Towards integration of XML in the Creol object-oriented language

Arild Torjusen, Olaf Owe, Gerardo Schneider
Department of Informatics, University of Oslo - Oslo, Norway
{aribraat,olaf,gerardo}@ifi.uio.no

Abstract

Creol is a high level object-oriented modeling language for distributed systems. In this paper we propose an extension to Creol for handling XML documents.

1 Introduction

XML (eXtensible Markup Language) [2] is a flexible and generic format for structured data aimed at being shared on the World Wide Web and intranets. The need for XML documents as first-class citizens is acknowledged by academic as well as by business-oriented communities [7].

XML *documents* are ordered labeled tree structures containing *markup* symbols describing their content. The document structure is described by a document type—or *schema*—written in a schema language. The integration of XML on current object-oriented languages is far from trivial. The initial approach has been to treat XML through APIs which use strings for representing literals. One problem with this approach is that it limits the use of static checking tools. Furthermore, the representation of programs as text involves potential security risks. See [7] for a more detailed description of the main problems arising with the integration of XML in object-oriented languages.

Creol Our research project concerns integration of XML into the object-oriented language Creol [6]. The main features of Creol are: It supports object-oriented classes and subclasses, as well as user defined data types and functions. This gives flexibility in our choices when representing XML. Creol is oriented towards open distributed systems, with support for concurrency and asynchronous communication. This gives an interesting setting for exploring processing and sharing of XML documents. Creol is strongly typed and has a formal operational semantics defined in rewriting logic with a small kernel consisting of only 11 rewrite rules. This makes it easy to extend the language and to formalize the extensions by reuse of the operational semantics. The interpreter for Creol written in Maude [3] provides a useful framework for implementation and testing.

Our Agenda In order to integrate XML documents in Creol, we intend to address the following issues: (1) Parsing and well-formedness checking of XML documents; (2) internal representation of XML in Creol with preservation of Creol static type-safety; (3) validity-checking of XML data-structures against schemas; (4) queries on XML documents; and finally (5) more complex transformations on XML documents.

This paper gives a sketch of the first steps towards integration of XML in Creol. We give a representation of XML in Creol and address the issue of validity-checking.

This paper was presented at the NIK-2007 conference; see http://www.nik.no/.

Validation can be done either by functions defined "on top" of the existing type system or by enhancing the Creol type system with *regular expression types* [5]. The former approach is taken here. A more detailed presentation of our work is available in the research report [9], which also contains a larger example and a survey of related work.

In the next section we show how XML documents are integrated in Creol. In Section 3 we show how schemas are represented in Creol. Section 4 is concerned with the validation of XML documents. In Section 5 we conclude and present further work.

2 A model for XML in Creol

The data model defined in the XPath 1.0 Recommendation [10] is the basis for canonical XML which we will take as the point of departure for the internal representation of XML in Creol. XPath models an XML document as an ordered tree containing nodes of seven different types. We will focus on the following four node types in our model: A *root* node represents the root of the XML tree; an *element* node has a name (corresponding to the XML tag for the element) and may have as its children nodes of other kinds and associated sets of attribute and namespace nodes; *text* nodes represent character data; and *attribute* nodes contain name/value pairs for attributes. The three remaining node types: *namespace*, *processing instructions*, and *comment* nodes are left out from the model for now since they are less relevant to demonstrating integration of XML in Creol. Leaving these out also simplifies the definition of element nodes and allows us to represent a root node with an element node (cf. [9]).

Since the Creol operational semantics is executable in Maude, we accommodate XML by extending the operational semantics with Maude sorts (type names) for XML names, element, text, and attribute nodes, as well as a sort for XML documents and a common supersort `ContentNd` for nodes that can occur as children of an element node (i.e. element and text nodes):

```
sorts   XMLName ElemNd TextNd AttNd ContentNd XMLDoc  .
subsort ElemNd TextNd < ContentNd .
```

We add the following constructors:

```
op (_=_)   : XMLName String                  -> AttNd  [ctor] .
op tx      : String                          -> TextNd [ctor] .
op _(_)[_] : XMLName AttNdList ContentNdList -> ElemNd [ctor] .
op _[_]    : XMLName ContentNdList           -> ElemNd [ctor] .
op xmlDoc  : ElemNd XMLSchema               -> XMLDoc [ctor] .
```

Example The XML fragment: `<rcp addr="vera@foo.com">Vera</rcp>` has the Maude syntax: `"rcp"("addr"="vera@foo.com")[tx("Vera")].`

3 Schemas and type checking

There are several generally adopted XML schema languages with different expressive power [8]. The DTD language is sufficiently expressive for our purpose of demonstrating XML integration in Creol and hence we adapt its restrictions to achieve simple validation (i.e. only deterministic regular expressions are allowed in the definition of an element).

The schema type for Creol A DTD is a list of markup declarations i.e. either declarations of *element type*, *attribute-list*, *entity*, or of *notation*. We only consider declarations of element type here. The `XMLSchema` constructor is:

```
op xmlSchema : XMLName ElemDeclList  -> XMLSchema [ctor] .
```

with an additional operator `noSchema` for use in XML documents with no XML Schema.

Element type declarations consist of a name and a specification of the legal content. There are four kinds of specification: either one of the keywords “EMPTY” or “ANY”, or the specification of a *content model*, or a *mixed-content declaration* of the form: $(\texttt{\#PCDATA} \mid e_1 \mid e_2 \mid \ldots \mid e_n) *$ where each e_i is an element name.

A content model is a context free grammar governing the allowed types of the child elements and the order in which they are allowed to appear. We model the content models as *regular expressions* over the alphabet Σ containing element names and the reserved name `PCDATA`. By including `PCDATA` in Σ, a mixed-content declaration may be modeled as a content model specification. The set of regular expressions over Σ^* is obtained using concatenation (`@`), union (`|`), and star as well as ’?’ and ’+’ with their standard interpretation:

```
op elDecl      : XMLName ContentModel   -> ElemDecl [ctor] .
ops empty any  :                        -> ContentModel  [ctor]  .
op elCt        : RegExp                 -> ContentModel [ctor] .
op PCDATA      :                        -> RegExp  [ctor] .
ops _? _* _+   : RegExp                 -> RegExp [ctor prec 40 ] .
op (_@_)       : RegExp RegExp          -> RegExp [ctor assoc prec 42 ]
op _|_         : RegExp RegExp          -> RegExp [ctor prec 44 ]
```

Example The DTD fragment:

```
<!DOCTYPE email [<!ELEMENT email (head, body, foot*)>
<!ELEMENT head (sender, rcp, subject?)>...]>
```

has the Maude syntax:

```
xmlSchema("email", (elemDecl("email",elCt("head"@"body"@("foot"*)))
elemDecl("head",elCt("sender"@"rcp"@("subject"?)))...)) .
```

4 Validating XML in Creol

Well-formedness of any value of type `XMLDoc` is ensured by Maude type checking. The XML specification defines an XML document to be *valid* “if it has an associated document type declaration and if the document complies with the constraints expressed in it” [2].

Validation of an XML document is performed by the function `validate : XMLDoc` $\rightarrow$ `ValResult`. A `ValResult` is a boolean/string pair where the boolean value indicates validity and the string contains an error message or a record of the processing. `validate` checks whether there is a schema with a name matching the document root node associated with the document, in which case the recursive function `val` is called, otherwise validation ends with a negative result. The function `val : ContentNdList ElemDeclList` $\rightarrow$ `ValResult` validates a content node list against the element declaration list defined by the schema by retrieving for each node the corresponding declared `ContentModel` and calling the function `check : ContentNd ContentModel ElemDeclList` $\rightarrow$ `ValResult`. If there is no `ContentModel` for some node the document is invalid. Note also that according to [2] an element type must not be declared more than once so uniqueness of declarations may be assumed.

For a `ContentNd` to be valid relative to a `ContentModel` we need to consider three cases: The `ContentModel` is `empty` and the element should have no content or the `ContentModel` is `any` and the element can consist of any sequence of (declared) elements intermixed with character data. These two cases are easy to check. The third case is where the `ContentModel` specifies a regular expression and it is checked by matching the list of names of children elements of the node against the regular expression specified in the `ContentModel` and in addition calling `val` on the list of children elements. Matching

of a list of element names against a regular expression is implemented by constructing a deterministic finite automaton from the regular expression and test whether the automaton accepts the string corresponding to the list of names.

5 Conclusion

Integrating XML documents in object-oriented languages is not easy in general as witnessed by the extensive research conducted in this area, and nicely presented in the survey [7]. We have outlined how to integrate XML documents into Creol, an object-oriented language with formal semantics in rewriting logic. We have also presented an algorithm for validating XML documents against XML schemas, to show that the former are instances of the latter.

This paper is a first step towards a full integration of XML into Creol, and we intend to pursue our work as to complete our agenda described in Section 1. In particular, we find it interesting to be able to manipulate and reason about XML documents, and to enhance the Creol type system with *regular expression types* and to adapt the semantic sub-typing algorithm from the related work on CDuce [1] and XDuce [4].

References

[1] V. Benzaken, G. Castagna, and A. Frisch. CDuce: an XML-centric general-purpose language. *SIGPLAN Not.*, 38(9):51–63, 2003.

[2] T. Bray, J.Paoli, C. Sperberg-McQueen, E. Maler, and F. Yergeau. *Extensible Markup Language (XML) 1.0*, third edition, February 2004. http://www.w3.org/TR/REC-xml/.

[3] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. F. Quesada. Maude: Specification and programming in rewriting logic. *Theoretical Comput. Sci.*, 285:187–243, Aug. 2002.

[4] H. Hosoya and B. C. Pierce. XDuce: A statically typed XML processing language. *ACM Trans. Inter. Tech.*, 3(2):117–148, 2003.

[5] H. Hosoya, J. Vouillon, and B. C. Pierce. Regular expression types for XML. *ACM SIGPLAN Notices*, 35(9):11–22, 2000.

[6] E. B. Johnsen and O. Owe. An asynchronous communication model for distributed concurrent objects. *Software and Systems Modeling*, 6(1):35–58, Mar. 2007.

[7] E. Meijer, W. Schulte, and G. Bierman. Programming with circles, triangles and rectangles. In *Proceedings of the XML Conference*, 2003.

[8] M. Murata, D. Lee, and M. Mani. Taxonomy of XML schema languages using formal language theory. In *Extreme Markup Languages*, Montreal, Canada, 2001.

[9] A. Torjusen, O. Owe, and G. Schneider. Towards integration of XML in the Creol object-oriented language. Res. Rep. 365, Dept. of Informatics, Univ. of Oslo, Oct. 2007.

[10] W3C (World Wide Web Consortium). *XML Path Language (XPath) Version 1.0*, 1999. W3C Recommendation 16 November 1999. http://www.w3.org/TR/1999/REC-xpath-19991116.