Implementation Issues in the ebXML CPA formation process - the Referencing Problem

Sacha Schlegel*
*Department of Computing
Curtin University of Technology
GPO Box U1987 Perth
Western Australia 6845
Email: sacha@schlegel.li

Abstract—**ebXML is a new framework for electronic business where each trading partner describes its capabilities in a Collaboration Protocol Profile (CPP). Two trading partners form a bilateral agreement, called a Collaboration Protocol Agreement (CPA), based on the two CPP's. This process is called the CPA formation process and can be implemented as a software system; CPP and CPA documents are XML structured documents. During the implementation phase of the system a problem was identified whereby XML elements were found to reference other XML elements within a CPP, with the possibility of overwriting previously accepted checks. This problem can be solved by copying the referenced XML elements prior to running the proposed algorithm. This paper provides an introduction to ebXML and describes where the CPA formation process takes place, explains the referencing problem and provides a subsequent solution. The paper closes with a discussion on the consequences and has an outlook for future work in the area of the CPA formation processes.**

I. INTRODUCTION

Electronic business in the 20th century is a paradigm to utilise computer systems in order to make business easier, more efficient, of higher quality, and to lower costs. Electronic business combines electronic infrastructure (e.g. information software systems, networks, security, and communication) with conventional business.

ebXML (electronic business XML) is a project which provides a framework for global electronic business, where individual trading partners describe their capabilities in a Collaboration Protocol Profile (CPP). For a concrete Business-To-Business (B2B) interaction a Collaboration Protocol Agreement (CPA) is formed from two CPP's. However, the ebXML Collaboration-Protocol Profile and Agreement Specification [1] does not provide a reference solution for the formation of a CPA from two CPP's, only a non-normative description of the algorithm.

The objective of a research was to implement an ebXML specification compliant CPA formation system providing a system which automates the CPA formation process to show that it is possible to develop a CPA formation algorithm. This system consists of three levels. The first level is an algorithmic process which parses two CPP's and creates a draft CPA. As different information elements in both CPP's are compared against each other, the CPA formation algorithm has to check matching elements.

The second level is an automated negotiation system which negotiates the various conflicting elements of the draft CPA, whilst the third level is a web-enabled human to human negotiation system to finalise the draft CPA.

During implementation of the first level of the ebXML specification for the CPA formation process, a problem was identified whereby XML elements were referencing other XML elements within the CPP. A solution for this referencing problem is described in section IV.

This paper is structured as follows. Section II gives an overview of ebXML, provides an ebXML scenario and introduces the CPA formation process. In section III, the CPA formation referencing problem is described in detail and a solution to the described problem is presented in section IV. Section V discusses the research and future considerations.

II. OVERVIEW

ebXML[1] is a joint initiative by the Organisation for the Advancement of Structured Information Standards (OASIS)[2] and the United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT)[3].

ebXML stands for electronic business XML (Extensible Markup Language[4]) and provides a framework for global electronic business. ebXML emphasises collaborative business processes between trading partners and provides the methodologies to describe those processes, to define the choreography of the corresponding messages and how to structure business documents which are based on core components. ebXML builds upon technologies such as XML, HTTP (Hypertext Transport Protocol)[5], SOAP[6] and tries to overcome the short-

---

[1]http://www.ebxml.org
[2]http://www.oasis-open.org
[3]http://www.uncefact.org
[4]http://www.w3c.org/XML/
[5]http://www.w3c.org/Protocols
[6]http://www.soaprpc.com

comings of previous electronic business approaches such as EDIFACT (Electronic Data Interchange for Administration, Commerce and Transport)[7].

The various ebXML specifications describe the different components of the overall ebXML framework; in particular

- the Messaging Specification [2],
- the Business Process Specification [3],
- the Collaboration Protocol Profile and Collaboration Protocol Agreement Specification [1],
- the ebXML Registry Service Specification [4], and
- the Business Process Analysis Worksheets & Guidelines [5].

The ebXML Technical Architecture Specification [6] shows in Figure 1 a typical ebXML scenario.

1) Company A browses the ebXML registry to see what is available online. At best, company A can reuse all the existing business processes, documents, and core components common to its industry that are already stored in the ebXML registry. Otherwise company A designs the missing parts, stores them in the ebXML registry and makes them available for its industry partners.

2) Company A decides to do electronic business the ebXML way and considers implementing a local ebXML compliant application. An ebXML Business Service Interface (BSI) provides the link between the company and the outside ebXML world. The company has to create a Collaboration Protocol Profile (CPP) which describes the supported business process capabilities, constraints and technical ebXML information such as choice of encryption algorithms, encryption certificates and choice of transport protocols.

3) Company A submits its CPP to the ebXML registry. From that point on, company A is publicly listed in the ebXML registry and is likely to be discovered by other companies looking/querying for new trading partners.

4) Company B is already registered at the ebXML registry and is looking for new trading partners. Company B queries the ebXML registry and receives the CPP of company A. Company B then has two CPP's: Company A's CPP and its own. The two companies have to come to an agreement on how to do business. This is called a Collaboration Protocol Agreement (CPA) in the ebXML terminology. Company B uses an ebXML CPA formation tool to derive a CPA from the requirements of the two CPP's.

5) In this scenario company B communicates with company A directly and sends the newly created CPA for acceptance to company A. Upon agreement of the CPA by company A both companies are ready for electronic business.

6) The companies then use the underlying ebXML framework and exchange business documents conforming to the CPA. This means that both companies follow the business processes defined in the CPA.

The 'CPA formation process' in the 5th action of Figure 1 is an important part. Without a valid CPA there is no agreement and without an agreement there is no electronic business.

A CPP represents a companies capabilities whereas the CPA represents the final agreement on how to conduct the collaborative business process. Both documents are structured XML documents with their corresponding XML schemas [8].

Of importance to this study is the question "How to form a CPA from two CPP's automatically". One approach would be to take the two CPP's and form a CPA by hand. Of course this is not the ideal solution and there is a need to have an algorithm (software program) that does the repetitive, tedious, simple parts of forming a CPA from two CPP's. The best case is, of course, if there are no problems and a valid CPA is formed by the algorithm. Unfortunately this is very unlikely. In the case where some problems occur, the outcome of the algorithm is called a draft CPA and human to human intervention is necessary to finalise the draft CPA to a valid CPA.

The ebXML project website provides two sample CPP's and a sample CPA. These documents are used as the main input (test cases) for the development of the algorithm of CPA formation.

The pseudo code in Figure 2 shows the outline of the algorithm. The algorithm takes the two CPP's and checks each XML element/attribute from one CPP against the XML element (or attribute) of the other CPP. For example if one CPP plays the "buyer" role in the request order business process and the other CPP also play the "buyer" role in the same business process there is a problem. Also if one CPP references a "Query Product Information" business process and the other CPP references a "Request Order" business process than there is again a problem.

```
open cpp-1, cpp-2
 while get element from cpp-1, cpp-2
  if elements complement
    write elements to draft-cpa
  else
    write dummy elements to draft-cpa
    write conflict to conflict-file
  end
end
close cpp-1, cpp-2, conflict-file, draft-cpa
```

Fig. 2. CPA formation algorithm pseudo code

III. THE REFERENCING PROBLEM

Throughout the CPP document XML elements reference other XML elements. This technique circumvents the need to "copy & paste" data, eg removes duplicate parts. This XML element referencing allows multiple problematic overwriting of values during the CPA formation process.

[7]http://www.unece.org/trade/untdid/welcome.htm

[8]http://www.ebxml.org provides the latest XML CPP, CPA schemas in the specification section .
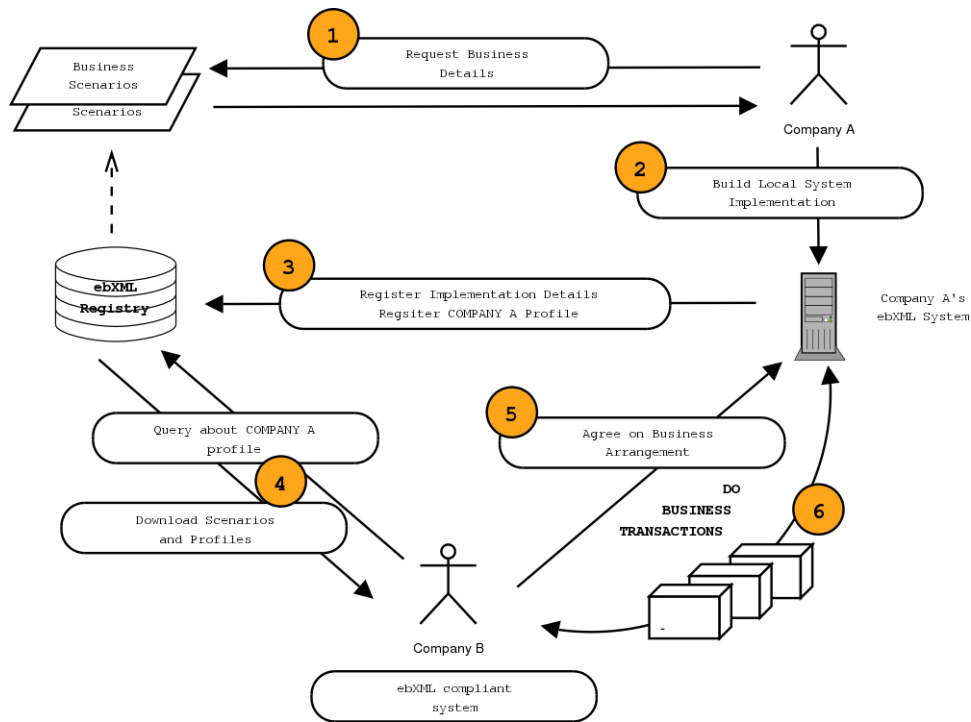
Fig. 1. The classic ebXML scenario (adapted from the ebXML Technical Architecture specification [6]).

This problem can be compared with the pointer aliasing problem [7] which is illustrated with ruby[9] code in Figure 3.

```
my-object = String.new(''Hello World'')
p-one = my-var
p-two = my-var
p-one = String.new(''Ooops'')
```

Fig. 3. The pointer aliasing problem

The example in Figure 3 has a String object called $my-object$ and is initialised to $'HelloWorld'$. The variables $p-one$ and $p-two$ reference the $my-object$ object. When changing the value of the $p-one$ reference to the value $'Ooops'$ also the value of reference $p-two$ and the main String object $my-object$ change.

An enhanced algorithm suggests changes to some elements in the input CPP's to avoid conflicts. The discussion in section V will talk about the validity of this approach, to change CPP's.

This paper does not use the exact CPP and CPA structure to illustrate a concrete implementation problem but uses simplified, mock XML structures. To illustrate the referencing problem the outline of two very simplified CPP's are described. The first CPP is shown in Figure 4.

This sample CPP with id 1 in Figure 4 holds some other XML elements like msg-1, msg-2, msg-3 and two protocol elements. These msg elements are used to show which protocol is used when sending/receiving these msg. The msg number

[9]Ruby is a 100% object oriented programming language and can be found at http://www.ruby-lang.org/en/

(1, 2, and 3) indicate the sequence of the messages. Msg-1 references protocol 'a' and both msg-2 and msg-3 reference protocol 'b'. If more than one msg uses the same protocol, it is easier to *reference* the protocol element instead of having the information about the protocol within each msg element.

The second CPP is shown in Figure 5 where we see that msg-1 references protocol 'z', msg-2 references protocol 'y', and msg-3 references protocol 'x'. [10]

The CPA formation process algorithm has to check element against element to determine any conflicts. Table I shows what the algorithm will check; msg-1 against msg-1, msg-2 against msg-2 and msg-3 against msg-3.

Because the msg elements reference protocol elements (see Figure 4 and Figure 5 for the CPP's), the algorithm in fact has

[10]In the official ebXML CPP's the DeliveryChannel XML element references DocumentExchange XML elements.

```
<cpp id=''1''>
  <msg-1 ref=''a''/>
  <msg-2 ref=''b''/>
  <msg-3 ref=''b''/>
  <protocol id=''a''>
    <name>FTP</name>
  </protocol>
  <protocol id=''b''>
    <name>HTTP</name>
    <version>1.1</version>
  </protocol>
</cpp>
```

Fig. 4. First CPP with id 1.

cpp '1'/cpp '2'	msg-1	msg-2	msg-3
msg-1	1		
msg-2		1	
msg-3			1

to compare the corresponding protocol elements against each other.

cpp '1'/cpp '2'	protocol 'a'	protocol 'b'
protocol 'z'	1	
protocol 'y'		1
protocol 'x'		1

The algorithm creates a matrix of the protocols (see Table II) to determine which elements (protocols) have to be checked against each other.

1) The first test case is protocol 'a' against protocol 'z'. The name of both protocol elements is "FTP" so this first check is OK.
2) Then the algorithm has to check protocol 'b' against protocol 'y'. Name of both protocol elements is "HTTP" and their version is "1.1"; again this check is OK.
3) Then the algorithm checks protocol 'b' against protocol 'x'. Both protocol names are HTTP which is OK *but* the version differ between "1.1" and "1.0". This is clearly a conflict. The enhanced algorithm suggests to change the version in protocol 'b' from "1.1" to "1.0" (the discussion in section V questions why the algorithm changes protocol 'b' and not protocol 'x').

A problem occurs when the algorithm changes values of protocol 'b'. It seems that with the change from the check of

```
<cpp id=''2''>
  <msg-1 ref=''z''/>
  <msg-2 ref=''y''/>
  <msg-3 ref=''x''/>
  <protocol id=''z''>
    <name>FTP</name>
  </protocol>
  <protocol id=''y''>
    <name>HTTP</name>
    <version>1.1</version>
  </protocol>
  <protocol id=''x''>
    <name>HTTP</name>
    <version>1.0</version>
  </protocol>
</cpp>
```

Fig. 5.   Second CPP with id 2.

msg-3 against msg-3 of the two different CPP's the algorithm could make a match by changing the protocol version value in the original cpp with id '1'. This would now seem to make the formation of the CPA possible. But with that change, the algorithm has negated the previous comparison of msg-2 against msg-2 which previously showed protocol 'b' against protocol 'y' and which was OK. Now the version of protocol 'b' is "1.0" and the version of protocol 'y' is "1.1".

So the changing of msg-3 has also caused a change in msg-2. Actually the algorithm changed each element which references protocol 'b'. This referencing can become arbitrary complex.

A solution for this referencing problem is shown below.

IV. A REFERENCING SOLUTION

The referencing problem can be bypassed by first copying the referenced object and applying changes to the copy only.

The matrix in Table II shows that protocol 'a' is compared against protocol 'z', protocol 'b' is compared against protocol 'y' and protocol 'x'. We see the potential problem case where protocol 'b' is compared against protocol 'y' and protocol 'x'.

The algorithm transforms the matrix into a new matrix without potential problems. Table III shows the transformed matrix.

cpp '1'/cpp '2'	protocol 'a'	protocol 'b'	protocol 'b-copy'
protocol 'z'	1		
protocol 'y'		1	
protocol 'x'			1

During the transformation the algorithm makes a copy of each problematic protocol. The problematic protocol is copied the number of times it is compared against other protocols. In this example only protocol 'b' is compared against more than one other protocol, so protocol 'b' is copied to protocol 'b-copy'. The algorithm does the transformation in x and y direction.

The algorithm has to update all the places where the protocol 'b' was referenced. This in fact does also modify the original CPP which is discussed in the discussion section V. In our example element msg-3 in cpp '1' no longer references protocol 'b' but protocol 'b-copy'. Theoretically the new/updated cpp '1' is shown in Figure 6.

The algorithm then runs and checks each pair as described previously. In our example protocol 'a' against protocol 'z' ("FTP" against "FTP", without any version) is OK. Then protocol 'a' against protocol 'y' ("HTTP" against "HTTP" with version "1.1" against "1.1") is also OK. Last protocol 'b-copy' is checked against protocol 'x' where we have a conflict in the version between "1.1" and "1.0". The algorithm now changes the version in protocol 'b-copy' from "1.0" to "1.1". After the element against element checking cpp with id

'1' looks as shown in Figure 7. This time the change did not overwrite previously accepted comparisons. For completeness the resulting draft CPA is shown in Figure 8.

The resulting CPA holds the two CPP's plus the extra corresponding attribute in each msg element. The corresponding attribute indicates which msg elements corresponds to each other.

Because the algorithm does copy protocol 'b' to protocol 'b-copy' without first checking if it is necessary at all to copy protocol 'b'. In case of no conflict when checking protocol 'x' against protocol 'b' we have protocol 'b-copy' which is the same as protocol 'b'. In that case protocol 'b-copy' is simply superfluous and can be removed again. Also all the updated references to protocol 'b-copy' have to be changed back so they reference again protocol 'b'.

Table IV shows another issue where a protocol 'a' is checked three times against protocol 'z'. If there are no conflicts between these two protocol elements then there is no problem but if there is a conflict then we have a problem. If we change protocol 'a' or protocol 'z' once then we change it for all three occurances. This is may not be in the interest of the CPP owner so here we have to provide an option to let the user choose whether to differentiate between the three occurances. The solution (matrix transformation) is the same as mentioned before.

TABLE IV
NUMBER OF COMPARISONS IS BIGGER THAN ONE.

cpp '1'/cpp '2'	protocol 'a'	protocol 'b'
protocol 'z'	3	
protocol 'y'		1
protocol 'x'		1

V. DISCUSSION

This paper gave an introduction to an ebXML scenario and showed at which stage the CPA formation process takes place. The paper outlined the formation algorithm in pseudo code and introduced the referencing problem. Further the paper provided a solution to the CPA formation process referencing problem.

The justification for changing a CPP needs discussion. To alter a CPP is a sensitive decision. Company A in the example ebXML scenario from Figure 1 carefully crafted its CPP. The

```
<cpp id=''1''>
  <msg-1 ref=''a''/>
  <msg-2 ref=''b''/>
  <msg-3 ref=''b-copy''/>
  <protocol id=''a''>
    <name>FTP</name>
  </protocol>
  <protocol id=''b''>
    <name>HTTP</name>
    <version>1.1</version>
  </protocol>
  <protocol id=''b-copy''>
    <name>HTTP</name>
    <version>1.1</version>
  </protocol>
</cpp>
```

Fig. 6.   CPP '1' with a new protocol 'b-copy'.

```
<cpp id=''1''>
  <msg-1 ref=''a''/>
  <msg-2 ref=''b''/>
  <msg-3 ref=''b-copy''/>
  <protocol id=''a''>
    <name>FTP</name>
  </protocol>
  <protocol id=''b''>
    <name>HTTP</name>
    <version>1.1</version>
  </protocol>
  <protocol id=''b-copy''>
    <name>HTTP</name>
    <version>1.0</version>
  </protocol>
</cpp>
```

Fig. 7.   CPP '1' with an updated version of protocol 'b-copy'.

```
<cpa id=''111''>

  <cpp id=''1''>
    <msg-1 ref=''a''corresponding=''z''/>
    <msg-2 ref=''b''corresponding=''y''/>
    <msg-3 ref=''b-copy'' corresponding=''x''/>
    <protocol id=''a''>
      <name>FTP</name>
    </protocol>
    <protocol id=''b''>
      <name>HTTP</name>
      <version>1.1</version>
    </protocol>
    <protocol id=''b-copy''>
      <name>HTTP</name>
      <version>1.0</version>
    </protocol>
  </cpp>

  <cpp id=''2''>
    <msg-1 ref=''z'' corresponding=''a''/>
    <msg-2 ref=''y'' corresponding=''b''/>
    <msg-3 ref=''x'' corresponding=''b-copy''/>
    <protocol id=''z''>
      <name>FTP</name>
    </protocol>
    <protocol id=''y''>
      <name>HTTP</name>
      <version>1.1</version>
    </protocol>
    <protocol id=''x''>
      <name>HTTP</name>
      <version>1.0</version>
    </protocol>
  </cpp>

</cpa>
```

Fig. 8.   The resulting CPA.

undertaking of writing a valid CPP needs valuable time and is a major achievement. If the CPA formation process algorithm changes that CPP of company A in order to avoid conflicts this might not be acceptable at all for company A. On the other hand, if neither company A nor company B change their CPP there will never be a valid CPA, so one of the input CPP's (or both) has to be adapted to allow an agreement (CPA).

To remove the need for changing the original CPP, the CPA formation process does not change the original CPP but suggests a new, updated CPP to its owner. The new CPP is then subject to be accepted or rejected.

The consequences of changing the CPP's does allow the CPA formation algorithm to get one step closer to a valid CPA.

The current algorithm has certain limitations such as missing XML elements of the CPP's being checked against each other or being untested with real world CPP's for robustness and stability.

An interesting question is where the formation process shall take place. In our example it was company B which ran the CPA formation process. With different CPA formation process tool providers, the CPA formation process algorithm will most likely be different. The use of different algorithms can result in different output draft CPAs. There is even the possibility to have algorithms which are developed to favour one CPP over the other. If in our example company B uses a "company B" favouring algorithm it could be a disadvantage for company A to accept the resulting draft CPA. This leads to the idea of letting the ebXML registry provide the CPA formation process service. If the ebXML registry provides the CPA formation process algorithm that could increase the overall trust in a fair the CPA formation process.

Further could the CPA formation process use some learning techniques. The algorithm could for example keep track of the common problems between CPP's or simply keep track of the most used protocols or most used encryption algorithms. If the ebXML registry would provide the CPA formation process algorithm, the algorithm would be used much more often compared to the algorithm of each company and thus would get much more valuable data. This collected data could then be shared with the owners of the stored CPP's so they can go over their own CPP's and maybe adjust their CPP's to the "most common practices".

Section IV mentioned, that after having copied protocol 'b' to protocol 'b-copy' the algorithm might change back to referencing protocol 'b' if there are no differences between protocol 'b' and protocol 'b-copy'. The current algorithm blindly copies protocol 'b' to protocol 'b-copy' in the example without first checking if there are any conflicts. The alternative method is to go ahead as described and only in the case of a problem copy the conflicting elements (protocol 'b' in this example). Considering an ebXML registry with hundreds or even thousands of CPP's forming a CPA of all available CPP's (or a selection of them) the CPA formation process will become a performance issue. In the ebXML registry CPA formation scenario it could be a performance decision to first

remove all references (as it is done in this algorithm) and then compare CPP against CPP. This is definitely an interesting question and subject to future work.

Future work is also needed for other issues such as nested referencing problems where element X references element Y which references element Z or which CPP of the two shall be updated as both CPP's are subject to change.

The algorithm can be tested online at `http://www.schlegel.li/ebXML/`. The source code is licenced under the GNU GPL (GNU General Public Licence) Version 2.0 [11].

REFERENCES

[1] ebXML Trading-Partners Team, "ebxml collaboration-protocol profile and agreement specification v.2.0," September 2002. [Online]. Available: http://www.ebxml.org/specs/ebcpp-2.0.pdf
[2] ebXML Transport-Routing-Packaging Project Team, "ebxml message service specification v.2.0," April 2002. [Online]. Available: http://www.ebxml.org/specs/ebMS2.pdf
[3] Business Process Project Team, "ebxml business process specification schema v.1.0.1," May 2001. [Online]. Available: http://www.ebxml.org/specs/ebBPSS.pdf
[4] ebXML Registry Project Team, "ebxml registry services specification v.2.0," December 2001. [Online]. Available: http://www.ebxml.org/specs/ebrs2.pdf
[5] Business Process Project Team, "ebxml business process analysis worksheets & guidlines v.1.0," May 2001. [Online]. Available: http://www.ebxml.org/specs/bpWS.pdf
[6] ebXML Technical Architecture Project Team, "ebxml technical architecture specification v.1.0.4," February 2001. [Online]. Available: http://www.ebxml.org/specs/ebTA.pdf
[7] W. Landi, "Interprocedural Aliasing in the Presence of Pointers, Tech. Rep. lcsr-tr-174, 1991.

[11] http://www.gnu.org/licenses/gpl.html