needs may be due to a refinement in the structure of an application document, or the inclusion of new elements into these documents to take new features into account. Consequently, the DSML must be extended in order to contemplate the new markup needs (i.e., the DSML evolves). Next the application documents can be refined using the new markup, and also the processor must be extended to deal with this markup. 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).

Therefore, in the document-oriented development, not only the application documents, but also the application DSML and the DSML's processor are objects of

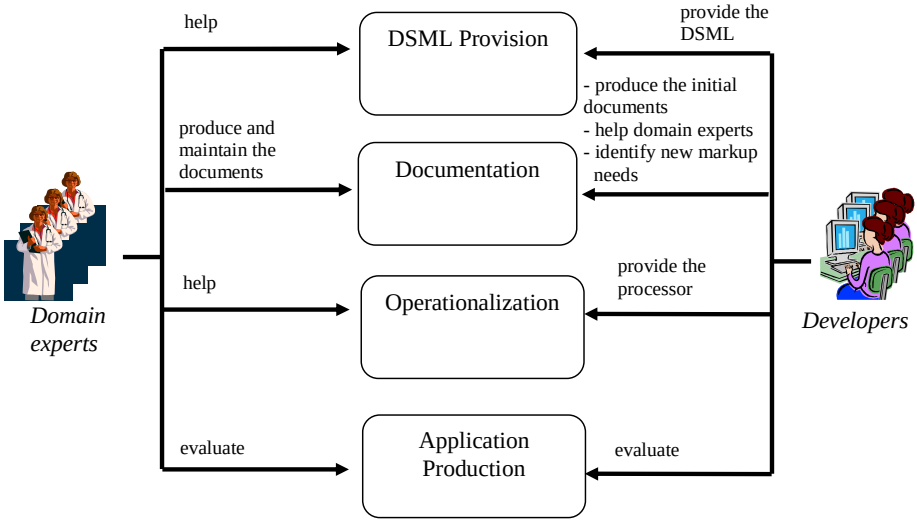

Fig. 10. Actors in the document-oriented development and their roles in the activities.

evolution. The evolutionary nature of DSMLs palliates the high initial costs of their formulation and operationalization. Besides, these costs can pay off along the development of new related applications. More importantly, this also improves the comprehensibility of these DSMLs, because the inclusion of very general or sophisticated descriptive artifacts in the languages is avoided. Nevertheless, the evolution of the DSMLs and their processors must be adequately managed. This can be done by adopting standard techniques for the systematic definition of languages and their processors [13–15] as well as for the modular construction of these processors [16–18]. For this purpose, we have used a modularization technique for the definition of DTDs in the style of that described in [19]. The details can be found in [20]. We have also formulated an operationalization model for the incremental construction of processors that is also described in [20].[b]

3.3. *Actors and their role in the activities*

The document-oriented development of content-intensive applications actively involves both domain experts and developers as indicated in Sec. 2. These actors collaborate in the whole process to produce the different products contemplated in Sec. 3.1. Therefore, they participate in all the activities, playing different roles. This participation, which is depicted in Fig. 10, can be summarized as follows:

- During the *DSML provision* activity the main role of the developers is to formalize the DSML as a document grammar. In this task, they are advised by the domain

[b]An earlier version can be found in [8] and a more updated summary in [21].

experts. The comprehensibility of the final DSML will strongly depend on this advice. Since domain experts know the application domain, they can present to the developers the kind of documents managed in this domain, and they can help them in the elicitation of the structure of these documents and of the terms used to refer to this structure (i.e., of the markup vocabulary).

- A similar scenario occurs during the *Operationalization* activity. This time the developers construct a processor for the DSML (i.e., they formalize a suitable operational meaning added to the descriptive markup). This processor will entail operations regarding the application domain that, in turn, will be clarified by the domain experts. Thus, participation of domain experts during *Operationalization* is also mandatory.
- During the *Documentation* activity, domain experts and developers collaborate in describing the application's features by creating and marking up the application documents. Drafts of these documents can be initially produced and marked up by the developers, but due to the comprehensibility of the application DSML, these can be subsequently understood and modified by domain experts. In addition, developers can help domain experts in understanding and using this DSML. Finally, they can also detect the new markup needs manifested by experts. The consequence of this detection is a new evolutive iteration.
- During *Application production* both experts and developers evaluate the application produced. While domain experts are able to validate the application against the functional requirements, developers can judge its adequacy to the non-functional ones (e.g., a bad response time in path calculation). The defects detected by domain experts usually start maintenance iterations, while those detected by developers lead to corrective ones.

4. Related Work

Our document-oriented development shares some features with *literate programming* [22, 23]. The literate programming paradigm, originally proposed by Knuth [22], enhances the comprehensibility of the programs by identifying these with their documentation. In literate programming, a hypertextual representation of the program code is promoted, which is interleaved with its documentation. The result (a *web*) is a narration of the program, in the same way that the program would be presented in a programming textbook. These documents are marked up for enabling both the assembling of working programs (*tangling*) and the production of documentation printouts (*webbing*). The ideas described in this paper differ from those of literate programming, because in our approach only the high level aspects of the main applications' features, but not the code of the programs implementing these applications, are documented and marked up. In our document-oriented approach documents are to be comprehended and authored by domain experts instead of by developers. The code itself is implicitly contained in the processor for the DSML used in the markup, in the same way that the assembler code for the

programs in a high-level programming language is *contained* in the compiler for this language. Because of this, in our work, suitable DSMLs are provided *for each* application domain or family of applications instead of using a fixed markup language, as in literate programming. Notice that processors themselves could be documented to enhance their comprehensibility by developers. In [8] we propose an initiative in this direction, where processors are documented following an *elucidative* style [24].

HyTime [25], an SGML extension for the description of hypermedia applications, demonstrated that in some domains, descriptive markup languages could be used for describing applications in terms of documents, and the applications described could be generated by processing these documents. XML and its related technologies have generalized the use of descriptive markup languages as a standard way of information interchange between applications and for many other uses. Whilst in almost all these approaches markup languages are pre-established, we have adopted a more dynamic approach, where markup languages are designed and evolve according to the markup needs of domain experts and developers.

Our work also shares many features with the approach to software development based on *Domain-Specific Languages* (DSLs [26, 27]). A pioneering work in the application of SGML/XML for the definition of DSLs is [28]. In [29] the relationships between markup languages and the DSL approach are highlighted. In [30] a comprehensive work about using XML technologies to implement the DSL approach is detailed. Although these works recognize the potentiality of markup metalanguages as a vehicle for defining DSLs, the stress is put on their use to formalize abstract syntax, instead of on their use as descriptive markup (meta) languages. The Jargons approach [31, 32] is similar to our work. In Jargons, DSLs are directly formulated, and even operationalized (using a scripting language), by domain experts. While the conception of this author-driven design of DSLs is consistent with our work, we consider it unrealistic to assign language design and operationalization responsibilities to domain experts. Instead, a community of developers is involved for this purpose. *Generative programming* [33] in general, and DSLs in particular, have been identified in [34] as complementary to program comprehension [1]. In [35] DSLs are proposed as a natural choice to express the result of the comprehension process.

The document-oriented development described in this paper was formerly suggested in [36, 37] as a vehicle to improve the production and maintenance of educational applications, and was consolidated after several experiences in the development of content-intensive applications in different domains. The work in [4] provides information about the application of the approach in the development of educational applications for the comprehension of texts written in a foreign language similar to that of the student's mother tongue. In [12, 38] a naive version of the approach is used in the broader field of hypermedia domain. In [39] the approach is proposed for the development of knowledge-based systems. The process model for the document-oriented development described in this paper is based on [8].

5. Conclusions and Future Work

This paper proposes to improve the production and maintenance of content-intensive applications using a document-oriented development approach. According to this approach, the main features of these applications are described by means of documents marked with application-specific DSMLs. Using these languages, domain experts can comprehend, produce and maintain the representations of the main features of content-intensive applications. In addition, in our approach, applications are produced by processing these marked documents with processors for the DSMLs used. We have successfully applied this approach in the development of educational and hypermedia applications, and also in the development of knowledge based systems. In this paper we systematize and generalize the approach with the formulation of a process model that promotes the iterative and incremental provision of both the applications and of the DSMLs and their processors.

Document-oriented development facilitates the maintenance and tuning up of content-intensive applications, because their main features can be described in terms of readable and editable documents, comprehensible by domain experts and developers. The approach also facilitates the reuse of the contents, because well-known markup standards (e.g., XML) can be used in the production of the documents describing them. This facet is especially relevant for content-intensive applications, where the representation of the contents in portable and standard formats is critical in order to extend their life cycles.

Our process model promotes an incremental formulation and operationalization of DSMLs. This helps to palliate the costs associated with these activities, because these costs can pay off during the development of more than one application in a given domain. This also provides the flexibility required by the development of complex applications, because the DSMLs can be extended when new markup needs are discovered. Finally, this feature also enhances the comprehensibility of the DSMLs by domain experts, because it avoids the inclusion of very general or complex markup structures.

Current work is oriented towards improving the pragmatic applicability of our document-oriented development process model by using it on several projects in the domain of distributed e-learning systems. With this work we hope to achieve further refinements and improvements in our approach. In addition, we are interested in a better characterization of the authoring problems in this process model, not only in the *Documentation* activity, but also in all the others. Finally, as future work, we are considering the experimentation with ways to enhance the comprehensibility of DSMLs' processors by developers in order to improve their production and maintenance. For this purpose we want to experiment with the definition of the DSMLs and their processors using object-oriented attributed grammars [15].

Acknowledgements

The Spanish Committee of Science and Technology (TIC2001-1462, TIC2002-04067-C03-02 and TIN2004-08367-C02-02) has supported this work.

References

1. A. von Mayrhauser and A. M. Vans, Program understanding: models and experiments, *Advances in Computers* **40** (1995) 1–38.
2. P. Fraternali, Tools and approaches for developing data intensive web applications: A survey, *ACM Computing Surveys* **3** (1999) 227–263.
3. N. Juristo and J. Pazos, Towards a joint life cycle for software and knowledge engineering, *IFIP Transactions* **A-27** (1993) 119–138.
4. A. Fernández-Valmayor, C. López-Alonso, A. Séré, and B. Fernández-Manjón, Integrating an interactive learning paradigm for foreign language text comprehension into a flexible hypermedia system, in *Proc. International Working Conference on Building University Electronic Educational Environments*, IFIP WG3.2 and WG 3.6, Irvine, CA, 1999.
5. C. F. Goldfarb, A generalized approach to document markup, *ACM SIGPLAN Notices* **16**(6) (1981) 68–73.
6. J. H. Coombs, A. H. Renear, and S. J. DeRose, Markup systems and the future of scholarly text processing, *Commun. ACM* **30**(11) (1987) 933–947.
7. C. F. Goldfarb, *The SGML Handbook* (Oxford University Press, Oxford, UK, 1990).
8. J. L. Sierra, A. Fernández-Valmayor, B. Fernández-Manjón, and A. Navarro, ADDS: A document-oriented approach for application development, *Journal of Universal Computer Science* **10**(9) (2004) 1302–1324.
9. D. Lee and W. W. Chu, Comparative analysis of six XML schema languages, *ACM SIGMOD Record* **29**(3) (2000).
10. International Standards Organization ISO, Standard Generalized Markup Language (SGML) (ISO/IEC IS 8879, 1986).
11. T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler (eds.), Extensible Markup Language (XML) 1.0 (Second Edition) (W3C Recommendation, 2000).
12. A. Navarro, B. Fernández-Manjón, A. Fernández-Valmayor, and J. L. Sierra, The plumbingXJ approach for fast prototyping of web applications, *Journal of Digital Information* **5**(2) (2004).
13. D. E. Knuth, Semantics of context-free languages, *Mathematical Systems Theory* **2**(2) (1968) 127–145.
14. A. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques and Tools* (Addison-Wesley, MA, 1986).
15. J. Paakki, Attribute grammar paradigms — A high-level methodology in language implementation, *ACM Computing Surveys* **27**(2) (1995) 196–255.
16. U. Kastens and W. M. Waite, Modularity and reusability in attribute grammars, *Acta Informatica* **31**(7) (1994) 601–627.
17. S. Liang, P. Hudak, and M. P. Jones, Monad transformers and modular interpreters, in *Proc. 22nd ACM Symposium on Principles of Programming Languages*, San Francisco, CA, 1995.
18. D. Duggan, A mixing-based semantics-based approach to reusing domain-specific programming languages, in *Proc. 14th European Conference on Object-Oriented Programming* (*ECOOP'2000*), Cannes, France, 2000.
19. M. Altheim, F. Boumphrey, S. Dooley, S. McCarron, S. Schnitzenbaumer, and T. Wugofski (eds.), Modularization of XHTML (W3C Recommendation, 2001).
20. J. L. Sierra, Towards a Document-Oriented Paradigm for Application Development (PhD Thesis — in Spanish, Universidad Complutense, Madrid, Spain, 2004).
21. J. L. Sierra, B. Fernández-Manjón, A. Fernández-Valmayor, and A. Navarro, Document-oriented software construction based on domain-specific markup languages, in *Proc. International Conference on Information Technology: Computing and Coding*

(*ITCC'05*), Las Vegas, 2005.
22. D. E. Knuth, Literate programming, *Computer Journal* **27**(1) (1984) 97–111.
23. D. Cordes and M. Brown, The literate programming paradigm, *IEEE Computer* **24**(6) (1991) 52–61.
24. K. Nømark, Requirements for an elucidative programming environment, in *Proc. 8th International IEEE Workshop on Program Comprehension IWPC'00*, Limerick, Ireland, 2000.
25. International Standards Organization ISO, Hypermedia/Time-based Structuring Language (HyTime) – 2nd Edition (ISO/IEC 10744, 1997).
26. P. Hudak, Domain-specific languages, in *Handbook of Programming Languages Vol. III: Little Languages and Tools*, ed. P. H. Salus (Macmillan Technical Publishing, Indianapolis, 1998), pp. 39–60.
27. A. van Deursen, P. Klint, and J. Visser, *Domain-specific languages: An annotated bibliography*, *ACM SIGPLAN Notices* **35**(6) (2000) 26–36.
28. M. Fuchs, Domain specific languages for ad hoc distributed applications, in *Proc. First USENIX Conference on Domain Specific Languages*, USENIX, Santa Barbara, 1997.
29. P. Wadler, The next 700 markup languages, *Invited Talk at the Second USENIX Conference on Domain Specific Languages*, USENIX, Austin, Texas, USA, 1999.
30. J. C. Cleaveland, *Program Generators with XML and Java* (Prentice-Hall, Englewood Cliffs, 2001).
31. L. H. Nakatani and M. Jones, Jargons and infocentrism, in *Proc. First ACM SIGPLAN Workshop on Domain-Specific Languages* (*DSL'97*), Paris, France, 1997.
32. L. H. Nakatani, M. A. Ardis, R. G. Olsen, and P. M. Pontrelli, Jargons for domain engineering, in *Proc. First USENIX Conference on Domain Specific Languages*, USENIX, Santa Barbara, 1997.
33. K. Czarnecki and U. Eisenecker, *Generative Programming: Methods, Techniques and Applications* (Addison-Wesley, New York, 2000).
34. D. Batory, Program comprehension in generative programming: A history of grand challenges, in *Proc. 12th IEEE International Workshop on Program Comprehension* (*IWPC'04*), Bari, Italy, 2004.
35. T. Bull, Comprehension of safety-critical systems using domain-specific languages, in *Proc. 4th IEEE International Workshop on Program Comprehension* (*IWPC'96*), Berlin, Germany, 1996.
36. B. Fernández-Manjón, A. Fernández-Valmayor, and A. Navarro, Extending web educational applications via SGML structuring and content-based capabilities, in *Proc. IFIP Internatinal Conference the Virtual Campus: Trends for Higher Education and Training*, Madrid, Spain, 1997.
37. B. Fernández-Manjón and A. Fernández-Valmayor, Improving world wide web educational uses promoting hypertext and standard general markup languages, *Education and Information Technologies* **2**(3) (1997) 193–206.
38. A. Navarro, A. Fernández-Valmayor, B. Fernández-Manjón, and J. L. Sierra, Conceptualization prototyping and process of hypermedia applications, *Int. J. Software Engineering and Knowledge Engineering* **14**(6) (2004) 565–602.
39. J. L. Sierra, B. Fernández-Manjón, A. Fernández-Valmayor, and A. Navarro, A document-oriented approach to the development of knowledge-based systems, in *Current Topics in Artificial Intelligence*, eds. R. Conejo, M. Urretavizcaya and J. L. Pérez-de-la-Cruz, LNAI 2040 (Springer, Berlin, 2004), pp. 16–25.