CSP-in-XML

CENG 599 TERM PROJECT REPORT

BY

KURTCEBE EROGLU

Supervisor: Asst. Prof. Dr. HALİT OĞUZTÜZÜN

Department of Computer Engineering
MIDDLE EAST TECHNICAL UNIVERSITY

FEBRUARY 2004

CSP-in-XML

Kurtcebe Eroğlu

Middle East Technical University, Computer Engineering Department, 06531, Ankara
e1272475@ceng.metu.edu.tr

Abstract: A notation based on 'Communicating Sequential Processes' (CSP) is used to specify behavior and coordination of components in architectural description language (ADL) Wright. Extending previous study on XML Schema for representing grammar of structural part of Wright descriptions, an XML Schema for grammar of CSP in these descriptions is developed. Because of heavy use of mathematical expressions in CSP, XML schema is augmented with MathML, a product of the W3C Math working group. As a sample application, a validating parser is developed which transforms Wright descriptions in XML to a Wright 'Configuration Behavior', a single CSP expression designating behavior of entire configuration, in a format accepted by a CSP analyzer for further analysis.

1 Introduction

Increasing size and complexity of the contemporary software systems cause the design of overall system structure to become main focus of software development process. To quote Len Bass et al. "a software architecture is the development work product that gives the highest return on investment with respect to quality, schedule and cost."

WRIGHT, being the Architectural Description Language (ADL) used in this study, utilizes explicit, independent connector types as interaction patterns, the ability to describe the abstract behavior of components using a CSP-like notation. This paper heavily depends on WRIGHT description by [allen]

XML is chosen to represent WRIGHT descriptions because of a wide range of known advantages. XML documents are easy to be investigated by humans, access to a large and growing community of tools, XML is license-free, prevents vendor dependencies, and is platform independent. Beyond XML, "the XML Family" is a growing set of modules that offer useful services to accomplish important and frequently demanded tasks. During development of XML schema, Schema, XSLT, XPath queries and MathML, a low level specification for describing mathematics as a basis for machine to machine communication.

The rest of the paper is organized as follows. Section 2 gives information about the topics on which the project depends such as CSP, extended form of WRIGHT, XML, MathML. Section 3 gives information about the XML Schema developed. Section 4 introduces the sample tool developed, and gives an examination of the XML schema developed. Section 5 is the conclusion. Remaining information such as samples, details of schema etc. that would be suitable to present at appendices are omitted in order to prevent report from getting too long. These information is provided at project website.

2 Background

2.1 CSP

CSP is a notation for concurrency based on synchronous message passing and selective communications designed by Anthony Hoare in 1978. CSP builds on the basic concept of *process* as a mathematical abstraction of the interactions between a system and its environment. Then operations to assemble the processes, concurrency, nondeterminism, and communication are added to this formalism [hoare].

In context of WRIGHT descriptions, a modified version of CSP is used. Differences between original and modified formalisms and reasons are discussed in [allen A.4.]. Additionally a notation is added to original CSP to distinguish between *initiating* an event and *observing* an event. Also being integral parts of a WRIGHT description, definition of CSP expressions must be able to utilize parameterization, implementing interface types and extending styles.

Interface Type DataInput = *[read data repeatedly, closing the port at or before end-of-data]*
Interface Type DataOutput = *[write data repeatedly, closing the port to signal end-of-data]*

Component SplitFilter(nout:1..)
Port Input = DataInput
Port $Output_{1..nout}$ = DataOutput
Computation = *[read from* Input *repeatedly, writing to* $Output_1$, $Output_2$, … , $Output_{nout}$ *in succession]*

Figure 1- Example of interface implementation and parameterization

These aspects of a WRIGHT description force the definition of CSP expression to inherit from template CSP expressions in interfaces and styles, transform these using renaming operations where necessary. Also all parameters defined in the surrounding WRIGHT description may as well be used in the CSP expression itself (even CSP expression itself may be parameter). As a result, it is evident that an XML schema designed for CSP expressions in WRIGHT descriptions must extend from the schema of WRIGHT descriptions itself.

In order to express complex behaviors, *naming, state* and *alternatives* must also be supported.

Naming:	$P = e \rightarrow P, \quad f \rightarrow P$ **where** $e \rightarrow p$
State:	P_i ***where*** $P_i = \overline{count!i} \rightarrow P_{i+1}$
Alternatives:	P_i ***where*** $P_i = \begin{cases} \overline{count!i} \rightarrow P_{i+1} \ when \ i < 3 \\ \overline{count!i \rightarrow P_i} \quad otherwise \end{cases}$

Figure 2- Example of naming, state and alternatives in CSP expressions

Thus, it is obvious that augmented with many additional facilities, the expressions defined are more complex in representation than original CSP expressions. So facilities required for above properties and operations of prefixing, internal and external choice, and interleaving are supported in XML Schema developed for defining CSP expressions in WRIGHT descriptions.

2.1 Extended WRIGHT

The developed XML schema has also facilities to use security annotations of extended WRIGHT description as described in [ulu],[yolacan]. Events initiating communication may be left as they are to indicate that default security label for the collaborating port is used, or these events may be given fixed security labels. Necessary features are present in the schema developed.

Sample tool developed generates an output given a description in extended WRIGHT with security annotations. The output of the tool is the 'configuration behavior' formally described in [allen 3.5]. Output of tool is simply a single CSP expression, which is fed in to a CSP analyzer together with a lattice, list of ports in the configuration and a mapping between ports and security labels. This single CSP expression is actually a combination of the behavior specifications of each instance of an architectural element in the system via parallel composition, as discussed in [allen 3.5], that is, one process for each component instance and one for each connector instance are combined. Example given in [allen] will be given below to give further explanation for this process.

Configuration ABC

Component A-type

Port Out = $\overline{a} \rightarrow Out \Pi \xi$

Computation = $\overline{Out.a} \rightarrow Computation \Pi \xi$

Component B-type

Port In = $c \rightarrow In[] \xi$

Computation = $In.c \rightarrow \overline{b} \rightarrow Computation[] \xi$

Connector C-type

Role Origin = $\overline{a} \rightarrow origin \Pi \xi$

Role Target = $c \rightarrow T \arg et[] \xi$

Glue = $Origin.a \rightarrow \overline{T \arg et.c} \rightarrow Glue[] \xi$

Instances

A: A-type
B: B-type
C: C-type

Attachments

A.out as C.origin
B.in as C.Target

Resulting Configuration Behavior

$A = \overline{A.Out.a} \rightarrow \Pi \xi$

$\| C = A.Out.a \rightarrow \overline{B.In.c} \rightarrow C[] \xi$

$\| B = B.In.c \rightarrow \overline{B.b} \rightarrow B[] \xi$

Figure 3- A simple configuration example by [allen]

Actually, the sample tool experiments with the XML schema developed by implementing the two transformations explained in [allen 3.5.3];

- *Conversion of local names to global names*: Each event is given a three level name: component name, port name and local event name (N.p.e). For example if we have two instances of A-type above, A1,A2: A-type, we would call event a in port definition A1.out.a. This way, naming clashes between different instances are avoided.

- *Matching up names of attached* ports *and* role: In the above mentioned example, *glue* definition of C-type connector instance C does not use the role names Origin and Target, but uses the names of the actual instance ports they are connected to (as defined in attachments section of *configuration*. If we did not make this transformation, *glue* would refer to an event with a role name and computation would refer to it by port name, resulting in the fact that we would not know that they are same events and miss the semantics of interactions between the instances of components.

The above-mentioned discussion also implies that 'configuration behavior' expression represents much more than independent CSP expression fragments defined within the components, and so is the real subject for analysis.

2.2 XML and MathML

Being a mature technology by now, XML by itself needs little introduction. However a brief introduction will be given for other technologies from the "XML family", that are used in this study.

Given excessive use of mathematics in CSP expressions, especially when used in context of WRIGHT descriptions, it is evident that embedding even the simplest mathematical formalism to the developed XML schema for CSP is not a simple task. Here a technology from XML family may prove handy; MathML, a W3C Recommendation released on February 2001 and work of W3C Math working group, is a standard for representing mathematical objects with their semantics, allowing them to be exchanged between computer programs, or published in worldwide web. Many implementations of MathML are available (browsers, authoring tools), many of which are open source software and can be reached from MathML homepage. Developing tools complying to this standard, automatically incorporates these effort to the tool.

In MathML, there are two styles of encoding; content and presentation. Content encoding tries to communicate the sense and meaning, the semantics. Presentation encoding tries to convey notation, usually for presentation to a human. MathML allows an author to use either kind of encoding, or mix them in a hybrid. The XML schema developed uses content encoding schema of MathML as an embedded module, using schema inclusion.

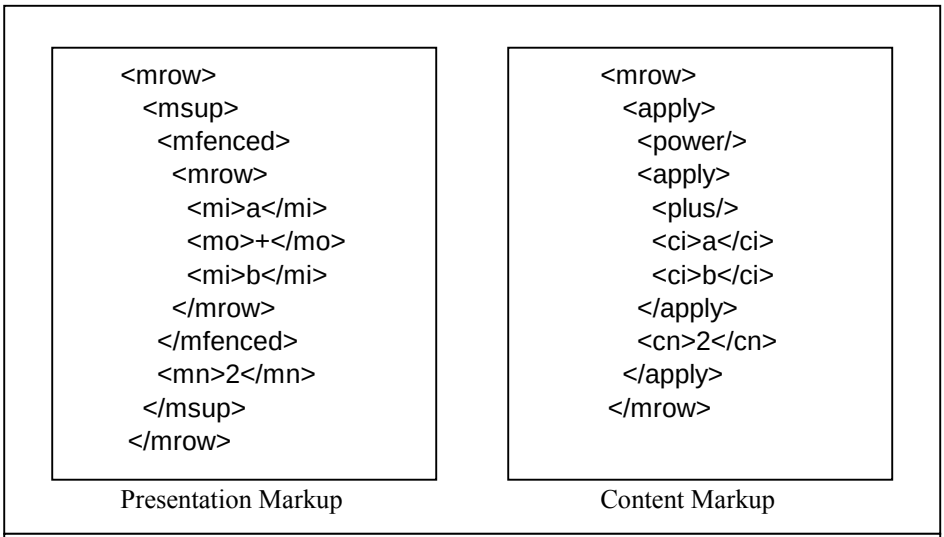

Figure 4- Markup of $(a+b)^2$ in presentation and content encoding styles

A viable alternative to using content encoding of MathML, is to use another emerging technology, OpenMath in place of it. To quote from Open Math homepage "While the original designers were mainly developers of computer algebra systems, it is now attracting interest from other areas of scientific computation and from many publishers of electronic documents with a significant mathematical content. There is a strong relationship to the MathML recommendation from the Worldwide Web Consortium, and a large overlap between the two developer communities. MathML deals principally with the presentation of mathematical objects, while OpenMath is solely concerned with their semantic meaning or content. While MathML does have some limited facilities for dealing with content, it also allows semantic information encoded in OpenMath to be embedded inside a MathML structure. Thus the two technologies may be seen as highly complementary". Because of time constraints on project, MathML is chosen because its implementation is more straightforward, but OpenMath clearly deserves attention.

XML schema is used, instead of DTD (document type definition) document, to model the grammar of CSP expressions, referring to the discussion presented in [yolacan]. Sample tool provided also makes use of several XML family products and technologies, namely Xerces2 DOM compliant XML parser and Xalan XSLT processor from Apache Software Foundation. Information about the products and underlying technologies can be found at Apache XML homepage, also available through dependencies link in project homepage.

3 XML Schema

Extending the work in [yolacan], first step is to modify XML Schema provided for grammar of extended WRIGHT descriptions, as to change CSP elements from simple string types, to structured types, which are defined in their own schema document, utilizing schema inclusion (extendedwright_2.xsd). Next similar step is to modify top file in hierarchy of MathML Schema files, which includes all other MathML files, as to be available to be imported as a namespace (MathML2.xsd). These files, and CSP XML Schema document hierarchy (Constructs.xsd, Implicit-Process-Expression.xsd, Explicit-Process-Expresion.xsd, Event-Expression.xsd) is then included in CSP.xsd.

The four files included define different aspects of CSP grammar. Contructs.xsd presents constructs for defining conditional definitions using 'piecewise, piece and otherwise' type semantics, and parameterized expressions using bounded variable semantics, both resembling their counterparts in MathML. Event-Expression.xsd provides a structure for defining events in CSP expressions, considering security annotations, communications, data definition, direction of data flow, naming requirements like optional port or role name. Implicit-Process-Expression.xsd defines all CSP operators used; naming, prefixing, internal choice, external choice, interleaving, sequencing (any operation quantified over a sequence), and special process of *success*. Application of these operations implicitly defines new processes. Explicit-Process-Expression.xsd defines structure for explicitly defined CSP expressions with optional names and state subscripts. This structure is used as a shell for implicit definitions. CSP.xsd puts all these files together adding flexibility to define explicit process expressions in where clauses, and defining process expressions as interface substitutions.

4 Sample Tool

The tool is prepared using Java programming language. Libraries for Xerces XML Parser and Xalan XSLT processor, and Log4j logging framework were obtained from Apache Foundation Website.

The tool takes an XML document confirming to developed XML Schema, validates conformance to schema, and produces 'configuration behavior' process expression in a format suitable for analysis with an existing CSP analyzer.

Tool is provided as a single executable including developed code and imported libraries, and packed in an achieve file with XML Schema developed, three example XML input files conforming to schema, and a configuration file. Configuration file allows modification of output style and logging facilities. Description of contents of the archive file, and instructions to use the tool are provided at project website. Project website also contains browsable schema documentation, browsable source code, javadocs generated from source code in an integrated fashion, utilizing Apache Foundation's project comprehension tool Maven.

It must be noted that implementation for notation is provided for basic mathematical operations only, such that addition, subtraction, logical and, or, set operations union, intersection, set difference, constructs as interval, list etc. These operations are enough for representing a fairly complex WRIGHT description given as a chapter in [allen] as AEGIS case study. Additional mathematical operations can be added easily by extending the code. A dummy operator represents other operations, which simply list its arguments.

5 Conclusion

XML Schema developed for CSP expressions in WRIGHT descriptions is exercised with preparation of three sample XML files representing WRIGHT descriptions appearing in [allen], and preparation of a sample tool to validate them and extract some meaningful information from these representations as if to show that representation captures the behavior and coordination of components.

To conclude, CSP expressions may successfully be encoded as XML documents. These documents may be used modularly in different contexts, WRIGHT descriptions being the one presented in this study. It must be noted that XML becomes a more viable choice as emerging technologies like MathML and OpenMath mature and tools for these technologies are developed, providing a momentum in representation of semantics of complex mathematical expressions that CSP heavily depends on.

REFERENCES

[allen] R. Allen and D. Garlan, "A formal approach to software architectures", Proceedings of IFIP'92, September 1992
http://citeseer.nj.nec.com/article/allen97formal.html

[hoare] C.A.R Hoare, "Communicating Sequential Processes", March 2003, http://www.usingcsp.com/

[ulu] C. Ulu, "Access control in software architectures", Progress Reports 2002, 2003

[yolacan] B. Yolacan, "An XML Schema for Wright Descriptions with Confidentiality Annotations", 1. Ulusal Yazılım Mühendisliği Sempozyumu, 2003
http://www.ceng.metu.edu.tr/~e112075/project/

XSLT Developers Guide, C. See, McGraw-Hill 2002

Elements of ML Programming, J.D. Ullman, Prentice, Hall 1998

XML Schema Homepage and Specifications: http://www.w3.org/XML/Schema

MathML homepage: http://www.w3.org/Math/

OpenMath homepage: http://www.openmath.org/

Apache XML Homepage: http://xml.apache.org/

Project web site: http://www.ceng.metu.edu.tr/~e1272475/tp/