Computing and Information Systems **8** (2001), p.1-8

Handling requirements with XML like system specifications

Antonio Domínguez and Juan Corchado

This paper shows how XML metalanguage capabilities and related tools could be used first to model data structures and operations of domain specific languages, and second to facilitate the transformation process from system specifications to software systems.

This approach allows to identify the subsystems of a software system using different domain specific languages. Such languages and the language transformer rules are the result of a domain analysis process adequately customized for this propose.

INTRODUCTION

A system specification is the result of the first step of the software engineering cycle. In it, software engineers must take into account business processes, and the relations the new system should have with its environment. Four decades of software engineering have shown that it is very difficult to verify the correctness of system specifications, and to prove they meet all user requirements. By now, most of the times it is only possible to verify the specification correctness when the final system is in operation, and behaves as users expected for a long period of time.

Software reuse techniques tend to ensure correctness of software systems building them from reusable blocks of software proved correct and stable [19]. This kind of software systems made of reusable components, are built in a dual software engineering process [22], the first part of the process focuses in obtaining reusable components, and the second one in the usual system engineering process. A dual software engineering approach can improve traditional software engineering [15], but there are some pitfalls, and there exist domains[1] where this technique fails. It happens mainly because it is mandatory to reuse each component several times to obtain the return of the initial investment in building reusable components [4].

This paper shows a method for the obtention of Domain-Specific Specification Languages (DSSL) as a result of the first activity (i.e. Domain Analysis) of the software engineering with reuse dual process [15], instead of software components [9]. These languages can be combined to build a system specification, which can be transformed [7] in a final running system. These languages are generated, combined, and handled using as base metalanguage the eXtensible Markup Language (XML) [10].

XML[2] has been selected because it provides a special kind of objects, named XML documents, made up of several units called entities. It provides mechanisms to impose constraints in the layout and structure of these XML documents. Furthermost XML processors can be used to read, and provide access to XML documents structure and content, and can work as front ends for other software systems, which are able to adapt its behavior according to the XML document contents.

Next section shows a short review of the main differences between generic specification languages, and domain specific specification languages, and how to join different domain specific specification languages to specify a single system. Following it is introduced a method to adapt the domain analysis process to obtain domain specific languages. The fourth section discusses the use XML related capabilities and tools to support system specification, and system specifications transformation. An example is also included to show how XML based specifications can be transformed into code using a very simple transformer from XML spec to UNIX shell script code. The paper ends with a short review of related work and some conclusions.

DOMAIN CHARACTERIZATION AND IDENTIFICATION.

The software community has shown great interest in understandable and useful specification languages, both for software engineers and users. The main disadvantage of generic specification languages is that they have an important learning curve, and are not user friendly [16]. One way to obtain simpler and more user friendly specification languages is to restrict its generality to a well known domain [1].

Domain restricted languages, also known as Domain Specific Languages (DSL), could only be collateral or a byproduct of the domain analysis phase. They are most of the times used to facilitate the process of documenting the domain components and the domain structure. But they can be used to abstract and hide key concepts about the domain, so the process of specifying new systems can be done uppon them [25].

[1] A domain is a set of several related systems [23,24]

[2] A very large amount of information about XML, XML related languages, and tools can be found in "the XML cover pages", URL:http://www.oasis-open.org/cover/

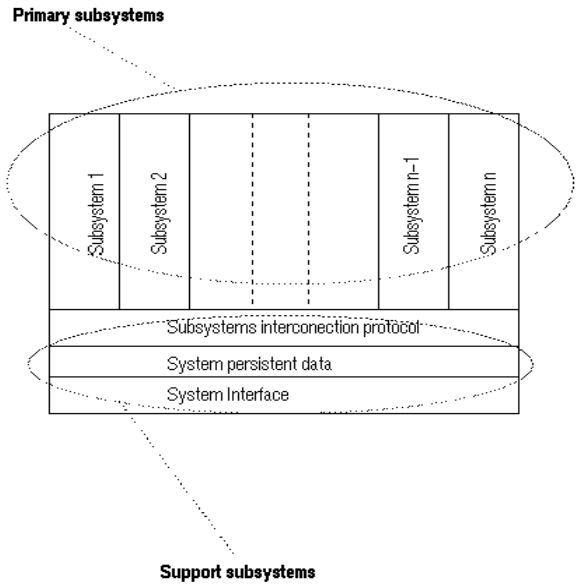

Figure 1: System decomposition in subsystems. Horizontal subsystems are those responsible for system main functionalities. Horizontal or support subsystem are typically, system interface, system data base, and system in the subsystem glue implementation.

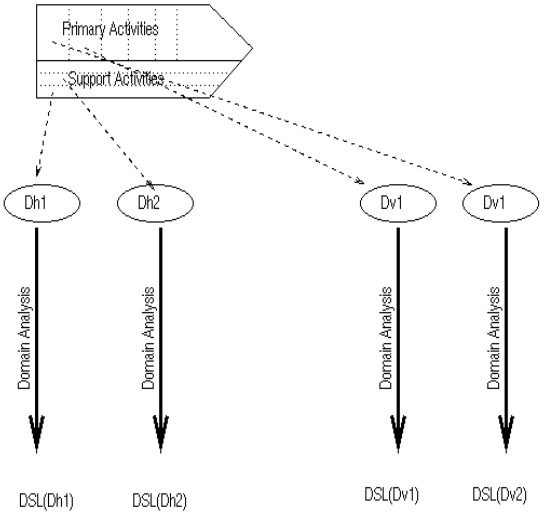

Figure 2: Process for obtaining a domain specific language for a given domain. Each one of the value chain activities is a domain. A domain analysis process will result in the corresponding domain specific language.

Nevertheless implementing a DSL is a difficult and expensive process due to its dependence with the compiler or translator used to translate them [6]. To solve this difficulties several approaches have risen based in the extension of a common base metalanguage [18,14] to obtain the DSL. Other important drawback against DSLs is that there is not a known and formalized process to obtain them from a domain [13]. Current approaches rely on the processes of analysis, design, implementation of the language, and building the compiler, but not on the techniques that can be used during the process [28, 6].

The intention of the approach presented in this paper is first to characterize such process and second to do it using XML as a common base metalanguage. In this case each domain is seen as a business area of the organization characteristic value chain[3]. At this point it is important to note that there are two different kind of value chain activities. There are activities related to horizontal domains, and activities which related domain is vertical.

Horizontal domain activities are the support activities of the organization: human resources, accounting, etc. DSL for this kind of domains could be used to model systems for a wide range of business. Vertical domain activities are the business primary activities (business core activities), like sales, inbounds logistics, or customers service. DSL obtained for this kind of domains have a more restricted application field, as they can only be re-used in similar businesses (banking, bookstores, universities...).

Similarly the subsystems of a software system can be characterized in two different ways, as support or horizontal subsystems, and as primary or vertical ones. Fig. 1 shows a hypothetic system characterized in this way. Horizontal subsystems are the ones commonly found in any software system, for example the systems interface or the persistent data storage. Vertical subsystems are responsible of the system main functionality. For example in a compiler: the parsing subsystem, the lexical analysis subsystem, the semantic analysis subsystem, etc.

Having characterized a system in this way, the process of system specification can be separated for each subsystem. A requirement analysis process should be run, so all system requirements have to be included in the system specification. The two kind of requirements that will emerge in the system specification are the following:

- Vertical subsystem requirements: The requirements related with each one of the vertical subsystems.
- Horizontal subsystem requirements: Requirements related with, persistent data storage, interface requirements and interconnection protocol.

Using small DSL to specify subsystems will avoid main problems in the requirement specification process, such us ambiguity, complexity of generic

[3] Value chain, as defined in the Porter's value chain model [21]

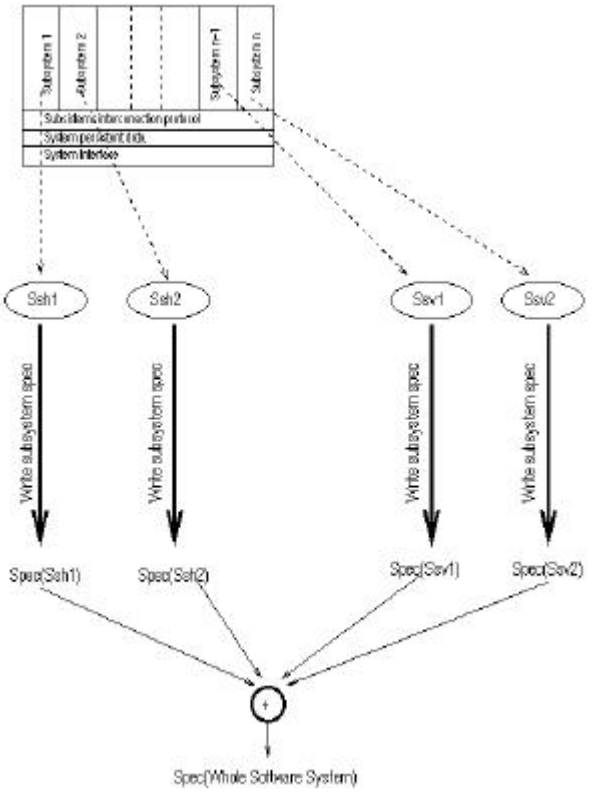

Figure 3: Process of obtention of a system requirements specification. Different domain specific languages are used for specifying each subsystem

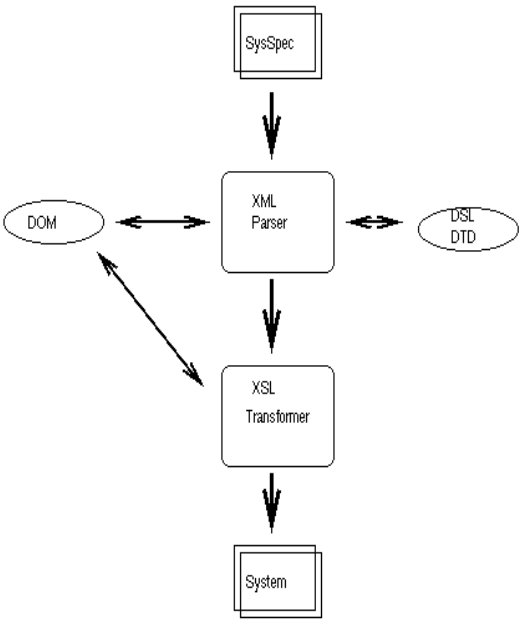

Figure 4: Process of automatic system generation. Given a system specification, written with appropriate DSSL for each subsystem, a XML parser, and a XML transformer based in XML, and in the DOM, can transform the specification in a final software system.

specification languages, incompleteness, etc. They are useful both to specify horizontal subsystems: system interface, subsystem interconnection protocol, and system persistent data, and also to specify vertical subsystems. Although horizontal DSLs are widely available, there is a necessity for vertical domain oriented DSL. For that it is necessary to adapt the traditional process of domain analysis. Next section discuses how it could be done.

DOMAIN ANALYSIS ADAPTATION TO DOMAIN-SPECIFIC SPECIFICATION LANGUAGES

The starting point of the domain engineering life cycle, is the identification of the domain where to run the Domain Analysis (DA) process. The business value chain concept is used to identify the objective domain. Therefore the target domain should be, or should represent, one of the activities of the business value chain.

The process of domain analysis starts with the identification of the domain using the business value chain model. Then an independent process of domain analysis should be carried out of each of the value chain activities, in order to obtain the corresponding domain model, the DSSL, and the transformation rules needed to translate DSSL specifications to a compiler understandable high level programming language (see figure 2).

Support activities, being common to the most of business structures, will result in DSSLs with a wider range of application, that is, it can potentially be used to specify a higher number of systems. In the other hand vertical activities (vertical domains) will result in DSSL with more restricted reusability potential.

The process of domain analysis to obtain each domain DSSL could run independently for each domain. There must be at least one DSSL to specify each subsystem in order of being able to specify a new system using a set of DSSLs.

The approach shown here is grounded, for building the domain specific language and the transformer, onto XML metalanguage capabilities and associated tools. This enables the possibility of mixing several DSLs in a single system specifications, because the result will allways be a valid XML document. Such document simplifies the process of language definition, enables the use of XML tools, and provides a simple mechanism to easily add up several subsystem specifications to build a full system (see fig. 3).

Each DSL should allow the writing of specifications to both, data entities and operations

```
<spec>
 <add_host name="mainserver" ip="193.146.11.13">
 </add_host>
 <add_host name="pepe" ip="193.146.11.12">
 <nick>lucas</nick>
 <nick>jose</nick>
 </add_host>
</spec>
```

Figure 5: System specification example. Representing that two new hosts will be added to the hosts database. The second one with two nicknames

inside the domain. Data structures of the domain constitute also a horizontal subdomain, and each data structure is seen as an Abstract Data Type (ADT) [3]. The building of the data structures DSL can be done using a generic data specification language (DSL (data)). New domain ADT are built upon the native or derived ADT of the generic data structures DSL(data). The subset of the language that enables the specification of operations is described building a XML Document Type Definition (DTD) [17].

USING XML BASED DSSLS TO SPECIFY AND TRANSFORM SOFTWARE SYSTEMS

A XML DTD and a related grammar can be used to check specifications correctness. This is done using a standard XML parser. The process of parsing the system specification generates the Document Object Model(DOM), that is the XML document hierarchic structure. The transformer also plays the role of front end to check the grammar of the DSL, and to transform the specification in a final system, using the Extended Styleshet Language (XSL) capabilities and related tools(see fig. 4).

Parsing and transforming DSL specifications, can be done upon current resources like XML parsers, XSL [12], and XML APIs. Solutions that enable the use of XML as basis for specify horizontal subsystems, are in a continue emerging process. For example:

- To specify interfaces between systems and users there are proposals like the Extensible User Interface Language (XUL)[4] , or the User Interface Markup Language (UIML) [27].
- XML-Data can be used to specify persistent data structures.
- The Bean Markup Language(BML) [11], or the Koala Bean Markup Language (KBML)[5] can be used for subsystem intercommunication protocols.

```
cp /etc/hosts /tmp/ghosts
echo "mainserver 193.146.11.13 " >>
/etc/ghosts
echo "pepe 193.146.11.12 lucas jose "
>> /etc/ghosts
cp /tmp/ghosts /etc/hosts
rm /tmp/ghosts
```

Figure 7:Resulting code generated directly from the specification

```
<XST__XMLscript="1.1">
 <_data file="adh1.xml" />
 cp /etc/ghosts /tmp/ghosts
 <_foreach element="\spec\add_host">
   # \spec\linea := .name " " .ip #
   <_foreach element="nick">
   #\spec\linea := \spec\linea " " ._content #
   </_foreach>
  echo "# \spec\linea # " >> /etc/ghosts
  </_foreach>
cp /tmp/ghosts /etc/hostsrm /tmp/ghosts
</XST>
```

Figure 6: Transformercode rules. It can obtain values form SML tag attributes (name, ip) to generate the final code.

The aim of the proposal presented in this paper is to extend this kind of languages, to the vertical value chain activities. There are several examples of vertical domain specific languages currently on development, thought they do not present a formalized domain analysis process. Some of them are the following:

- The real State Transaction Markup Language (RETML) [8].
- The Bank Internet Payment System (BIPS)[6] .
- The Business Rules Markup Language (BRML)[7] .

FROM A XML SPECIFICATION TO CODE: AN UNIX SHELL SCRIPT EXAMPLE

In order to shown the XML capabilities to build a system specification, and obtain code from it, this section describes a very simplified scenario. The aim of the example system is to maintain information stored on UNIX like operating systems */etc/inet/hosts* system file. For that, shell scripts automatically generated from XML specs will do the work.

The hosts file is a local archive file used in UNIX SVR4 operating systems to associate names of hosts with their corresponding IP addresses. This file has an

[5] http://www-sop.inria.fr/koala/kbml

[6] http://www.fsct.org/projects/bips

[7] http://www.research.ibm.com/rules

entry for each host IP address. Each of this entries is a line with the following format:

IP-address host-name optional-nick-names

In this example XML specifications are used to add or remove hosts lines. That is, the entries we want to add or remove from the hosts database are specified using a XML specification language. Then it is

```
<?xml
version="1.0"?>
<!DOCTYPE uiml PUBLIC "-//UIT//DTD UIML 2.0 Draft//EN"
"UIML2_0d.dtd">
 <uiml>   <interface name="hostsfile">
<structure>
       <part class="Frame" name="frame">
       <part class="Panel"  name="addhostForm">
        <part class="Label" name="title"/>
         <part class="Label" name="hosts"/>
         <part class="TextField" name="hostsField"/>
         <part class="Label" name="ipaddress"/>
         <part class="TextField" name="ipaddressField"/>
         <part class="Label" name="nick"/>
         <part class="TextField" name="nickField"/>
         <part class="Panel" name="buttonPanel">
         <part class="Button" name="addButton"/>
         <part class="Button" name="cancelButton"/>
         </part>
          </part>
     </part>
   </structure>
  <style>
<property part-name="title"
 name="text">Add Host Name</property>
  <property part-name="hosts" name="text">Host name:</property>
  <property part-name="hostsField" name="columns">25</property>
  <property part-name="ipaddress" name="text">Address:</property>
  <property part-name="ipaddressField"  name="columns">25</property>
  <property part-name="nick" name="text">Nick:</property>
  <property part-name="nickField" name="columns">25</property>
  <property part-name="addButton" name="label">AddHost</property>
  <property part-name="cancelButton"  name="label">Cancel</property>
  <property part-name="title" name="font">Serif-bold-16</property>
  <property part-name="addhostForm" name="layout">
                          java.awt.GridBagLayout</property>
  <property part-class="Label" name="anchor">WEST</property>
  <property part-class="TextField" name="anchor">WEST</property>
  <property part-class="Label" name="fill">HORIZONTAL</property>
  <property part-class="TextField" name="fill">HORIZONTAL</property>
  <property part-class="Label" name="gridwidth">1</property>
  <property part-class="TextField" name="gridwidth">1</property>
  <property part-name="title" name="anchor">NORTH</property>
  <property part-name="title" name="fill">NONE</property>
  <property part-name="title" name="gridwidth">REMAINDER</property>
  <property part-name="hostsField" name="gridwidth">REMAINDER</property>
  <property
part-name="ipaddressField" name="gridwidth">REMAINDER</property>
  <property part-name="nickField" name="gridwidth">REMAINDER</property>
   <property part-name="buttonPanel" name="anchor">SOUTH</property>
  <property part-name="buttonPanel" name="fill">HORIZONTAL</property>
  <property part-name="buttonPanel" name="insets">5,0,0,0</property>
  <property part-name="buttonPanel" name="gridwidth">4</property>
 </style>
  </interface>
 </uiml>
```

Figure 8: Interface UIML code. The first part of the specification defines the interface structure, and the second one the interface style properties

automatically generated a shell script that does the job.

The domain language used to specify the hosts database changes consists in tree XML tags.

> The <*add_host name= ip=*> tag specifies that a host entry has to be added. Hosts IP address and name are specified using the tag attributes *ip* and *name* respectively.
>
> The <*nick*> tag can be used inside a <add_host> hierarchy to specify several optional hosts nick names.
>
> The <*remove_host name= ip=*> tag is used to specify a hosts entry that has to be removed from the hosts database.

An example of a spec built using this tags is shown in fig 5. XMLScript is used to transform the specification to shell script code . XMLScript[8] is a scripting language designed specifically for XML transformation tasks.

The process of transforming the specifications into code is done using the XMLScript Xtrac transformer. Fig 6 shows the transformer rules that guide the transformation process from specification to shell script. Finally fig 7 shows the resulting shell script code.

Having seen the process of specification and generation of system operations. Now it is time to focus in how to specify the interface subsystem. For that propose the User Interface Markup Language (UIML) is used [27]. The whole specification of the interface is shown in fig. 8, and its final look in fig. [9].

Finally both interface and operations have to be linked in order to collaborate and behave as a cohesive system. For that it is necessary to use new XML-tags to encapsulate interface and process specification to conform a whole system (fig 10):

The <system_spec> tag encapsulates the whole system specification, formed from several subsystem specifications.

The <interface_subsystem> tag encapsulates the interface subsystem specification.

The <interconnection_mechanism> tag encapsulates the specification of the intercommunication subsystem.

From the system specification the parser will obtain the following three results:

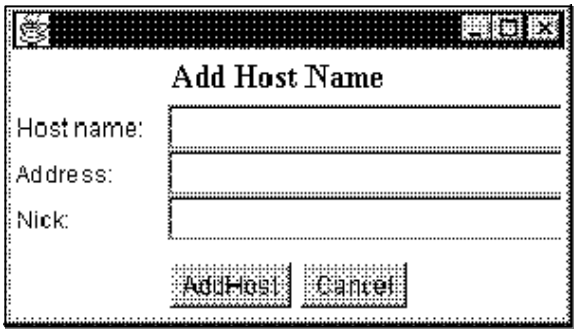

Figure 9: Interface generated with UIML transformer to Java code.

- A UIML specification that will be transformed with the UIML parser into Java code.
- A IMEC specification that can be void if the glue code is inserted into the subsystems
- A process specification, to be transformed using XMLScript into Unix shell script code.

The domain model of the example is shown in fig 11. The notation used is based in the Unified Modeling Language (UML) use-cases [26]. The model shows the main interactions between the user and the system. Associations between use-cases and actors will result in the DSL specification tags.

The interconnection mechanism handles two types of communication actions (see fig 12):

- Asynchronous events. All subsystems can tiger or receive events, and events can have parameters.
- Synchronous remote calls. All subsystems can call remote methods and receive calls in his published methods.

CONCLUSION AND RELATED WORK

Since XML is an emerging technology, continuous changes, new approaches, and tools are being pushed into the research community and the market. It is expected that upcoming related XML standards and tools like XQL, SML-Link, etc, will play a important role in the approach presented here, and will also made it became easier.

There are several important research efforts directed to obtain a way to describe software systems in a high abstraction level, as close to the user conceptual view as possible.

[8] XMLScript is a language developed by Decision Soft, writen in XML.

```
<system_spec>
<spec>
 <add_host name="mainserver"
ip="193.146.11.13">
 </add_host>
 <add_host name="pepe" ip="193.146.11.12">
    <nick>lucas</nick>
    <nick>jose</nick>
 </add_host> </spec>
<interface_subsystem>
<uiml_interface name="hostsfile">
   <form title="Add host Name">
   <textfield tag="Hostname" length=25 />
   <textfield tag="IpAdress" length=15 />
   <textfield tag="Nicknames" length=50 />
   <button tag="AddHost" />
   <button tag="Cancel" />
   </form>
   </uiml_interface>
</interface_subsystem>
</system_spec>
```

Figure 10. Wholesystem specification, it comprises three parts, process spec, interfacespecification and interconnection mechanism specification.

Application generators operate similarly as a compiler translating specifications into application programs. Although they can produce the whole system, usually they are used to create only a part of the system. Their main disadvantages are that they can only be used in few situations, and are difficult to build, because they require the previous design of specification languages, user interfaces, and generic units of software for the application domain [6].

The Eli language implementation system enables the implementation of domain specific languages from a high abstraction level. The tool generates an executable language processor for the domain language. The language processor translates DSL code into source code. Input specifications had to be parameterized with preprocessor switches and macros to select or deselect certain language features and supply user data [20]. Other approaches tend to use a general propose language to embed domain specific languages. Haskel has been used to build DSLs in several domains such as parser generation, VLSI design, graphical user interfaces, etc. [13].

InfoWiz is a common base language used to build domain specific languages, named jargons. They all share the same syntax, inherited from InfoWiz, and reflect the semantics of a specific domain. The base language is able to represent complex hierarchical information structures that can be composed across several domains, to introduce new domain specific terms, and to encode arbitrary data and operations in the domain. Nevertheless InfoWiz does not provides mechanisms to extend the language syntax. It is possible to compose specifications written with

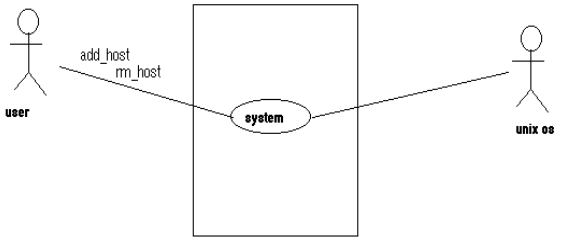

Figure 11: Domain model

different jargons, first because all have the same syntax, and second because the semantics of the application is not built inside the interpreter, but is provided using added modules. So each jargon has to provide the interpreter with an additional module able translate it [18].

Other approaches tend to build a dedicate translator for each domain specific language. As for example a domain abstract machine defined from the operations identified in the domain. The abstract machine is implemented as a set of highly parameterized software library [25]. Mawl is a domain specific language for programming form based web services using cgi programs. A Mawl specification can be compiled to a CGI executable or to a HTTP server. The Mawl translator can generate C++ and Standard ML. Main drawback of Mawl is that it loses user page browse history [2]. Apostle is a parallel event simulation language. Apostle specifications are translated to C++ code [5] .

Building software systems with a transformational approach, from a system specification in which each subsystem is independently specify using a DSL, could improve the software engineering process, facilitating the process of writing complete specifications, verifiable, and easily tested.

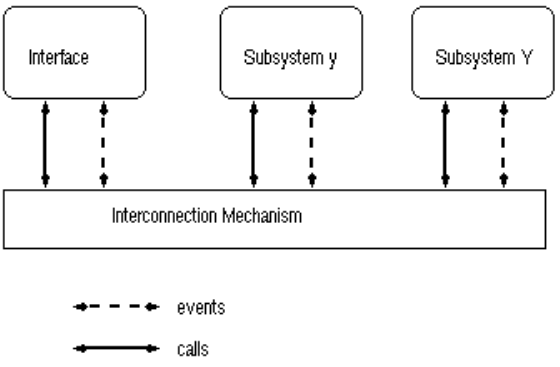

Figure 12: Interconnection mechanism

DSSL are more easy obtained than reusable components, and can be used in a more flexible way. First because the initial software engineering dual cycle, domain engineering, is reduced to just the domain analysis phase, avoiding domain design and domain implementation, to reduce use of resources. Second because each DSSL can be used to obtain system specifications in the same domain where it was obtained, or in any other, because of the relation that exists between DSSLs and the subsystems requirements that can be described with them.

REFERENCES

[1] Valeri N. Agafonov. Reuse of general specio/cation notions and specio/cation languages. In Proceedings of the Eighth Workshop on Institutionalizing Software Reuse, 1997.

[2] David L. Atkins, Thomas Ball, Glenn Bruns, and Kenneth Cox. Mawl: A Domain-Specific language for form-based services. IEEE Transactions on Software Engineering, 25(3):234-246, May 1999.

[3] D. Batory, V. Singhal, and M. Sirkin. Implementing a domain model for data structures. International Journal of Software Engineering & Knowledge Engineering, 2(3):375-402, 1992.

[4] Ted J. Biggerstaff. The library scaling problem and the limits of concrete component reuse. In Third International Conference on Reuse, 1994.

[5] David Bruce. What makes a good domain specific language? In Proceedings of First ACM SIGPLAN Workshop on Domain-Specific Languages DSL'97, pages 17-35, 1997.

[6] Craig Cleaveland. Building application generators. IEEE Software, (7):25-33, July 1988.

[7] Craig Cleaveland. Domain Analysis and Software Systems Modelling, chapter Building Application Generators, pages 9-33. IEEE Computer Society Press, 1991.

[8] Larry Colson. Nar mls/client data standards white paper. Technical Report RETS-V-1.0, Moore Data Management Services, 1999.

[9] K. Czarnecki. Leveraging reuse through Domain-Specific architectures. In Proceedings of the Eighth Workshop on Institutionalising Software Reuse, 1997.

[10] Bob DuCharme. XML: The Annotated Specification. Prentice Hall, 1999.

[11] David A. Epstein. Bean Markup Language: Using XML to dynamically construct, configure, and augment java. In XTech '99. XML Application Developers Conference and XIO Expo, 1999.

[12] Ken G. Holman. Introduction to XSLT(XSL Transformations). Crane Softwrights Ltd., 1999.

Antonion Domiguez is at the University of Vigo, Spain, J. M. Corchado is at the University of Salamanca, Spain.

[13] Paul Hudack. Building Domain-Specific embedded languages. ACM Computing Surveys, 28(4), December 1996.

[14] Samuel Kamin. Moving functional languages into real world. In Joint Brazilian/US Workshop on Formal Foundations of Software Systems, 1997.

[15] E.-A. Karlsson, editor. Software Reuse: A Holistic Approach. John Wiley & Sons, 1995.

[16] Charles W. Krueger. Software reuse. ACM Computing Surveys, 24(2):132-183, June 1992.

[17] Simon S. Laurent and Robert Biggar. Inside XML DTDs: Scientific and Technical. McGraw Hill, 1999.

[18] Lloyd Nakatani and Mark Jones. Jargons and infocentrism. In Proceedings of First ACM SIGPLAN Workshop on Domain-Specific Languages DSL'97, pages 59-74, 1997.

[19] James Neighbors. Software Construction Using Components. PhD thesis, University of California at Irvine, 1981.

[20] Peter Pfahler and Uwe Kastens. Language design and implementation by selection. In Proceedings of First ACM SIGPLAN Workshop on Domain-Specific Languages DSL'97, pages 97-108, 1997.

[21] Michael E. Porter. Competitive Advantage: Creating and Sustaining Superior Performance. Free Press, 1998.

[22] M. Simos. The domain-oriented software life cycle: Towards an extended process model for reusability. In Proceedings of the Workshop on Software Reuse, Boulder, CO, October 1987.

[23] Yellamraju V. Srinivas. What is a domain? Technical Report ASE-RTP-102, University of California, Irvine, Department of Information and Computer Science, October 1988.

[24] Yellamraju V. Srinivas. Domain Analysis and Software Systems Modelling, chapter Algebraic specification for Domains, pages 90-119. IEEE Computer Society Press, 1991.

[25] Scott A. Thibault, Renaud Marlet, and Charles Consel. Domain-Sspecific Languages: From design to implementation. Application to video device drivers generation. IEEE Transactions on Software Engineering, 25(3):363-377, March 1999.

[26] UML. UML semantics. Technical Report ad/97-08-04, Rational Software Corporation, 1997.

[27] Universal Interface Technologies, Inc. UIML Java Rendering Engine Manual, 2000.

[28] Arie van Deursen and Paul Klint. Little languages, little maintenance? In Proceedings of First ACM SIGPLAN Workshop on Domain-Specific Languages DSL'97, pages 109-127, 1997.