CPN Modelling of an Object Based System: The ODP Trader

A. Tokmakoff and J. Billington,
Telecommunications Systems Engineering Centre,
University of South Australia,
Warrendi Road,
The Levels, 5095, Australia
E–Mail:[A.Tokmakoff, J.Billington]@UniSA.Edu.Au

Abstract

Advances in computing power and networking technologies are leading the way to a new generation of information systems, where heterogeneous computers share resources and cooperate in a distributed manner. Such systems are inherently complex due to concurrency and synchronisation of activities operating in parallel, making design and implementation much more challenging than yesterday's stand–alone personal computers. In order to engineer these Open Distributed Systems, tools are required to aid in the design and analysis processes. In this paper, Coloured Petri Nets and the Design/CPN™ tool are used to model the ODP Trader which has been recently standardised by ISO and ITU–T. The Trader is presented in an electronic commerce environment, where interworking of Traders is demonstrated.

Keywords
Coloured Petri Nets; ODP Trader; Electronic Commerce.

1 INTRODUCTION

The Reference Model for Open Distributed Processing (RM–ODP) (ITU/ISO ODP, 1994) is a joint effort by the international standards bodies ISO (International Organisation for Standardisation) and ITU–T (International Telecommunications Union). RM–ODP aims to provide transparent sharing of resources and services over different architectures, networks and operating systems (Bietz, Berry, Lister and Raymond, 1993). As an important part of the RM–ODP standard, the Trading function (ITU/ISO Trading, 1995) allows a service consumer (importer) to be "matched up" with a service provider (exporter) by a trusted third party.

In a distributed system, locating an appropriate server should ideally be performed at run time in a dynamic manner to allow for the introduction of new service providers. The Trading function has many areas of possible application including Load Balancing in a Distributed Operating System, User–Oriented Resource Location in the World Wide Web (WWW) and as a commercial service location utility. In order to improve our understanding of Trader, a formal, executable specification of the Trader has been created using Coloured Petri Nets (CPNs) (Jensen, 1992). This paper will discuss a model in which two Traders interwork to provide a service in an electronic commerce environment, providing advanced information services.

2. THE TRADING FUNCTION

The Trader holds service parameters that describe attributes of exported services. These parameters may be used by the Trader to match a request with a previously exported service. When a set of offers which fulfil the request is obtained, the Trader may then select the "best" service based upon selection criteria provided by the initial client request.

The sequence of interactions between the Exporter, Importer and Trader to locate a service are shown in Figure 1 (ITU/ISO Trading Tutorial, 1995).

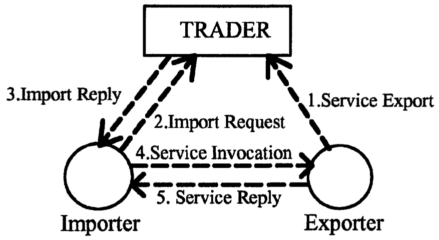


Figure 1 Sequence of Interactions between Exporter, Trader, and Importer.

- 1. **Service Export**: The Trader receives a service export from the Exporter. This request includes a description of the service, a location at which the service may be accessed and properties of the service. This information is stored in a database.
- 2. **Import Request**: The Trader receives a service import request from a client. This request includes the type of service required and a list of desired attributes. The Trader checks with an "Infrastructure Object" called the Type Repository (not shown in Figure 1) to ensure that the requested service and properties are legal.
- 3. **Import Reply**: The Trader searches its database of exported services and returns any successful matches after applying selection criteria if requested by the Importer.
- 4. **Service Invocation**: The Importer is now free to interact with the Exporter independently of the Trader and utilise the service.
- 5. **Service Reply**: The Exporter replies to the Importer's service request.

Interworking of Traders is where multiple autonomous, possibly heterogeneous Traders cooperate to provide an increased range of matching services. This is facilitated by the creation of "Links" which describe the knowledge one Trader has of another and with whom it may Interwork. Using these Links, a Trader may act on behalf of one of its clients as the client of another Trader. At this point in time, interworking is an area of considerable interest and research activity (Vogel, Bearman and Beitz, 1995).

3. UNDERSTANDING TRADER WITH FORMAL METHODS

An important first step is to increase one's understanding of Trader by the development of models. A formal specification allows there to be an unambiguous statement of Trader's functionality. We believe that it is important to develop "specification prototypes" which can be executed to gain insight into Trader's operation.

To do this, we are using Coloured Petri Nets (CPNs) to provide a specification prototype, and the tool Design/CPN™ (Meta, 1992) for creating and managing the prototype and for performing simulation experiments. The aims here are to increase our understanding of the dynamics of Trader, while at the same time, evaluating the utility of CPNs as a modelling language.

A number of analysis methods have been automated for CPNs, including reachability analysis and invariants analysis. Reachability analysis can be performed by the creation of an Occurrence Graph. This graph contains one node for each reachable state, and an arc for each possible change of state. This method effectively performs an exhaustive simulation run from a given initial condition and covers all possible states the model may reach. It is not surprising therefore, that it suffers from computational limitations, where a model may include too many states for the tool to accommodate. Design/CPN provides an Occurrence Graph Analyser (OGA) which may be used to transverse the state space and detect properties such as Deadlock and reaching undesirable states. In addition, TORAS (Tool for Reachability Analysis of Systems) (Wheeler, Valmari and Billington, 1990), developed by Unico Computer Systems under a contract with Telstra Research Laboratories, promises to be a superior tool for analysis of concurrent systems as it supports advanced techniques for reducing the state space (Valmari, 1989).

Place Invariants analysis is a method which attempts to find equations that are satisfied by all reachable markings. It does not suffer from the state space explosion problem associated with OGA analysis since it uses algebraic symbolic calculations rather than a brute force method. We can use invariants analysis to prove properties of the model including assertions such as the set of tokens of a given colour remain constant (i.e. neither produced nor consumed by the CPN). A tool for invariants analysis (Design/CPN Place Invariants Tool) is not yet widely available as it is currently at research prototype stage (Jørgensen, Mortensen and Sousa, 1994).

Having used these analysis methods to verify the Trader, it may be possible to use implementation tools such as PROMPT (Billington, Wheeler and Wilbur–Ham 1988), to transform a CPN model into executable code. When using CPNs as a modelling tool, we may illustrate the flow of control and data flow by observing the flow of tokens through the net on execution.

4. COLOURED PETRI NETS

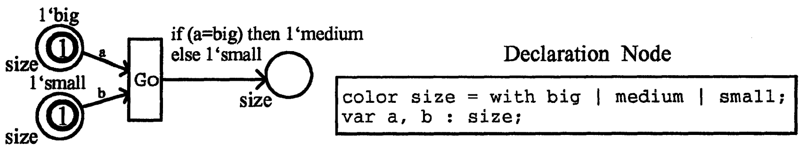
CPNs were developed in the early 1980's by Kurt Jensen at Aarhus University in Denmark (Jensen, 1992). CPN models have a graphical form, and a well–defined semantics which allows for formal analysis. They have been applied to a variety of applications including Description of Services in Intelligent Networks (Dibold, 1992), Specification and Verification of Protocols (Billington, Wheeler and Wilbur–Ham, 1988), Modelling Organisation Workflows (Mortensen and Pinci, 1994) and Design and Validation of Hardware at the Register level (Jensen, 1992). Graphical simulations may be executed where the modeller sees tokens moving throughout the system as the transitions fire. As the tokens are arbitrarily complex data values, the simulation allows data flows to be modelled. CPNs are high level Petri Nets which provide for modelling with hierarchical structure and representation of complex data types. Figure 2 provides a simple example of a CPN. A CPN is composed of the following:

- Colours: Analogous to a "type" in a Programming Language.
- Places: Represented by ellipses, which are typed (thus being of a certain colour).
- Transitions: Represented by rectangles.
- Arcs: Represented by arrows, which define relationships between places and transitions.
- Tokens: Data values contained in places (consistent with the place's colour).
- Inscriptions: Arcs and transitions may be inscribed with expressions containing constants, variables and functions. In the computer support tool, Design/CPN, these are written in Standard ML, a functional programming language.
- A set of declarations, defining Colours and functions and declaring variables and constants.

The association of tokens with places in the CPN is known as the marking (or global state) of the CPN. The marking of a CPN is changed by the occurrence of transitions. A transition may occur if:

- there are appropriate tokens residing in its input places (determined by the the arcs and inscriptions connecting places to the transition) and,
- the transition inscription (guard) is true.

We will introduce these concepts with the help of Figure 2 which shows three places and one transition. A single colour `size` has been defined in a Declaration Node. `Size` is a set of three values: `big,medium, small`. The two variables which have been declared (`a` and `b`), are restricted to values which are given by their colour (`size`). All the places in this example are of colour `size`, which limits the tokens they may contain to that colour. The presence of tokens at a place is indicated by a bold circle which contains the number of tokens located there. The text above places indicates the multi–set of tokens which are present. For the top place, `1'big` indicates that there is one token with value `big` in the place. If desired, this information may be hidden allowing the model to remain uncluttered.



```
color size = with big | medium | small;
var a, b : size;
```

Figure 2 A simple CPN Transition with two input places and one output place.

There are a number of inscriptions in this CPN. Both of the input arcs to transition "Go" have variables for inscriptions, and the output arc has a more complex inscription which involves a conditional test.

When Design/CPN assigns a value to a variable, it is known as "binding". For a particular binding to variables, if all the input places connected to a transition by arcs have tokens of the required number and colour (as defined by the arc inscriptions), then the transition is said to be enabled. More than one transition may be enabled at once. A transition that is enabled may fire. When a transition fires, tokens are removed from the input places and added to output places, as indicated by the arc inscriptions. In Figure 2, we can see that transition `Go` is enabled when `a` is bound to the value `big` and `b` is bound to `small`. On firing, both of the tokens in the input places are consumed, and a token with value `medium` is placed into the output place.

Design/CPN supports hierarchical modelling. This means that a model may be spread out over a number of "pages", where each page models different aspects of the system allowing modularity. A page which is "used" by another is known as a sub–page of the super–page. The CPN on the sub–page is known as a sub–net. The main construct for this is a substitution transition.

5. OBJECTS AND PETRI NETS

A great deal of work has been done on the topic of Petri Nets and objects, and a survey of these techniques is given in Bastide (1995). There are two main approaches to combining the concepts of Object–Orientation and Petri Nets.

The first method, known as "Objects inside Petri Nets" is where tokens are considered to be objects and each token is an instance of the object class. In order to perform functions (or methods) on objects, transitions fire where inscriptions on outward arcs may be evaluated using object method operations. Figure 3 (Bastide, 1995) illustrates this concept, where objects of different colour (residing in Places `A` and `B`) are bound to variables `x` and `y`. When transition `T1` occurs, the outgoing arcs are evaluated, with the result of a method invocation on object `x` (using parameter `y`) being placed into C which is of colour `Colour1`. Output arcs indicate the flow of objects throughout the system. This philosophy allows objects (tokens) to be created and destroyed dynamically which occurs in real dynamic object–based systems.

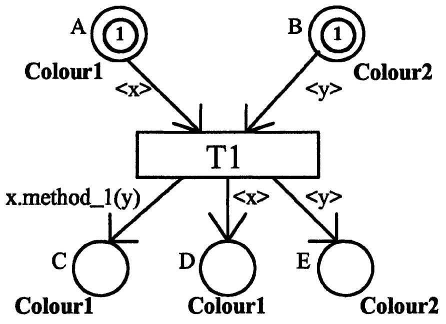


Figure 3 The "Objects inside Petri Nets" concept.

Thus, CPNs may be considered to be object based, where tokens are objects with their internal data structure defined by their colour. Transitions associated with a place provide the methods which may operate on the data within the object (token). The net structure determines the ordering of these methods, hence defining the system's functionality. CPNs provide for concurrency and synchronisation and thus, are a concurrent object–based modelling language.

The second method, known as "Petri Nets inside Objects" is where a Petri net is used to model the internal workings of an object, and the inter–object communication of an Object–Based system. In this modelling methodology, a token is used to hold data (depending upon its colour) and the marking of a net represents the internal state of an object. Using this methodology, it is not possible to dynamically create objects, as they are part of the CPN structure. A clearly defined interface may be created which indicates the "methods" which an object may perform and data hiding is easily represented. It is this methodology which has been used to model the ODP Trader using Design/CPN.

6. MODELLING TRADER'S DYNAMICS

Firstly, the hierarchy of, and interaction between pages in the model will be discussed. A detailed explanation of some of the pages (due to a lack of space) will follow and an explanation of the colours (data types) which have been used in the model will be included where appropriate. This paper only covers the Import operation, the most complex of the four functions (Import, Export, Withdraw, Modify) which a Trader can perform.

A. The Model Hierarchy

Design/CPN provides us with the opportunity to create a hierarchy of nets which as a whole, model the Trader. These nets may represent objects which exist in the Trading Environment, or may be used to hide complexity within the model. Figure 4 shows the Hierarchy page of the model (Hierarchy#1001), where pages are represented by a node containing a page name and number.

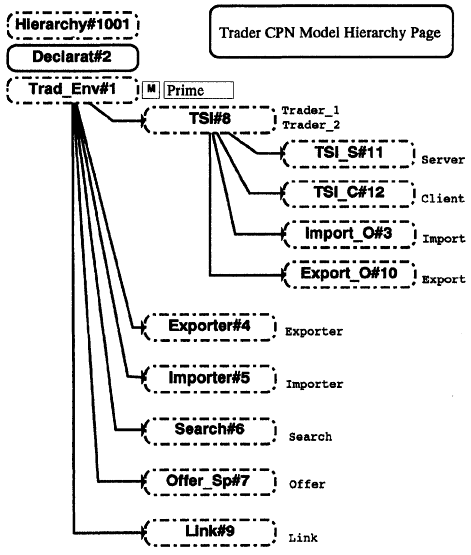


Figure 4. The Hierarchy page of the Trader Model.

Relationships between pages are indicated by a directed acyclic graph. This page indicates a "uses" relationship between pages and is created by Design/CPN upon request (page names are truncated to 8 letters). The function of the pages is given below.

- Declarat#2 is a page which contains the Global Declaration Node, where all colour sets (types) and variables are declared and SML function code for the model is located.
- Trad_Env#1 is the top–most page of the model which defines a topology for communication between objects in the Trading Environment. It is also the "Prime" Page which specifies which pages of the CPN should be simulated.
- TSI#8 represents a Trading Service Interface (TSI) within the Trading Environment. It may be instantiated multiple times to model an environment where Traders interwork.
- TSI_S#11 and TSI_C#12 represent Trading Service Interfaces (Server and Client mode respectively) which are objects instantiated by the Trader to provide well–defined interfaces to the Trading Community. TSI_S#11 operates in Server mode, where it accepts incoming requests and invokes appropriate operations on the Trader itself. TSI_C#12 operates in Client mode, where it initiates requests and receives replies on the Trader's behalf.
- Import_O#3 and Export_O#10 model the operations which the Trader performs when servicing an Import and Export Request respectively. These pages are sub–pages of TSI#8.
- Exporter#4 represents the Exporting Object. The model assumes that valid Export Offers have been submitted by this Object and have been stored in the Offer Space Object. This is easily achieved in Design/CPN by creating tokens at the appropriate place using initial markings.
- Importer#5 represents the Importing Object. This page models an importing object which submits an Import request to a Trader. In practice, this object may be a computational object in a Distributed System, or even a WWW browser.
- Search#6 represents the Search Policy Object. This Object is used to determine the Search Scope when an Import Request is being processed. Information held by this object indicates whether a Trader should search the local database, forward the Import Request to linked Traders, or both. It holds initial search policies for both Traders in the Trading Environment.
- Offer_Sp#7 represents the Offer Space Object which is used by the Trader to manage the offers which are submitted to the Trader by Exporting Objects. It is a Generic Service Object which may be used by any Trader to store Offers.
- Link#9 represents the Link Space Object which manages the Trading Links. It maintains a record for each Trader in the Trading Environment. The record indicates which other traders may be contacted for joint trader interworking.

The Search#6, Offer_Sp#7 and Link#9 pages all represent objects which are yet to be specified by the ODP Standardisation process. Thus, in this model, these objects have been modelled using a "common sense" approach. That is, they are modelled to provide limited functionality to the Traders to enable them to perform the Trading Function. This allows us to test the Trading Environment Model by performing simulation runs using the Design/CPN Simulator.

B. Trader Environment Top Level Page

The top level page of the model is Trad_Env#1 (Figure 5). It can be seen that are 7 instances of Computational Objects in the distributed system, each of which has been modelled at a lower level on a separate page. These sub–pages are linked to Trad_Env#1 through the use of a shared place which models the Communication Medium used by objects for message passing. It is assumed that there is a Communication Protocol stack in place which provides a service in which:

- messages are neither lost, nor duplicated,
- ordering of messages is unimportant (overtaking is allowed),
- there is no mutilation (corruption) of messages.

Each object in the system is modelled by a substitution transition which encapsulates the object and places it in the context of the environment in which it resides. Substitution transitions are indicated by the presence of a HS within a box located in each transition on the page. The objects communicate through input/output ports which have a mapping from the high–level Trader Environment page to the corresponding sub–page which represents the object. Thus, the place "Comms_Medium" has a place in each substitution transition sub–page which has exactly the same marking.

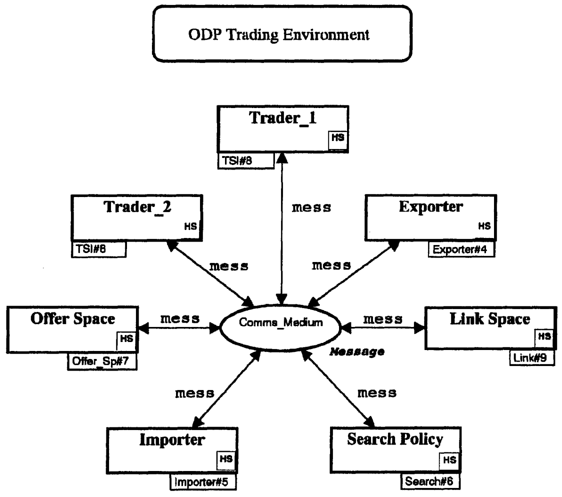


Figure 5 The Top Level page (Trad_Env#1).

Objects in the system have unique identifiers required by the communications sub–system to allow inter–object communications. These identifiers are defined by colour `Id` which is similar to an enumerated type found in most programming languages.

```
color Id = with importer     | (* Generic Importing Object *)
                exporter     | (* Generic Exporting Object *)
                TSI_S        | (* Generic Trading Service Interface Server *)
                TSI_C        | (* Generic Trading Service Interface Client *)
                search       | (* Search Space Object *)
                link_space   | (* Link Space Object *)
                offer_space  | (* Offer Space Object *)
                t1_tsi_c     | (* Trader 1 Trading Service Interface Client Role *)
                t1_tsi_s     | (* Trader 2 Trading Service Interface Client Role *)
                t2_tsi_c     | (* Trader 2 Trading Service Interface Client Role *)
                t2_tsi_s       (* Trader 2 Trading Service Interface Client Role *)
```

An important colour in the model is `Message` which defines the structure of the messages passed between objects in the system. It is defined as a tuple of two colours:

```
color Addresses = product Id*Id;
color Data = product Operation*Op_Data;
color Message = product Addresses*Data;
```

`Addresses` is a pair of sender and receiver identifiers for the message, and `Data` is a pair containing an Operation to be performed and Operation Data, which are parameters for the operation.

The colour `Operation` defines all valid operations which objects may perform including:

```
color Operation = with get_offers     |  (** Offer Space Operation **)
                       import         |  (** Import operations **)
                       query_offer_graph |
                       export         |  (** Export Operation **)
                       unify_policy   |  (** Policy Operation **)
                       get_links      |  (** Link Space Operation **)
                       error          |  (** General Purpose Operation **)
                       respond        |
                       acknowledge;
```

In addition to `Operation`, subsets of Operation are defined. These subsets correspond with specific sets of operations which are associated with an interface. Having declared the sub–set, the `in' (subset_id)` operator may be used as a boolean test for an element's membership within the subset. `Import_Ops` is defined as a subset of Operations by the following declaration.

```
color Import_Ops = subset Operation with
   [import, query_offer_graph] declare in;
```

All Operations may have an associated `Op_Data` which contains Operation parameters. `Op_Data` is defined as a Union of different colours representing data required for different `Operations`.

```
colour Op_Data = union err:Error + imp:Serv_Type + ack:Acknowledge + none:Null+
                       off:Serv_Off + s_pol:S_Policy + link_l:Link_List + off_l:Offer_List;
```

Thus, any place with colour `Op_Data` may store a token which is a member of the union. We need only to specify the union identifier of the colour (such as `link_l` for data of colour `Link_List`), when we remove tokens from this place. This indicates the colour of the data and is used extensively throughout the model.

The colours which model Service Offers are based upon the Trader Standard. As before, high level colours are made up of products of lower level colours. A service type is defined by a tuple consisting of an Interface Type (`Int_Type`) (which defines what the service is), and a list of Service Properties (`Serv_Prop`) (which is a list of attributes associated with the service). From this, a Service Offer includes a reference to where the service may be obtained (`Serv_Int_Id`). An Offer_List is simply a list of Service Offers. It is used in the model for storing the exported offers and for creating a list of retrieved offers which will undergo a selection process to obtain the "best".

```
colour Serv_Type = product Int_Type*Serv_Prop;
colour Serv_Off = product Serv_Type*Serv_Int_Id;
colour Offer_List = list Serv_Off;
colour Offer_Store = product Id*Offer_List;
```

As a demonstration of the Trader's application to the provision of advanced information services in an electronic commerce environment, the colour definitions of the Services focus on some typical services which a "net hacker" might require.

```
(** Service Interface Types **)
color Int_Type = pizza | software | used_car;
(** Service Properties **)
color Properties = the_lot | vegetarian | simcity | spreadsheet | six_cylinder | sports;
color Serv_Prop = list Properties;
(** Service Providers **)
color Serv_Int_Id = with pizza_haven | pedros_pizza | software_supermarket |
                         virgin | bob_moran | champion;
```

A service exporter could submit a `Serv_Off` token with structure:

```
((used_car,[six_cylinder,sports]),bob_moran)
```

The shared place `Comms_Medium` is of colour Message which indicates that only message tokens may exist in the place. Each of the substitution transitions has an associated sub–net whose input/output port is connected to a transition with a guard. This guard checks the addressee (receiver) of the message and evaluates to `true` only if the message was addressed to that object. Only transitions with a guard which evaluates to `true` will be enabled, thus consuming the message when fired.

C. Trading Service Interface

The page TSI#8 (Figure 6) represents a Trading Service Interface, which is the interface between the Trader and other objects in the Trading Environment. It accepts messages and depending upon whether they are for the server (TSI_S#11) or client (TSI_C#12) mode, directs them to the appropriate substitution transition. Both of these modes are encapsulated in a page to reduce the "clutter" of this page. A TSI inherits the functionality of both the Import and Export operations as seen by the fact that both the Import_O#3 and Export_O#10 pages are sub–pages of TSI#8.

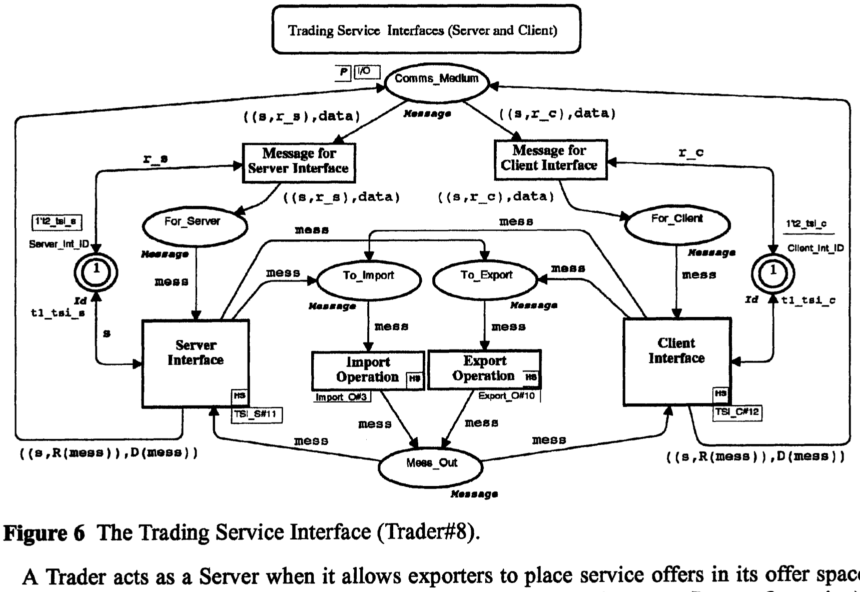


Figure 6 The Trading Service Interface (Trader#8).

A Trader acts as a Server when it allows exporters to place service offers in its offer space (Export operation), and when it accepts import requests from an importer (Import Operation). When the Trader is required to propagate an import request, it does so using the client mode of the TSI. The preceding import request is issued to all linked Traders, and the results are passed back to the Import Operation.

In order to allow unique instances of the TSI, two places are required to hold tokens with unique instance identifiers (`Server_Int_ID` and `Client_Int_ID`). These places are used in order to identify messages addressed to a specific instance of the TSI in either server or client mode. Design/CPN allows multiple instances of a page and thus, for each instance, the initial marking must be altered to contain a unique identifier.

Figure 6 shows `Server_Int_ID` having a marking of `1't2_tsi_s` which means "Trader 2, Trading Service Interface, Server mode". Since the only value which the variable `r_s` may be bound to is `t2_tsi_s`, any message which arrives in the `Comms_Medium` place must have a Receiver field `r_s` containing the same value in order for the "Message for Service Interface" transition to become enabled. This is an example of where pattern matching has been used to select a specific type of message.

In order to access information which is "hidden" inside tokens, some "Projection Functions" have been written. They allow the model to move data around as high order colours, whilst still allowing access to data within the token. Four basic functions (`S(mess)`, `R(mess)`, `Op(mess)`, `Op_Dat(mess)`) are defined to access the most commonly used data in a `Message` token, all of which follow a basic pattern. The input parameter to the functions is a message. For function S, the variable `s` is returned as the sender of the parameter message.

```
fun S(((s,r),dat:Data):Message):Id = s;
fun R(((s,r),dat:Data):Message):Id = r;
fun Op((addr:Addresses,(opn,op_d)):Message) :Operation = opn;
fun Op_D((addr:Addresses,(opn,op_d)):Message) :Operation = op_d;
```

The R(mess) function call returns the `Id` of the receiver of the Message and is used in the inscription on the arc from the transition "Server Interface" to the `Comms_Medium` I/O port.

D. Server Interface

Page TSI_S#11 models the Server Interface, where messages from clients are sorted according to operation type through the use of the `Op(mess)` function call (Figure 7). Initially, the token in place `Interface_Mode` is set to `idle`. When a message is received, if `m` is bound to `idle`, the message is checked to see if `Op(mess)` is in the set of either `Import_Ops` or `Export_Ops`. Depending upon this test, the token in `Interface_Mode` is modified to reflect the new Operation in process. While the state variable is not `idle`, messages are forwarded to the appropriate operation until `Op(Mess)=respond` which indicates the end of an Operation and the return to the `idle` state.

E. Import Operation Page

The Import operation is modelled on the Import_O#3 page (Figure 8) and is the most significant page of the Trader model. It models the processes which are performed when the Trader services an Import Request. The Trader Standard (ITU/ISO Trading, 1995) specifies the steps which are to occur to perform an Import Operation in the Computational Specification of the Trader. The CPN model uses the places "Incoming Mess" and "Outgoing Mess" as input and output ports respectively. All input and output occurs through these places. When a message arrives from an external object, the guards on the transitions connected to "Incoming Mess" allow only messages from the appropriate object to proceed.

From left to right, the vertical streams of token flow have the following functionality:

- The initial Import Request is checked to see if the parameters are correct. This has been modelled through the use of non–determinism. A boolean variable `result` of colour `Test` (which may take only two values (FAIL or PASS)) is bound to one of its values arbitrarily. The effect is that the transition will pass or fail the test non–deterministically, eliminating the need to write a function to perform the test. At this level of abstraction in modelling the Trader, we are interested only in the behaviour under certain conditions, not the implementation to determine whether parameters are correct or not. Given that the message passes both the Parameter and Security checks, a message is sent to the Search Policy Object (located on page Search#6), requesting it to send the current search policy.

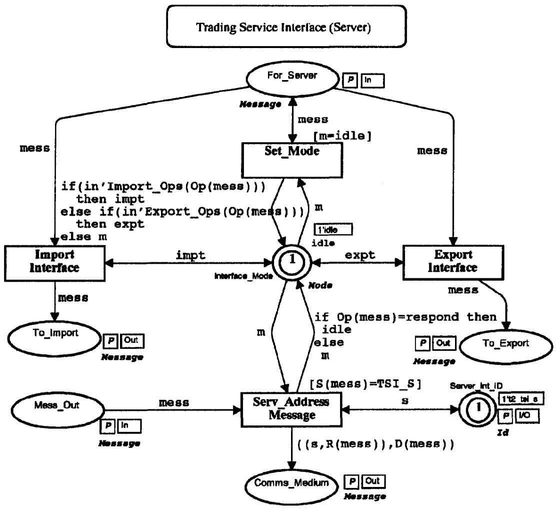


Figure 7 The Server Interface page (TSI_S#11).

The value of the token in Interface_Mode determines the path of incoming messages, either to the import or export operations. When the selected operation has completed, it sends a message to the requesting object where Op (mess) =respond, indicating that the request has been serviced. This message indicates the end of a service transaction, and the token in Interface_Mode is reset to idle. The Interface_Mode place limits the TSI to service an import or export operation, but never both at the same time.

- The Search Policy Object replies with the Search Policy which is used to determine whether a local search, a link search or both should be performed. A function call to Link_Local(S_Policy, Mess) determines if local and/or link searches are required. If a local search is required, a message is sent to the Offer Space object (Offer_Sp#7), requesting it to get_offers matching the initial Import Request (using the stored mess). If a link search is required, then a message is sent to the Link Space object (Link#9) with a request for it to get_links.
- When the Link Space object replies, the links are obtained and used to send off multiple requests (one to each linked trader) for an Import based upon the initial request. In addition, a counting token, maintaining the number of links being searched in place Number_of_Links, is updated from its initialised value of zero.

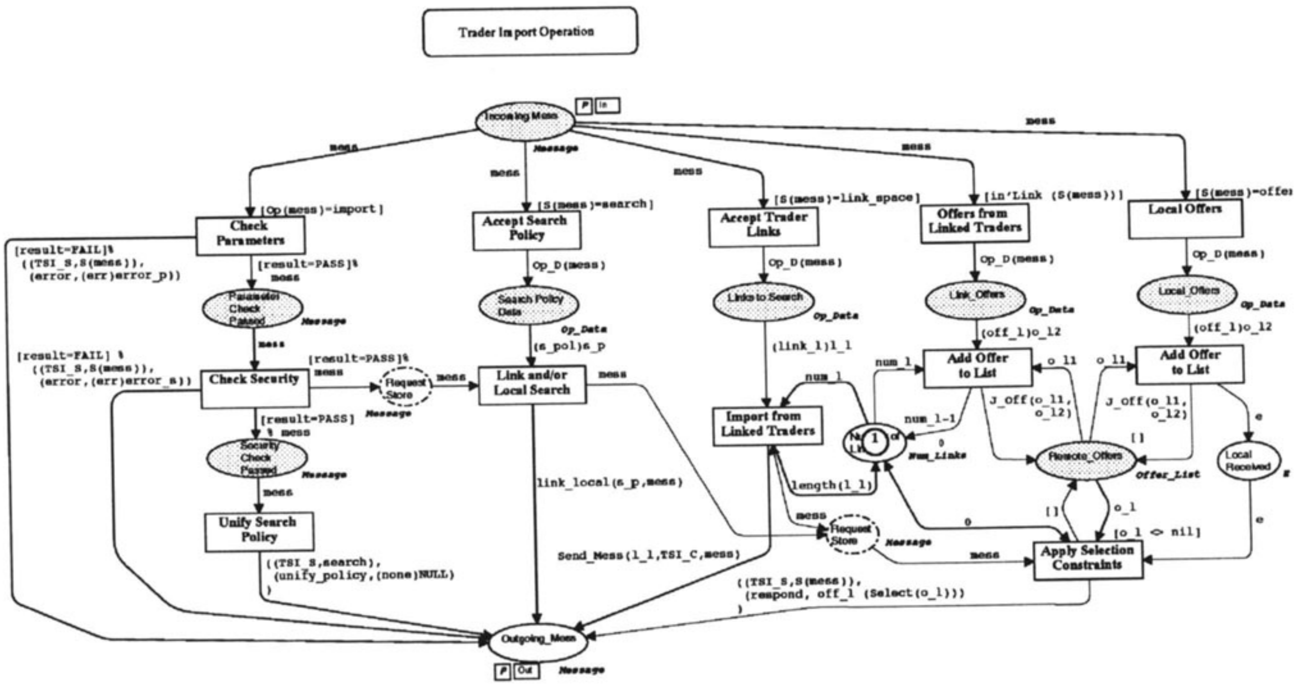


Figure 8 The Import Operation page (Import_Op#3).

- When the linked traders reply, offers are added to an initially empty list. The `Number_of_Links` counting token is used to indicate how many linked trader responses are still outstanding. Transition "Apply Selection Constraints" will not be enabled until the local offer space has been searched, and the counting token has reached zero. Thus, it will wait until all responses have been received and upon firing, apply selection constraints to the list of matching offers and return the "best" offer(s) to the importer.

In addition to the functions mentioned in section C, the model includes functions which operate in a recursive manner on lists of tokens. They are:

```
Send_Mess(Link_List, Id, Mess)
Match_Off(Serv_Type, Offer_Store)
```

`Send_Mess()` sends an import request to all of the linked Traders given in the `Link_List`, from the object specified by `Id` (used on page Import_O#3). Match_Off() searches the Offer_Store and attempts to find a match to the Serv_Type token which contains the match criterion (used on the Offer_Sp#7 page).

6. COMMENTS ON THE MODEL

6.1 Analysis

There are three methods which may be used in the analysis of a CPN – Simulation, Occurrence Graph Analysis and Invariants Analysis.

- Simulation is effective in debugging a model as the modeller has direct graphical feedback from the simulator and can monitor the passage of tokens as they flow throughout the model. It is limited, however, in that it is slow and requires direct human supervision to obtain results. Automatic Simulation Mode disables graphical feedback and terminates the simulation run after a user defined number of steps have occurred. Prior to simulation, Design/CPN performs a syntax check which ensures that arc and transition inscriptions are valid, and that places have an associated colour set.
- Occurrence Graph Analysis can be used to detect deadlocks, livelocks or forbidden states in the model. This is a particularly useful method in order to prove that a system will not "lock up". Its drawback is that it may require large amounts of computational resources for complex systems, and only gives results for a particular initial marking.
- Invariants analysis is more computing resource efficient (compared with the OGA memory requirements) and can be used to prove certain assertions about the system. It requires the modeller to determine which checks are necessary and to write appropriate queries for the tool to use which is itself in a prototype stage.

At this point, extensive debugging of the design has been performed through the use of the Design/CPN Simulator. In order to improve the performance of Formal Analysis using the OGA tool, the model should be modified to reduce the possible state space. This may be performed by removing intermediate places and through reduction of the number of colours which are used (sample data). In general, this results in reduced "readability" which is the trade–off. This model is intended for use in Simulation runs to gain a feel for Trader operation. The modified model will be derived from the simulation model, but will be used for verification of model properties.

It is intended to "port" the model into TORAS which has advanced analysis techniques including stubborn sets which promises to be very effective in reducing the state space of the model. The Trading system which has been modelled is a concurrent system where objects are loosely coupled. That is, there are sections which may operate with very little interaction between transitions. This is particularly conducive to analysis using the stubborn set method. Stubborn sets are sets of transitions that are not affected by external transitions in a given state [14]. This concept is used to select representative transition sequences and discard the rest. Most properties of the CPN are guaranteed to be preserved although not all.

6.2 Benefits

The model allows for true concurrency of operation. This is evident in the structure of the CPN model, where objects operate autonomously and in parallel. When an Import Request is processed, the local and link searches operate concurrently; that is, two messages are sent to different objects which process them concurrently. This is particularly attractive as it models the distributed nature of linked traders.

The CPN method allows an integration of data and control flow in one model. This is advantageous as it provides a more "complete" picture of the system being modelled. In addition, the model is executable which provides increased confidence in the model, through simulation runs. Although these runs provide only a limited traversal of the model's state space, use of tools such as TORAS allow exhaustive state space verification methods to be applied.

The model is both graphical and textual, allowing formalisation of data definitions in addition to a graphical description of the system relationships. The combination of these features improves comprehension of the model which is important when the model is used as a design/implementation tool.

6.3 Limitations

At present, the model does not include any Management functions which are implemented by the Trader Management Interface, nor a detailed offer space graph which allows offers to be partitioned. This may included through modification of the Offer Space Object (Offer storage algorithm). Policy information (import, export, trader) is limited to search policy information and is not updated to reflect Traders which have already been searched. This is a matter of including a list of visited traders in the search policy of a request.

7. CURRENT WORK

The model will be extended to include pages to represent all Trader functions (Import, Export, Withdraw, Modify), thus providing a complete formal, executable, specification of the Trader. Interfaces between Trader and external objects will be included, allowing multiple channels of communication between Trader instances and client objects. This will eliminate the need for the TSI's to manage messages from external objects during the servicing of requests.

Support for concurrent servicing of requests will be included by creating multiple TSI's and assigning internal transaction identifiers when processing import/export requests. This will allow the Trader to offer multiple TSI's and process multiple requests as they arrive. Formal analysis of the Trader using the OGA will be performed, with the aim of finding deadlocks and/or unwanted states.

8. CONCLUSIONS

We believe that the Trader has an important role to play when electronic commerce becomes widespread which is foreseeable in the near future. In order to verify the correctness of the emerging standard, a model has been created which is formally verifiable and executable through simulation runs. This allows us greater understanding of the system, and increases our confidence in an implementation. CPNs are an appropriate choice for the modelling of Trader as they have a graphical representation, are excellent for modelling concurrency, and have a formal semantics which allows us to prove properties of the model. Our research will verify the Trader and subsequently apply the Trader to an electronic commerce environment where an obvious need exists.

REFERENCES

Bastide, R. (1995) *"Approaches in unifying Petri nets and the Object–oriented Approach"*, Available at: **http://www.dsi.unimi.it/Users/Labs/PetriLab/ws95/abstract/bastide.html**

Bietz, A., Berry, A., Lister, A. and Raymond, K.A. (1993) *"Introduction to Open Distributed Processing"*, Proceedings of the ACS Queensland Branch Conference – Overcoming Isolation: The Human–Computer Connection, Townsville, Australia, pp. 27–34.

Billington, J., Wheeler, G. R. and Wilbur–Ham, M. C. (1988) *"PROTEAN : A High–level Petri Net tool for the Specification and Verification of Communication Protocols"*, IEEE Transactions on Software Engineering, Vol 14, No. 3.

Billington, J. (1991) *"FORSEEing Quality Telecommunications Software"*, Proceedings of the First Australian Conference on Telecommunications Software, (ACTS) April 1991.

Dibold, H. (1992) *"Hierarchical Coloured Petri Nets for the Description of Services in an Intelligent Network"*, International Zurich Seminar on Digital Communications : IN and their Applications, 1992.

ITU/ISO ODP (1994) *"Reference Model of Open Distributed Processing – Part 1: Overview and Guide to Use"*, Draft International Standard 10746–1 , Draft ITU–T Recommendation X.901, 1994.

ITU/ISO Trading Tutorial (1995) *"Reference Model of Open Distributed Processing – Trading Function Annex A: Tutorial of the Trading Function"*. Available at: **http://www.dstc.edu.au/AU/research_news/odp/trader/tr_tutorial.html**

ITU/ISO Trading (1995) *"Reference Model of Open Distributed Processing – Trading Function"*, ISO/IEC DIS 13235 | Draft Recommendation. X.950, June 1995. Also available from: **http://www.dstc.edu.au/AU/research_news/odp/trader/standards.html**

Jensen, K. (1992) *"Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1 : Basic Concepts"*, EATCS Monographs on Theoretical Computer Science, Springer–Verlag.

Jørgensen, J.B., Mortensen, K. H. and Sousa, A. V. (1994) *"Modelling and Analysis of Distribution in BETA Using Coloured Petri Nets"*, Technical Report, Computer Science Department, Aarhus University.

Meta Software Corporation (1992) Design/CPN User's Manual. Cambridge, Mass, U.S.A.

Mortensen, K. H. and Pinci V. (1994) *"Modelling the Workflow of a Nuclear Waste Management Program"*, Application and Theory of Petri Nets 1994, LNCS 815, R. Valette (Ed), Springer–Verlag, pp. 376–395.

Valmari, A. (1989) *"Stubborn sets for reduced state space generation"*. Advances in Petri Nets 1990, LNCS 483, Springer–Verlag 1990 pp. 491–515, originally appeared in Proceedings of 10th International Conference on Application and Theory of Petri Nets, Bonn, Vol II, pp. 1–22.

Vogel, A., Bearman, M. and Beitz, A. (1995) *"Enabling Interworking of Traders"*. Open Distributed Processing: Experiences with distributed environments, K. Raymond, L. Armstrong (Eds), Chapman and Hall, 185–196. Also available at: **http://www.dstc.edu.au/AU/staff/andreas–vogel/papers/icodp95.ps**

Wheeler, G., Valmari, A. and Billington, J. (1990) *"Baby TORAS Eats Philosophers but thinks about Solitaire"*, Proceedings of the Australian Software Engineering Conference (ASWEC) 1990.