Verifying Inter-Organisational Trade Procedures by Model Checking Synchronised Petri Nets

Robert Zimmer[*], Aspassia Daskalopulu[**] and Alan Hunter[*]

[*]Brunel University, Department of IS and Computing, Uxbridge, Middlesex UB8 3PH, U.K.
E-mail: {Robert.Zimmer, Alan.Hunter}@brunel.ac.uk
[**]The Open University, Department of Computing, Walton Hall, Milton Keynes MK7 6AA, U.K.
E-mail: A.K.Daskalopulu@open.ac.uk

Abstract

Effective use of business to business electronic commerce requires that the trade procedures of the companies involved in an exchange be well enough understood and modelled to allow possible conflicts to be discovered and resolved during negotiation, before they occur in practice. This paper describes requirements analysis systems that have been useful in doing exactly this kind of modelling and reasoning for microelectronic design. We argue that the same mathematics is applicable to inter-organisational electronic commerce, and that, in fact, some microelectronic design debugging and verification software—interacting Petri net systems and model checkers—can be used to debug and verify trade procedures. We believe that this can lead to safer and more powerful forms of electronic interaction in a commercial setting, as it has in an electronic design setting.

Keywords: Electronic commerce, Formal methods and verification.

1 Introduction: Electronic Design and Electronic Commerce

Business–to–business interaction in a commercial setting is normally implemented within the context of a contract that parties have negotiated and agreed upon. Contractual situations can be identified in business exchanges ranging from the relatively straightforward (for example the purchase of a ticket for a rail journey) to the complex (for example the establishment of a long-term trading agreement between organisations or a complex trading procedure involving third parties). Current technology makes it possible for such contractual situations to be entered into and realised electronically. Contractual situations at the simple end of the scale are typically based on unilateral models of exchange. The transaction is implemented on the basis of the goods or service provider's terms, its duration is relatively short and any negotiation between the parties is limited to agreeing a price, a quantity and a time for delivery. EDI enables such transactions to be implemented efficiently through the exchange of structured messages that are machine processable.

Contractual situations at the complex end of the scale typically involve bilateral agreements and are usually realised over a longer period of time, rather than instantaneously. During negotiation, parties communicate their goals and their preferred ways of operating and agree on a set of rules that will govern their activities while the business exchange is in operation. Such sets of rules are in other words specifications of the parties' (future) behaviour and by extension, specifications of the parties' agreed *ideal* model of their exchange. They characterise deontically the parties' actions during the business exchange; that is, they specify what actions are permitted, obligatory or prohibited by each party under certain circumstances. Typically, such permissions, obligations and prohibitions are temporally bound and sanctions or remedies are specified for the cases where a party's actual behaviour departs from the stipulated one (the ideal).

Researchers in the field of electronic commerce refer to such sets of rules as trade procedures [3], or business protocols [31], legal theorists refer to them as contracts [2], social scientists and philosophers refer to them as norms (cf. [25], [30]) and software and hardware engineers usually refer to them as specifications. We shall use the term 'trade procedures' in the rest of this paper to refer to such rules because this is more familiar to the electronic commerce community. It is however important to remember that such trade procedures describe *ideal* models of inter-organisational exchanges, rather than the *actual* exchanges themselves—in the same spirit that contracts describe how a business exchange *should* happen rather than how it *actually* happens and system or program specifications describe what the system or program *should* do, which may be different from what the actual system or program (the implementation of the specification) *does*.

There are some proposals by researchers in the area of electronic commerce that aim to facilitate the process by which organisations communicate their goals and preferred trade procedures. Such proposals stem from the observation that trade procedures should be specified in a common, formal, machine processable language in order to decrease negotiation costs and are responses to the Open-EDI initiative ([15]; [20]). Kimbrough and Lee ([16]; [17]) have noted that existing EDI messaging standards (for instance EDI X12, UN/EDIFACT) are lacking flexibility and expressive power and have proposed illocutionary logic as the suitable framework for a formal language for business communication.

Kimbrough and Moore [19] have developed such a language based on speech act theory ([26]; [27]) and event semantics [24]. Their language supports the expression of a variety of communications between parties, which are useful at the negotiation stage (for instance promises, assertions of fact, avowals of fact, predictions, requests for information and so on) and at the operationalisation stage of a business exchange (for example, requests for delivery, acknowledgements for such requests, invoice issuing and so on). Kimbrough & Moore [18] have also examined issues relevant to the representation and processing of temporal and deontic aspects of electronic transactions during the operationalisation of a business exchange. Bons *et al.* [3] have developed a formalism—documentary Petri nets—for defining trade procedures themselves.

Although the development of a formal representation language is instrumental in making the negotiation of trade procedures more effective and efficient, a natural extension to such efforts is the development of appropriate tools to *verify* such trade procedures. In paper-based negotiation, which seeks to establish a mutually agreed set of rules for the parties' future behaviour, both individualistic and collectivist concerns arise for the parties. The former centre around questions such as what obligations, permissions or prohibitions are implied upon one party under a given trade procedure. The latter are relevant to both parties and centre around questions such as whether a given trade procedure is complete and consistent, that is whether it covers all the intended cases without conflicts, or whether the described ideal model of the business exchange has the appropriate intended safety ("nothing bad will happen") and liveness ("something good will happen") properties. These issues for paper-based contractual negotiation are discussed in more detail in [9] and [8]. Ill-defined trade procedures may result in undesirable situations when they are put to practice, with parties finding that they cannot execute them or that unanticipated circumstances arise, which they cannot resolve without resorting to costly and lengthy litigation. The same issues are relevant to electronic specification of trade procedures. If a trade procedure can be both formally specified and verified, that is, checked for undesirable pathological features, then not only is its negotiation more effective but its subsequent performance is smoother.

In this paper, we describe a suite of formal techniques that have proved effective in finding pathological features in hardware (and software) systems, prior to implementation. We also indicate how these same techniques can be re-deployed in an electronic commerce setting to verify trade procedures as they are negotiated and agreed, before they are put into practice. Much of the theory is explained in terms of an e-commerce example that is drawn as an analogy of an electronic protocol. Before describing any of these techniques in detail, a brief overview of the whole process is given in the next section.

2 An Overview of the Verification Technique

The first step in the verification process is to describe the system using a number of Petri nets that model the system from different points of view. From an electronic design perspective, the component Petri nets model the behaviour of different modules, or of different requirements. From an electronic commerce perspective, each Petri net represents one of the trading partner's views of the transaction system. In either case, a composition algebra combines the separate nets into one large net. This final Petri net is likely to be too large and complicated to have been reliably designed and safely input by hand. The composition and every other process mentioned in this section, is fully automated. The only human inputs to the system are the descriptions of the initial Petri nets, and some behavioural requirements. For more information on the use of this algebra in micro-electronic design see [32]; [33]).

The next step is to form a model of the possible states of the composed Petri net. As a first step, this would be modelled as a discrete state space. However, these spaces are very large: they are exponential in the number of variables. These large state spaces are transformed into another model, called a binary decision diagram, or BDD. These are still exponential in the number of variables, but they admit reduction techniques, which can render them much smaller. The result of the reduction technique is called a reduced ordered binary decision diagram, or ROBDD. The ROBDD is usually considerably smaller than the original state space. These techniques have been employed to reason effectively with systems with more than 10^{20} states [6].

The ROBDD is then queried using a species of temporal logic. Responding to these queries involves boolean operations that efficiently perform searches on all possible cases. This process is called *Model Checking* ([13]; [22]). The temporal logic enables the expression of properties about all future times from initialisation. Examples of the kinds of questions that can be expressed are:

(i) Is there a possible future in which a given request will never be answered?

(ii) Can the system ever be in a position in which one party must perform two conflicting actions?

Such questions can be answered very quickly. And these systems have reported some notable successes, including the discovery (in minutes) of bugs that had eluded months of NASA testing [14].

The verification and debugging models and processes involved in model checking will be explained in terms of an example that is described below.

3 Specification of a Multi-party Trading Procedure

In this section, we present a contractual scenario, in which a single seller interacts with multiple purchasers: the parties all agree that delivery will happen if and when all purchasers can take delivery of goods. This may, for example, be the procedure for a business that has a high delivery overhead, and wants to avoid unnecessary trips. The scenario is based on an IEC (International Electrotechnical Commission) standard 625-1 1979 interface protocol for programmable measuring instruments. The protocol includes a 3-wire handshake, which is intended to synchronise the transmission of data bytes over a bus. The handshake is designed to allow a single source to synchronise with a number of acceptors.

The trade procedure is intended to operate as follows: the seller can indicate that he has goods available for delivery (GAV). The purchasers can indicate that they are ready to take goods—that is, they request goods—(RFG) and can also indicate when they have received goods (GAC). The required sequence of events for the transfer of goods from the seller to the purchasers is as follows: GAV, RFG, GAC are initially false (that is GAV_F, RFG_F, GAC_F) and remain false until all purchasers are ready to receive goods, at which time RFG becomes true (RFG_T). The seller may then assert GAV_T and RFG becomes false. When all purchasers have accepted a specified quantity of goods, GAC becomes true and only then may GAV become false. Finally the purchasers may set GAC back to false and the cycle may repeat.

This sequence is illustrated in Figure 1 below:

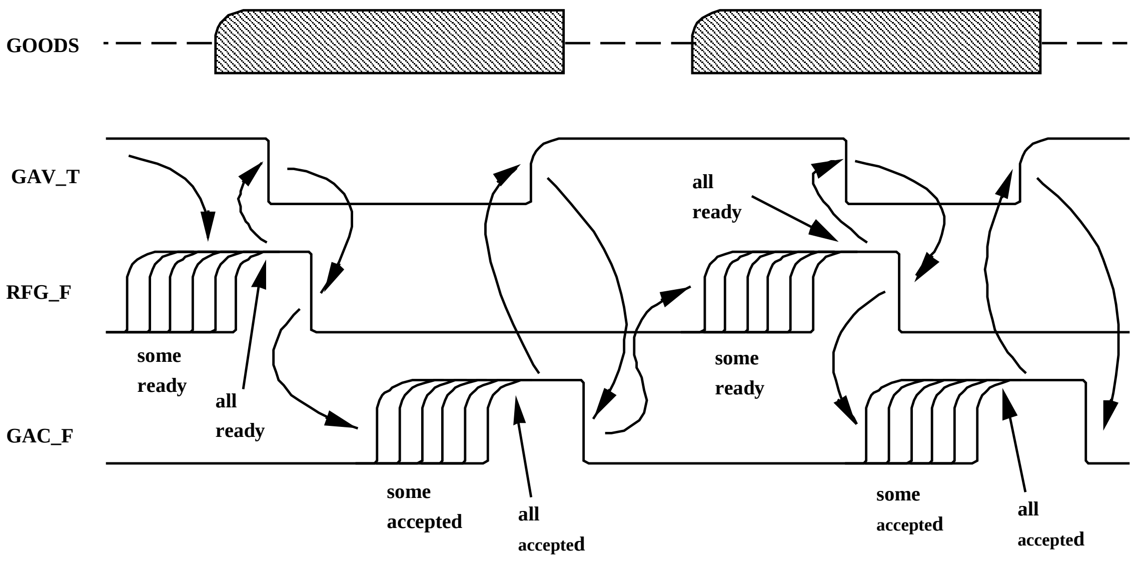


Figure 1 Sequence of events in trade procedure

For the original 3-wire handshake problem, the interface is specified semi-formally in IEC 625 as a set of state diagrams each corresponding to an interface function. The following state diagrams, based on IEC 625, illustrate the seller's and the purchaser's views of the trade procedure:

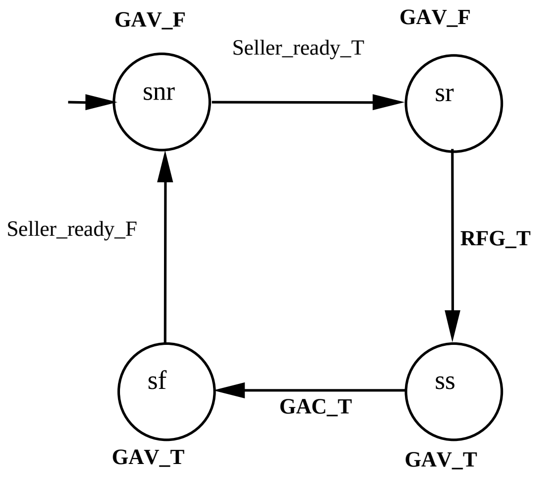


Figure 2 Seller's view of trade procedure

The seller is initially in state snr (seller not ready—for example because he has no goods available or because for whatever reasons private he does not want to enter the transaction). When the seller is ready (in state sr) if he receives a request for goods from a purchaser, he moves to a state where he starts the sale (ss). When the goods have been delivered to the purchaser (GAC_T) the seller is in a state where he has finished the sale (sf) and may terminate the transaction.

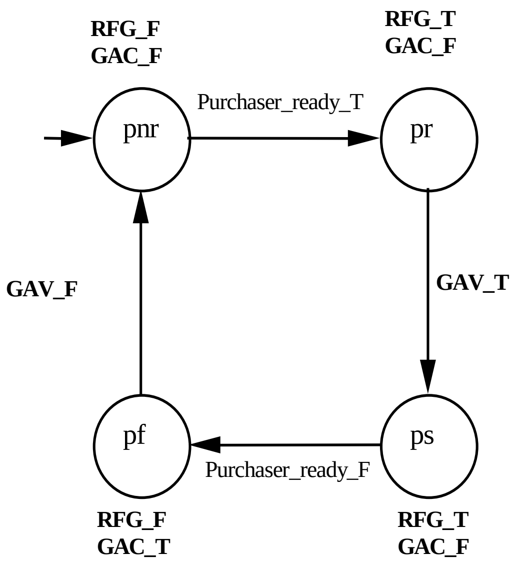


Figure 3 Purchaser's view of trade procedure

The purchaser is initially in state pnr (purchaser not ready—for example because he wants no goods or because for reasons private to him he does not want to enter the transaction). When the purchaser is ready (in state pr) if goods are available (GAV_T) he moves to a state where he starts the purchase (ps). When the goods have been delivered the purchaser is in a state where he has finished the purchase (pf) and may terminate the transaction by indicating that he has accepted the goods.

The part of the trade procedure that we have discussed so far specifies how goods are transferred from the seller to the purchasers. An isomorphic specification can be given for the payment of received goods by the purchasers (viewing payment as the transfer of funds). In that, the purchaser would indicate that funds are available (FAV) and the seller would indicate that it requests payment (RFF) and that it has accepted payment (FAC). State diagrams similar to the ones above would show the seller's and purchaser's views of the payment part of the trade procedure.

The initial specification of this system was given in terms of state diagrams. Our system works with specifications given as Petri nets, which are generalisations of state machines, so that this specification is already in a form that we can use. In the next section, the definition of Petri nets is given and we explain how the state diagrams can be completed into a full verifiable specification of the trade procedure.

4 Petri nets and Their Composition

A Petri net consists of a set, P, of places, a set, T, of transitions and two relations from T to P giving, for each transition, the pre-conditions and post-conditions of that transition. Petri nets are usually given pictorially as in the following example of a Petri net with a single transition whose pre-conditions are p1 and p2, and whose post-conditions are p1 and p3:

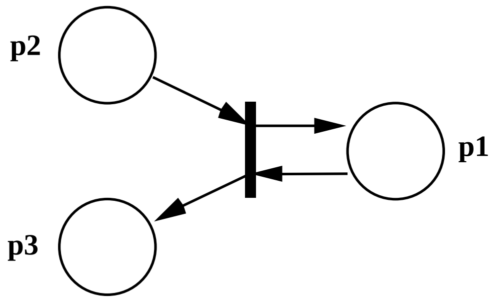


Figure 4 Example Petri Net

There are various flavours of Petri Nets, allowing, for example, the transitions to have outputs [11]. Bons *et al* [3] have introduced another variant, which they use to model trade procedures. In this version, called *Documentary Petri Nets*, some of the places are distinguished as representing documents. Bons *et al* find this a useful distinction because of the way they compose different views of the same transaction. We compose Petri nets quite differently, and in this paper, we restrict ourselves to the simplest flavour of nets. However, it should be noted that all of the mathematics and software alluded to have variants for several different flavour of Petri net. Moreover, since Petri nets are a generalisation of finite state machines (a finite state machine is a net all of whose transitions have one pre-condition and one post-condition), the techniques described in this paper are directly applicable to any modelling technique based on finite state machines, such as Kumar & Feldman's work modelling auctions [21].

The interpretation of a Petri net is that each transition has certain requirements and certain effects. The requirements are the pre-conditions for the transition; the effects are the post-conditions. At a given time, each place of a net may or may not have a token in it; the ones with tokens are said to be *marked.* A *marking* of a Petri net is a snapshot of the net that indicates what commodities are available at a given time. A transition is said to be *fireable* in a given marking if all of its pre-conditions are marked. The effect of firing a *fireable* transition is to unmark its pre-conditions and to mark all of its post-conditions.[1] The dynamics of a Petri net consists of sequences of markings, each derived by firing either one or all of the fireable transitions from the previous marking.

4.1 Composing Petri Nets

The composition of Petri nets is an instance of a general categorical notion of concurrent composition of labelled objects that is defined in [10] (see also [28]). The Petri nets are labelled in the sense that some of the transitions are given names from elements of a labelling set L. The labels are used in providing synchronisation information in the composition: those transitions without labels are not available for synchronisation. Unlabelled transitions will exist in the composite nets, but cannot be synchronised with any transitions in other component nets.

The synchronisation information that is used in the composition is given by partial functions from an event set E to the various labelling sets. A transition, t, from one Petri net synchronises with a transition, t', from another if the labels of the two transitions are both mapped to by same element, e, of the event set. Notice that there may be another element e' that maps to t and not t'.

For example, we may wish to model an acknowledge signal that will not go high until a request signal is already high. We can get this behaviour by having an event that is mapped to such labels as ack_go_high and req_is_high. However, we may wish to allow a request to go high without being acknowledged. This is accomplished by having a second event that maps to req_is_high but not to ack_go_high. This kind of asymmetry is very awkward to model in other synchronisation formalisms, such as CCS [23] and CSP [12].

4.2 Composed Petri Net Versions of the Specification

The state diagrams in Figures 2 and 3 translate into the following two Petri nets.

[1] A slightly more general notion is afforded by allowing more than one token per place. This allows one to say, for example, that a transition requires 4 units of some commodity and produces 7 units of some other commodity. Everything said in this paper applies to this more general case, but we stay with the simpler case for ease of exposition.

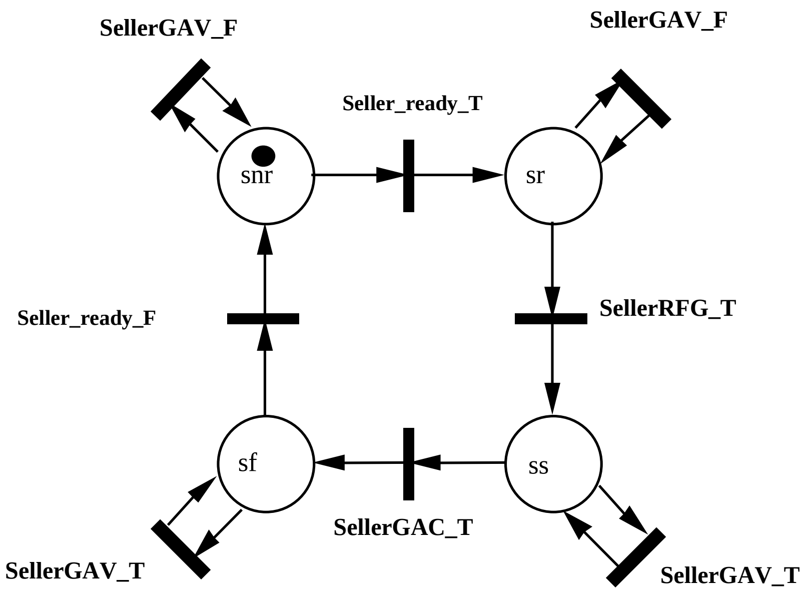


Figure 5 Seller's trade procedure Petri Net

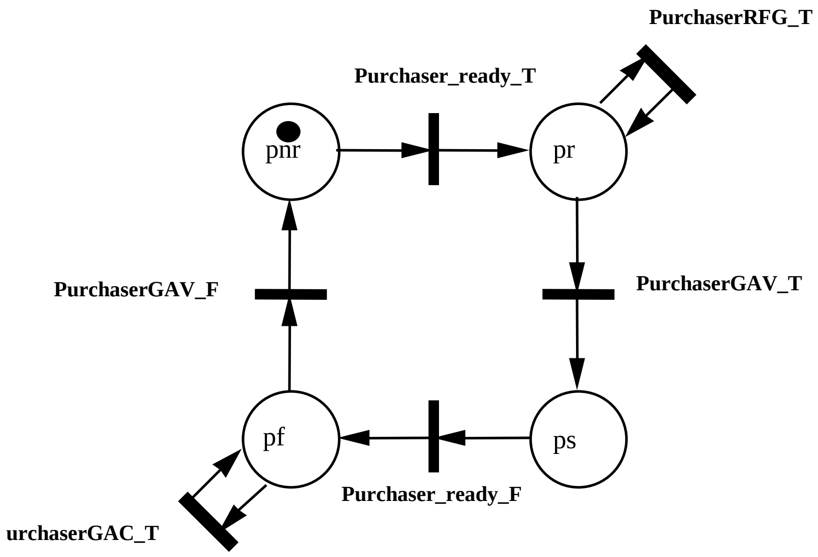


Figure 6 Purchaser's trade procedure Petri Net

The only thing that is not transparent is the self-loops on some of the places. They are there because they are necessary in synchronisations. The specification is completed by adding the synchronisation information, that is, by providing an event set and the information about how each event is viewed in each net. In this case the event set is:

E = { RFG, GAC, P_GAV_T(i), P_GAV_F(i)}, where $1 \le i \le n$.

The events are "Ready for goods," "Goods accepted," and Goods becoming available and becoming unavailable to each of the purchasers.

Information about each of the parties views of these events given by the following partial functions:

Seller :	RFG $\mapsto$ SellerRFG,
	GAC $\mapsto$ SellerGAC,
	P_GAV_T(i) $\mapsto$ SellerGAV_T,
	P_GAV_F(i) $\mapsto$ SellerGAV_F.
Purchaser(i):	RFG $\mapsto$ PurchaserRFG,
	GAC $\mapsto$ PurchaserGAC,
	P_GAV_T(i) $\mapsto$ PurchaserGAV_T,
	P_GAV_F(i) $\mapsto$ PurchaserGAV_F

The difference between the behaviour of P_GAV_T(i) and RFG should be noted. The former is only mapped to labels in two component Petri nets: the seller and purchaser i; while the latter is mapped to every component Petri net. Therefore, if there were two purchasers, the RFG event would be:

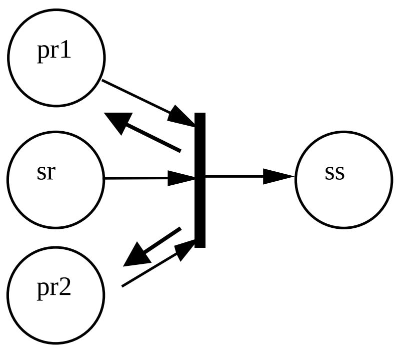


Figure 7 RFG event for two purchasers and one seller

Whereas the P_GAV_T(i)'s all have two input places and two output places.

This example demonstrates how simple it is to compose as many transitions as required. In a calculus such as CCS, for example, the synchronisation operator does not allow more than two events to be synchronised directly, and therefore would not easily model systems that involve several parties, or several procedures.

4.3 The Composed Petri Net

The combined Petri net with one purchaser and one seller is given by:

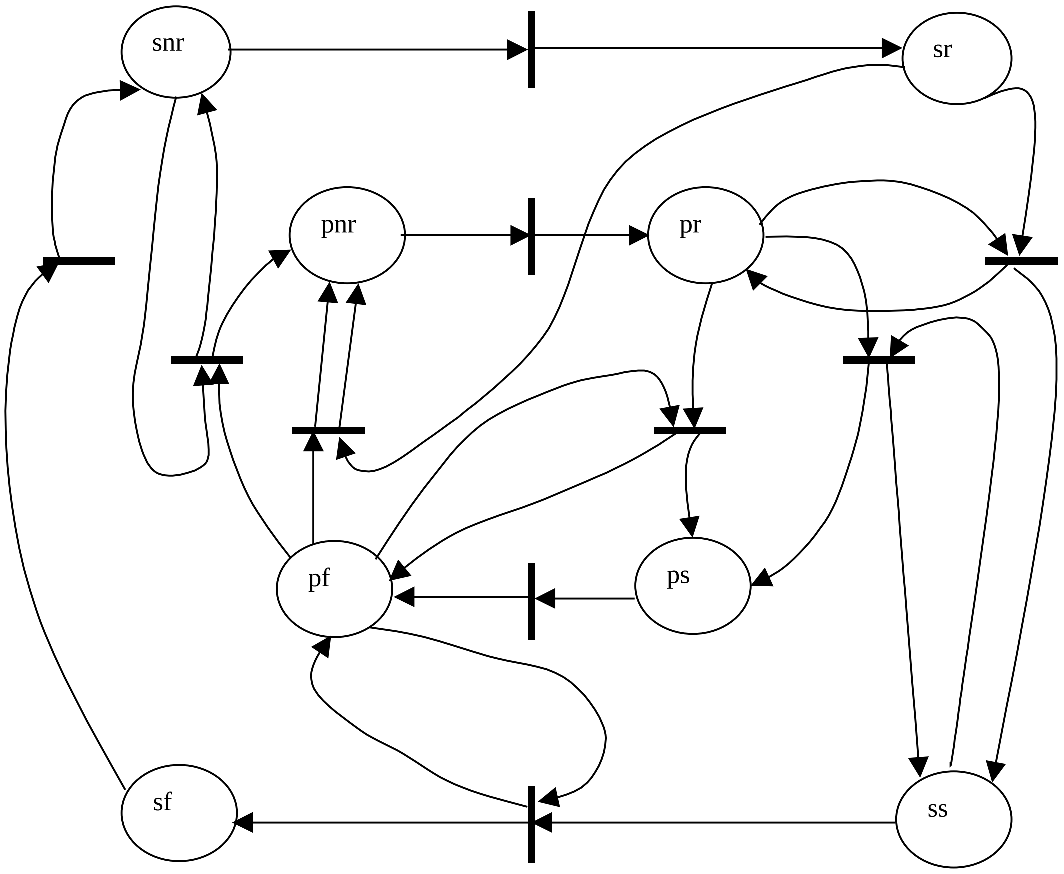


Figure 8 Composed Petri Net for one purchaser and one seller

The composition allows us to combine as many purchasers as we need, but this example shows how difficult it would be to describe directly the resulting Petri net for more than one purchaser. (This exposition of composing Petri nets was necessarily brief for a more detailed exposition see [32]; [33]).

The next step in the process is to map this composed Petri net to a state space or Kripke model.

5 Kripke Models

Kripke models (from a systems modelling point of view) are just directed graphs, or finite state machines without the inputs, that layout the possible state changes of a system. They are generated as the markings of the Petri nets. In this case, the initial marking has snr and pnr marked in the Petri nets of seller and purchaser. This leads to the following model:

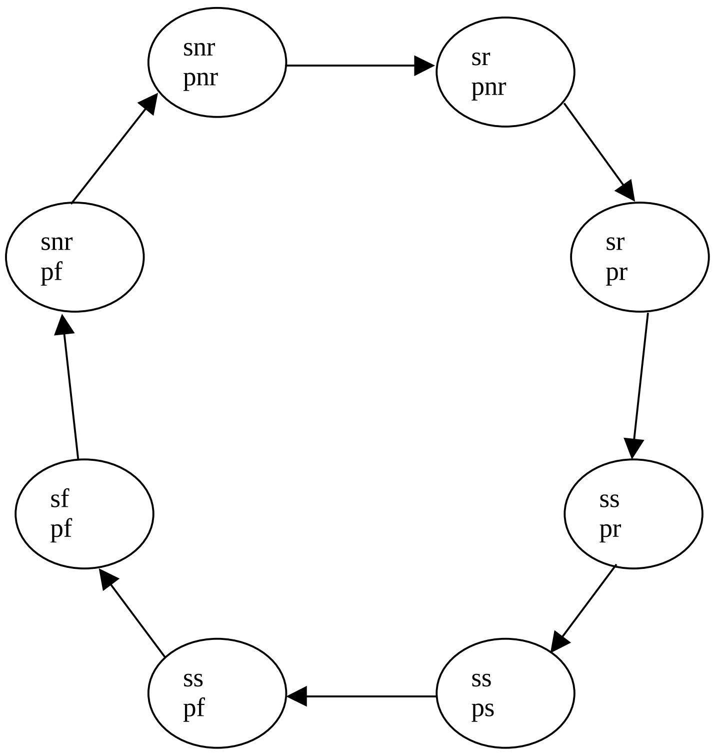


Figure 9 Kripke model of composed Petri Net

If we had started with more than one purchaser, the Kripke model would not have been deterministic. The next stage of verification is to test the truth of various conditions of the Kripke model. The logics used in model checking are all species of propositional temporal logic; the logic that is most frequently used for model checking is called *Computation Tree Logic* or *CTL* [7].

6 Computation Tree Logic

As in all other propositional temporal logics, the formulae in Computation Tree Logic include the usual propositional ones—that is, as well as variables, the connectives "and", "or", "not", and "implies" are allowed—and some temporal operators.

The operators and their meanings will be defined at the same time. The semantic (meaning) relations are defined between states and formulae. The relation is expressed by s |= ϕ, where s is a state and ϕ is a formula, and is to be interpreted as saying that ϕ is true in state s. Then the definition of |= is built up as follows:

$s \models \neg\phi$	Iff s does not satisfy ϕ
$s \models \phi \vee \psi$	Iff s satisfies at least one of ϕ and ψ
$s \models \phi \wedge \psi$	Iff s satisfies both of ϕ and ψ
$s \models \phi \rightarrow \psi$	Iff $s \models (\neg\phi \vee \psi)$
$s \models E(\phi)$	Iff in some immediate successor state of s, ϕ holds. That is, it is possible for ϕ to hold in the next state
$s \models A(\phi)$	Iff in all immediate successor states of s, ϕ holds. That is, it is necessary that ϕ will hold in the next state.
$s \models F(\phi)$	Iff in some future state reachable from s, ϕ holds. That is, ϕ holds eventually.
$s \models G(\phi)$	Iff in all future states reachable from s, ϕ holds. That is, ϕ holds (globally) for ever after s.
$s \models (\phi \text{ Au } \omega)$	On all paths from s, ϕ holds at all states until the first state in which ω holds (ω is not assumed to ever hold).
$s \models (\phi \text{ Eu } \omega)$	On some maximal path from s, ϕ holds at all states until the first state in which ω holds (ω is not assumed to ever hold). A maximal path is either a path that comes to a dead-end or an infinite path.
$s \models (\phi \text{ EU } \omega)$	On some path from s, ϕ holds at all states until the first state in which ω holds (ω is assumed to eventually hold).
$s \models (\phi \text{ AU } \omega)$	On all paths from s, ϕ holds at all states until the first state in which ω holds (ω must eventually hold on every maximal path).

The differences between the small and large u's are subtle. To get a feel for it consider the following Kripke model:

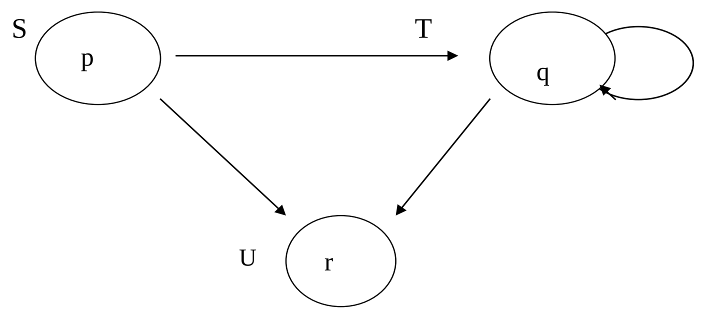


Figure 10 Example Kripke model

The letters outside of the circles are the names of the states, and the letters inside the circles represent properties that are true in the states. There are infinitely many maximal paths from S: there is the length-one path going straight to U; there are paths going from S to T, staying in T for some number of cycles, and then moving to U; and there is the infinite path that goes to T and stays there (by repeatedly going around the loop).

On all of those paths, q remains true until r is true. Therefore

$$S \models q \text{ Au } r$$

However on the infinite path r is never true, so

$$S \models \neg (q \text{ AU } r)$$

Meanwhile there are some paths in which r is eventually true (in fact, on almost all of them) so:

$$S \models q \text{ EU } r$$

Because the example trade procedure (with one purchaser) is so simple, this distinction does not occur in the example worked out above. However, the distinction does show up as soon as more than one purchaser is admitted: a purchaser making a request will continue to make the request until he is satisfied, however, if there are two purchasers and the second purchaser never requests goods, the first purchaser’s request will never be satisfied.

7 Verification of the Trade Procedure

One condition, a form of liveness, that we wish to check is that from every state the trade procedure can eventually return to its initial state. In other words, we wish to establish that the transaction between the seller and any of the purchasers will eventually terminate. The condition is expressed below (S0 is the initial state).

$$S0 \models G\ (EF\ (snr \wedge A\ (pnr(1) \wedge pnr(2) \wedge \ldots \wedge pnr(n)))).$$

Some safety conditions we have also checked are:

(i) The seller should assert GAV_T only when all purchasers are ready—all purchasers should be in state pr. In other words, we wish to establish whether there is a situation where goods are available for delivery on the seller's part but the transaction cannot be realised because some purchaser cannot accept delivery. The condition is expressed below:

$$S0 \models G\ ((\neg SellerGAV\_T \wedge E\ SellerGAV\_T) \Rightarrow (pr(1) \wedge pr(2) \wedge \ldots \wedge pr(n)).$$

(ii) Once GAV is true, it should remain true until all purchasers have accepted goods—all purchasers should have entered state pf. In other words, we want to establish whether it is possible once the transaction has been entered for the seller to retract the goods before they are delivered (to cancel his part of the bargain). The condition is expressed below:

$$S0 \models G\ (SellerGAV\_T \Rightarrow (SellerGAV\_T\ AU\ (pf(1) \wedge pf(2) \wedge \ldots \wedge pf(n))).$$

(iii) No purchaser should accept goods until GAV is true—i.e. none should enter ps until GAV is true. In other words we want to establish that there is no situation where a purchaser's obligation to pay for goods is activated before such goods are available and delivered to him. So we require:

$$\forall i: 1 \leq i \leq n,\ S0 \models G\ ((\neg ps(i) \wedge E\ ps(i)) \Rightarrow SellerGAV\_T).$$

(iv) No purchaser should accept more than one lot of goods while GAV is true—i.e. none should enter ps more than once (this is what was stipulated in our trade procedure), so we require:

$$\forall i: 1 \leq i \leq n,\ S0 \models G\ (ps(i) \Rightarrow A\ (\neg ps(i) \Rightarrow (\neg ps(i)\ AU\ \neg SellerGAV\_T))).$$

These liveness and safety conditions among others were checked by our system, and all hold for the defined trade procedure.

8 Model Checking State Spaces

Checking simple conditions, like the propositional ones or E(ϕ), is straightforward and quite clearly computable: the algorithm involves looking ahead at most one time step. For each of the more complicated temporal constructs, the model checking algorithm is an iterative searching technique. The main theorem that makes the method computable is that these iterative techniques all finish after a number of steps bounded by the length of the longest non-looping path in the state space. The model checking techniques is to keep performing iterations until the same result is reached twice. This is called a *fixed point.* The theorem states that a fixed point will be reached in bounded time.

However, the bound on the time is the number of states of the graph, and the number of states is, at worst, exponential in the number of variables. This is called the *state explosion problem*. Where this technique really gets its power is from a new representation of the state transition graph as a boolean function. The representation is called a Reduced Ordered Binary Decision Diagram (ROBDD). The first step along this path of representation is to view the state space as a boolean function, as will be explained in the next section.

9 From State Spaces to Boolean Expressions

The easiest way to explain how to get from a state space to a boolean expression is by way of an example. Consider the following trading protocol: A request can be made for goods, and the goods can be sent. A possible state diagram is given below:

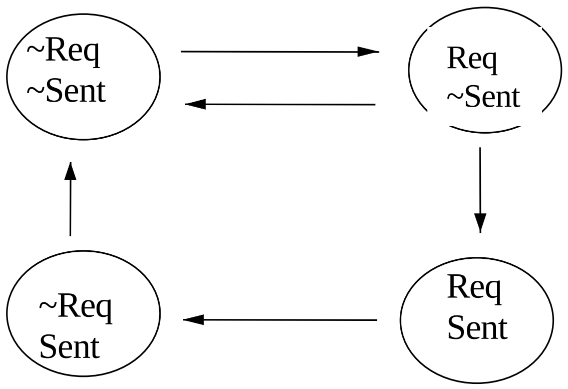


Figure 11 Example trading protocol

To translate this into a boolean expression, it is considered to be a system with four boolean variables: Req, Sent, Req', and Sent'. Req and Sent represent variables in the present state, while Req' and Sent' represent variables in a next state. Each transition (arrow) in the state space is modelled as a conjunction of these variables. For example the bottom arrow is modelled as: Req&Sent&~Req'&Sent'. The whole graph is then a disjunction (an OR) of these conjunctions. Therefore, in this example the boolean expression for the state space is:

$$
\begin{aligned}
B = \quad & (\neg Req \wedge \neg Sent \wedge Req' \wedge \neg Sent') \vee \\
& (Req \wedge \neg Sent \wedge \neg Req' \wedge \neg Sent') \vee \\
& (Req \wedge \neg Sent \wedge Req' \wedge Sent') \vee \\
& (Req \wedge Sent \wedge \neg Req' \wedge Sent') \vee \\
& (\neg Req \wedge Sent \wedge \neg Req' \wedge \neg Sent')
\end{aligned}
$$

The next step is to turn the boolean expression into a binary decision diagram.

10 Binary Decision Diagrams

An unordered binary decision diagram is simply a tree form of a truth table. Consider, as a first example, the expression a $\wedge$ b. The binary decision diagram is given by:

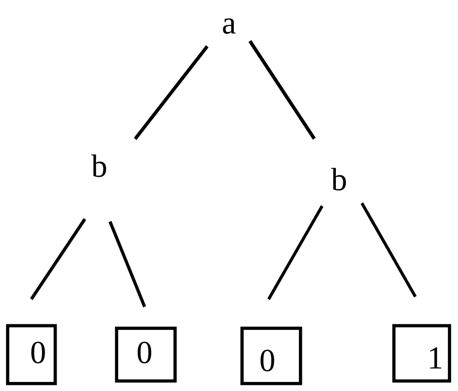


Figure 12 Binary decision diagram for (a $\wedge$ b)

As you go down the tree, left means false and right means true. In this tree, for example, to find out the truth value of the function when a is true and b is false, start at the top, go down the right branch (corresponding to 'a' being true), and then at the junction, go down the left branch (corresponding to 'b' being false). Diagrams like these were first introduced in [1] .

These diagrams were greatly refined by Bryant in [4]. Bryant's improvement was to reduce the diagrams using three reduction operations:

(1) Join all the 0's and join all the 1's at the bottom of the tree

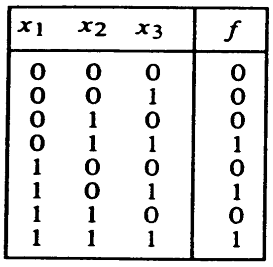
(2) Join nodes that do the same thing in the tree

(3) Remove redundant tests

To understand the way these work, consider the following example is taken from Bryant [5]. The function is

$$F = \neg x_1 x_2 x_3 \vee x_1 \neg x_2 x_3 \vee x_1{*}\, x_2{*}\, x_3$$

The binary decision diagram is:



x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

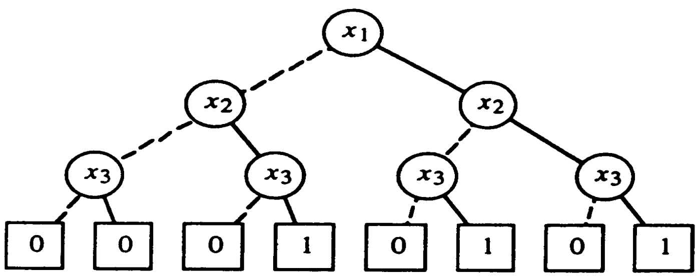


Figure 13 Example BDD from [5]

And after the reductions it is:

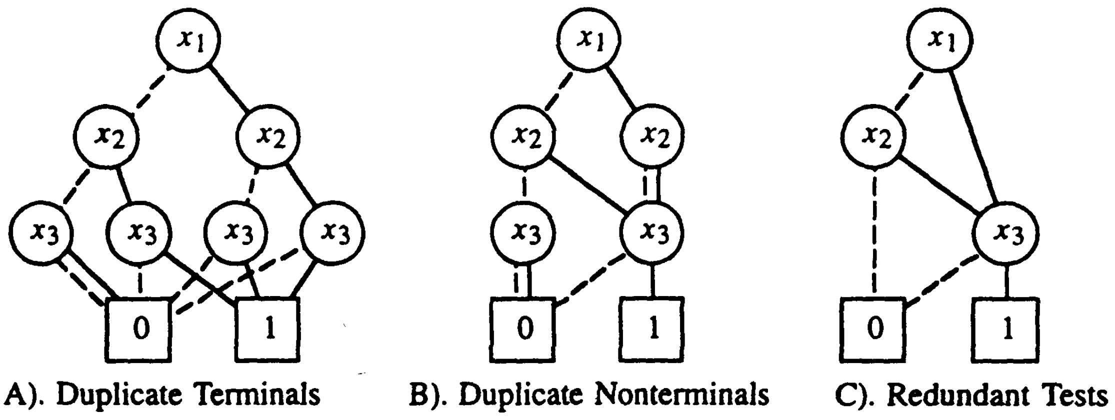


Figure 14 Reduced BDD from [5]

In this example, the move from (A) to (B) entails noticing that the three rightmost x_3 nodes in BDD (A) all have arrows leading to the same places. They are therefore conglomerated as one node. The move from (B) to (C) entails noticing that from the rightmost x_2 in (B) both arrows lead to the same place: this implies that the x_2 test is unnecessary since the same behaviour is given by both results. The same holds for the leftmost x_3 node. Therefore, these two nodes are removed. The boolean function represented by the final BDD is:

$$G = \neg x_1 x_2 x_3 \vee x_1 x_3$$

which is equivalent to F.

All of the standard model checking operations can be performed as boolean operations on Reduced Ordered BDDs (see [22] for details). This is frequently a very efficient method in time (and especially) in space of performing these operations. To each boolean operation, there corresponds an operation on the associated BDDs that is polynomial in the size of the BDDs.

11 Conclusions and Future Work

Business-to-business electronic commerce would greatly benefit from a system that could model and evaluate trade procedures before they are put into practice. This will enable misunderstandings and potential mistakes to be found before they become actual and expensive problems. This paper has reported the techniques behind a system that performs much the same task for micro-electronic design. That these techniques are applicable in an electronic commerce setting has been demonstrated by following a simple multi-party trade procedure through the major steps in the modelling and verification procedure.

In the example, the trade procedure was checked against some temporal conditions. Essentially, we showed the behavioural equivalence of two specifications: one of which is declarative (given in terms of temporal logic formulae), and one of which is operational (given in terms of Petri nets). The two kinds of specification tend to suffer different problems. Errors in a declarative specification tend to be those of omission (for example a liveness or safety condition is left out); errors in an operational specification tend to be of commission. One enhancement of this work will be to use an equational reasoning system to help develop the declarative specification, by making them complete. On the other hand, future work on this system will involve the use of Deontic Logic and Speech Act Theory to help move from a natural language description to the operational, Petri net, specification (see [8] for prelimirary discussion of these matters). However, the main direction for future work will be to study real-life, and possible future, trade procedures, with a view to understanding exactly what needs to be modelled.

We believe that synchronised Petri nets will prove an effective means of modelling these procedures. We know that Petri nets can be used as a front end to Model Checking. Moreover, we know that model checking has proved an extremely powerful technique for verifying high-level behaviour of hardware systems. The evidence so far suggests they will prove equally effective in electronic commerce.

Acknowledgements: We would like to thank Alan McDonald for his valuable contribution to the method for composing Petri Nets.

12 References

1. Akers S. B. *Binary Decision Diagrams*, IEEE Transactions on Computers, C-27, 6, pp. 509–516.

2. Atiyah P. S. *An Introduction to the Law of Contract*. Clarendon Law Series (4th edition), Clarendon Press, Oxford, 1989.

3. Bons R., Lee R. M., Wagenaar R., and Wrigley C. D. Modelling Inter-organizational Trade Procedures Using Documentary Petri Nets. *HICCS 95*.

4. Bryant R. E. Graph based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35, 6, 1986, pp. 677–691.

5. Bryant R. E. Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams. Carnegie Mellon Technical Report CMU-CS-92-160, 1992

6. Burch J. R., Clarke E. M., McMillan K. L., Dill D. L., and Hwang L. J. Symbolic Model Checking: 10^{20} states and beyond. *Information and Computation* 98 (2), 1992, pp. 142–170.

7. Clarke E. M., Emerson E.A., and Sistla A. P. Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages* 8 (2), 1986, pp. 244–263.

8. Daskalopulu A. Logic-Based Tools for the Analysis and Representation of Legal Contracts. Ph.D. dissertation, Imperial College London, 1999 (forthcoming).

9. Daskalopulu A. and Sergot M. J. The Representation of Legal Contracts. *AI and Society*, 11 (1/2), 1997, pp. 6–17.

10. Fourman M. P. and Zimmer R. M. Modular Design as Algebraic Composition. *Intelligent CAD 1* (edited by P. Ten Hagen), Springer-Verlag, 1987.

11. Genrich H. J. and Lautenbach K. System Modelling with High Level Petri Nets. Theoretical Computer Science, 13, 1981, pp. 109–136.

12. Hoare C. A. R. *Communicating Sequential Processes*, London: Prentice-Hall International, 1985.

13. Holzmann. G. J. The model checker Spin. *IEEE Transactions on Software Engineering*, 23 (5), 1997, pp. 279–295.

14. Holzmann G. J. Seminar at Royal Holloway and Bedford New College, 1998

15. ISO/IEC JTC1/WG3. The Open-EDI Reference Model. Working draft document N305, 1994.

16. Kimbrough S. O. and Lee R. M. On Illocutionary Logic as a Telecommunications Language. *Proceedings of the International Conference on Information Systems*, San Diego, December 1986.

17. Kimbrough S. O. and Lee R. M. Formal Aspects of Electronic Commerce: Research Issues and Challenges. *International Journal of Electronic Commerce*, 1 (4), 1997, pp. 11–30.

18. Kimbrough S. O. and Moore S. A. On Obligation, Time, and Defeasibility in Systems for Electronic Commerce. *Proceedings of the 26th Annual Hawaii International Conference on System Sciences*, vol. 3, Los Alamitos CA, 1993, IEEE Computer Society Press, pp. 493–502.

19. Kimbrough S. O. and Moore S. A. On Automated Message Processing in Electronic Commerce and Work Support Systems: Speech Act Theory and Expressive Felicity. *Transactions on Information Systems*, 15 (4), 1997, pp. 321–367.

20. Knoppers J. Importance of the “OPEN-EDI” Reference Model from a User and Business Perspective. *Proceedings of Conference on Interorganizational Systems in the Global Environment*, Bled, 1992.

21. Kumar M. and Feldman S. I. Business Negotiations on the Internet. IBM Technical Report.

22. McMillan. K.L. *Symbolic Model Checking*, Kluwer 1993.

23. Milner. R. *Communication and Concurrency,* London : Prentice Hall, 1989.

24. Parsons T. *Events in the Semantics of English: A Study in Subatomic Semantics.* Current Studies in Linguistics. The MIT Press, Cambridge MA, 1990.

25. Pörn I. *Action Theory and Social Science: Some Formal Models.* Synthese Library, 120, D. Reidel, Dordrecht.

26. Searle J. R. *Speech Acts*. Cambridge University Press, Cambridge, England, 1969.

27. Searle J. R. and Vanderveken D. *Foundations of Illocutionary Logic*. Cambridge University Press, Cambridge, England, 1985.

28. Winskel, G. A category of labelled Petri nets and compositional proof system (extended abstract). *Proceedings Third Annual Symposium on Logic in Computer Science*, Edinburgh, Scotland, 5-8 July 1988. IEEE Computer Society, pp. 142–154,

29. Winskel, G. Petri Nets, Algebras, Morphisms, and Compositionality. *Information and Computation*, 72, 1987, pp. 197–238

30. von Wright G. H. Deontic Logic. *Mind,* 60, 1951, pp. 1–15.

31. Wrigley C. D. EDI Transaction Protocols in International Trade. *Proceedings of the Conference on Interorganizational Systems in the Global Environment*, Bled, 1992.

32. Zimmer R. M. MacDonald A. and Holte R. Representation Engineering and Electronic Design: CAD as Automated Representation Change, Patricia European Community ESPRIT Project Deliverable, 1992.

33. Zimmer R. M. and MacDonald A. The Algebra of System Design: A Petri Net Model of Modular Composition, *IEEE International Symposium on Circuits and Systems*, 1993.