

Modelling and Analysis of Distributed Program Execution in BETA Using Coloured Petri Nets

Jens Bæk Jørgensen and Kjeld Høyer Mortensen

University of Aarhus, Computer Science Department
Ny Munkegade, DK-8000 Aarhus C, Denmark
{jbj,khm}@daimi.aau.dk

Abstract. Recently, abstractions supporting distributed program execution in the object-oriented language BETA have been designed. A BETA object on one computer may invoke a remote object, i.e., an object hosted by another computer. In this project, the formalism of Coloured Petri Nets (CP-nets or CPN) is used to describe and analyse the protocol for remote object invocation. In the first place, we build a model in order to describe, understand, and improve the protocol. Remote object invocation in BETA is modelled on the level of threads (lightweight processes) with emphasis on the competition for access to critical regions and shared resources. Secondly, the model is analysed. It is formally proved that it has a set of desirable properties, e.g., absence of dead markings.

1 Introduction

In this project, the formalism of Coloured Petri Nets (CP-nets or CPN) [11] is used to describe and analyse the protocol for remote object invocation in the object-oriented language BETA [14].

The project is divided into a construction stage and an analysis stage. In the construction stage a model is built in order to describe and understand the considered protocol. Several meetings are held between the modellers and the designer of the protocol. In the process, the designer increases his own understanding. As a consequence, a number of changes are made.

In the analysis stage, the protocol is verified. It is formally proved that it has a set of desirable properties. E.g., we prove that the protocol has no deadlocks, that certain BETA objects always have the chance to do remote object invocations (liveness), and that a monitor construction correctly ensures exclusive access to a critical region. We apply recently developed computer tools for formal analysis of CP-nets, an occurrence graph tool and a place invariant tool.

The rest of this paper is structured as follows: In Sect. 2, the system supporting distributed program execution in BETA and the protocol for remote object invocation are introduced. Sect. 3 describes the constructed CPN model and Sect. 4 its analysis. In Sect. 5 related work is discussed. Finally, in Sect. 6, we draw some conclusions.

2 Description of the DistBETA System

The system considered in this project will be called the *DistBETA system* [2,3]. The DistBETA system is a framework for distributed program execution in BETA. It includes the protocol for remote object invocation. In this section we first introduce a set of concepts from the DistBETA system that are used to describe the protocol. Then we explain the protocol itself.

The following three concepts are relevant for the remote object invocation protocol:

- *Ensemble*: Is a representation of the operating system on a computer connected to a network.
- *Shell*: Is similar to a process. Shells exist inside ensembles. A shell can communicate with another shell in a remote ensemble or in its own ensemble. Moreover, a shell can communicate directly with its ensemble.
- *Thread*: Each shell contains at least one *user thread* executing the main program and exactly one *listener thread* taking care of incoming requests from the network.

The framework that the application programmer uses to support distributed program execution contains a class called the *RPC handler*. The RPC handler includes the necessary primitives for serialization (marshalling) and communication. The framework is more general than RPC¹. An object can act both as a client and as a server, allowing arbitrarily long invocation-chains wandering through many different computers. The parameters passed in an invocation are not just values as for RPC. They may be objects or references to objects, to which messages can be sent resulting in object invocations.

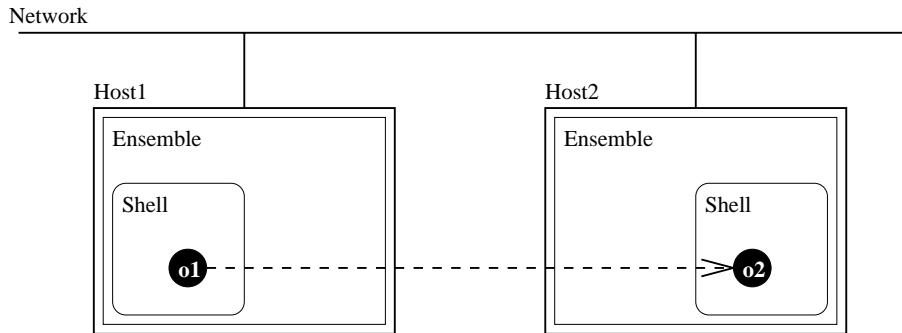


Fig. 1. A remote object invocation.

¹ RPC is an acronym for remote procedure call. For an introduction, see [16]. In this paper the term “RPC” means remote object invocation.

In the following we describe the protocol for remote object invocation (see Fig. 1). Suppose that an active object `o1` wants to invoke another object `o2`. The two objects are physically separated on two computers, `Host1` and `Host2` respectively. Each object has a unique object identifier (OID)². The sequence of events is as follows:

1. `o1` looks up the OID of the object to be invoked (`o2`) in a table containing OIDs of remote objects. The table is local to the shell of `o1`. `o1` allocates a resource containing the necessary primitives for serialization. All parameter objects are serialized. `o1` invokes a method for sending remote object invocation requests in the RPC handler.
2. The remote object invocation together with its associated serialized parameter objects are sent to `Host2`. The invoking object `o1` is blocked.
3. The RPC handler in the shell enclosing `o2` receives the incoming request. A worker thread containing the necessary primitives for invoking objects is allocated. The worker thread unserializes the received parameter objects. While doing so it also looks up the OID of the receiver object in a table containing OIDs of local objects. The table is local to the shell of `o2`.
4. The object `o2` is invoked with the unserialized parameter objects.
5. The worker thread gets the result which is serialized. Control is handed over to the RPC handler again. The worker thread is released and the result is sent back to `Host1`.
6. The result is unserialized by `o1`, the resource allocated is released, and finally the result is given to `o1`.

When objects are passing the boundary of shells their OIDs are looked up in tables of remote or local OIDs. Upon serialization it is checked if new objects cross the boundary. If a new OID is needed for an object not yet in any of the tables, it is necessary to communicate with the relevant ensemble, since it is the ensembles that generate unique OIDs.

Many shared resources and critical regions are involved in the sequence of events described above: Allocating and releasing resources, allocating and releasing worker threads, looking up OIDs in tables of remote and local objects, and communication with ensembles upon requests for new OIDs. Monitors and semaphores are used to grant exclusive access for competing threads. Threads compete with other threads belonging to the same shell. The DistBETA system ensures that no thread is starved, by associating queues with monitors and semaphores.

3 Description of the CPN Model

The CPN model of the DistBETA system describes the basic flow of control inside active objects. The model is a hierarchical CP-net consisting of global

² The purpose of an OID is to have a database key. Some objects may be persistent, i.e., they may survive between program executions, and are typically stored in a database on a permanent storage (as a hard-disk). The OID is then used to retrieve the object again or can even be used to get type information about the object.

declarations plus 12 pages with net structure. It aims at describing the remote object invocation protocol of the DistBETA system, emphasising how objects compete for shared resources and access to critical regions. In this section we provide a description of the CPN model. In Sect. 3.1 some of the important declarations are outlined. The net structure is explained in Sect. 3.2. Sect. 3.3 addresses the limitations of the model.

The model is built with Design/CPN [12], a general editing, simulation, and analysis tool for CP-nets. Design/CPN uses the language CPN ML for declarations and inscriptions. CPN ML is an extension of the functional programming language Standard ML [18].

3.1 Global Declarations

The basic components of the DistBETA system are ensembles, shells, and threads, which are all active objects, plus messages and packets, which are both passive objects.

Colour sets for these components are declared in a straightforward fashion in CPN ML. Most places in the model have colour set **Thread**. The movement of **Thread**-tokens describe the main flow of the model. A **Thread**-token being in a certain place is similar to a program counter having a certain value.

A token from colour set **Thread** is a pair. The first component identifies the thread. The colour set identifying a thread is called **ThreadInfo**. It is a record colour set with fields for identities of an ensemble, a shell, and a thread. The second component of a **Thread**-token is an environment holding information that the thread needs at certain points in its lifetime, e.g., values of local variables. It is modelled by the record colour set **Environment**.

There are different kinds of **Thread**-tokens used for different purposes: User threads are making requests for remote object invocations, listener threads are receiving requests, and worker threads are handling the requests and subsequently returning results.

Threads communicate by sending packets over a network. Packets are modelled in CPN ML by the colour set **Packet**, which is a record type holding a sender, a receiver, and a message. The set of messages is very coarse. Basically, a message is either a request or a result because we are only concerned with its direction. As an example a real BETA object may want to send a message like $(2, 5) \rightarrow \text{add}$ to a remote object and expect the result 7 to be returned, but this level of description is not necessary for our purpose. We are concerned with communication patterns only, not with the data involved. In addition to the basic types of requests and results, a message may be a network error message.

3.2 Net Structure

The net structure can be seen from the hierarchy page of the CPN model, shown in Fig. 2.

The hierarchy page has a node for each page of the model. An arc between two nodes indicates that the source node contains a transition whose behaviour

is described on the page of the destination node. Such a transition is called a substitution transition. The page of the destination node is called a subpage.

The model consists of four parts, each one with its own well-defined meaning: A top-level part, a network part, a sender part, and a receiver part. Some pages are shared between the sender and receiver part. This means that these pages have more than one page instance. Places, transitions, and arcs on a page with more than one instance, accordingly appear in more than one instance³.

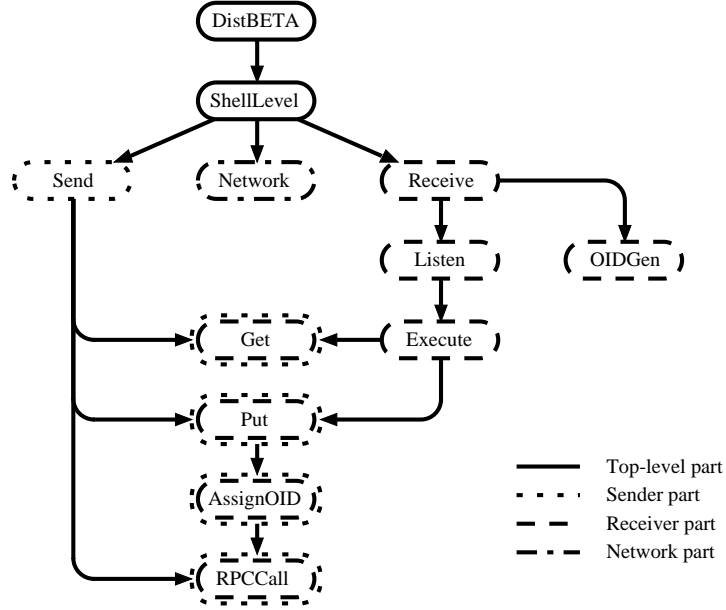


Fig. 2. The hierarchy page.

A detailed description of the model can be found in [13]. Below we describe two typical pages. The other pages are comparable with respect to the number of places, transitions, and arcs; and the complexities of the arc inscriptions.

Page RPCCall The page RPCCall shown in Fig. 3 is first described. The four places with thick border have colour set **Thread**. The two places with a dot-dashed border model the interface to the network. Conceptually they contain packets. Each one of them contains a list of packets for each shell. These lists act as input and output buffers to the network. To enhance readability, all colour sets are hidden. The variable **thrinf** is over colour set **ThreadInfo** and the variable **envr** is over colour set **Environment**.

³ In this paper, when no confusion is possible, we say “page” instead of the proper term “page instance”. Similar remarks apply to places, transitions, and arcs.

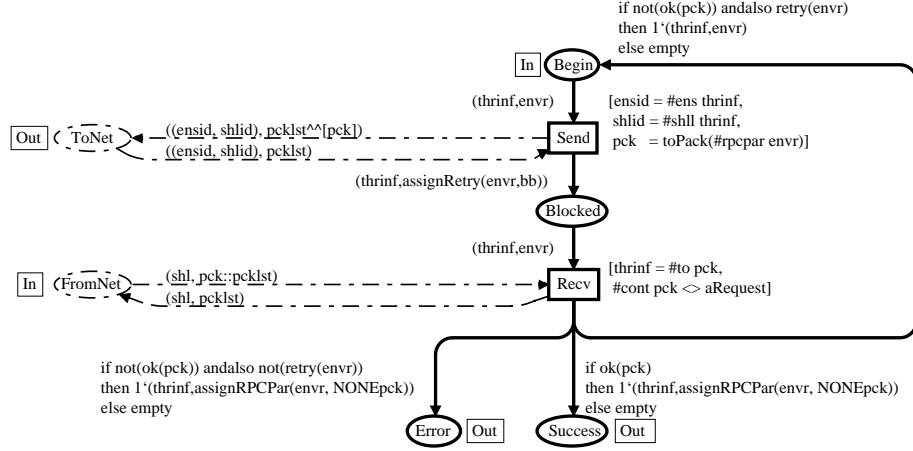


Fig. 3. The page `RPCCall`.

The page `RPCCall` is a central part of the CPN model. It models the behaviour of the sender side⁴ in a remote object invocation assuming the necessary preceding work has been done, e.g., serialization of parameters. When a user thread is in the place `Begin`, it is ready to initiate the remote object invocation. When the transition `Send` occurs, a packet is directed to the network, modelled by the network interface place `ToNet` receiving a certain token. A packet is appended to the network input buffer for the shell enclosing the user thread, modelled by the arc inscription⁵ $((ensid, shlid), pcklst^^[pck])$. After having sent, the user thread is blocked waiting for an answer. It sits in the place `Blocked`. This models the synchronous nature of the communication. An answer will eventually emerge from the network, modelled by a certain token being available on the other network interface place `FromNet`. When this is the case, the transition `Recv` becomes enabled for the waiting user thread. In the normal case (the expression `ok(pck)` is true), an ordinary result comes back. In this case, the remote object invocation went well and the user thread ends up in the place `Success`. If an error appeared (`ok(pck)` is false), the user thread may end up in the place `Error`, or it may return to the place `Begin` and try to do the failed invocation once again depending on the value of the boolean variable `retry` in its environment.

Page `AssignOID` Now the page `AssignOID` shown in Fig. 4 is described. It models a monitor construction ensuring unique `OIDs` upon request from user threads.

⁴ `RPCCall` is also included in the receiver part because a receiving object sometimes does an RPC with its ensemble in order to obtain an `OID`.

⁵ The operator `^^` concatenates two lists.

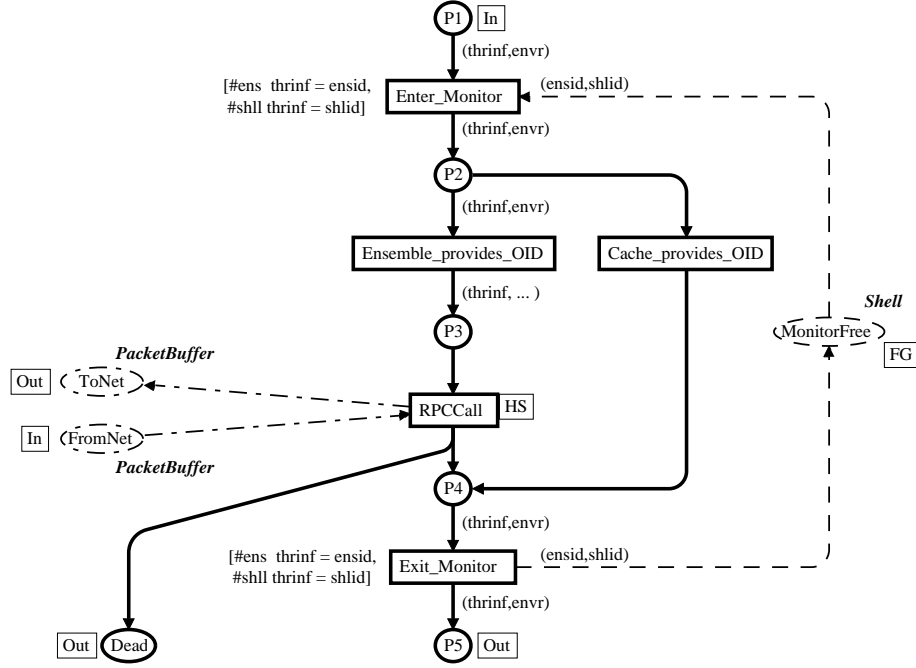


Fig. 4. The page `AssignOID`.

The monitor is controlled by the place `MonitorFree` which initially contains a token for each shell. A user thread is inside the monitor exactly when the corresponding token is in a place that is inside the monitor. The inside of the monitor is made up of places both on the page `AssignOID` itself and its subpage `RPCCall`. On `AssignOID`, the places `P2`, `P3`, `P4`, and `Dead`⁶ are inside the monitor. In addition, the monitor includes the places on the subpage `RPCCall`. The entry to the monitor is modelled by the transition `Enter_Monitor`. When it occurs, a user thread enters the monitor, and the appropriate token is removed from `MonitorFree`. This disallows any other user thread from the same shell from entry before the appropriate token is redeposited in `MonitorFree`. The transition `Exit_Monitor` models exit from the monitor. When it occurs, the token corresponding to the shell of the exiting user thread is added to `MonitorFree`.

3.3 Limitations

The model explained in this section is an abstraction of the protocol for remote object invocation of the DistBETA system. We have aimed at keeping the model as simple and clear as possible. Simplicity and clarity are of course natural goals

⁶ A user thread is terminated, i.e., ends at the place `Dead`, if the ensemble fails to provide an OID upon request.

by themselves, but in this project they are more than just worth striving for. They are necessary in order for the model to be tractable for formal analysis. In agreement with the designer of the DistBETA system, we choose to leave out certain aspects from the model, though still keeping it realistic and usable. The most important limitations are listed below. They are justified in detail in [13].

In the DistBETA system, a thread may invoke an object on a remote computer. Upon invocation, a new thread is created on the other computer. This thread may invoke an object on yet another remote computer causing the creation of a new thread on that computer, etc. In this way an active object can act both as client and server at the same time. We do not include this aspect in our model. It would cause the number of reachable markings to be infinite.

As mentioned at the end of Sect. 2, the DistBETA system cannot starve any threads competing for access to critical regions and shared resources. The CPN model can. It is easy to construct an infinite occurrence sequence starving any given user thread. Preventing starvation in the model requires use of complex data structures.

In the DistBETA system, communication errors may happen and be detected both on the sender side and on the receiver side. In the CPN model errors can only happen on the sender side. This is straightforward to model since errors here can be handled locally. Error handling on the receiver side is much more complex.

4 Analysis of the CPN Model

Both informal and formal analysis methods are applied to the CPN model in this project. The informal analysis consists in simulating the model. Simulation is an important activity in any CPN modelling project. In this paper however, our focus is on describing the formal analysis of the CPN model.

This section consists of two subsections, one for each of the two formal analysis methods used. Occurrence graph analysis is described in Sect. 4.1 (occurrence graphs are also known as state spaces and reachability graphs). Place invariant analysis is described in Sect. 4.2. Using the formal methods, we are able to prove that the CPN model has certain properties. Both techniques are supported by computer tools. For a thorough introduction to formal analysis of CP-nets, see [11].

4.1 Occurrence Graphs

In this section, first the main characteristics of the occurrence graph method are recalled. The goals of our analysis and the results obtained are described next. Finally, an attempt to alleviate the state explosion problem using net reductions is discussed.

The Method An *occurrence graph* for a CP-net is a directed graph with a node for each reachable marking and an arc for each occurring binding element⁷. An arc is going from the node of the marking in which the associated binding element occurs to the node of the marking resulting from the occurrence.

All standard dynamic properties for a CP-net⁸ can be derived from its occurrence graph, e.g., boundedness, home, liveness, and fairness properties. It is worthwhile also to construct the *SCC-graph* for the occurrence graph. The SCC-graph has a node for each of the strongly connected components of the occurrence graph. If there is an arc in the occurrence graph between two nodes from different strongly connected components, then the corresponding arc is in the SCC-graph. Investigating the SCC-graph instead of the full occurrence graph may significantly speed up the check of a dynamic property. Using Tarjan's algorithm (see, e.g., [8]), the construction of the SCC-graph is an inexpensive operation. It is linear in the size of the occurrence graph.

The most serious drawback of the occurrence graph method is the *state explosion problem*: Very often, even for relatively small CP-nets the occurrence graphs get so big that they cannot be generated, even with the most powerful computers available. Another limitation inherent to the occurrence graph method is *dependency of the initial marking*: Each possible initial marking of a considered CP-net may yield a new occurrence graph. Thus verifying properties for all values (perhaps infinitely many) of some system parameter requires generation of an occurrence graph for each value.

The tool used for the occurrence graph analysis is called the Design/CPN Occurrence Graph Tool (Occ Tool) [4]. It is an application that is integrated with the basic simulation tool offering functionalities to generate occurrence graphs, to draw them, and to do queries. All our analysis including generation of the occurrence graphs is conducted on a Sun Sparc 20 with 256 MB physical RAM.

Analysis Goals The state explosion problem and the dependency of the initial marking are general problems of the occurrence graph method. However, in a concrete application, we can try to find workarounds.

This specific CPN model is created with formal analysis in mind. Therefore, we have carefully tried to choose colour sets and to build the net structure so that the state explosion is controlled, at least for small initial markings, i.e., initial markings containing only a few user threads (and consequently only a few ensembles and shells).

Our model does have an infinite number of legal initial markings. What we will call a *configuration* is determined by the number of ensembles; for each ensemble, the number of shells; and for each shell, the number of user threads. Each configuration uniquely induces an initial marking. The model has an infinite number of legal configurations. Because of the dependency of the initial marking problem, we have no chance of verifying the model in general. However, if we

⁷ A *binding element* is a pair containing a transition and a binding. The binding assigns values to all variables in the surroundings of the transition.

⁸ Here we consider only CP-nets with a finite number of reachable markings.

show that the considered properties are satisfied for a number of configurations, our confidence in the CPN model is increased.

The occurrence graph analysis focusses on proving the two vital dynamic properties stated below.

1. The CPN model has no dead markings.
2. Each user thread can forever participate in the basic send/receive communication. Formally this can be stated as two specific sets of binding elements being live — one set of binding elements for each of the transitions **Send** and **Recv** on the page **RPCCa11** (see Fig. 3) of the sender part.

If the CPN model has these two properties, there is strong evidence that the protocol is well-functioning. Property 2 means that each user thread remains active, i.e., it will always have the chance to request a remote object invocation, and a request will always be followed by a result (which might be an error notification). It is obvious that property 1 is a consequence of property 2. Thus it suffices to prove 2. As an aid we show:

3. The CPN model is always able to return to its initial marking, i.e., the initial marking is a home marking.
4. For any given user thread there is an occurrence sequence starting in the initial marking and containing an occurrence of each of the transitions **Send** and **Recv** on the page **RPCCa11** of the sender part. Both occurrences with the variable **thrinf** bound to the given user thread.

Property 3 states that from any given reachable marking, it is possible to find an occurrence sequence leading back to the initial marking. Property 4 says that from the initial marking we can find an occurrence sequence containing a binding element from the set we are analysing for liveness. Hence, together 3 and 4 imply 2. It is well-known that 5 below implies 3. In summary, it is sufficient to prove 4 and 5.

5. The SCC-graph consists of exactly one component.

The original CPN model does not have the listed properties 1 and 2. The reason is rather technical and is explained in [13]. The violation is not a modelling error, but is inherent to the DistBETA system. Fortunately, we can easily construct a modified version of the model which we will prove to satisfy the properties. All we have to do is to make the communication between user threads asking for OIDs and their ensembles error-free. For this reason the CPN model analysed with the occurrence graph method is a slight variation of the original one. The modification is accomplished just by giving the transition **Ensemble_provides_OID** on the page **AssignOID** (see Fig. 4) the guard **[false]**.

Analysis Results It is our aim to prove the properties 1 and 2 for as many configurations as possible. However, due to the state explosion problem, we cannot expect to be able to handle configurations with many user threads. The experiments will take us as far as we can get before the computer runs out of memory. Our results are summed up in Fig. 5 and are commented below. In the figure a configuration is visualised as in Fig. 1: Boxes represent ensembles, rounded boxes represent shells, and black filled circles represent user threads. E.g., configuration 1 has one ensemble with one shell containing two user threads. The time indicated in the figure is wall-clock time for the occurrence graph generation, not CPU time. The computer was only spending little time running other processes while the occurrence graph generations were going on.





	<i>Configuration</i>	<i>Time</i> seconds	<i>Nodes</i>	<i>Arcs</i>
1		96	5,501	13,725
2		760	21,554	54,793
3		653	21,554	54,793
4		$\geq 5,247$	$\geq 75,018$	$\geq 183,827$

Fig. 5. Statistics for generation of occurrence graphs.

Intuitively configuration 1 is more likely to lead to a dead marking than configurations 2 and 3. All monitors, resources, and critical regions are local to shells. Configuration 1 has two user threads running within the same shell, so conflicts will arise. In this case, it is possible to verify directly that property 5 holds. Property 4 is established in the simulator. As an alternative, property 4 may be proved by finding an appropriate path in the occurrence graph. Thus for configuration 1, the desired properties 1 and 2 hold.

For configurations 2 and 3, properties 1 and 2 are shown similarly. As an aside, we note that the two graphs have exactly the same size. This fact is no surprise, because in both cases, the two user threads are independent, i.e., they never have to wait for each other because they run in different shells.

For configuration 4, the full occurrence graph is too big to be generated. Thus this configuration cannot be analysed with the available tool and computers. This result is of course negative for the verification of our CPN model, but it does exhibit the state explosion problem very clearly. Increasing the number of user threads from two to three causes the occurrence graph to grow to a size that we presently cannot handle. We will return with an attempt to tackle configuration 4 at the end of this section. For now, we can only generate a partial occurrence graph, i.e., a graph where only a subset of the reachable markings

and the occurring binding elements is included. Therefore, we only have lower bounds for the number of nodes and arcs, and for the generation time.

There are a number of configurations with a total of three user threads, e.g., three ensembles, one shell in each, and one user thread in each shell. Of these possibilities configuration 4 is the one with the lowest number of reachable markings: In a system with three user threads running in the same shell, the user threads are forced to wait for each other once in a while. Thus the behaviour is more restricted than, e.g., the system mentioned above. Obviously, generation of a full occurrence graph is not possible for a configuration with more than three user threads. Therefore, we have not tried to generate occurrence graphs for more configurations. In summary, with the software and hardware used for these experiments, configurations with less than three user threads can be analysed with the occurrence graph method. Larger configurations cannot.

Above the Occ Tool was used to verify properties for the final version of the CPN model. It is not the only way to use it. It is a convenient tool for debugging in the model creation stage as well. It provides a systematic way to investigate all occurrence sequences, in contrast to simulation where only one occurrence sequence at a time is considered. The process of finding and correcting modelling errors in this project was eased by using occurrence graphs. For more details see [13].

Alleviating State Explosion Using Reductions As mentioned at the beginning of this section, we make an attempt to alleviate the state explosion problem. We do so using *reductions*. Reductions of Petri nets are among the classical approaches to analysis. The basic idea is to derive a net from the original one in a systematic way using a set of rules that are known to preserve a selected set of dynamic properties. We certainly expect the derived net to have a smaller occurrence graph than the original.

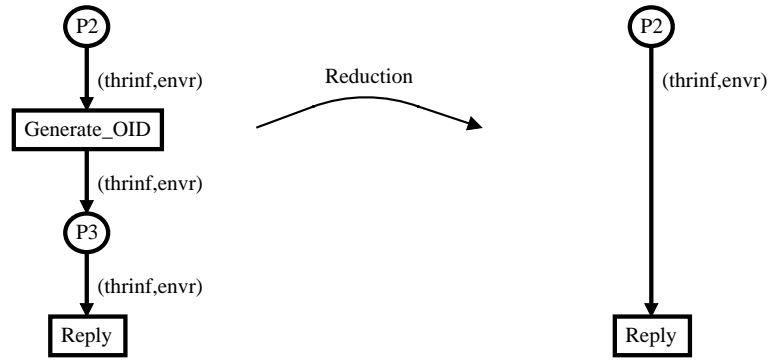


Fig. 6. Reduction of the CPN model using pre-agglomeration.

To illustrate how reductions are applied to our model, consider the extract of the model shown in Fig. 6. Removal of the transition `Generate_OID` and the place `P3` results in a simpler model. This reduction is a special case of *pre-agglomeration of transitions* defined in [10]. In this reference it is proved that the reduced model has the same properties as the original with respect to the properties we are considering: Home markings, liveness, and dead markings.

We make a number of pre-agglomerations of transitions in the model. In total, five places and five transitions are discarded. The relevant statistics for the occurrence graph generation is shown in Fig. 7.





	<i>Configuration</i>	<i>Time</i> seconds	<i>Nodes</i>	<i>Arcs</i>
1		47	2,541	6,877
2		338	9,810	27,497
3		345	9,810	27,497
4		$\geq 10,340$	$\geq 75,001$	$\geq 245,794$

Fig. 7. Statistics for generation of occurrence graphs for the reduced model.

The occurrence graphs for the reduced model are significantly smaller than for the original. For each of configurations 1, 2, and 3, the graph for the original model has approximately twice as many nodes as the graph for the reduced version. The same remark applies to the number of arcs. Smaller occurrence graphs for the reduced model is of course the expected result. Anyway, it is remarkable that removal of only five places and five transitions yields occurrence graphs of sizes half of the original. The original model has 70 places. Thus removing about 7% of the places gives decent reductions in the sizes of the occurrence graphs. The lesson learned is that it is sensible to apply as much as possible reduction strategies preserving the properties we are interested in before an occurrence graph is generated. Small reductions in the net structure may yield significant reductions. But alas, even with the reduction technique, we cannot handle configurations with more than two user threads. The occurrence graph for configuration 4 is partial, also for the reduced model.

4.2 Place Invariants

In this section, first the main characteristics of the place invariant method are recalled. The goals of our analysis and the results obtained are described next.

The Method The basic idea of the place invariant method is to find equations that are satisfied for all reachable markings of a considered CP-net. To explain how in more details, we recall the concepts of weights and sets of weights. In this context, a *weight* is a linear function associated with a certain place in the considered CP-net. Its domain is the place colour set. It can also be applied to multi-sets over the place colour set. A *set of weights* contains exactly one weight for each place, and all its weights have a common range. Given a set of weights, the weight of a place is computed by providing its marking as argument to the weight. The weight of a marking of the net is the multi-set sum of the weights of all the places. A set of weights can be viewed as a means to extract some specific information that we are interested in, from the complex information of the full marking. Typically, a set of weights will contain zero weights⁹ for a large number of places and non-zero weights for a relatively small number of places. The set of places for which the weight is non-zero is called the *support* of the set of weights. A *place invariant* is a set of weights for which the weights of all reachable markings are identical.

Compared to the occurrence graph method, an advantage of the place invariant method is that it does not suffer from the state explosion problem. To check if a proposed set of weights is a place invariant does not require generation of all reachable markings. Instead, the check may be done statically and locally: For each binding element it is checked that the weight of the multi-set of tokens removed is identical to the weight of the multi-set of tokens produced upon occurrence. In this case, we will say that the binding element preserves the weight. Typically, all binding elements of a given transition may be checked simultaneously. A transition preserves the weight when all its binding elements preserve the weight. The set of all binding elements is known in advance, thus investigating all reachable markings is not needed. We just have to check that all transitions preserve the weight.

Another advantage of the place invariant method is that it is not dependent of the initial marking. When proving that all transitions preserve the weight, only the static inscriptions appearing in arcs and guards in the net are considered.

The place invariant analysis is conducted using the Design/CPN Place Invariant Tool (Inv Tool) [17]. It is a research prototype. For a given set of weights, it checks if all transitions preserve the weight: The CPN ML expression for the weight of the net effect of a transition is translated into a lambda expression. The Inv Tool uses lambda reduction rules to rewrite this expression. When no more reductions are possible, it checks if the expression equals the zero function.

Analysis Goals The aim of the place invariant analysis is to increase our confidence in the CPN model. Throughout the model construction stage, we have in mind a set of important properties that a sensible model must satisfy. Using place invariants, we are able to prove that the final model actually has these properties. Most of them are very similar. They state that certain sets of

⁹ A zero weight maps each multi-set into the empty multi-set.

tokens remain constant. Another property says that a monitor construction is correct. Specifically we prove the following:

1. The set of user threads is constant, i.e., no user thread ever disappears and no new user thread is ever created.
2. The set of listener threads is constant.
3. The set of network input buffers is constant.
4. The set of network output buffers is constant.
5. The monitor ensuring unique OIDs on the page **AssignOID** (see Fig. 4) is correct, i.e., there are never two user threads from the same shell inside the monitor at the same time.

Analysis Results First we consider the verification of property 1: We are aiming at defining a suitable set of weights. We take the weight for any place that has colour set different from **Thread** to be zero. All places with colour set **Thread** get the same weight: If the argument is a user thread, it returns the identity of the thread, i.e., it ignores the environment. If the argument is a thread which is not a user thread, it returns the empty multi-set. The Inv Tool is able to verify that the weight set thus defined is a place invariant. Thus property 1 is proved. A set of weights corresponding to property 2 is constructed similarly.

Properties 3 and 4 are shown by constructing appropriate sets of weights whose support are the network buffer places **ToNet** and **FromNet** (see Fig. 3) respectively.

To establish property 5, we need a more sophisticated place invariant. Consider the page **AssignOID** shown in Fig. 4 in Sect. 3. The set of weights proving 5 contains the same weight for all places inside the monitor — a weight *shell* that maps a **Thread**-token to its enclosing shell. The set of weights contains the identity-function for **MonitorFree** and zero for all other places in the model. If this set of weights is a place invariant, it states that the places with non-zero weights in any reachable marking M together contain exactly one token from each shell (given the initial marking). In terms of an equation:

$$M(\text{MonitorFree}) + \sum_{p \in \text{InMonitor}} \text{shell}(M(p)) = \text{AllShells}$$

where *AllShells* is a multi-set containing exactly one appearance of each shell, and *InMonitor* is the set of all places inside the monitor. Thus assuming two or more user threads from the same shell inside the monitor at the same time is a contradiction: The multi-set on the left-hand side of the equation contains an element with coefficient at least two and the right-hand side does not.

The Inv Tool tool is not able to establish that the set of weights defined for property 5 is a place invariant. E.g., it cannot verify that the transition **Recv** on the page **RPCCall** (see Fig. 3) preserves the weight. This is a shortcoming in the tool. The involved arc expressions, e.g., on the **RPCCall** page are too complex for the present prototype. If we analyse the model that is modified by making the communication between user threads asking for OIDs and their ensembles

error-free, property 5 can now be established by the Inv Tool. The modification must be done by explicitly deleting parts of the model. The reason is that it is thus not necessary to check the complex transitions on the page `RPCCall`, which were causing the problems in the original model. The model modified as described above has the same behaviour as the original one simplified by giving the transition `Ensemble_provides_OID` on the page `AssignOID` the guard `[false]`, as we did for the occurrence graph analysis.

Thus all five properties are established (property 5 only for a modified version of the model).

An attempt was made to prove a place invariant capturing the following property: When a user thread is blocked after a send, either there is a packet from that user thread on its way on the network to the receiver side, or the receiver side is working on providing a result, or there is a packet addressed to the blocked user thread on the network. The set of weights defined in order to conduct the proof has a large support. Due to a shortcoming in the Inv Tool, it is unfortunately not possible to verify that this set of weights is a place invariant. However the tool is able to provide a partial check of the proposed place invariant. Partial in the sense that it can verify that some but not all transitions preserve the weight. In this way, the tool may be an aid in a semi-automatic checking of a proposed place invariant, by simply reducing the number of transitions the user has to consider manually.

It is possible to prove all of the listed properties in this section using occurrence graphs — for a fixed configuration. Each property can be verified by a traversal of the full occurrence graph where it is checked that every marking satisfies the considered property. In fact, we did this for property 5 above for the configurations for which the full occurrence graphs were generated.

5 Related Work

A large number of papers describing system modelling and simulation using Petri nets exist. Significantly fewer reports on formal analysis. The explanation is natural: Formal analysis of interesting models require tool support. So far, while having excellent tools for editing and simulation, the Petri nets community has been lacking high-quality tools for formal analysis of anything but very small models. Nevertheless, some papers documenting modelling and formal analysis projects do exist. In this section, we relate our work to a number of these.

In [5] the modelling and simulation of a network management system using CP-nets is described. A large model was built and to some extent analysed using place invariants. However, the authors note that they only made very limited use of formal analysis due to the lack of tool support. They propose to use formal analysis during the model construction stage. Viewing our project in the light of these remarks, we note that formal analysis actually was used during our model construction: The Occ Tool assisted us finding and correcting errors in the process as described in Sect. 4.1. Moreover, we had a set of place invariants in

mind throughout the model construction. These invariants were formally proved with the Inv Tool, thus increasing our confidence in the model.

The occurrence graph analysis of our model could only verify properties for a few initial markings. It is of course highly desirable if the model can be verified independently of some initial marking. Alas, this is prohibited in general because of the nature of the occurrence graph method (refer to the discussion of the dependency of the initial marking problem in Sect. 4.1). Sometimes, the problem may be overcome though. In [7], the modelling and analysis of a hardware chip (an arbiter cascade) is reported. The model is characterised with one single integer system parameter d , the depth of the cascade. The authors verify the model for all possible values of the system parameter in the following way: For $d = 0$ and $d = 1$, the number of reachable markings is small, and occurrence graphs are easily constructed. With them, the desired properties of the model are verified directly. Mathematical induction establishes the proof for all values $d > 1$. Using occurrence graphs in conjunction with induction is very appealing whenever applicable. This strategy solves the two most serious problems with occurrence graph analysis: The state explosion problem is simply eliminated because it is only necessary to generate occurrence graphs when the number of reachable markings is small. The dependency of the initial marking is elegantly overcome with the inductive step. However, this approach relies upon the considered system being regular in the sense that it is characterised by a natural number parameter, and there is a well-defined relation between the behaviour of the system with parameter d and the system with parameter $d + 1$. Unfortunately, our communication protocol does not adhere to these requirements. In fact, we think it will be hard in general to apply this technique to complex communication protocols. It seems that hardware designs are more regular, and thus more adequate for inductive proofs.

In our place invariant analysis, we concentrated on proving place invariants whose existence we presumed. Thus our place invariant analysis was a checking of properties that we expected the CPN model to have. An alternative is to try calculating all place invariants for a given model automatically. With a model of this size, we believe that the computational complexity of this approach is prohibitive. Moreover from some representation of all place invariants it is not necessarily easy to pick out the ones of interest, i.e., the ones that express relevant properties of the model. Our attitude towards place invariant analysis is shared by the authors of [1].

In [13] our project is compared with three others that exploited formal analysis methods. They are described in [9], [15], and [6].

Finally in this section, we make clear the contribution of our project in comparison with the other projects mentioned here. We are convinced that the availability of suitable formal analysis tools is the reason why we obtained quite powerful analysis results compared to the earlier projects. Moreover, we exploit both occurrence graphs and place invariants. We are well aware that we do not use some combination of or interaction between the two methods. However, we believe that the two methods inspire the user to investigate different aspects of

the model. Therefore it is sensible to apply both, if possible. It will typically produce a more comprehensive set of analysis results. It did in this specific example, where the two methods supplemented each other nicely.

6 Conclusion

In this project we have modelled and analysed the protocol for remote object invocation in BETA.

The first stage was to build the CPN model. It had some impact on the protocol. We had a number of meetings with the designer. The discussions gave both him and us a better insight into the behavioural aspects. As a consequence, a number of changes were made to the protocol. More specifically, some superfluous critical regions were removed.

The second stage was to analyse the model using formal methods. For this purpose we used two standard methods, occurrence graphs and place invariants. The results obtained were non-trivial. Careful choice of colour sets and net structure for the model implied a relatively successful use of the occurrence graph method: It was possible to prove important dynamic properties such as absence of dead markings and liveness of specific sets of binding elements for small initial markings. Place invariants were used to prove quite different dynamic properties of the model, e.g., that certain sets of threads remain constant, and that a monitor construction correctly ensures exclusive access to a critical region. We took advantage of two recently developed tools, the Occ Tool and the Inv Tool.

The analysis stage did not influence the protocol, because the design was already sensible before the analysis began. If the formal analysis had revealed, e.g., an unexpected deadlock, then of course this problem would have to be fixed in the design and implementation of the protocol. Thus the formal analysis would have had an impact. The designer of the protocol has the viewpoint that the formal analysis was mostly usable to increase his confidence in the CPN model. Therefore, the key question is if the model is a proper reflection of the design and implementation of the protocol. In this specific project, both the designer and ourselves are confident that the model sensibly captures relevant and important aspects of the protocol. The verification of the model thus does increase our confidence in the protocol. It is theoretically possible though that something left out from the model (e.g., error handling on the receiver side) is exactly what is causing a serious problem.

Advantages and drawbacks of both formal methods were discussed. The main drawbacks inherent to the occurrence graph method, state explosion and dependency of the initial marking, were exhibited. Although the problems are generally recognised in the theory, it is valuable to see their impact on a specific real-world example. Here we saw that only configurations with less than three user threads could be handled. When the number of user threads was increased, occurrence graphs could not be generated in full. We proposed net reductions as a means

to alleviate the state explosion problem and demonstrated decent savings in the sizes of the occurrence graphs.

The model built in this project is complex. It is thus very hard to get certain information about its dynamic behaviour using informal methods such as simulation only. E.g., ruling out the possibility of a dead marking requires formal verification. Unfortunately we have to accept the fact that currently full occurrence graphs can only be generated for small initial markings. Thus, e.g., absence of dead markings is only established for a few markings. Hence on one hand, we saw an example of a model where formal analysis proved really useful when applicable. On the other, formal analysis was somewhat obstructed. There are two categories of sources limiting the applicability of formal analysis. One concerns the analysis methods, the other the tools. The most severe limitations are inherent to the methods, e.g., the state explosion problem. However, the present theoretical work is much more advanced than the present tool support. An important field for future work is development of better tools for formal analysis of CP-nets. With respect to the Occ Tool, generation of larger occurrence graphs will be possible when a version storing markings in a more economic fashion is implemented. The current version does not use memory in an optimal way. Moreover, only ordinary occurrence graphs are presently supported. It will definitely be valuable reconsidering the analysis done in this project when the Occ Tool is matured to support occurrence graphs with equivalences [11]. With respect to the Inv Tool, it needs to be matured to handle more complex expressions.

Acknowledgements We thank Søren Brandt, Søren Christensen, Alexandre Valente Sousa, and Jan Toksvig for help in this project. Thanks to Kurt Jensen, Rikke Drewsen Andersen, Vincent Becuwe, Torben Bisgaard Haagh, Ludovic Joly, Lars Kristensen, and René Wenzel Schmidt for proof-reading various versions of this paper.

This work has been supported by grants from the Danish Research Councils SNF and STVF, and from the Faculty of Science at University of Aarhus.

References

1. J. Billington, G. R. Wheeler, and M. C. Wilbur-Ham. PROTEAN - A High-level Petri Net Tool for the Specification and Verification of Communication Protocols. In K. Jensen and G. Rozenberg, editors, *High-level Petri Nets, Theory and Application*. Springer-Verlag, 1991.
2. S. Brandt. Implementing Shared and Persistent Objects in BETA. Technical report, Computer Science Department, University of Aarhus, 1994.
3. S. Brandt and O. L. Madsen. Object-Oriented Distributed Programming in BETA. In R. Guerraoui, O.M. Nierstrasz, and M. Riveill, editors, *Object-Based Distributed Programming*, Lecture Notes in Computer Science, Kaiserslautern, Germany, 1993. Springer-Verlag.

4. S. Christensen, K. Jensen, and L. Kristensen. *The Design/CPN Occurrence Graph Tool. User's manual version 3.0*. Computer Science Department, University of Aarhus, 1996.
Online: <http://www.daimi.aau.dk/designCPN/>.
5. S. Christensen and L. O. Jepsen. Modelling and Simulation of a Network Management System using Hierarchical Coloured Petri Nets. In E. Mosekilde, editor, *Proceedings of the 1991 European Simulation Multiconference*, Copenhagen, Denmark, 1991. Springer-Verlag.
6. H. J. Genrich, H.-M. Hanisch, and K. Wöhlhaf. Verification of Recipe-based Control Procedures by Means of Predicate/Transition Nets. In R. Valette, editor, *Proceedings of the 15th International Conference on Application and Theory of Petri Nets*, Lecture Notes in Computer Science, Zaragoza, Spain, 1994. Springer Verlag.
7. H. J. Genrich and R. M. Shapiro. Formal Verification of an Arbiter Cascade. In K. Jensen, editor, *Proceedings of the 13th International Conference on Application and Theory of Petri Nets*, Lecture Notes in Computer Science, Sheffield, UK, 1992. Springer Verlag.
8. A. Gibbons. *Algorithmic Graph Theory*. Cambridge University Press, 1985.
9. C. Girault, C. Chatelain, and S. Haddad. Specification and Properties of a Cache Coherence Protocol Model. In K. Jensen and G. Rozenberg, editors, *High-level Petri Nets, Theory and Application*. Springer-Verlag, 1991.
10. S. Haddad. A Reduction Theory for Coloured Nets. In K. Jensen and G. Rozenberg, editors, *High-level Petri Nets, Theory and Application*. Springer-Verlag, 1991.
11. K. Jensen. *Coloured Petri Nets — Basic Concepts, Analysis Methods and Practical Use. Volume 2, Analysis Methods*. Monographs in Theoretical Computer Science. Springer-Verlag, 1994.
12. K. Jensen, S. Christensen, P. Huber, and M. Holla. *Design/CPN. A reference manual*. Computer Science Department, University of Aarhus, 1996.
Online: <http://www.daimi.aau.dk/designCPN/>.
13. J.B. Jørgensen and K.H. Mortensen. Modelling and Analysis of Distributed Program Execution in BETA Using Coloured Petri Nets. Technical report, Computer Science Department, University of Aarhus, 1995.
14. O. L. Madsen, B. Møller-Pedersen, and K. Nygaard. *Object-Oriented Programming in the BETA Programming Language*. Addison Wesley, 1993.
15. W. M. McLendon, Jr. and R. F. Vidale. Analysis of an Ada System Using Coloured Petri nets and Occurrence Graphs. In K. Jensen, editor, *Proceedings of the 13th International Conference on Application and Theory of Petri Nets*, Lecture Notes in Computer Science, Sheffield, UK, 1992. Springer Verlag.
16. A. S. Tanenbaum. *Modern Operating Systems*. Prentice-Hall International, 1992.
17. J. Toksvig. Tool Support for Place Flow Analysis of Hierarchical CP-nets Version 2.0. Technical report, Computer Science Department, University of Aarhus, 1993.
18. J. D. Ullman. *Elements of ML Programming*. Prentice-Hall, 1993.