An XML Schema for Wright with Confidentiality Extensions

Burak YOLAÇAN
Department of Computer Engineering,
Middle East Technical University,
Ankara, 06531, TURKEY
e-mail: burak@yolacan.name.tr

Cemil ULU
Department of Computer Engineering,
Middle East Technical University,
Ankara, 06531, TURKEY
e-mail: cemil@ceng.metu.edu.tr

Halit OĞUZTÜRKÜN
Department of Computer Engineering,
Middle East Technical University,
Ankara, 06531, TURKEY
e-mail: oguztuzn@ceng.metu.edu.tr

Abstract

The eXtensible Markup Language (XML), a World Wide Web Consortium (W3C) standard, is utilized to encode descriptions in Wright/c, the architectural description language Wright extended with confidentiality-related specifications. The grammar of Wright/c is converted to an XML Schema. A common format is thus formed which can be used by applications accepting architectural descriptions as input. This approach facilitates XML tool support. As a case study, a parser that transforms the Wright/c descriptions encoded in XML into the representation accepted by an existing data flow analysis tool is developed, leveraging the power of the Java-based Xerces XML parser, which has the W3C standard DOM interface.

Key Words: XML, architecture description languages, confidentiality.
1. Introduction

Software source codes and related formal descriptions are plain texts that do not present any special difficulty for reading and writing by humans. However, such a linear representation is not quite convenient for the tools aimed to parse and process such material. Hence, using an easy-to-process representation should lead to software development with less effort and time. Further, a wide range of tool support for that particular representation can be leveraged.

In this work, the aforementioned approach is realized by using XML to encode descriptions written in the confidentiality-related extension of the architecture description language Wright [1], dubbed Wright/c [2]. For this purpose the Wright/c syntax is transformed to an XML Schema.

The rest of this paper is organized as follows. Section 2 provides information about Wright and Wright/c, on which this work depends. Section 3 points to related works. In section 4, the approach followed in this work is expounded. Section 5 reports on a case study trying the adopted approach on a newly developed front-end for an existing dataflow-based confidentiality analysis tool. Finally, section 6 draws conclusions, and appendices present material about the Wright/c Schema and the case study.

2. Background

2.1. Wright

During the architectural design of software systems, it is likely that imprecise and incomplete descriptions arise due to the utilization of notations whose syntax and semantics do not have a well-defined basis (such as the traditional box-and-line
diagrams). Architecture description languages have been proposed as a solution based on formal languages.

Wright is such a language which has the aforesaid purpose and lies in the scope of this work [1]. A software system design is described in Wright in terms of components, connectors, interfaces, styles and configurations. A component is an architectural building block that provides a computational service through its interface points, called ports. A connector is an architectural element that mediates interaction among components by defining the roles the ports will play. Connectors come with a glue that connects the computations of each participating component. The processes, both for the interaction patterns of components and connectors, and for the glues and computations, are expressed with Hoare’s CSP (Communicating Sequential Processes) language [3]. Styles are groups of components and connectors together with a set of constraints from which system designers can select according to the particular architecture they want to develop. Lastly, a configuration defines the behaviour and overall topology of a specific system gathering together components and connectors.

2.2. Wright/c

Security is an important extra-functional aspect of a software system. Taking security issues into consideration in the architectural design phase of system development may contribute to the reduction of losses that can occur during operation. Wright/c, an extension of Wright with confidentiality-related definitions and authorizations for secure systems, is put forth as a realization of this approach. The extensions support three functions: (i) defining (presumably based on a mandatory access control policy) the lattice of security classes (ii) assigning clearance to the ports of components in an
architectural configuration, and (iii) assigning (optionally in a parametric way) security classes to the variables holding output data. It then becomes possible to provide an assurance at the architectural level that the information with particular sensitivity is permitted to flow only among the ports having sufficient clearance. The rationale for Wright/c extensions along with their role in the system development cycle are discussed in [2].

3. Related Work

There are basically two kinds of approaches followed by XML-based ADLs, namely DTD (Data Type Definition) oriented and Schema-oriented. The Architecture Description Markup Language (ADML) [4] makes use of DTD for the representation of architecture description interchange language Acme [5]. Another work in the area of XML-based representation is Xarch [6] which provides a core XML Schema to describe typical architectural elements such as components, connectors and interfaces. Xarch can be used both as a description language for the representation of software architectures and a meta-language to define new description languages owing to the extensibility provided by the inheritance of schemas. XAcme [7] is a language created in that way by the addition of schemas defined for the core language Acme to the basic schema of XArch. Another description language introduced by extending XArch schema is xADL 2.0 [8]. XC2, xDarwin and xSADL [9, 10] extensible description languages constituted by DTD-based xADL 1.0 are other related languages.

Apart from XML-based ADLs, there are efforts on XML-based representation for programming languages. C++ Markup Language (CppML) and Java Markup Language
(JavaML) define DTDs for the grammars of C++ and Java languages to encode source codes [11].

4. Approach

4.1. Why XML?

First, the availability of a wide range of both free and commercial XML tools facilitates the processing of XML-encoded representations. In that way, the effort to develop front-ends for such tools from scratch could be directed to the analysis and processing of documents [12].

Second, changes to source code representation can be handled by utilizing the extensibility features of XML.

Third, thanks to the textual encoding of XML, any source code represented in XML can be interchanged among virtually all computer systems.

Fourth, due to the application independent nature of XML and the facility to define DTD and Schema, the interoperability of the applications sharing XML documents is achievable.

Lastly, XML is an open standard on which no single vendor can make changes unilaterally, thereby disrupting the interoperability of systems.

4.2. Why XML Schema?

In spite of DTD’s wide range of use, the more recent XML Schema approach has the following advantages:

- Schemas have the same syntax as XML. Thus, there is no need for an extra learning effort, and special tools to analyse the syntax.
• They provide a rich set of datatypes compared with DTD’s limited support.
• It is possible to create new datatypes.
• Extension or restriction can be applied on an existing datatype.
• They allow to define elements with the same name in different contexts.
• They provide enhanced control over markup.

4.3. Extended BNF to Schema Transformation

Some methods that can be applied for a transformation from a context-free grammar in Extended BNF (EBNF) [13] notation to an XML schema are as follows. If a non-terminal symbol generates a markup, it can be mapped as a schema element with an appropriate tag name. A non-terminal symbol that does not directly produce a markup can be mapped in an implicit way by making use of the ‘group’ indicator so that no tag is generated. Terminal symbols can be mapped to user-defined simple datatypes or basic datatypes such as string or integer. The right hand side of a rewrite rule can be enclosed by the ‘sequence’ indicator to define consecutive symbols. In the event that a rewrite rule provides at least two alternative symbols, the ‘choice’ indicator is suitable. In case of repetitive symbols defined by a rewrite rule, the occurrence constraints ‘minOccurs’ and ‘maxOccurs’ can be utilized for constraining the cardinality. An optional entity can be indicated by setting the ‘minOccurs’ constraint to zero. An item with a basic datatype content can be mapped as either an obligatory or an optional attribute through the usage of the ‘required’ or ‘optional’ indicators, respectively. In case of an entity that can be represented both as an element and attribute, it is better to opt for the latter to achieve a more compact and comprehensible encoding. In addition to these methods, syntactic reductions can be applied to rewrite rules to enhance their readability. As an illustration of transformation methods, the definitions of non-terminal symbols ‘Spec’ and ‘Style’
are mapped to their corresponding schema representations in Figure 1 and Figure 2, respectively.

Figure 1. EBNF definitions for ‘Spec’ and ‘Style’ symbols

```xml
<xs:group name="Spec">
  <xs:choice>
    <xs:element name="Configuration" type="Configuration"/>
    <xs:element name="Style" type="Style"/>
  </xs:choice>
</xs:group>
<xs:complexType name="Style">
  <xs:sequence>
    <xs:element ref="ImportLattice" minOccurs="0"/>
    <xs:group ref="Type" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="Constraints" type="NormalizedString" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="name" type="SimpleName" use="required"/>
</xs:complexType>
```

Figure 2. XML schema representation for ‘Spec’ and ‘Style’ symbols

In the example, as the non-terminal symbol ‘Spec’ does not generate a markup directly, it is mapped with the ‘group’ indicator. For the right hand side of the ‘Spec’ rewrite rule, the two alternative symbols are represented with ‘choice’ indicator such that each alternative non-terminal symbol is mapped as a schema element with a suitable tag name generating a markup. The consecutive symbols defined on the right hand side of the ‘Style’ rewrite rule are described by the ‘sequence’ indicator. Due to the optional definition of lattice entity, zero-valued ‘minOccurs’ constraint is utilized. As the ‘Type’ entity can be repeated zero or more times, ‘minOccurs’ and ‘maxOccurs’ constraints are set to zero and ‘unbounded’, respectively. As the content of the
‘SimpleName’ symbol is a basic datatype and its existence is mandatory, it is preferred to be mapped as an attribute with ‘required’ indicator to yield a shorter representation compared to alternative element description.

The XML schemas for the EBNF definitions for Access Control Lattice Model syntax and the Wright/c syntax [14] are presented in Appendix A and Appendix B, respectively. The grammar that defines the original syntax of Wright is available in [15].

5. Case Study

In order to examine whether encoding of source code in XML is a beneficial way for the representation of Wright/c descriptions, a parser is implemented which is responsible for the construction of the abstract syntax that is used by an existing data-flow based confidentiality verifier programmed in the standard ML [2]. For this purpose, the parser creates a set of datatype definitions in ML, for architectural entities that can appear in a Wright/c description. Besides, it has to carry out adjacency analysis among ports, create data structures related with the security classes of received and sent data at ports, and to implement the algorithms for lattice operations [2].

5.1. Why Xerces and DOM?

Xerces-J XML parser [16] has been chosen for the processing of Wright/c descriptions due to the following reasons. Firstly, it is the only current parser that strictly follows W3C XML Schema Recommendation [17, 18]. Secondly, it supports the DOM [19] interface, and lastly it is based on the programming language Java, which supports platform-independence [20].
There are mainly two types of interface for parsing and processing XML documents: Simple API for XML (SAX) [21] and DOM. In the present context the latter has some advantages over the former. First, SAX is an event-based interface in which user defines certain actions to be performed at certain locations throughout the document. This can be an inefficient method in case of dispersed queries due to multiple passes over the document’s tree representation. On the other hand, it is possible for DOM to read the whole document into memory and reply the queries in less time. However, SAX is able to carry on its performance without being subject to any restriction on document size. The lack of this property of SAX in DOM can cause memory-related problems in large descriptions. Second, the SAX user has to combine the partial information obtained from individual events for a new interpretation. On the contrary, it is possible for a DOM user to obtain the information in a more direct way by already having the tree structure of the document on hand. Lastly, DOM’s being a W3C standard makes it capable of interchanging data with the other tools complying with same standard.

5.2. Secure Print Server

To illustrate our approach, the Secure Print Server (SPS) example from [2] is selected. The SPS system is composed of two printers at different locations, of which one is public and the other is secret, and a print server handling the printer requests. The function of SPS is to get documents printed according to their confidentiality (secret/public) so that a secret document is not sent to the public printer. There are two kinds of users: ordinary (everyone) and authorized. Furthermore, there are connectors which provide user-server and server-printer communication. The Wright/c description for SPS is shown in figure 3.
Figure 3. Lattice of security classes along with architectural style and configuration of SPS

The XML descriptions of the Lattice Model and Wright/c syntax of SPS are in Appendix C. Both the definition of the abstract syntax produced by the parser and the source codes thereof can be obtained from [22]. Figure 4 presents the XML-encoded representation of ‘Printer’ component as an example.
The parser is further tested with a more complex system architecture taken from the literature [1] and accommodated to a security policy. The descriptions and the abstract syntax related to this example are found in [2, 22].

6. Conclusion

The main contribution of this work is the XML representation for the syntax of Wright architecture description language with confidentiality extensions. This style of representation offers advantages that are hard to be obtained by traditional plain text encoding. In the case study, it has been observed that relatively small effort is needed for the development of applications that process documents by utilizing existing XML tools. Further, a common platform for the interoperability of tools operating on architectural descriptions has been created on the basis of XML.

Although these facilities simplify the work of tool developers, the user side may suffer from XML-based representation, which is difficult to read and write. Therefore, there is a need for tools that transform plain text to XML. This will be taken up as a future work.

Acknowledgement
The authors would like to thank Kurtcebe Eroğlu, who produced the part of the XML Schema for CSP, and Ali Ferhat Tamur, who developed the data-flow analyzer for CSP.

References


[22] Project Web Site, http://www.ceng.metu.edu.tr/~e112075/project
Appendix A: XML Schema for Access Control Lattice Model

```xml
<x:schema xmlns:x="http://www.w3.org/2001/XMLSchema">
  <x:simpleType name="IDENTIFIER">
    <x:restriction base="x:NCName">
      <x:pattern value="w(w/d)*"/>
    </x:restriction>
  </x:simpleType>
  <x:simpleType name="SimpleName">
    <x:restriction base="IDENTIFIER"/>
  </x:simpleType>
  <x:simpleType name="NodeList">
    <x:restriction base="x:string">
      <x:pattern value="( ( |\t) *(w(w/d))*( ( |\t) *(w(w/d))*( ( |\t) *))*)"/>
    </x:restriction>
  </x:simpleType>
  <x:simpleType name="Chain">
    <x:restriction base="NodeList"/>
  </x:simpleType>
  <x:element name="Lattice" type="Lattice"/>
  <x:complexType name="Lattice">
    <x:sequence>
      <x:element name="SecurityLabels" type="NodeList"/>
      <x:element name="Ordering" type="EdgeSet"/>
      <x:element name="ClearanceList">
        <x:complexType>
          <x:sequence>
            <x:element name="Clearance" type="ClearanceList" minOccurs="0" maxOccurs="unbounded"/>
          </x:sequence>
        </x:complexType>
      </x:element>
    </x:sequence>
    <x:attribute name="name" type="SimpleName"/>
  </x:complexType>
  <x:complexType name="EdgeSet">
    <x:sequence>
      <x:element name="Order" type="Chain" minOccurs="0" maxOccurs="unbounded"/>
    </x:sequence>
  </x:complexType>
  <x:complexType name="ClearanceList">
    <x:attribute name="names" type="NodeList"/>
    <x:attribute name="labels" type="NodeList"/>
  </x:complexType>
</x:schema>
```
Appendix B: XML Schema for Wright/c

```xml
<x:schema
xmlns:x="http://www.w3.org/2001/XMLSchema">
  <!-- Type Definitions -->
  <xs:simpleType name="IDENTIFIER">
    <xs:restriction base="x:NCName">
      <xs:pattern value="w(\w|d)*/"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="NormalizedString">
    <xs:restriction base="x:string">
      <xs:whiteSpace value="collapse"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="StaticRangeExpression">
    <xs:restriction base="x:string">
      <xs:pattern value="(\d)+\.(\d)+"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="DynamicRangeExpression">
    <xs:restriction base="x:string">
      <xs:pattern value="(\w(\w|d]*)\.(\w(\w|d]*)")"></xs:restriction>
      <xs:pattern
        value="(\w(\w|d]*)\.(\w(\w|d]*)\.(\d)+\.(\d)+"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="FiniteRangeExpression">
    <xs:restriction base="x:string">
      <xs:pattern value="(\d)+\.(\d)+\.(\w(\w|d]*)"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="SubjectRangeExpression">
    <xs:union memberTypes="StaticRangeExpression DynamicRangeExpression"/>
  </xs:simpleType>
  <xs:simpleType name="SimpleName">
    <xs:restriction base="IDENTIFIER"/>
  </xs:simpleType>
  <xs:simpleType name="NameList">
    <xs:restriction base="x:string">
      <xs:pattern value="(\w(\w|d]*)*,(\w(\w|d]*)*)+"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="NodeSet">
    <xs:restriction base="x:string">
      <xs:pattern value="(\w(\w|d]*)*,(\w(\w|d]*)*)+"/>
    </xs:restriction>
  </xs:simpleType>

  <!-- The start symbol -->
  <xs:element name="Descriptions" type="SpecList"/>
  <xs:complexType name="SpecList">
    <xs:group ref="Spec" maxOccurs="unbounded"/>
  </xs:complexType>
  <xs:group name="Spec">
    <xs:choice>
      <xs:element name="Configuration" type="Configuration"/>
      <xs:element name="Style" type="Style"/>
    </xs:choice>
  </xs:group>
</xs:schema>
```
<complexType name="Style">
  <sequence>
    <element ref="ImportLattice" minOccurs="0"/>
    <group ref="Type" minOccurs="0" maxOccurs="unbounded">
      <element name="Constraints" type="NormalizedString" minOccurs="0"/>
    </group>
    <attribute name="name" type="SimpleName" use="required"/>
  </sequence>
</complexType>
<complexType name="ImportLattice">
  <attribute name="name" type="SimpleName" use="required"/>
  <attribute name="filename" type="FilePath" use="required"/>
</complexType>
<group name="Type">
  <choice>
    <element name="Component" type="Component"/>
    <element name="Connector" type="Connector"/>
    <element name="InterfaceType" type="IG"/>
    <element name="Process" type="IG"/>
  </choice>
</group>
<complexType name="Component">
  <sequence>
    <element name="param" type="FormalCCParam" minOccurs="0" maxOccurs="unbounded"/>
    <element name="Port" type="PR" minOccurs="0" maxOccurs="unbounded"/>
    <element name="Computation" type="BehaviourDescription"/>
  </sequence>
  <attribute name="name" type="SimpleName" use="required"/>
</complexType>
<complexType name="Connector">
  <sequence>
    <element name="param" type="FormalCCParam" minOccurs="0" maxOccurs="unbounded"/>
    <element name="Role" type="PR" minOccurs="0" maxOccurs="unbounded"/>
    <element name="Glue" type="BehaviourDescription"/>
  </sequence>
  <attribute name="name" type="SimpleName" use="required"/>
</complexType>
<complexType name="FormalCCParam">
  <attribute name="names" type="NameList" use="required"/>
  <attribute name="type" type="FormalParamType" use="optional"/>
  <attribute name="range" type="StaticRangeExpression" use="optional"/>
</complexType>
<complexType name="PR">
  <sequence>
    <element ref="CSPExp"/>
  </sequence>
  <attribute name="name" type="SimpleName" use="required"/>
  <attribute name="range" type="FiniteRangeExpression" use="optional"/>
</complexType>
<complexType name="BehaviourDescription">
  <choice>
    <element ref="CSPExp"/>
    <group ref="Subconfiguration"/>
  </choice>
</complexType>
<group name="Subconfiguration">
  <sequence>
    <element name="Configuration" type="Configuration"/>
    <element name="Bindings" type="Bindings"/>
  </sequence>
</group>
<complexType name="Bindings">
  <sequence>
    <element name="Binding" type="Binding" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</complexType>
<complexType name="Binding">
  <sequence>
    <element name="Outer" type="Interface"/>
    <element name="Inner" type="ActualPRName"/>
  </sequence>
</complexType>
<complexType name="ActualPRName">
  <attribute name="pr" type="SimpleName" use="required"/>
  <attribute name="index" type="x:integer" use="optional"/>
</complexType>
<complexType name="Interface">
  <attribute name="cc" type="SimpleName" use="required"/>
  <attribute name="ccIndex" type="x:integer" use="optional"/>
  <attribute name="pr" type="SimpleName" use="required"/>
  <attribute name="prIndex" type="x:integer" use="optional"/>
</complexType>
<complexType name="Configuration">
  <sequence>
    <element ref="ImportLattice" minOccurs="0" maxOccurs="unbounded"/>
    <group ref="Type" minOccurs="0" maxOccurs="unbounded"/>
    <element name="Instances">
      <complexType>
        <sequence>
          <element name="Instance" type="Instance" maxOccurs="unbounded"/>
        </sequence>
      </complexType>
    </element>
    <element name="Clearance" minOccurs="0">
      <complexType>
        <sequence>
          <element name="ClearanceList" type="ClearanceList" minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
      </complexType>
    </element>
    <element name="Attachments">
      <complexType>
        <sequence>
          <element name="Attachment" type="Attachment" minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
      </complexType>
    </element>
  </sequence>
  <attribute name="name" type="SimpleName" use="required"/>
  <attribute name="style" type="SimpleName" use="optional"/>
</complexType>
<complexType name="ClearanceList">
  <sequence>
    <element name="CCName" type="Subject" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="clearance" type="SimpleName" use="required"/>
</complexType>
<complexType name="Subject">
  <sequence>
    <element name="PRName" type="PRName" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="id" type="SimpleName" use="required"/>
  <attribute name="index" type="SubjectRangeExpression" use="optional"/>
</complexType>
<complexType name="PRName">
  <attribute name="id" type="SimpleName" use="required"/>
  <attribute name="index" type="SubjectRangeExpression" use="optional"/>
</complexType>
<complexType name="Instance">
  <sequence>
    <element name="name" type="InstanceName" maxOccurs="unbounded"/>
    <element name="param" type="ActualCCParam" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="type" type="SimpleName" use="required"/>
</complexType>
<complexType name="InstanceName">
  <attribute name="id" type="SimpleName" use="required"/>
  <attribute name="range" type="StaticRangeExpression" use="optional"/>
</complexType>
<complexType name="ActualCCParam">
  <choice>
    <element ref="CSPExp"/>
    <element ref="IntegerExp"/>
    <element name="LatticeFunction" type="LatticeFunction"/>
  </choice>
</complexType>
<x:complexType name="LatticeFunction">
  <x:attribute name="lattice" type="SimpleName" use="required"/>
  <x:attribute name="function" type="SimpleName" use="required"/>
  <x:attribute name="nodes" type="NodeSet" use="optional"/>
</x:complexType>
<x:complexType name="Attachment">
  <x:sequence>
    <x:element name="From" type="Interface"/>
    <x:element name="To" type="Interface"/>
  </x:sequence>
</x:complexType>
<x:complexType name="IG">
  <x:sequence>
    <x:element name="param" type="NormalizedString" minOccurs="0" maxOccurs="unbounded"/>
    <x:element ref="CSPExp"/>
  </x:sequence>
  <x:attribute name="name" type="SimpleName" use="required"/>
</x:complexType>
</x:schema>
Appendix C: Description of SPS in XML

```xml
<Lattice xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="lattice.xsd"
name="CSL">
  <SecurityLabels> PUBLIC, SECRET </SecurityLabels>
  <Ordering>
    <Order> PUBLIC, SECRET </Order>
  </Ordering>
  <ClearanceList>
    <Clearance names="EVERYONE" labels="PUBLIC"/>
    <Clearance names="PRIVATE" labels="SECRET"/>
  </ClearanceList>
</Lattice>

<Descriptions
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="extendedwright.xsd">
  <Style name="ClientServerPrinting">
    <ImportLattice name="CSL" filename="ACLattice.xml"/>
    <Component name="Client">
      <param names="T" type="Security Label"/>
      <Port name="PrintServiceP">
        <CSPExp> request! x^t   -> PrintServiceP </CSPExp>
      </Port>
      <Port name="PrintServiceS">
        <CSPExp> request! x^SECRET  -> PrintServiceS </CSPExp>
      </Port>
      <Computation>
        <CSPExp>
          PrintServiceP. request! x^t -> Computation
          PrintServiceS. request! x^SECRET-> Computation
        </CSPExp>
      </Computation>
    </Component>
    <Component name="Printer">
      <Port name="Receive">
        <CSPExp> request? x   -> Receive </CSPExp>
      </Port>
      <Computation>
        <CSPExp> Receive. Request? x -> DoPrint-> Computation </CSPExp>
      </Computation>
    </Component>
    <Component name="PrintServer">
      <Port name="PrintPublic">
        <CSPExp> request?x -> PrintPublic </CSPExp>
      </Port>
      <Port name="PrintSecret">
        <CSPExp> request?x -> PrintSecret </CSPExp>
      </Port>
      <Port name="OutputPublic">
        <CSPExp> Print!x -> OutputPublic </CSPExp>
      </Port>
      <Port name="OutputSecret">
        <CSPExp> Print!x -> OutputSecret </CSPExp>
      </Port>
      <Computation>
        <CSPExp>
          PrintPublic.Request?x -> OutputPublic.Print!x -> Computation
        </CSPExp>
      </Computation>
    </Component>
    <Connector name="PrintConnector">
      <Role name="ClientP">
        <CSPExp> request? x  -> ClientP </CSPExp>
      </Role>
      <Role name="ServerP">
        <CSPExp> request!x   -> ServerP </CSPExp>
      </Role>
      <Glue>
        <CSPExp> ClientP.request?x -> ServerP.request!x -> Glue </CSPExp>
      </Glue>
    </Connector>
  </Style>
  <Configuration name="PrintExample"
    style="ClientServerPrinting">
    <Instances>
      <Instance type="Client">
        <name id="U1"/>
        <param>
          <LatticeFunction lattice="CSL" function="min"/>
        </param>
      </Instance>
      <Instance type="Client">
        <name id="U2"/>
        <param>
          <LatticeFunction lattice="CSL" function="min"/>
        </param>
      </Instance>
      <Instance type="PrintServer">
        <name id="PS"/>
        <param>
          <LatticeFunction lattice="CSL" function="min"/>
        </param>
      </Instance>
      <Instance type="Printer">
        <name id="SECUREPRINTER"/>
      </Instance>
    </Instances>
  </Configuration>
</Descriptions>
```
Gizlilik Eklentili Wright için bir XML Şeması

**Burak YOLAÇAN**
Bilgisayar Mühendisliği Bölümü,
Orta Doğu Teknik Üniversitesi,
Ankara, 06531, TÜRKİYE
e-posta: burak@yolacan.name.tr

**Cemil ULU**
Bilgisayar Mühendisliği Bölümü,
Orta Doğu Teknik Üniversitesi,
Ankara, 06531, TÜRKİYE
e-posta: cemil@ceng.metu.edu.tr

**Halit OĞUZTÜRKÜN**
Bilgisayar Mühendisliği Bölümü,
Orta Doğu Teknik Üniversitesi,
Ankara, 06531, TÜRKİYE
e-posta: oguztuzn@ceng.metu.edu.tr

**Özet**

Anahtar Kelimeler: XML, mimari betimleme dilleri, gizlilik.
**Figure Legends**

Figure 1. EBNF definitions for ‘Spec’ and ‘Style’ symbols

Figure 2. XML schema representation for ‘Spec’ and ‘Style’ symbols

Figure 3. Lattice of security classes along with architectural style and configuration of SPS

Figure 4. XML representation of ‘Printer’ component