Automated Generation of Electronic Procedures: Procedure Constraint Grammars

Ronald M. Lee
email: rlee@fac.fbk.eur.nl
EURIDIS, Erasmus University Rotterdam, PO 1738, 3000 DR Rotterdam, the Netherlands
http://www.euridis.fbk.eur.nl/Euridis/

Abstract

This paper focuses on a generalization of the notion of electronic trade procedures. Instead of designing specific trade procedures for each possible variation of a transaction, we develop a generative grammar formalism that defines entire families of related trade procedures. Since trade procedures are influenced by various situation-specific factors from multiple sources, we have been led to modify the usual grammar formalism to take into account a variety of constraints. The formalism is therefore called a procedure constraint grammar (PCG). The resulting functionality allows for a far greater level of flexibility to customized electronic trade procedures to specific contracting situations. Numerous examples are provided.

Keywords: electronic commerce; electronic trade procedures; transaction modeling; Open-EDI; EDI; Petri-nets

1. Procedural Controls for Open Electronic Commerce

Global information infrastructures have become a reality. Such worldwide networks help companies to operate not only on a local or regional level, but also on a global level. Especially for small and medium enterprises (SMEs), this would offer tremendous opportunities to do global business electronically. However, communication networks alone are not sufficient to enable international electronic trade.

In the past it has been shown that the introduction of Electronic Data Interchange (EDI) can offer important benefits for the efficiency of trade and accuracy of trade, especially for business-to-business electronic commerce.

On the other hand it can also been observed that EDI usually requires substantial implementation efforts, which must be re-programmed for each new type of transaction.

As a result, most successful EDI implementations have been realized in what could be called 'closed trading relationships', i.e. long-lasting trading relationships, involving a high number of transactions, between parties that have a high level of trust and possibly a close coordination of the parties' business processes.

However, when the partnership is established for a limited period, covering a few transactions only and on an "at arms' length" basis, EDI linkages are seldom worthwhile since the costs of the necessary implementation cannot be recovered from the benefits. These shorter-term partnerships could be called 'open trading relationships'. The main aim of our research is to contribute to the lowering of the barriers for using EDI in these open trading relationships.

One of the reasons for the complexity of this implementation process is the fact that parties have to know about each others’ “way of doing business” before they can start exchanging data electronically. Extra knowledge about the preferred way of doing business of one trading partner has to be conveyed to the other; in other words, the parties have to agree upon the trade procedure they are going to follow. We define a trade procedure as the mutually agreed upon set of actions and associated rules that governs the activities of all parties involved in a set of related business transactions. Thus, a trade procedure controls all interactions between the roles involved. A trade procedure stipulates which actions should be undertaken by which parties, the order in which these actions should be performed and possibly the timing constraints on the performance of these actions. Actions of parties include the sending and/or receiving of goods, documents or funds.

The need and usefulness of trade procedures is easy to demonstrate. Consider only a simple post-payment contract for goods. The buyer assumes that an invoice will be sent after delivery to trigger the payment obligation. The seller, on the other hand, abides by the practice that payment becomes due from the time of delivery, and does not send an invoice. Thus, the goods arrive, and the buyer does not pay, waiting for an invoice. Meanwhile the seller becomes irked, and initiates collection proceedings. This is an example of the so-called 'battle of the forms'. Each party utilizes standardized documents such as a purchase order, delivery agreement, etc. which contain (typically on the backside, in small print) the terms and conditions that are their style of doing business. Unfortunately, the small print is often ignored by the receiving party.

For trade in a well-established industry area, standardized practice becomes generally accepted, and there usually is not a problem1. However, in more open trading

1In some cases, guidelines by international bodies such as the International Chamber of Commerce or the UNCID have been issued to diminish these ambiguities (an example is the Uniform Customs and Practices for Documentary Credits, issued by the ICC [8,9]).
situations, that cross national, cultural or sectorial boundaries, such conflicts are much more likely to arise.

1.2 Trade Scenarios

One approach to decrease these negotiation costs is to define standard trade procedures. Although EDI messages can be structured using an international standard like UN/EDIFACT or ANSI X.12, there are no standards yet for the semantics and context of those messages, i.e. the business scenarios that describe the trade procedures used by the several parties involved in a business transaction [31, 10]. For example, the type of response to the receipt of a purchase order can differ from company to company; one company might reply with a purchase order acknowledgment, another company might reply with a shipping notice.

Many existing EDI applications of course embed the types of document exchange sequencing of a trade procedure. However, these sequences are normally 'hard coded' into the application programs, as specified in the terms of the trading partner agreement [23], a legal, textual document. A key aspect of the architecture presented here is that trade procedures are generic, in a declarative, rule-based form [3]. This has the virtue that they may be down-loaded from e.g. a central library to meet the needs of a particular contractual situation.

More significant, however, is that such procedures can be analyzed and managed using computational tools. For example, analytical techniques can be applied to check for formal correctness (boundedness, etc.), as well as fraud potential and other audit controls. Further, soft-coding allows for the representation of generic models that are parameterized for specific circumstances. Additionally, generic trade procedures enable the navigation, synthesis and negotiation of procedures from different trading sectors or regulatory environments. These aspects are discussed further, below.

2. Representation: Documentary Petri Nets

A central problem for the management of electronic trade procedure is the question of how they should be represented both from the modeler's perspective, and as well as from a computational perspective. The representation we have developed (over nearly a decade) is an extended form of petri net, called a documentary petri net (DPN) [3,4,5,16,18,20,21,22]. The main extensions over classical petri nets are as follows.

One important aspect of modeling complex scenarios is the ability to model roles as separate documentary petri nets. This allows the decomposition of an trade procedure into a number of logically separate sub-nets. This modeling style results in a clear "geographical" separation between the roles -- both visually, as well as literally, in actual operation [20].

In documentary petri nets transitions represent actions, labeled to indicate the agent (role) as well as the kind of action performed:

\texttt{<role> : <action>}

Various place types and predicate labels are also used in documentary petri nets to specify the different document types, goods, funds and deontic states.

- The conventional places in a classical petri net are called control places in DPN. These model the flow of control with a particular role in the transaction.
- The sending and receiving of of documents is modeled using document places. A document place is represented by a square box. These kind of places have labels that identify the sender/recipient role, as well as one or more document types (thus, a collection of documents may also be sent as a single unit).
- The sending and receiving of funds is modeled in the same fashion as other document exchanges. Since the concept of money is closely related to documents (a 100 dollar bill is a performative document), we use the document places to denote funds transfer. In the description of these documents, the amount and currency are specified in the structure of these documents.
- The sending and receiving of physical goods is also represented in a similar way to documents, though the place is sometimes drawn as a cube or special icon for visual distinction. Nonetheless, semantically, they function in the same way as document places in the DPN.

\footnote{Petri nets are a graphical representation for modeling discrete concurrent processes (like a PERT or marked graph), that also represent choice (like a state-transition diagram, or decision tree.) A classical petri net is a bipartite, directed graph. It has two disjoint sets of nodes: places (represented as circles) and transitions (represented as bars). Arcs connect places with transitions or vice versa (it is not allowed to connect two places or two transitions). The dynamic behavior of the modeled system is represented by tokens flowing through the net (represented as dots). Each place may contain several tokens (the marking of the place); a transition is enabled if all its input places (i.e., arcs exist from those places to the transition) contain at least one token. If this is the case, the transition removes one token from each input place and instantaneously produces one in each output place (i.e., an arc exists from the transition to the place). This is called the 'firing' of a transition. For further background see e.g. [24,25] extended forms of Petri nets are discussed in [1,6,7,32].}

\footnote{We use the term 'procedure' to refer to the formalized, computable sequence of document exchanges and related deductions; the term 'scenario' is used in a more informal and generic sense, referring not only to such procedures, but also to related informal explanations and contextualizations.}

\footnote{In the terminology of programming languages, this is like the distinction between interpreted (soft) vs compiled code. In AI terms, these are declarative representations, as used for instance in expert systems [20].}
• Deontic states are represented as predicate labels on a control place. As such, they are local to a role. The marking of the control place indicates the deontic predicate becomes true. This state persists until it is explicitly negated by another deontic state predicate.

A DPN transition (action) is enabled when its input places (control places or document places) are marked (darkened), indicating the fulfillment of its pre-conditions. Firing the transition causes its input places to be unmarked, and all of its output places to become marked, indicating the initialization of its post-conditions.

There are several variations in the DPN notation. One form, called a linked DPN, shows the entire transaction in a single graph, as shown in Figure 1. In this example the Buyer sends a request for quotation to the Seller, who prepares the quote and sends it to the Buyer. An alternate notation, called role DPNs, uses a separate sub-net for each role. This is shown in Figures 2a and 2b for the same transaction. It is the role DPN form that is used for distributing scenarios among geographically remote contracting parties. A feature of this form is that each of the role scenarios executes autonomously, with no central control of the transaction. The only coordinating communications are done by means of the (electronic) documents. This accurately reflects the legal assumptions for contracting and trade procedures.

3. Constraint-Based Generation of Electronic Trade Scenarios

A limitation of the manual design approach (indeed, more generally of other open EDI approaches) is that the scenarios produced are fixed; that is, they cannot be adapted or adjusted to meet additional needs of a given situation. In this part of the project, we address this problem with an expert system approach, by which scenario components are broken down into reusable component parts, which can be flexibly reassembled to meet the needs of a wide variety of situations.

The computational formalism we introduce is called a procedure constraint grammar (PCG). As its name suggests, an objective of the PCG representation is to describe procedures by their temporal ordering constraints, rather than the absolute sequence of steps. This allows for more flexible re-combination of procedural components (doing and control tasks).

Using a procedure constraint grammar, the user interacts with the system, specifying constraints and objectives of the contracting situation. Based on these specifications, the system composes a trade procedure, which is presented in graphical form, and which can then be compiled and simulated. Here, the term ‘grammar’ is used in the linguistic sense of generative grammars, i.e. a set of rules for generating syntactically correct or well-formed sentences in a language. The objective of PCG rules is to generate procedures that are not only well-formed syntactically, but also from a control standpoint. In this aspect, a procedure constraint grammar operates like an expert system shell that may be used to develop knowledge bases about contracting and associated legal and documentary requirements.

Unlike language grammars, however, which are typically represented as an integrated hierarchy of rules, PCG’s are organized as constraints on a target procedure. It is the job of the PCG constraint solver to identify a (minimal) solution procedure (according to some preference ordering of the user -- e.g. minimal duration vs minimal risk).
ordering of actions may be either explicit, or implicit, relative to some state condition. These temporal ordering constraints use the following notation:

* explicit: \(<\text{action-label}> .<. \text{action-label}>\)
* implicit: \(<\text{condition}> .<. \text{action-label}>\)

The actions in the constraints may be either basic or non-basic, defined in terms of other constraints.

Based on these constraint specifications, the constraint solver expands non-basic constraints into lowest level, basic constraints. It then creates a partial ordering of actions based on explicit temporal ordering constraints. It then adds additional constraints to satisfy implicit temporal ordering constraints, based on state conditions (like a state space search in AI). Where multiple actions need to occur simultaneously, these are replaced by a joint action.

4. PCG Examples

4.1 Example 1: get quotation

As a starting example we present a tiny grammar that generates the example DPN procedure introduced previously:

```plaintext
[Buyer, Seller]: get_quotation ==>
[Buyer, Seller]: transmit([request_quote]) <<
[Seller, Buyer]: transmit([quote])
```

Other approaches to this objective have been considered, but proved not to be sufficient. These include a sub-procedure approach, where actions of the trade procedure can refer to other substitutable procedures. This proved too inflexible in that the situational variations often need to include a combination from different sources of procedural knowledge: those related to the task; those relating to controls; and those relating to the specific communications media employed. We also considered an object-oriented approach, but found it to have similar difficulties. Object oriented methods handle procedural knowledge mainly by overriding routines in the parent procedure. For our purposes, this is much like the sub-procedure approach. Additionally, the requirement of multiple knowledge sources leads to multiple inheritance, with the associated difficulties of contention.

In our current approach, knowledge specified as independent 'constraints', which can be at arbitrary levels of abstraction. Where different levels of abstraction need to be combined, this is handled by the constraint solver by expanding them to a common lowest level of detail. Where contentions occur, these are dealt with by preference orderings, which are specified in the knowledge base.

Actions are considered to be temporally unordered (concurrent), unless specified in the constraints. The
the linked-DPN, a single-flow DPN shows the activities of all roles in a single graph, but compressed to show just a single flow of transaction control. In order to reduce the detail of the graph, paired actions, such as 'transmit' above, are not expanded, but presented as a single action.

**Figure 4. Example 1: Single-flow DPN for 'get_quotation'**

### 4.2 Example 2: choice of payment mode

This example illustrates how parameters may be inserted in the grammar to provide customization options to the end user. In this example the parameter '@pay_mode' offers three different choices of payment. Figure 5 shows a dialog where the user selects a single choice. The resulting procedure is shown in Figure 6.

```plaintext
[Buyer, Seller]: postpaid_sale(Goods, Price) =>
    [Seller, Buyer]: transmit([Goods]) <<
    [Buyer, Seller]: payment(Price).
/* further payment details */

[X,Y]: payment(Price) =>
    { @pay_mode == 'bank_check'},
    [X,Y]: transmit(bank_check(Price)).

[X,Y]: payment(Price) =>
    { @pay_mode == 'money_order'},
    [X,Y]: transmit(money_order(Price)).

[X,Y]: payment(Price) =>
    { @pay_mode == 'electronic_funds_transfer'},
    [X,Y]: transmit(digital_cash(Price)).
```

**Figure 5. Example 2: Dialog for choice of payment mode**

**Figure 6. Example 2: Expanded procedure after single choice**

### 4.3 Example 3: multiple choices of payment mode

This example illustrates the difference between design time choices and execution time choices. By selecting multiple modes at design time, the generated procedure will offer these choices to the end-user when the procedure is executed.

```plaintext
[Buyer, Seller]: transmit([Goods])

[Buyer, Seller]: transmit([money_order(Price)])

[Buyer, Seller]: transmit([digital_cash(Price)])
```

**Figure 7. Example 3: Dialog for multiple choices of payment mode**
4.4 Example 4: sales contract with carrier subcontract

The following scenario grammar illustrates a concurrency situation. The resulting procedure is shown in Figure 9.

\[
\begin{align*}
\text{[Buyer, Seller, Carrier]}: & \text{carrier_contract(Goods, SalePrice, DeliveryPrice) } \Rightarrow \\
A & \text{:: Seller to Carrier: } \\
& \text{delivery_order(Goods, Buyer, DeliveryPrice),} \\
B & \text{:: Carrier: transport(loc(Buyer), Goods),} \\
C & \text{:: Seller to Carrier: payment(DeliveryPrice),} \\
D & \text{:: Buyer to Seller: payment(SalePrice),} \\
& \text{A } .<. B, B .<. C, B .<. D.
\end{align*}
\]

4.5 Example 5. transport contract with performance evidence requirement

The following scenario grammar illustrates heuristic search, based on evidence requirements, to assemble the controls of the procedure.

\[
\begin{align*}
\text{[Seller, Carrier]}: & \text{carrier_transport(Goods, DelPrice) } \Rightarrow \\
A & \text{:: Carrier: delivery(loc(Buyer), Goods),} \\
B & \text{:: Seller to Carrier: payment_for(A, DelPrice).}
\end{align*}
\]

\[/* \text{‘payment_for’ constraint}*/
\]

\[
\text{X to Y: payment_for(Action, Amt) } \Rightarrow \\
A & \text{:: X to Y: payment(Amt),} \\
& \text{has(X, evidence(Action)). } .<. A.
\]

Figure 8. Example 3: Expanded procedure after multiple choice

Figure 9. Example 4: Expanded procedure with concurrency

Figure 10. Example 5: Initial constraints
4.6 Example 6: documentary credit

A documentary credit is a common trade procedure used when goods are being transported long distances, and the seller (in the role of Shipper) wants to be paid immediately, whereas the buyer (in the role of Consignee) does not want to pay until the goods arrive. Guidelines for this procedure are set forth in the Uniform Customs and Practice for Documentary Credits [9]. In the simplified form presented here, a credit for the price of the goods is made by a bank, called the Issuing Bank (typically in the buyer’s country), which is conveyed to the Shipper (seller) by another bank, called the Coordinating Bank. A sketch of the document flows is shown in Figure 13. A scenario grammar is as follows:

\[
\begin{align*}
\text{[Buyer, Seller, Carrier, Bank]} : & \quad \text{letter_credit_payment(Goods, Price, DelPrice, Interest) } \\
& \Rightarrow \\
A & : \quad \text{Seller to Carrier: dispatch(Goods)}, \\
B & : \quad \text{Bank to Seller: payment_for(A, Price)}, \\
C & : \quad \text{Carrier: transport(whse(city(Buyer)), Goods)}, \\
D & : \quad \text{Carrier to Buyer: notify(A)}, \\
E & : \quad \text{Buyer to Bank: payment(Price + Interest)}, \\
F & : \quad \text{Carrier to Buyer: hand_over Goods}, \\
G & : \quad \text{Seller to Carrier: payment_for(F, DelPrice)}, \\
D & .<. E, \\
& \quad \text{has(Carrier, evidence(own(Buyer, Goods)))) .<. F}. \\
\end{align*}
\]

/* documentary evidence rules */

- \text{evidence_for X to C: dispatch(Goods)}
  is \text{doc bill_lading(C, Goods)}.
- \text{evidence_for own(X, Z)}
  is \text{doc title(X, Z)}.
- \text{evidence_for X to Y: pchange(Z)}
  is \text{doc receipt(Y, Z)}.

---

Figure 11. Example 5: Add transmit of receipt

<table>
<thead>
<tr>
<th>Carrier: delivery(loc(Buyer), Goods))</th>
</tr>
</thead>
<tbody>
<tr>
<td>- has(Buyer, Goods)</td>
</tr>
<tr>
<td>= has(Carrier, receipt(Buyer, Goods))</td>
</tr>
</tbody>
</table>

Figure 12. Example 5: Final procedure, after replace delivery constraint.

<table>
<thead>
<tr>
<th>Carrier: delivery(loc(Buyer), Goods)) &amp; [Buyer, Carrier]: transmit([receipt(Buyer, Goods))]</th>
</tr>
</thead>
<tbody>
<tr>
<td>= has(Buyer, Goods), hasCarrier, receipt(Buyer, Goods))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Carrier: delivery(loc(Buyer), Goods)) &amp; [Buyer, Carrier]: transmit([receipt(Buyer, Goods))]</th>
</tr>
</thead>
<tbody>
<tr>
<td>= has(Buyer, Goods), hasCarrier, receipt(Buyer, Goods))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Carrier: delivery(loc(Buyer), Goods)) &amp; [Buyer, Carrier]: transmit([receipt(Buyer, Goods))]</th>
</tr>
</thead>
<tbody>
<tr>
<td>= has(Buyer, Goods), hasCarrier, receipt(Buyer, Goods))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Carrier: delivery(loc(Buyer), Goods)) &amp; [Buyer, Carrier]: transmit([receipt(Buyer, Goods))]</th>
</tr>
</thead>
<tbody>
<tr>
<td>= has(Buyer, Goods), hasCarrier, receipt(Buyer, Goods))</td>
</tr>
</tbody>
</table>

Figure 13. Example 6: Roles and Document Flows for Documentary Credit
/* official copy */

evidence for Cond_or_Action is ocopy Doc
   if evidence for Cond_or_Action is Doc.

/* notify constraint */

X to Y: notify(X: transport(Loc, Goods) ) =>
   X to Y: send doc arrival_notice(Loc, Goods).

---

Figure 14. Example 6, step 0: initial constraint set

Figure 15. Example 6, step 1: add bill of lading as evidence of dispatch

Figure 16. Example 6, step 2: add bill of lading as evidence of ownership.
Figure 17. Example 6, Step 3: Final procedure, after add receipt as evidence of delivery

5. Concluding Remarks

The notion of electronic trade scenarios was described as a means of achieving "plug and play" capabilities for complex, business to business trade transactions, especially for international trading. However, such trade scenarios are fixed, and do not allow trading parties the flexibility to adapt them to different circumstances. In this paper, the notion of a scenario grammar was introduced as a generic class of trade transaction, whose variations are generated depending on user-specified parameters. The specific formalism introduced here is called procedure constraint grammar (PCG).

A generalization of this problem is to include the pertinent governmental regulations that apply to the transaction being generated. In the current version, such constraints are presumed to be included in the scenario grammar as additional rules. However, more realistically, one must cope with the problem of 'distributed governance' -- that such regulations are under the control of distant government agencies, and are subject to change. A desirable solution would be to poll these agencies 'on the fly' as the transaction procedure is being synthesized. An approach for this, utilizing computational agents called messengers, builds on the PCG formalism described here [17].

Appendix: PCG Syntax

PCG Syntax:

<constraint-definition> ::=  
<action> ==>  
{<conditions>}
<rule-constraints>.

<constraint-definition> ::= 
<action> ==> 
{<conditions>}
<PN pattern>.

<constraint-definition> ::= 
<action> ==>  
{<conditions>}
from <pre-conditions>
next <post-conditions>.

<rule-constraint> ::= <label> :: <action>  
| <label> :: <label-or-cond> .=. <label-or-cond>  
| <label> :: <label-or-cond> .<. <label-or-cond>  
| <label> :: <label-or-cond> .=<. <label-or-cond>

<rule-constraints> ::= < rule-constraint>  
<rule-constraints> ::= 
< rule-constraint >, <rule-constraints>

<pre-conditions> ::= <conditions>

<post-conditions> ::= <conditions>

<conditions> ::= <condition>
<conditions> ::= <condition>, <conditions>

<condition> ::= <predicate>

<actions> ::= <action>
<actions> ::= <action>, <actions>

<action> ::= <agents> : <condition>

<agents> ::= <agent>
<agents> ::= <agent> to <agent>
<agents> ::= [<agent>, ..., <agent>]

<agent> ::= <variable> | <atom>

<label-or-cond> ::= <label>
<label-or-cond> ::= <condition>

<label> ::= <variable>
References