

- During the semantic composition stage, the engine calculates an admissible evaluation order on the hypergraph and proceeds to apply the composition functions in this order.
- Finally, the engine delegates in the evaluation procedure Ev the evaluation of the tree semantic value.

Obviously, such behavior for a generic engine is only conceptual. An implementation can be free to carry out an observationally equivalent, yet more efficient, processing.

A particular processor can be now considered as the addition of a set of markup interpreters $\langle \text{Leo}, \text{Peo}, \text{Ceo} \rangle \dots \langle \text{Len}, \text{Pen}, \text{Cen} \rangle$ to a generic engine E . Because new interpreters can be added, the processor can be extended to deal with other markup structures. This leads to *extensible processors*.

For instance, for the shape markup language, we can begin by providing two interpreters $\text{IRectangle} = \langle \text{LRectangle}, \text{PRectangle}, \text{CRectangle} \rangle$ and $\text{IUnion} = \langle \text{LUnion}, \text{PUnion}, \text{CUnion} \rangle$ to obtain a processor observationally equivalent to that sketched in Figure 2. Then, we can extend this processor with a new interpreter $\text{IIntersection} = \langle \text{LIntersection}, \text{PIntersection}, \text{CIntersection} \rangle$ to obtain a processor equivalent to those sketched in Figures 3 and 4.

What happens if the new interpreter is not compatible with the existing ones? The idea is to apply adaptors to the existing interpreters, the new incorporated interpreter, or both. An *adaptor* is a procedure for generating interpreters from interpreters. In addition, because interpreters are structured in different procedures, adaptors can be structured accordingly into procedures for adapting those procedures.

For instance, when the processor induced by the interpreters $\{ \text{IRectangle}, \text{IUnion}, \text{IIntersection} \}$ must be extended with $\text{IScale} = \langle \text{LScale}, \text{PScale}, \text{CScale} \rangle$, some adaptation is required, because the scale factor must now be propagated by the composition functions of IUnion and IIntersection , and it must be used by IRectangle . In this way, if we suppose CScale is defined as:

```
CScale(pars,sems) =  $\lambda$ scale(  
  1. get the scale factor f from pars  
  2. get the semantic value s associated with the child element  
  from sem  
  3. apply s to f*scale  
)
```

(i.e., as a function that, for each parameter set and for each semantic assignment, gives a function that accepts a scale parameter and behaves as described), and CUnion is defined as:

```
CUnion(pars,sems) = (  
  1. get from sems the list of semantics ls for the children  
  elements.  
  2. make a Union with ls.  
)
```

we need to transform this function with:

```
C'Union(pars,sems) =  $\lambda$ scale (  
  1. get from sems the list of semantics ls for the children  
  elements.  
  2. make a new list ls' evaluating each element in ls with scale  
  3. make a Union with ls'.  
)
```

This can be done by applying an adaptor in the form:

```
AddScale(c) =
```



```

</Links>
</LinkSpec>

```

The description of process specifications is performed in terms of the following language:

```

<!ELEMENT ProcSpec (Adaptors?,Interpreters,Namespaces)>
<!ATTLIST ProcSpec evaluator CDATA #IMPLIED>
<!ELEMENT Adaptors (Adaptor|Composition)+>
<!ELEMENT Adaptor ((LinkerAdaptor | LinkSpec)?,CompositorAdaptor?)>
<!ATTLIST Adaptor id ID #REQUIRED>
<!ELEMENT LinkerAdaptor EMPTY>
<!ATTLIST LinkerAdaptor ref CDATA #REQUIRED>
<!ELEMENT CompositorAdaptor EMPTY>
<!ATTLIST CompositorAdaptor ref CDATA #REQUIRED>
<!ELEMENT Composition EMPTY>
<!ATTLIST Composition id ID #REQUIRED
                    a1 IDREF #REQUIRED
                    a2 IDREF #REQUIRED>
<!ELEMENT Interpreters (Interpreter)+>
<!ELEMENT Interpreter ((Linker |LinkSpec)?,Compositor)>
<!ATTLIST Interpreter id ID #REQUIRED
                    adaptor IDREF #IMPLIED>
<!ELEMENT Linker EMPTY>
<!ATTLIST Linker ref CDATA #IMPLIED>
<!ELEMENT Compositor EMPTY>
<!ATTLIST Compositor ref CDATA #REQUIRED>
<!ELEMENT Namespaces (Namespace)+>
<!ELEMENT Namespace (Bind)+>
<!ATTLIST Namespace ns CDATA #REQUIRED
                    default IDREF #IMPLIED>
<!ELEMENT Bind EMPTY>
<!ATTLIST Bind tag CDATA #REQUIRED
                    interpreter IDREF #REQUIRED>

```

This language allows declaration of the adaptors and interpreters to be used, and the association of interpreters with element types.

Each required adaptor is introduced using an `Adaptor` element. The adaptor can introduce a linker adaptor and a compositor adaptor. If one of these two adaptors is not present, the corresponding identity adaptor is taken instead. For linker adaptors, an explicit description can be given in terms of the linker description language (here, a *DeclarativeLinkerAdaptor* is used); alternatively, the Java class implementing the adaptor to be used can be referenced (using the `ref` attribute of the `LinkerAdaptor` element). For compositor adaptors, the name of the Java class implementing it must be explicitly referenced. Finally, using a `Composition` element, it is possible to create an interpreter adaptor composing other two adaptors. For the shape markup language, a possible description of the adaptors to be used is:

```

<Adaptors>
  <Adaptor id="addScaleA">
    <CompositorAdaptor ref="AddScaleAdaptor"/>
  </Adaptor>
  <Adaptor id="rectangleA">
    <CompositorAdaptor ref="RectangleAdaptor"/>
  </Adaptor>
</Adaptors>

```

Each interpreter to be used is introduced by an `Interpreter` element. Here, an adaptor to be applied to the interpreter can be referenced (using the `adaptor` attribute). As with linker adaptors, linkers can be described either in terms of the linker description language, or by referencing an external Java implementation. If a linker is not specified, the default linker is used instead. In turn, the `Compositor` element allows reference (using the `ref` attribute) to the Java class implementing the compositor. For instance, the declaration for the interpreter to be associated with the `Ref` element type can be:

Our assembler adaptors are similar to the so-called *mixins* in the object-oriented paradigm [Bracha 1990]. The informal description of a mixin is a class that is parametric in its superclass (i.e., which *super* variable can be dynamically changed). “A Mixin-Based Semantic-Based Approach to Reusing Domain-Specific Programming Languages” [Duggan 2000] describes another mixin-based Java framework for the development of modular interpreters. There, mixins represents interpreters that can be assembled together in a mixin chain. Each interpreter maintains its own state and encapsulates inner classes for the construction of abstract syntax trees made from expressions. These expressions can be evaluated into computations (the equivalent to our semantic values). The chaining of mixins leads to an associated chaining of states. To recover the appropriate state in each interpretation step, the framework enforces the writing of computations using a monadic style. However, the framework does not includes an equivalent to monad transformers. Instead, the monadic operations must be explicitly defined in each mixin to recover the appropriate state from the state chain.

There are several works describing the application of syntax-directed translation techniques to markup languages. Kuikka and Penttonen have described a method to perform document transformation by automatically deriving a *translation schema* from a source to a result document grammar [Kuikka 1993]. In “SIMON: A Grammar-based Transformation System for Structured Documents” [Feng 1993], document transformations are specified using *higher order attribute grammars* [Vogt 1989]. F. Neven [Neven 1999] has shown how to apply the attribute grammar formalism to extended BNF grammars (and, hence, to document grammars) oriented to be used as document query mechanisms. In “Adding Semantics to XML” [Psaila 1999], a method to associate semantic functions with documents (following the attribute grammar paradigm) is described. While these approaches perceive the syntax-directed translation engine as the *processor*, our approach conceives this engine as a way to *generate the actual processor* by assembling smaller processors, according to the *analyze eval* spirit.

Note that the *analyze eval* approach adopted here can also be identified in the processing model underlying publication approaches such as XSL [W3C 2001] (and its predecessor DSSSL [ISO 1996]). In effect, in XSL, documents are transformed into a markup language of formatting objects. The resulting specification must be subsequently evaluated to generate the presentation. The first phase corresponds with an analysis phase, while the second one is a particular type of evaluation phase.

The work described here relies on our previous work on the DTC [structured Documents, document Transformations and software Components] approach [Sierra 2000a] [Sierra 2000b] [Sierra 2001]. DTC is an approach to the development of applications that are described using domain-specific languages [van Deursen 2000]. We use XML to define domain-specific languages describing the contents managed by the application. Then, we transform these contents to languages supported by pre-existing software components. Hence, software components are interpreters for markup languages that must be combined to obtain more complex interpreters able to give operational support to the desired application domain. We have applied this approach both to the domain of route searching in transport networks [Sierra 2000a] and the domain of educational hypermedia applications [Navarro 2000]. Work described here substantially improves the previously reported research. In other studies [Sierra 2000a] [Sierra 2000b], we adopted a syntactic-based approach and focused on component composition instead of language composition. In this way, components might be reorganized for different applications in the same application domain. Later, we adopted a semantic-based approach and focused on the composition of languages [Sierra 2001]. Components were automatically assembled by interpreting documents trees. Components carried out both the analysis and the evaluation phase. In addition, we did not include any special adaptation mechanism in our framework. Adoption of the *analyze eval* approach described in this paper raises the true linguistic nature of DTC and alleviates many of its previously encountered shortcomings.

- [**Feeley 1987**] Feeley, M., Lapalme, G.: "Using Clousures for Code Generation". Journal of Computer Languages 12(1). 1987
- [**Feng 1993**] Feng, A., Wakayama, A.: "SIMON: A Grammar-based Transformation System for Structured Documents". Electronic Publishing. 6(4). 1993
- [**Friedman 2001**] Friedman, D. P., Wand, M., Haynes, C. T.: "Essentials of Programming Languages (Second Edition)". MIT Press/McGraw-Hill. 2001
- [**Goldfard 1990**] Goldfard, C. F.: "The SGML Handbook". Oxford University Press. 1990
- [**ISO 1986**] ISO. "Standard Generalized Markup Language (SGML). ISO/IEC IS 8879". International Standards Organization. 1986
- [**ISO 1996**] ISO. "Document Style Semantics and Specification Language (ISO/IEC 10179)". International Standards Organization. 1996
- [**Kuikka 1993**] Kuikka, E., Penttonen, M.: "Transformation of Structured Documents with the Use of Grammars". Electronic Publishing. 6(4). 1993
- [**Liang 1995**] Liang, S., Hudak, P., Jones, M.: "Monad Transformers and Modular Interpreters". 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. 1995
- [**Maruyama 1999**] Maruyama, H., Tamura, K., Uramoto, N.: "XML and Java: Developing Web Applications". Addison-Wesley. 1999
- [**Navarro 2000**] Navarro, A., Sierra, J. L., Fernández-Manjón, B., Fernández-Valmayor, A.: "XML-Based Integration of Hypermedia Desing and Component-Based Techniques in the Production of Educational Applications". In "Ortega, M., Bravo, J. (eds). Computers and Education in the 21st Century". Kluwer. 2000.
- [**Neven 1999**] Neven, F.: "Extensions of Attribute Grammars for Structured Document Queries". 7th International Workshop on Database Programming Languages. Kinloch Rannoch - Scotland. December 1 - 3. 1999
- [**Nilsson 1980**] Nilsson, N. J.: "Principles of Artificial Intelligence". Tioga Publishing Company, Palo Alto, CA. 1980
- [**Ousterhout 1990**] Ousterhout, J. K. "TCL: An Embeddable Command Language". Proceedings of the USENIX Association Winter Conference. 1990
- [**Psaila 1999**] Psaila, G., Crespi-Reghezzi, S.: "Adding Semantics to XML". Second Workshop on Attribute Grammars and their Applications. WAGA'99. Amsterdam. The Netherlands. March 26. 1999
- [**Rees 1982**] Rees, J., Adams, N. I. I.: "T: A dialect or LISP or, lambda: The Ultimate Software Tool". Conference Record of the 1982 ACM Symposium on LISP and Functional Programming. 1982.
- [**Sierra 2000a**] 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.
- [**Sierra 2000b**] Sierra, J. L., Fernández-Manjón, B., Fernández-Valmayor, A., Navarro, A.: "Developing Applications with XML Documents, Document Transformations and Software Components". 10th International Conference on Computing and Information ICCI-2000. Kuwait. November 18-21. 2000.

- [**Sierra 2001**] 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.
- [**Steele 1994**] Steele, G.: "Building Interpreters by Composing Monads". 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. Portland, Oregon. January 17-21. 1994
- [**Stoy 1977**] Stoy, J. E.: "Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory". The MIT Press. 1977
- [**van Deursen 2000**] van Deursen, A., Klint, P., Visser, J.: "Domain-Specific Languages: An Annotated Bibliography". ACM SIGPLAN Notices. 35(6). 2000.
- [**Vogt 1989**] Vogt, H. H., Swierstra, S. D., Kuiper, M. F. "Higher-Order Attribute Grammars". Proceedings of the ACM SIGPLAN'89 Conference on Programming Language Design and Implementation. 1989
- [**W3C 1999a**] W3C. "XML Path Language (XPath) Version 1.0 (W3C Recommendation)". World Wide Web Consortium. 1999
- [**W3C 1999b**] W3C. "Namespaces in XML (W3C Recommendation)". World Wide Web Consortium. 1999
- [**W3C 2000a**] W3C. "Document Object Model Level 2 (W3C Recommendation)". World Wide Web Consortium. 2000a
- [**W3C 2000b**] W3C. "Extensible Markup Language (XML) Second Edition (W3C Recommendation)". World Wide Web Consortium. 2000b
- [**W3C 2001**] W3C. "Extensible Stylesheet Language (XSL) (W3C Recommendation)". World Wide Web Consortium. 2001
- [**Wadler 1993**] Wadler, P. "Monads and Functional Programming". Proceedings of the 1992 Marktoberdorf International Summer School in Program Design Calculi. Springer Verlag. 1993

The Authors

José Luis Sierra

Dpto. Sistemas Informáticos y Programación, Universidad Complutense
Madrid
Spain

jlsierra@sip.ucm.es

Mr. José Luis Sierra is a Computer Science assistant professor at UCM. He is preparing a Ph.D. dissertation on the use of domain-specific markup languages in the development of applications.

Baltasar Fernández-Manjón

Dpto. Sistemas Informáticos y Programación, Universidad Complutense
Madrid
Spain

balta@sip.ucm.es

Dr. Baltasar Fernández-Manjón is a Computer Science professor at UCM. His research interests lie in the educational uses of computers, markup languages, and user modelling.

Alfredo Fernández-Valmayor

Dpto. Sistemas Informáticos y Programación, Universidad Complutense

Madrid

Spain

alfredo@sip.ucm.es

Dr. Alfredo Fernández-Valmayor is a Computer Science professor at UCM, and his research interests include markup languages and its application to hypermedia and educational systems construction.

Antonio Navarro

Dpto. Sistemas Informáticos y Programación, Universidad Complutense

Madrid

Spain

anavarro@sip.ucm.es

Dr. Antonio Navarro is a Computer Science assistant professor at UCM. His research interests include markup languages, hypermedia, and software engineering.

Extreme Markup Languages 2002

Montréal, Québec, August 6-9, 2002

This paper was formatted from XML source via XSL

Mulberry Technologies, Inc., August 2002