AN XML SCHEMA FOR WRIGHT DESCRIPTIONS WITH CONFIDENTIALITY ANNOTATIONS

MASTER PROJECT REPORT

BY

BURAK YOLAÇAN

Supervisor : Asst. Prof. Dr. Halit OĞUZTÜZÜN

Department of Computer Engineering
MIDDLE EAST TECHNICAL UNIVERSITY

JUNE 2003

An XML Schema for Wright Descriptions with Confidentiality Annotations

Burak Yolaçan

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

Abstract. It is decided that the eXtensible Markup Language (XML) standard developed by the WorldWide Web Consortium (W3C) is an appropriate tool to encode the descriptions expressed with the confidentiality annotated form of the architectural description language (ADL) Wright and then the grammar of the language is converted to an XML Schema. Thus, a common format is constituted which can be used by applications accepting architectural descriptions as input together with XML tool support. As a sample application, a parser which transforms the confidentiality annotated Wright descriptions written in XML into a representation accepted by a security checking tool is developed, utilizing the java-based Xerces XML parser which has the DOM interface as a W3C standard.

1 Introduction

Source codes have traditionally been represented in the form of plain text which makes them easy to read and write by humans. However, this kind of notation makes it hard for the tools aimed to analyze and process them. In that sense, using a representation which is easy to manipulate may lead to the development of tools with less effort and in less time. Together with its flexible nature, the existence of a wide range of tool support for that particular representation may further increase the facilities.

In this project, aforementioned approach is realized by using XML as a means of representation to encode the confidentiality annotated descriptions of Wright ADL. For this purpose, the notation used for describing a software system in the extended form of Wright language is converted to a format that can be used by XML parsers for the analysis of the documents.

The rest of the paper is organized as follows. Section 2 gives information about the topics on which the project depends such as Wright, extended form of Wright with security annotations and XML. Section 3 mentions works related with XML-based representation. In section 4, approach used in this work is described. Section 5 is about the examination of the approach by a tool developed and test of it with a sample description. Finally, section 6 concludes and appendices present material about the test sample.

2 Background

2.1 Wright

During the architectural design of software systems, it is possible that vague and indefinite descriptions may occur due to the informal notations used. Architectural description languages, in that sense, have been addressed as a solution to this problem by providing an approach based on formal notations.

Such an ADL in the range of this work is Wright [1]. A typical software system designed with Wright may consist of components, connectors, interfaces, styles and configurations. A component is an architectural element which provides computational service through its interface called port. A connector is a building block that makes the interaction and communication possible among components by defining the roles the ports will play. A connector also defines a glue which unions the computations of each participating component. Interfaces are used to define processes for the glues and computations. Styles are groups of components and connectors arranged together with a set of rules from which system designers can select according to the architecture they want to develop and lastly a configuration defines the behaviour and overall topology of the system gathering together the components and the connectors.

2.2 Extended Form of Wright

Security attacks has emerged as a serious problem causing not only vast amount of financial losses but also vital damages. For the solution to this problem, the idea of utilizing security extensions for software systems can act as an effective way. A realization of this approach, extended form of Wright [2], is a version of Wright supported with confidentiality annotations for the design of such secure systems. These annotations are used to assign security levels, namely clearances, to the ports of components and roles of connectors. In that way, a control mechanism is formed to regulate the accesses to system resources by letting information flow only between certain entities.

2.3 Overview of XML

XML is a subset of Standard Generalized Markup Language (SGML) [3]. Due to the complexity of SGML, a refined and simpler to use version, XML, was developed.

XML is extensible since user can define new tags and it is a markup language for the facility that data can be markedup between the opening and closing tags. An example is:

```
<ticket>
   <source> Ankara </source>
   <destination> Istanbul </destination>
   <price currency=”TL”> 25000000 </price>
</ticket>
```

An XML document is said to be well-formed if there is only one root element, attributes are quoted and every opening tag is matched with a closing tag in a way that case sensitivity and nesting levels are preserved.

The grammar of the language used in an XML document can be specified either in a DTD (Data Type Definition) or XML Schema [4, pp. 5-7; 5]. These entities imitate the function of BNF (Backaus Normal Form) metalanguage [6, pp.529] which is the langauge used to develop language. One major difference between DTD and schema is that the latter uses the syntax of XML while the former not.

3 Related Work

Basically, there are two kind of approaches used in XML-based ADLs., namely DTD-oriented and schema-oriented. Architecture Description Markup Language (ADML) [7], makes use of the DTD technology for the representation of architectural description interchange language Acme [8]. Another effort in the arena of XML-based representation is xArch which provides a core XML Schema describing typical elements such as components, connectors and interfaces that may be used in an architectural design. xArch can be used either as it is to represent software architectures or it can be extended to define new description languages by using the inheritance property of schemas. One language created in this way is Xacme [10] adding its own schemas defined for its core language Acme to the basic schema of xArch.

Apart from the XML-based ADLs, there are also 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 of all, XML is well-supported in the sense that it is easy to parse XML-encoded representations due to the potential support from many number of both freely and commercially existing processing tools. In that way, effort needed for developing such tools can be used in the analysis and manipulation of such documents [12].

Second, in case of probable changes in the original source code representation, XML can handle the situation by making use of its extensibility property.

Third, since XML is platform independent due to its textual encoding, any source code represented in XML can be interchanged among different computer platforms.

Fourth, XML is application independent. By being able to define DTDs and schemas, it is possible to make two or more XML documents and also the applications interchanging them speak the same language.

Lastly, XML is an open standard by W3C [13]. Existing and potential works can be guaranteed not to be affected from any changes in the future.

4.2 Why XML Schema?

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

- Schemas use the same syntax with XML. There is no need for an exstra learning effort and tool to process the grammar defined in the schema.
- Schemas 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 type.
- It is permitted to define elements with the same name in different contexts.

4.3 BNF to Schema Transformation

There are some policies that are followed during the conversion process from BNF to XML Schema as follows. If a non-terminal symbol generates a markup, then it can be mapped as a schema element with a proper tag name. In case of a non-terminal symbol which is an intermediate one not directly generating a markup, it is to be defined in a way that no tag is generated and the 'group' indicator may be useful in such a case. Terminal symbols can be mapped to suitable simple datatypes defined by the designer or basic datatypes such as string or integer. In condition that a sequence of elements are defined by a non-terminal, it is appropriate to use the 'sequence' indicator. In the event that a non-terminal provides alternative productions by using the symbol '|', the 'choice' is the convenient indicator to utilize. In case of repetitive structures defined at the right hand side of a BNF production, occurrence indicators, 'minOccurs' and 'maxOccurs', serve as the candidates. If there is an optional entity, it can be defined by setting the 'minOccurs' attribute to zero. In case of an entitiy that can be represented both by an element and attribute, it is better to use the latter to lead to more compact design.

The extended Wright XML schema obtained by transforming the corresponding BNF using mentioned policies and the BNF itself are available in [14].

5 A Case Study

To investigate whether encoding of source code in XML is a suitable way for the representation and analysis of annotated Wright descriptions, a parser is implemented. It is responsible for the construction of an abstract syntax which will be given as an input to a confidentiality checker tool [15]. For this purpose, parser creates a corresponding datatype definition for architectural entities that can be seen in extended Wright descriptions. In additon to these, it has to make dependency analysis between ports, create data structures related with securities of received and sent data between ports and also implement algorithms for lattice functions [2].

5.1 Why Xerces and DOM ?

Xerces-J XML parser [16] is chosen for the processing of extended Wright descriptions due to the following reasons. First, it is the only current parser which gives support for W3C XML Schema Recommendation [17], second it supports DOM interface and lastly it is based on a platform independent programming language, Java.

There are two main approaches used in processing and analyzing XML documents, namely Simple API for XML (SAX) [18] and DOM [19]. 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 is an inefficient method causing delay because of the need for multiple passes in case of different queries whereas DOM reads the document once into memory and therefore is able to reply queries with less effort. Second, SAX user has to join the information obtained from individual events for further interpration. However a DOM user is usually able to reach the information in a more direct way by already having the complete tree structure of the document in hand. Lastly, DOM's nature of being a W3C standard makes the tools developed with it capable of interchanging data.

5.2 Implementation of Lattice Operations

A lattice is a non-empty partially ordered set where for any two elements of the set, both a greatest lower bound and a least upper bound exist. A lattice is complete if any subset of elements has both a greatest lower bound and a least upper bound [20]. Hence, a complete lattice has the minimum and maximum elements. Also note that finite lattices, which are of interest in this work, are always complete. For such a lattice, it is possible to evaluate min, max, meet and join functions in an efficient way as described in [21] and [22]. For this purpose, adjacency matrices for 'immediately dominated by' and its inverse relation 'dominates' must initially be calculated. Based on them, their transitive-reflexive closures can be evaluated by using Warshall's algorithm [23]. In that way, each row of the corresponding matrices will be the code vectors for the lower and upper bounds, respectively. It would be appropriate to explain these algorithms by using a sample lattice with security class relationships 1 and 2:

$$\text{Unclassified (U)} \leq \text{Confidential (C)} \leq \text{Top secret (TS)} \quad (1)$$

$$\text{U} \leq \text{Secret (S)} \leq \text{TS} \quad (2)$$

Any destination node that can be reached directly or indirectly by a source node can be found in Table 1. Since U is dominated by all other nodes, in other words row U is entirely 1, 'min' function evaluates to U. To find the least upper bound for C and S, formally join(C, S), the corresponding rows must be logically conjucted as "0101 $\wedge$ 0011" resulting in 0001 and this outcome vector is to be compared with the existing row vectors for equivalance. For this particular example, 0001 vector is the code of row TS. Therefore join(C, S) evaluates to TS.

Table 1. Transitive-reflexive closure for the 'immediately dominated by' relation

≤	**U**	**C**	**S**	**TS**
U	1	1	1	1
C	0	1	0	1
S	0	0	1	1
TS	0	0	0	1

Reachability of nodes for the inverse relation is shown in Table 2. Since TS dominates all other nodes, leading to row TS to be wholly 1, max function evaluates to TS. meet(C, S), namely the greatest lower bound for C and S, can be evaluated by logically conjucting "1100 $\wedge$ 1010" that results in 1000 vector which is equivalent of row U when compared with the existing row vectors. Therefore meet(C, S) evaluates to U.

Table 2. Transitive-reflexive closure for the 'immediately dominates' relation

≥	**U**	**C**	**S**	**TS**
U	1	0	0	0
C	1	1	0	0
S	1	0	1	0
TS	1	1	1	1

5.3 Secure Print Server

To check the parser together with the lattice operations, a sample taken from [15], Secure Print Server (SPS), is used. The aim of SPS is to construct a mechanism in which documents are printed according to their confidentialities such that a secure document is not printed in a public printer. The sample system is composed of two printers at different locations, one is public and other is secret, a print server controlling the printer requests, and two kind of users, an ordinary client and an authorized client. There are also connectors between the users and printserver and between the printserver and printers to provide the communication. The detailed system description can be found in [15].

The annotated Wright descriptions of SPS in plain text and in XML can be followed in appendix A and appendix B, respectively. The abstract syntax produced by the parser and the parser itself is available in [14].

6 Conclusion

Representing extended Wright descriptions in XML leads to facilities which could not otherwise be obtained in traditional plain text encoding. In that sense, one of them is the less effort needed for developing applications that process and manipulate documents by utilizing the wide range of support from the repository of existing XML tools. In this way resources needed for the related phase can be leveraged for the consecutive issues. Another important facility is the common platform created for the interoperability of potential tools which are expected to communicate through the same XML interface. In addition, XML standard also prevents any kind of vendor specific dependencies.

Although these facilities make the work of tool developers easier, the user side may suffer from the XML-encoded representation which is not easy to read, write and follow. Therefore, there will probably be a need for the development of tools which transform the plain text form to XML in order to make both architecture designers and application developers satisfied.

References

1. R. Allen and D. Garlan, "A formal approach to software architectures", Proceedings of IFIP'92, September 1992, pp. 134-141. http://citeseer.nj.nec.com/article/allen97formal.html
2. C. Ulu, "Access control in software architectures", Progress Report, Computer Engineering Department, METU, December 2002, pp. 22-31
3. ISO 8879. Information Processing – Text and Office Systems – Standard Generalized Markup language (SGML), 1986. http://www.iso.ch/cate/d16387.html
4. W. R. Stanek, XML Pocket Consultant, Microsoft Press, 2002, ISBN: 0-7356-1183-1
5. XML Official Web Site, http://www.xml.org
6. W. K. Grassmann and J. Tremblay, Logic and Discrete Mathematics, Prentice-Hall, 1996, ISBN: 0-13-209008-2
7. ADML Web Site, http://www.opengroup.org/architecture/adml/adml_home.htm
8. D. Garlan, R. Monroe, and D. Wile, "Acme: An architectural description interchange language", Proceedings of CASCON'97, November 1997, pp. 169-183. http://www-2.cs.cmu.edu/afs/cs/project/able/ftp/acme-cascon97/acme-cascon97.pdf
9. xArch Web Site, http://www.isr.uci.edu/architecture/xarch/
10. xAcme Web Site, http://www-2.cs.cmu.edu/~acme/pub/xAcme/
11. E. Mamas and K. Kontogiannis, "Towards portable source code representation using XML", WCRE'2000, November 2000, pp. 172-182. http://www.swen.uwaterloo.ca/~evan/Papers/wcre2000.pdf

12. S. Pruitt, D. Stuart, W. Sull and T. W. Cook. “The merit of xml as an architecture description language meta-language”, Microelectronics and Computer Technology Corporation, October 1998. http://citeseer.nj.nec.com/pruitt98merit.html

13. World Wide Web Consortium Web Site, http://www.w3c.org

14. Project Home Page. http://www.ceng.metu.edu.tr/~e112075/project

15. C. Ulu, “Access control in software architectures”, Progress Report, Computer Engineering Department, METU, March 2003, pp. 1-8.

16. The Apache Group. Xerces Java Parser. http://xml.apache.org/xerces-j/

17. World Wide Web Consortium. XML Schema W3C Recommendation, May 2001. http://www.w3.org/TR/xmlschema-0/

18. SAX. http://www.saxproject.org

19. World Wide Web Consortium. Document Object Model (DOM) Level 1 Specification, W3C Recommendation, October 1998. http://www.w3.org/TR/REC-DOM-Level-1

20. B. A. Davey and H. A. Priestly, Introduction to Lattices and Order, 2nd edition, Cambridge University Press, 2002.

21. S. G. Pulman, “Unification encodings of grammatical notations”, Computational Linguistics, Vol 22/3, 1996, pp. 295-328.
http://www.clp.ox.ac.uk/people/staff/pulman/pdfpapers/unif.pdf

22. H. Ait-Kaci, B. Boyer, P. Lincoln, and R. Nasr, “Efficient implementation of lattice operations“, ACM Transactions of Programming Languages and Systems, Vol 11/1, January 1989, pp. 115-146. http://citeseer.nj.nec.com/ait-kaci89efficient.html

23. Y. Langsam, M. J. Augenstein, and A. M. Tenenbaum, Data Structures Using C and C++, Prentice-Hall, 1996, ISBN: 0-13-529322-7.

Appendix A: Annotated Wright Description of SPS

A.1 Lattice for SPS

```
(* lattice file “ACLattice.txt” *)
Lattice CSL
       Security Labels
              PUBLIC, SECRET
       Ordering
              PUBLIC, SECRET
       ClearanceList
              EVERYONE      : PUBLIC
              PRIVATE       : SECRET
End Lattice
```

A.2 SPS Description

```
Style ClientServerPrinting
   Import Lattice CSL “ACLattice.txt”

   Component Client(t : SecurityLabel) =
       Port PrintServiceP = request! x^t   -> PrintServiceP
       Port PrintServiceS = request! x^SECRET  -> PrintServiceS
       Computation  =  PrintServiceP. request! x^t -> Computation
              [] PrintServiceS. request! x^SECRET-> Computation

   Component Printer() =
       Port Receive = request? x   -> Receive
       Computation = Receive. Request? x -> DoPrint-> Computation

   Component PrintServer() =
       Port PrintPublic = request?x -> PrintPublic
       Port PrintSecret = request?x -> PrintSecret
       Port OutputPublic = Print!x  -> OutputPublic
       Port OutputSecret = Print!x -> OutputSecret
          Computation = PrintPublic.Request?x –>
                           OutputPublic.Print!x ->
                        Computation
                           []  PrintSecret.Request?x ->
                                  OutputSecret.Print!x ->
                               Computation

   Connector  PrintConnector ()   =
       Role ClientP = request? x  -> ClientP
       Role ServerP = request!x   -> ServerP
       Glue = ClientP.request?x -> ServerP.request!x -> Glue
           [] §
End Style
```

```
Configuration PrintExample
       Style ClientServerPrinting
       Instances
              U1           : Client(CSL.min())
              U2           : Client(CSL.min())
              PS           : PrintServer()
              SECUREPRINTER: Printer()
              PUBLICPRINTER: Printer()

              CONN1        : PrintConnector()
              CONN2        : PrintConnector()
              CONN3        : PrintConnector()
              CPRINTS      : PrintConnector()
              CPRINTP      : PrintConnector()

       Clearances
              U1                         : EVERYONE
              U2                         : PRIVATE
              U2.PrintServiceP           : EVERYONE
              PS.PrintPublic             : EVERYONE
              PS.PrintSecret             : PRIVATE
              PS.OutputPublic            : EVERYONE
              PS.OutputSecret            : PRIVATE
              CONN1, CONN3, CPRINTP      : EVERYONE
              CONN2, CPRINTS             : PRIVATE
              SECUREPRINTER              : PRIVATE
              PUBLICPRINTER              : EVERYONE

       Attachments
              U1.PrintServiceP as CONN1.ClientP
              PS.PrintPublic  as CONN1.ServerP
              U2.PrintServiceS  as CONN2.ClientP
              PS.PrintSecret as CONN2.ServerP
              U2.PrintServiceP  as CONN3.ClientP
              PS.PrintPublic as CONN3.ServerP
              PS.OutputPublic  as CPRINTP.ClientP
              PUBLICPRINTER.Receive as CPRINTP.ServerP
              PS.OutputSecret  as CPRINTS.ClientP
              SECUREPRINTER.Receive as CPRINTS.ServerP

End Configuration
```

Appendix B: Annotated Wright Description of SPS in XML

B.1 Lattice for SPS in XML

```
(* lattice file “ACLattice.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>
```

B.2 SPS Description in XML

```
<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 []
         PrintSecret.Request?x -> OutputSecret.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"/>
              </Instance>
              <Instance type="Printer">
                   <name id="SECUREPRINTER"/>
              </Instance>
              <Instance type="Printer">
                   <name id="PUBLICPRINTER"/>
              </Instance>
              <Instance type="PrintConnector">
                   <name id="CONN1"/>
                   <name id="CONN2"/>
                   <name id="CONN3"/>
                   <name id="CPRINTS"/>
                   <name id="CPRINTP"/>
              </Instance>
         </Instances>

         <Clearances>
              <ClearanceList clearance="EVERYONE">
                   <CCName id="U1"/>
              </ClearanceList>
              <ClearanceList clearance="PRIVATE">
                   <CCName id="U2"/>
              </ClearanceList>
              <ClearanceList clearance="EVERYONE">
                   <CCName id="U2">
                        <PRName id="PrintServiceP"/>
                   </CCName>
              </ClearanceList>
              <ClearanceList clearance="EVERYONE">
                   <CCName id="PS">
                        <PRName id="PrintPublic"/>
                   </CCName>
              </ClearanceList>
```

```
     <ClearanceList clearance="PRIVATE">
          <CCName id="PS">
               <PRName id="PrintSecret"/>
          </CCName>
     </ClearanceList>
     <ClearanceList clearance="EVERYONE">
          <CCName id="PS">
               <PRName id="OutputPublic"/>
          </CCName>
     </ClearanceList>
     <ClearanceList clearance="PRIVATE">
          <CCName id="PS">
               <PRName id="OutputSecret"/>
          </CCName>
     </ClearanceList>
     <ClearanceList clearance="EVERYONE">
          <CCName id="CONN1"/>
          <CCName id="CONN3"/>
          <CCName id="CPRINTP"/>
     </ClearanceList>
     <ClearanceList clearance="PRIVATE">
          <CCName id="CONN2"/>
          <CCName id="CPRINTS"/>
     </ClearanceList>
     <ClearanceList clearance="PRIVATE">
          <CCName id="SECUREPRINTER"/>
     </ClearanceList>
     <ClearanceList clearance="EVERYONE">
          <CCName id="PUBLICPRINTER"/>
     </ClearanceList>
</Clearances>

<Attachments>
     <Attachment>
          <From cc="U1" pr="PrintServiceP"/>
          <To cc="CONN1" pr="ClientP"/>
     </Attachment>
     <Attachment>
          <From cc="PS" pr="PrintPublic"/>
          <To cc="CONN1" pr="ServerP"/>
     </Attachment>
     <Attachment>
          <From cc="U2" pr="PrintServiceS"/>
          <To cc="CONN2" pr="ClientP"/>
     </Attachment>
     <Attachment>
          <From cc="PS" pr="PrintSecret"/>
          <To cc="CONN2" pr="ServerP"/>
     </Attachment>
     <Attachment>
          <From cc="U2" pr="PrintServiceP"/>
          <To cc="CONN3" pr="ClientP"/>
     </Attachment>
```

```
                <Attachment>
                     <From cc="PS" pr="PrintPublic"/>
                     <To cc="CONN3" pr="ServerP"/>
                </Attachment>
                <Attachment>
                     <From cc="PS" pr="OutputPublic"/>
                     <To cc="CPRINTP" pr="ClientP"/>
                </Attachment>
                <Attachment>
                     <From cc="PUBLICPRINTER" pr="Receive"/>
                     <To cc="CPRINTP" pr="ServerP"/>
                </Attachment>
                <Attachment>
                     <From cc="PS" pr="OutputSecret"/>
                     <To cc="CPRINTS" pr="ClientP"/>
                </Attachment>
                <Attachment>
                     <From cc="SECUREPRINTER" pr="Receive"/>
                     <To cc="CPRINTS" pr="ServerP"/>
                </Attachment>
           </Attachments>

     </Configuration>

</Descriptions>
```