

Modelling and Analysis of a Flowmeter System

Louise Lorentsen
Department of Computer Science
University of Aarhus
Ny Munkegade, Bldg. 540
DK-8000 Aarhus C, Denmark
E-mail:louisel@daimi.au.dk

Abstract. In this paper the practical use of Coloured Petri Nets and Design/CPN is demonstrated through an industrial cooperation project. The project is based on studies of a concrete industrial product and on methods well-known and currently used in the design of the product in the company. The paper describes the modelling in Design/CPN of the system of the product followed by an analysis of the model in relation to a set of desired properties defined by the company. The properties are formally verified by means of occurrence graphs and the Design/CPN Occurrence Graph tool.

1 Introduction

This paper is based on a project in the Centre for Object Technology (COT), which is a joint project between industry and universities in Denmark. The project is a cooperation project between the Danish industrial production company Danfoss and the CPN group at University of Aarhus. Three persons, representing both sides, have been involved in the project as primary project members supported by the leaders from both cooperating parties. The project group has held a number of meetings and communicated electronically throughout the whole project. However, most work has been based on internal documents kindly made available for the project by the company.

The overall aim of the paper, and the underlying project, is to show the applicability of Coloured Petri Nets (CPNs) [7] and occurrence graphs (OGs) [6] in industrial settings. In order to succeed in this endeavour it is important to relate to concrete industrial products on one hand and to the solutions used so far in industry on the other hand. In the project, which this paper is based on, this means that a specific product of the cooperating industrial company, a flowmeter (Fig. 1), is chosen as a relevant study object, and that present design techniques of this system is related to and combined with the new design and analysis techniques based on CPNs and OGs.



Fig. 1. A flowmeter

Flowmeters are primarily used for measuring the flow of water through pipes. In the concrete flowmeter system of this project different processes are used to measure the flow, e.g., a flow measurement process, a temperature measurement process and a calculation process. These processes cooperates to carry out the overall task - to measure the flow of water in the pipe.

In recent years Danfoss has changed the concept of the flowmeters from a system with a centralised operating system to a distributed and more flexible system with hardware modules which can be combined in many ways. With the new distributed flowmeter system a customer can design his own system and construct it from only the needed modules. However, this makes new problems arise. In the distributed setting based on communication between the processes it is difficult to reason about the individual processes and their influence on each other.

Practical tests in Danfoss have shown that the process communication in the first design version of the flowmeter system contained at least one deadlock, and therefore a new design was needed. As part of the new design a set of desired properties of a flowmeter system have been formulated and a solution has been presented using new design methods, but the correctness of the solution still has to be formally verified. This project should be seen as part of these experiments with new methods in Danfoss. The aim of the underlying project has been to secure the quality of the design process in Danfoss introducing Coloured Petri Nets (CPNs or CP-nets) and the tool Design/CPN [3].

In the following the flowmeter system is described in more detail, a CPN model of the protocol is introduced, a set of desired properties of the flowmeter system are motivated and formulated, different design alternatives for the processes are modelled in Design/CPN and it is analysed by means of occurrence graphs using the Design/CPN OG tool [8] whether the design alternatives fulfil the desired properties of a flowmeter system.

2 The flowmeter system

The flowmeter system consists of one or more modules connected via a Controller Area Network (CAN) [9]. The processes are called CAN Applications (CANAPPs) and are placed in the modules. The placement of the CANAPPs in the modules is flexible, however, which means that the concrete placement of the CANAPPs should not affect the functionality of the total system. All communication in the system consists of message passing between the CANAPPs. To control the communication among the CANAPPs in the modules a superior communication system is needed. The communication system consists of parts in every module called drivers. This means that a flowmeter system consists of a number of modules, each containing a driver and one or more CANAPPs. A system of three modules each containing four CANAPPs is shown in Fig. 2. In the following the protocol defining the transaction procedure and message structure is presented.

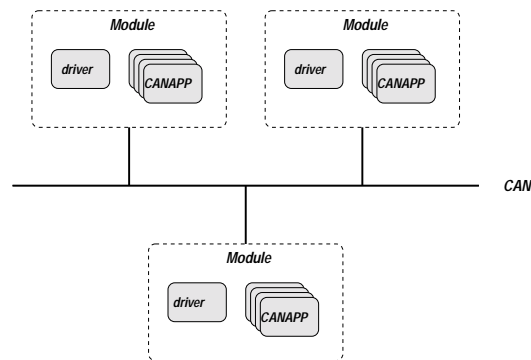


Fig. 2. A flowmeter system

2.1 The protocol

The protocol is based on a three layered architecture. The layers constitute a collapsed form of the OSI seven layer architecture, mapping onto the physical, data link and application layers of the OSI Reference Model [4].

The application layer provides six services: read, write, action, broadcast, command and event as shown in Fig. 3. The services fall into two groups, which will be dealt with separately in the following description: two-way asynchronous communication and one-way asynchronous communication.

| Service | Function |
|-----------|---|
| read | Address an other CANAPP and read the value of an attribute |
| write | Address an other CANAPP and modify the value of an attribute |
| action | Address an other CANAPP and ask it to execute an action |
| broadcast | Cyclically distribution of actual values without any acknowledgements |
| command | Address all other CANAPPs and ask them to execute a system command |
| event | Address all other CANAPPs and report a single event |

Fig. 3. The application layer services

The system considered in the rest of this section is a system of three modules each containing one CANAPP. The system is shown in Fig. 4.

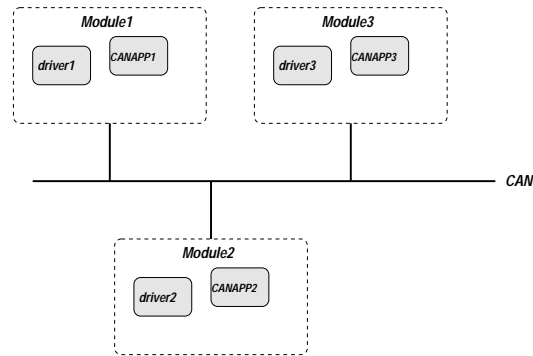


Fig. 4.

2.2 Two-way asynchronous communication

When a CANAPP invokes a read, write or action service, the CANAPP acts like a client. A destination CANAPP, which is going to act like a server in the interaction, is addressed in a request. A response is returned from the server CANAPP.

In case CANAPP 2 invokes a read service to read an attribute of CANAPP 3, the sequence of events shown in Fig. 5 occurs. Write and action services are performed in a similar way.

1. CANAPP 2 delivers the request to the driver in the same module (driver 2) to have the request sent to CANAPP 3.
2. Driver 2 transmits the request to the driver in the module in which CANAPP 3 is located (driver 3) via the CAN bus.
3. An INCAN OK acknowledgement is returned from driver 3 to driver 2 (i.e. driver 2 is ready to accept messages from any other driver).
4. Driver 3 delivers the request to CANAPP 3, which starts processing the request.
5. CANAPP 3 completes the processing, generates a response and delivers it to the driver in the same module (driver 3) to have the response sent to CANAPP 2.
6. Driver 3 sends the response to the driver in the module in which CANAPP 2 is located (driver 2) via the CAN bus.
7. An INCAN OK acknowledgement is returned from driver 2 to driver 3.
8. Driver 2 delivers the response to CANAPP 2.

CANAPP Communication (READ)

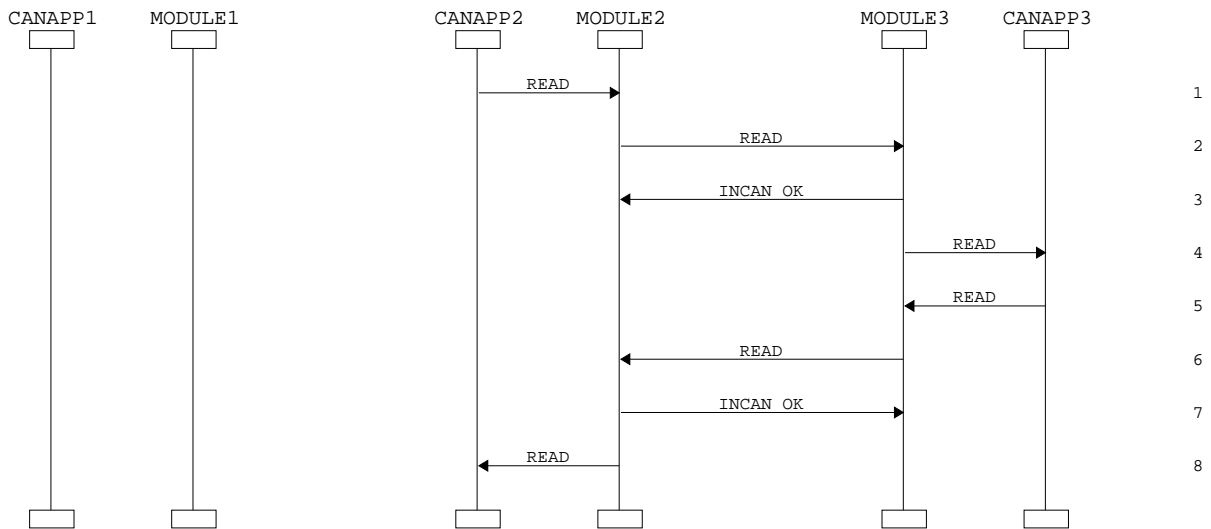


Fig. 5. Two-way asynchronous communication

2.3 One-way asynchronous communication

The one-way asynchronous communication may be used to distribute information to all the other CANAPPs in the flowmeter system - either as broadcast, as shown in Fig. 6, without any guaranty of delivery, or as commands or events, as shown in Fig. 7, with guaranteed delivery.

In case CANAPP 3 invokes a broadcast service the sequence of events shown in Fig. 6 occurs.

CANAPP Communication (BROADCAST)

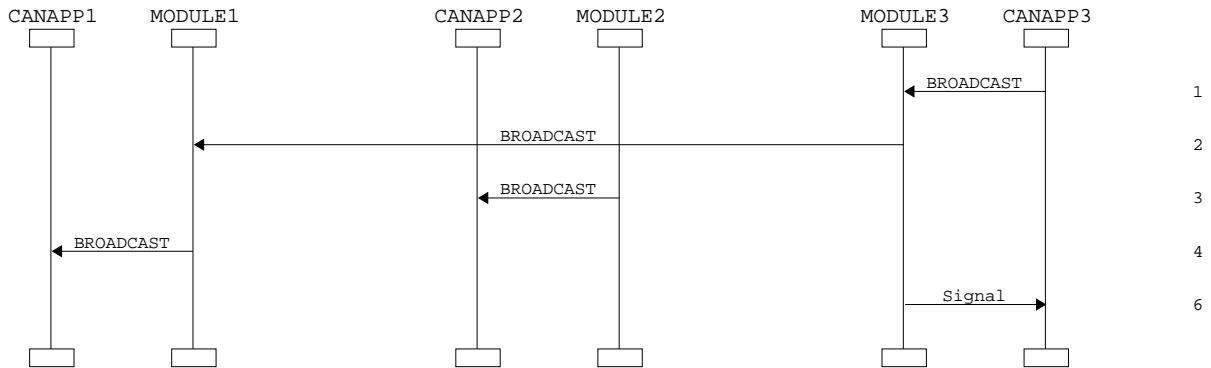


Fig. 6. One-way asynchronous communication - broadcast

1. CANAPP 3 delivers the broadcast message to the driver on the same module (driver 3) to have the message broadcasted to all other CANAPPs in the flowmeter system.
2. Driver 3 sends the message to all other drivers in the system via the CAN bus. No acknowledgements are returned.

3. Driver 2 delivers the broadcast message to its local CANAPPs (CANAPP 2).
4. Driver 1 delivers the broadcast message to its local CANAPPs (CANAPP 1).
5. CANAPP 3 is informed about completion of the transmission. This is not an acknowledgement from the individual receiver CANAPPs but a final synchronisation signal from the transmitter to the sending CANAPP. Notice that the signal can occur before the message has been received by the CANAPPs in the system.

In case CANAPP 3 invokes an event service, the sequence of events shown in Fig. 7 occurs. The command service is performed in exactly the same way as the event service. The delivery of the message is guaranteed by use of the INCAN OK acknowledgement mechanism.

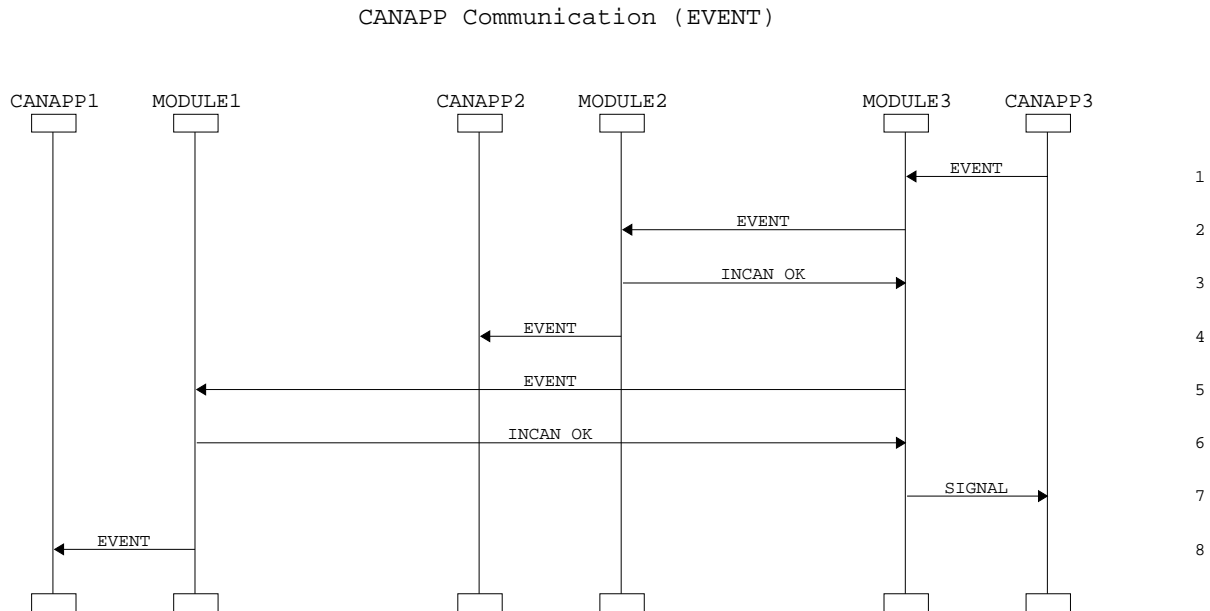


Fig. 7. One-way asynchronous communication - event

1. CANAPP 3 delivers the message to the driver in the same module (driver 3) to have the message sent to all other CANAPPs in the flowmeter system.
2. The requests are sequentially distributed to all other drivers in the system with an acknowledgement from each individual receiver.
3. When an acknowledgement is received, the request is sent to the next receiver.
4. CANAPP 3 is informed about completion of the transmission. This is not an acknowledgement from the individual receiver CANAPPs but a final synchronisation signal from the transmitter to the sending CANAPP. Notice that the signal occurs before the message has been received by CANAPP 1.

3 CPN model of the flowmeter system

The constructed model of the flowmeter system consists of 11 pages. An overview is given via the hierarchy page in Fig. 8.

The system modelled and presented in this section is a system consisting of two modules, each containing two CANAPPs. This system is shown in Fig. 9.

The topmost page `Flowmeter_System` of the CPN model is shown in Fig. 10. This page contains a top level description of a flowmeter system consisting of a number of CANAPPs communicating by means of the three layered protocol presented in Sect. 2. This is reflected by the four parts of the page: A part modelling

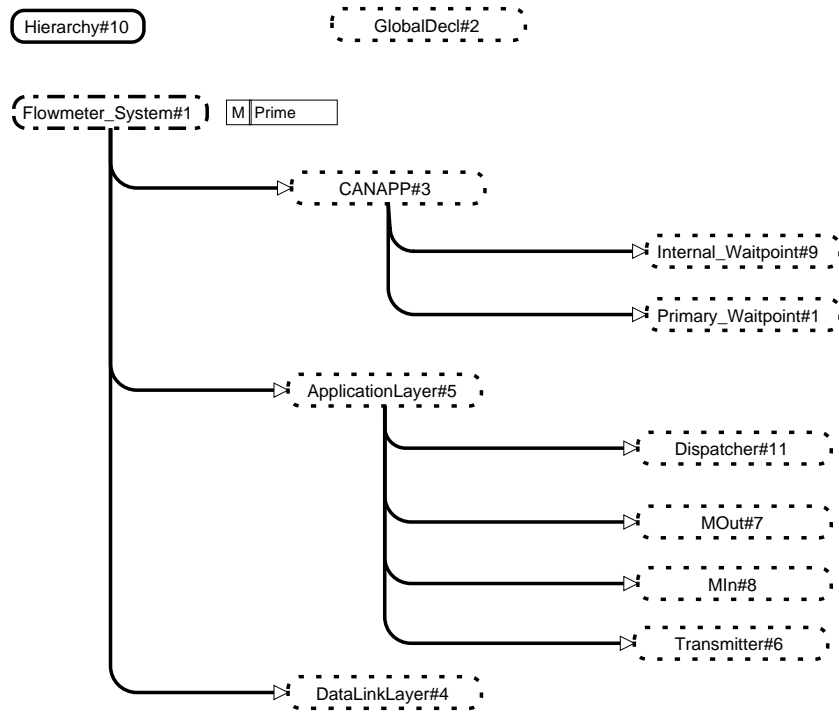


Fig. 8. The hierarchy page

the CANAPPs and three parts modelling the three layers of the protocol. CANAPP, ApplicationLayer and DataLinkLayer are all substitution transitions which means that the detailed behaviours are shown on the subpages with the corresponding names. The four CANAPPs in the system are modelled by use of colour sets. Therefore only one page modelling the CANAPPs is needed in the CPN model. The page CANAPP is further split into two parts modelling two different design alternatives for the CANAPPs. The presentation here concentrates on the parts of the model relevant for experimenting with the two different high level design alternatives of the CANAPPs. Therefore only the pages ApplicationLayer and CANAPP are presented in detail. First the page ApplicationLayer is presented as a basis for understanding the following CANAPP presentation and analysis. Then the desired properties of a flowmeter system are motivated and presented, the page CANAPP containing the design alternatives is presented and it is analysed whether the system modelled fulfils the properties formulated.

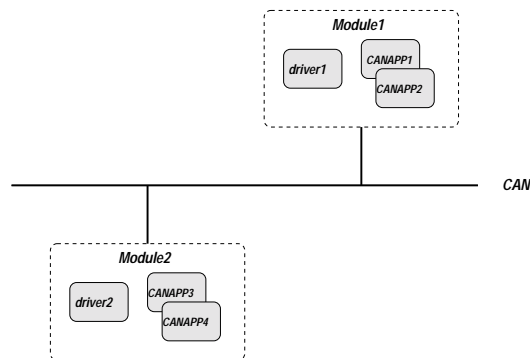


Fig. 9. The flowmeter system modelled

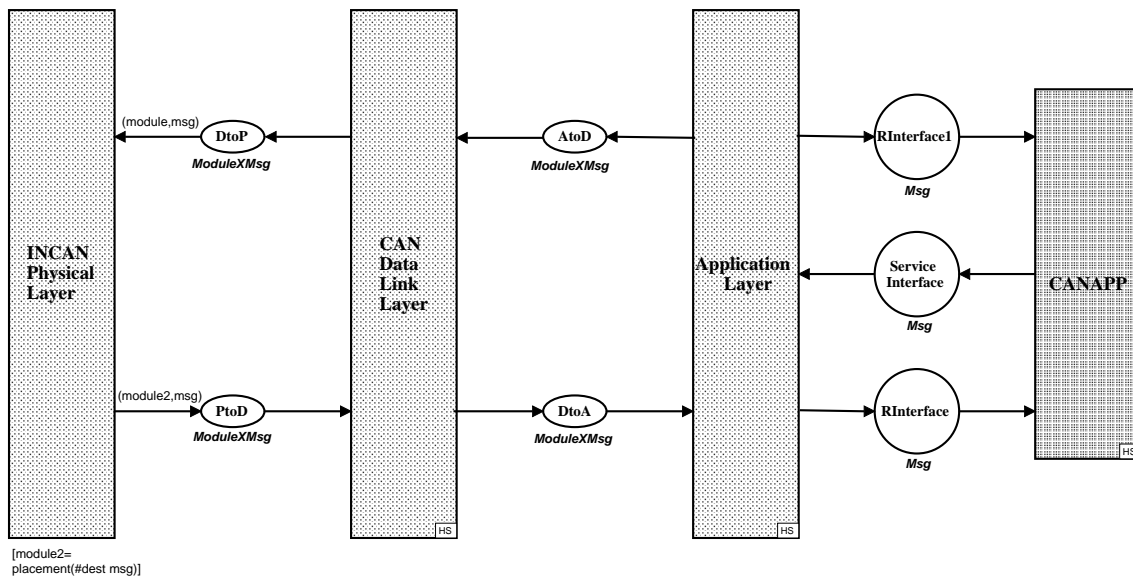


Fig. 10. Page Flowmeter_System

3.1 CPN model of the protocol

The page ApplicationLayer is shown in Fig. 11. This page contains the model of the driver, which is the part of the communication system in each module. The part of the driver responsible for receiving and delivering the messages from and to the CANAPPs in the module is modelled in the page Dispatcher. Messages are received from the CANAPPs via the place Service Interface and delivered to the CANAPPs via the places RInterface and RInterface1. The dispatcher is responsible for the internal communication in the module. There is no need to send messages from a CANAPP in a module to another CANAPP in the same module via the CAN bus. These intra module messages is delivered to the receiving CANAPP by the dispatcher. The inter module messages are handled by the rest of the driver. The dispatcher and the rest of the driver communicate via the 7 places to the left of the substitution transition Dispatcher in Fig. 11. We will return to these places later in this section.

To avoid deadlock between two modules the driver is implemented to support some concurrency. All incoming requests are received and handled to the CANAPPs for processing even if the driver is waiting for a response to an outstanding request from a CANAPP in the module. Specific buffers are allocated to handle incoming and outgoing broadcasts. This means that independent on other activities the modules are always able to transmit and receive broadcast messages. To obtain the wanted concurrency the part of the driver responsible for sending and receiving messages and to wait for responses is implemented as four state machines, two state machines for outgoing requests and the corresponding incoming responses and two state machines for incoming requests and the corresponding outgoing responses. This is reflected in the model by the four parts: BOut, MOut, MIn and BIn. MOut and MIn are substitution transitions, which means that the detailed behaviours are shown on the subpages with the corresponding names. These subpages will be presented later in this section.

Messages received from the data link layer on place DtoA are directed through the IN-buffers MessageIN and BroadcastIN, depending of the type of message. Broadcast messages are directed through BroadcastIN, the rest is directed through MessageIN. If the message is a read, write or action message the corresponding response is directed through the same IN-buffer in the opposite direction. Messages from the CANAPPs are directed through the OUT-buffers MessageOUT and BroadcastOUT to the data link layer on place AtoD. If the message is a read, write or action the corresponding response flows through the same OUT-buffer. In the rest of this section the modelling of the four statemachines BroadcastOUT, BroadcastIN, MessageOUT, and MessageIN is presented.

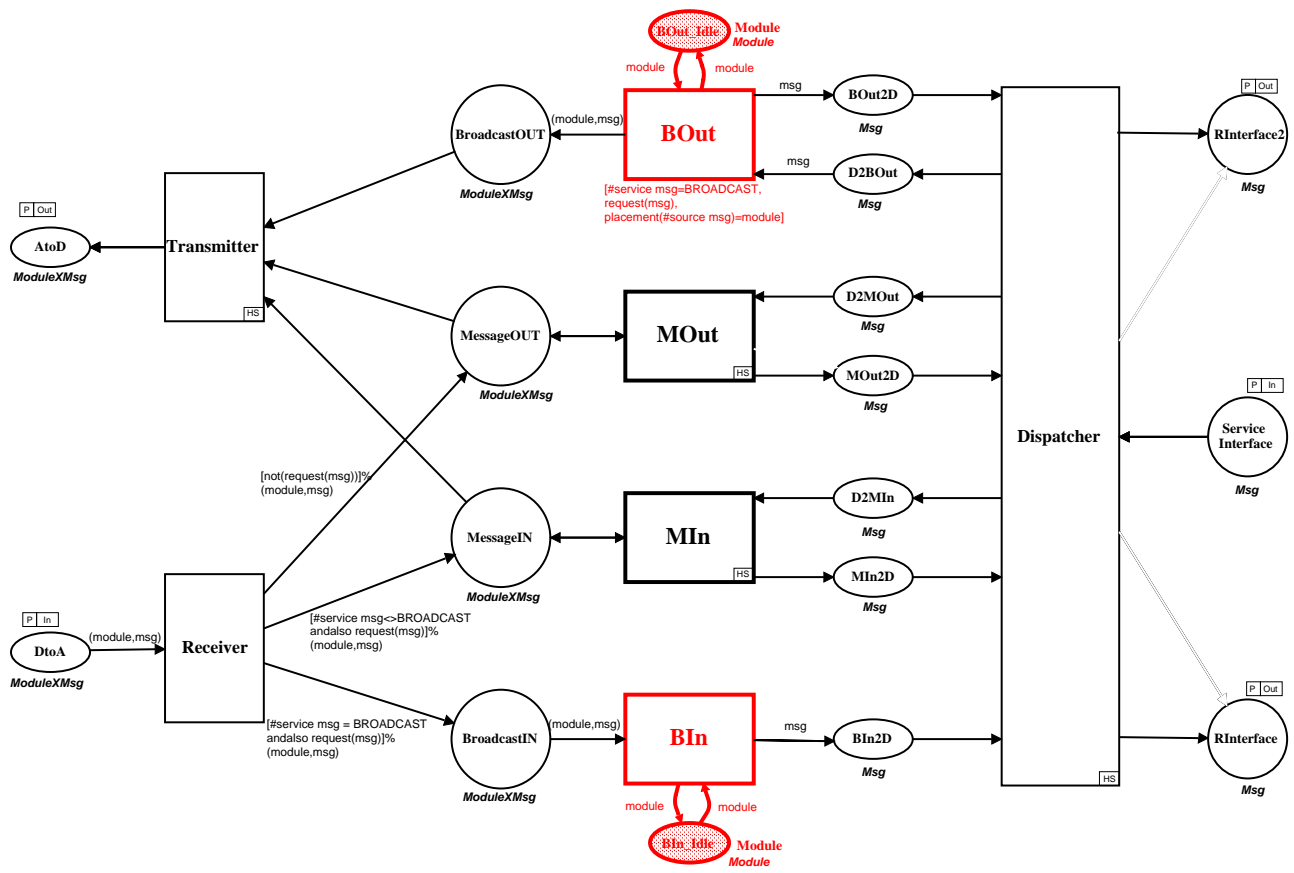


Fig. 11. Page ApplicationLayer

The BroadcastOUT statemachine The BroadcastOUT statemachine for sending broadcast messages can be seen in page ApplicationLayer which is shown in Fig. 11. The statemachine is very simple and consists only of the place *BOut_Idle* and the transition *BOut*. Initially there is a token corresponding to each module in the system on place *BOut_Idle* modelling that all drivers in the system are idle and ready to transmit broadcast messages. When a CANAPP sends a broadcast message a token modelling the message is by the dispatcher put on place *D2BOut*. This message is to be sent on the CAN bus to all other drivers in the system. The dispatcher is responsible for sending the broadcast message to all other CANAPPs in the same module. When there is a token corresponding to a broadcast message on place *D2BOut* the transition *BOut* can occur. The message is put on place *BroadcastOUT* to be put on the CAN bus and a token modelling a signal is put on place *BOut2D*. This signal is by the dispatcher delivered to the sending CANAPP. The tokens on place *BOut_Idle* modelling the modules in the system ensure that each module only transmits one broadcast message at a time.

The BroadcastIN statemachine Also the BroadcastIn statemachine for receiving broadcast messages can be seen in page ApplicationLayer which is shown in Fig. 11. The statemachine consists of the place *BIn_Idle* and the transition *BIn*. Initially there is a token corresponding to each module in the system on place *BIn_Idle* modelling that all drivers in the system are idle and ready to accept broadcast messages from the data link layer to be delivered to the CANAPPs in the module.

When a broadcast message is received from the data link layer, the message is put on the place *BroadcastIN*. Now the transition *BIn* can occur and the message is put on place *BIn2D*. The dispatcher now distributes the message to all CANAPPs in the module. The tokens on place *BIn_Idle* modelling the modules in the system ensure that each module only receives one broadcast message at a time.

- The message is removed from place `D2MOut`
- A token consisting of a pair where the first component is the module and the second component is a list of the messages which are to be sent is put on the place `Send`. The list is constructed by use of the function `SendSet`. Depending on the type of the message being sent different actions occur. If the message is a read, write or action message, the list has the message sent from the CANAPP as the only element. If the message instead is an event or command message the elements in the list are a message to each of the other CANAPPs in the system which is placed on another module as the sending CANAPP. All of these messages are to be sent one by one and an acknowledgement is awaited between each individual transmission.
- A token modelling the message is put on place `Service`. The function of this place is to “remember” the type of message being sent. In this way it can later be decided whether a response should be awaited (in case of read, write or action messages) or a signal should be sent to the sending CANAPP (in case of event and command messages).

If the list of messages on place `Send` is non-empty the transition `ReqTrans` can occur. The effect is

- The pair `(module,msg:msglist)` is removed from the place `Send`.
- The first message in the list is put on place `MessageOUT` to be put on the CAN bus.
- A token corresponding to the pair of the module and the list of messages (including the message just sent) is put on place `Wait` modelling that an acknowledgement is awaited between each transmission.

The acknowledgement will be directed through the same OUT-buffer. When an acknowledgement to the message just sent arrives on the place `MessageOUT` the transition `Collect` can occur. The effect is

- The pair `(module,msg:msglist)` is removed from the place `Wait`.
- The pair `(module,msglist)` is put on place `Send` to have the rest of the messages in the list sent.
- If the message is a read, write or action message a response which will be directed through the same OUT-buffer has to be awaited and a the pair `(module,msg)` is put on place `WaitResp`.

When the transmission is finished (i.e, the list of messages on place `Send` is empty), the transition `EndTrans` can occur. The effect is

- A token corresponding to the module is put on place `TrxIdle` modelling that the driver is now idle and ready to send a new message.
- The pair of the module and the message on place `Service` is removed. If the message is an event or command message a token modelling a signal is put on place `MOut2D` to be delivered to the sending CANAPP by the dispatcher. The signal indicates that the transmission is finished, not that the individual messages actually have been received by the CANAPPs.

When a response arrives at the place `MessageOUT` transition `RecResp` can occur. The effect is

- The pair `(module,msg)` is removed from place `WaitResp` modelling that no response is awaited anymore.
- The token corresponding to the response message is removed from `MessageOUT`.
- The response is put on place `MOut2D` to be delivered to the receiving CANAPP by the dispatcher.

The MessageIN statemachine The statemachine `MessageIN` is modelled by the substitution transition `MIn`. The page `MIn` is shown in Fig. 13.

Initially the driver in each module is idle and ready to accept request messages from the data link layer on place `MessageIN` to be delivered to the CANAPPs in the module. This is modelled by the place `ReclIdle` which contains a token modelling each of the modules in the system.

If the driver in the module in which the receiving CANAPP is placed is idle and a request message arrives on place `MessageIN` the transition `Collect` can occur. The effect of the occurrence is

- The token corresponding to the module in which the receiving CANAPP is placed is removed from the place `ReclIdle` modelling that the driver in the module is no longer idle but in the process of receiving a message.
- The token modelling the message is removed from the place `MessageIN`

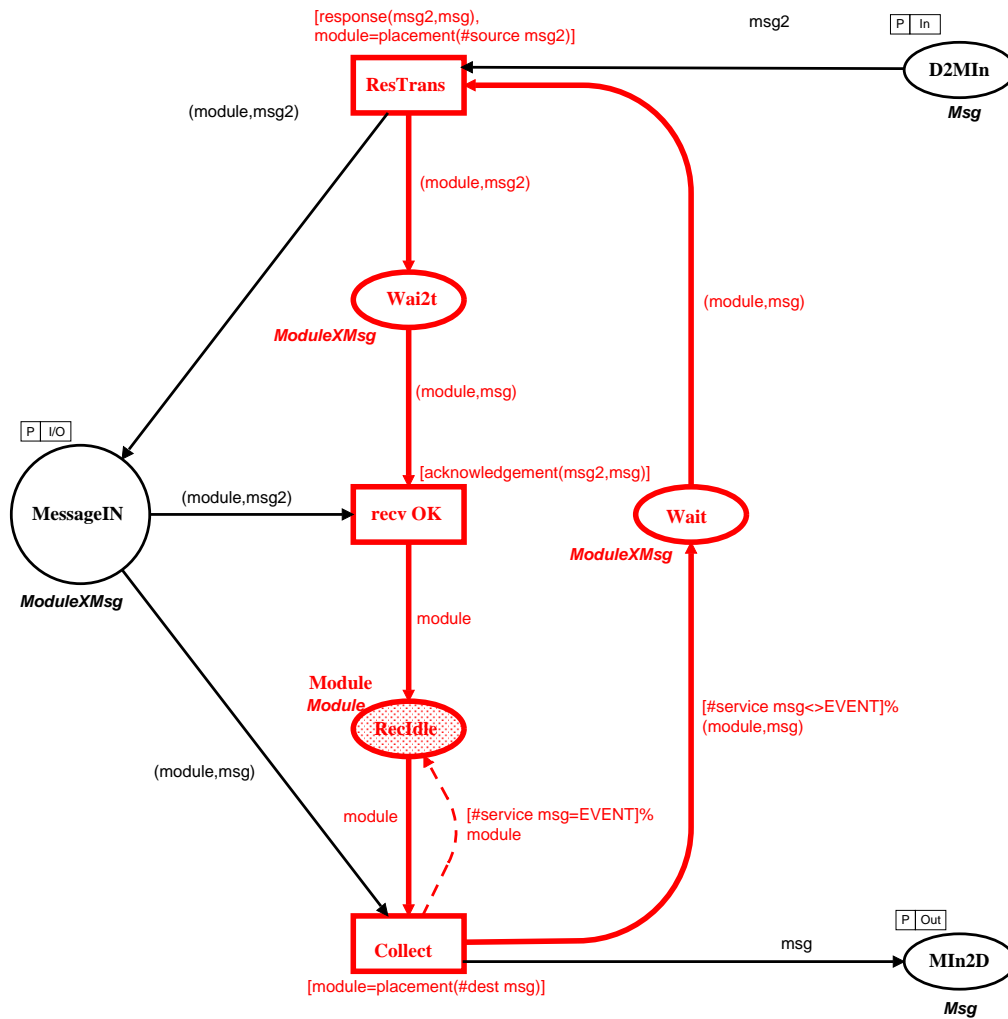


Fig. 13. Page MIn

- A token modelling the message is put on the place MIn2D to be delivered to the receiving CANAPP by the dispatcher.
- If the message is an event or command message no response is to be returned and a token corresponding to the driver is put back on place Wait modelling that the driver is again idle and ready to receive non-broadcast messages from the data link layer. If instead the message is a read, write or action message the driver has to wait for an response message to deliver to the sending CANAPP. In this case a token is put on the place Wait.

When the receiving CANAPP has generated the response a token corresponding to the message is by the dispatcher put on the place D2MIn. The transition ResTrans can now occur. The effect is

- The message is removed from the place D2MIn.
- The pair of the module and the request message is removed from the place Wait modelling that the response from the CANAPP is no longer awaited.
- The message is put on place MessageIN to be put on the CAN bus.
- A token corresponding to the pair of the module and the message is put on place WaitAck modelling that an acknowledgement is awaited.

The acknowledgement will be directed through the same IN-buffer. When an acknowledgement to the response message sent arrives the transition recv OK can occur. The effect is

- The pair (module,msg) is removed from the place WaitAck modelling that an acknowledgement is no longer awaited.
- A token corresponding to the module is put on place RecIdle modelling that the module is now idle and ready to receive messages from the data link layer to be delivered to the CANAPPs in the module.

The CPN model for the protocol defining the transaction procedure for message passing among the CANAPPs in a flowmeter system has now been introduced. In the following sections the CPN model of two different design alternatives for the CANAPPs are presented and analysed.

3.2 CPN model of the CANAPPs

When two CANAPPs communicate they do it in an asynchronous way: when receiving a read, write or action message a response message is sent back to the sender of the request. As we will see in this section the CANAPPs need to be carefully designed to avoid deadlocks and other problems.

When designing the flowmeter system Danfoss used the OCTUPUS method [1], where two basic design alternatives are presented: internal wait point and primary wait point.

As it is most important in collaboration projects to relate to existing knowledge and concepts in the field examined, this section will combine these two well known design alternatives with the CPN modelling of the system. Therefore a design of the CANAPPs based on each of the wait points above is modelled.

After the modelling each of the two models has to be analysed concerning the presence of the desired properties of a flowmeter system. These are, as stated by the producer:

- No deadlocks in the flowmeter system
- No change in the variables of a CANAPP when it is in the process of doing asynchronous communication

Internal wait point When a request is sent the CANAPP is blocked. If the CANAPP cooperates with other CANAPPs on the same module all CANAPPs in the task are blocked. The CANAPPs in the task are not released until the response message (or signal) has been received.

A CPN model of the CANAPPs based on the internal wait point approach is modelled on page InternalWaitpoint which shown in Fig. 14. The system modelled is the system in Fig. 9 of two modules each containing two CANAPPs. CANAPP 1 and 2 cooperate in the same task in module 1 and CANAPP 3 and 4 cooperate in the same task in module 2.

Initially all CANAPPs in the system are idle and ready to generate request messages to or receive messages from the other CANAPPs in the system. This is modelled by the place Idle which contains a token for each of the CANAPPs in the system. The place Attr models the attributes of the CANAPPs. Each of the CANAPPs modelled has an attribute (a local variable) which is modelled as an integer. When a CANAPP receives a write request or receives a response to a previously sent read request the CANAPP updates its attribute according to the value sent in the message.

If the CANAPP is idle the transition Request can occur. The effect is

- The tokens modelling the sending CANAPP and the other CANAPP in the task are removed from the place Idle modelling that both of the CANAPPs in the task are blocked until a response message or a signal returns.
- A token modelling the message is put on the places Service Interface to be delivered to the receiving CANAPP.
- A pair of the sending CANAPP and the message sent is put on place Wait modelling that the message is sent and the sending CANAPP has to wait for a response message or a signal.
- The places Packet and Attr1 are inspected to generate a packet with the value of the attribute of the sending CANAPP.

When a response message arrives on place RInterface1 the transition Confirm can occur. The effect is

- The message is removed from place RInterface1.

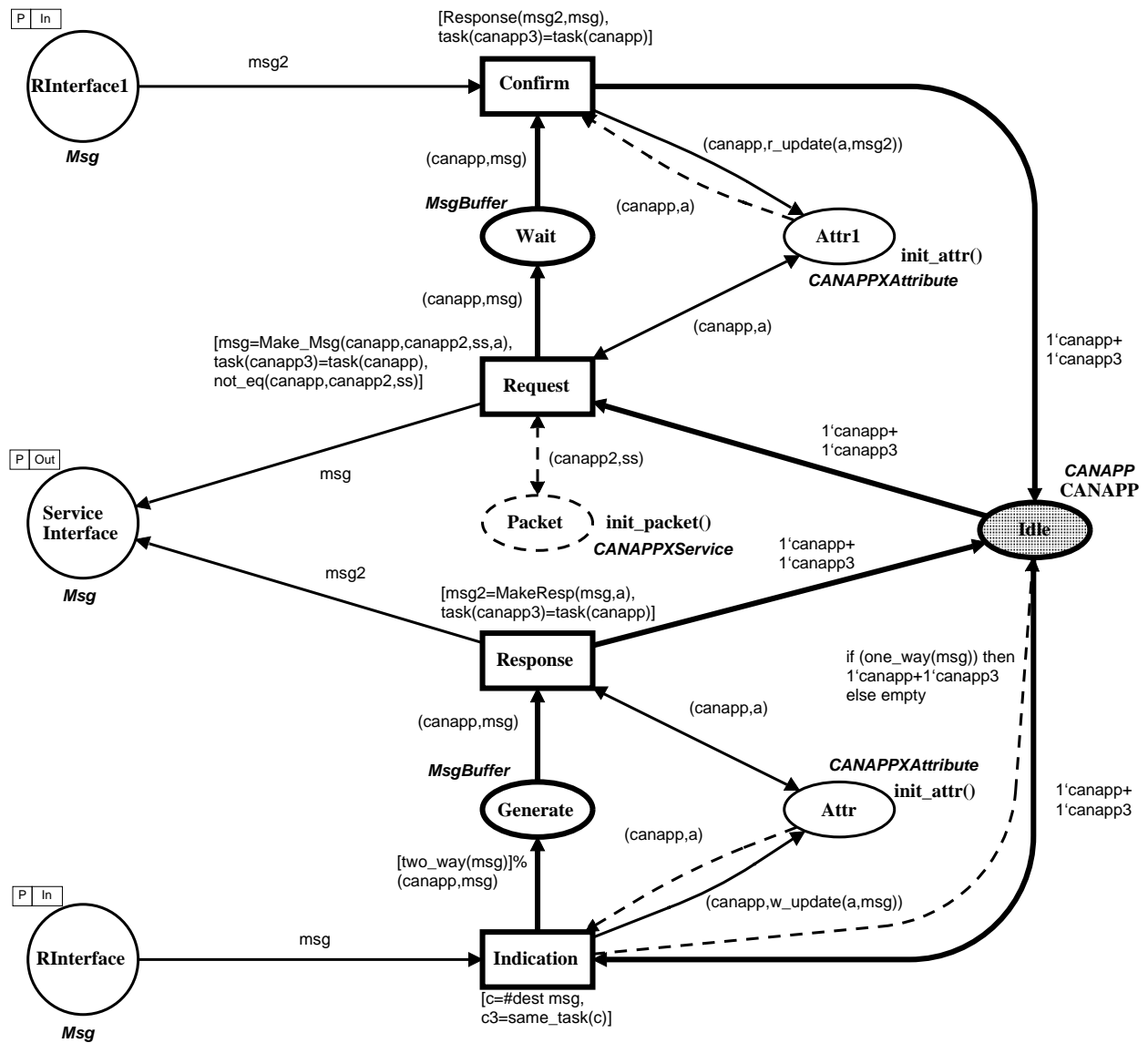


Fig. 14. Page InternalWaitpoint

- The pair modelling the sending CANAPP and the message sent is removed from place Wait modelling that the CANAPP no longer waits for a response.
- The token modelling the attribute of the CANAPP is removed from the place Attr. If the response message is a read message the attribute is updated to the value of the message. Otherwise a token with the original value of the attribute is put back on the place Attr.
- Two tokens corresponding to the two CANAPPs in the task are put back on place Idle modelling that both CANAPPs in the task are now idle and ready to either generate a message to another CANAPP in the system or to receive a message from another CANAPP.

When a message is sent to a CANAPP a token modelling the message is put on place RInterface. If the both of the CANAPPs in the task of the receiving CANAPP are idle the transition Indication can occur. The effect is

- The message is removed from place RInterface1.

- The tokens modelling the two CANAPPs in the task are removed from the place `Idle` modelling that they are no longer `Idle` but in the process of receiving a message.
- The token modelling the attribute of the receiving CANAPP is removed from the place `Attr`. If the message is a write message the attribute is updated to the value of the message. Otherwise a token with the original value of the attribute is put back on the place.
- If the message is a one-way message (i.e., a broadcast, event or command message) no response has to be generated and a two tokens modelling the two CANAPPs in the task are put on place `Idle`. If the message instead is a read, write or action message a response has to be generated and a pair of the CANAPP and the message is put on place `Generate`.

Now transition `Response` can occur. The effect is

- The pair is removed from the place `Generate` modelling that the response has been generated.
- Two tokens modelling the two CANAPPs in the task are put on place `Idle` modelling that the CANAPPs in the task are now idle and ready to either generate a message to another CANAPP in the system or to receive a message from another CANAPP.

Primary wait point In the primary wait point approach the return message is treated as an event and the sending CANAPP or any other CANAPPs in the task are not blocked. This means that a CANAPP can receive a request even if it is temporarily waiting for a response to a previously sent request.

A CPN model of the CANAPPs based on the primary wait point approach is modelled on page `Primary-Waitpoint` which is shown in Fig. 15. The system modelled is the system in Fig. 9 of two modules each with two CANAPPs. CANAPP 1 and 2 cooperate in the same task in module 1 and CANAPP 3 and 4 cooperate in the same task in module 2.

The CPN model in Fig. 15 differs from the CPN model in Fig. 14 by having two idle places (`Idle` and `Idle2`) instead of just one. This means that a CANAPP can receive and send messages independently. Furthermore are the other CANAPP in the not blocked when a CANAPP is sending or receiving messages.

Properties of the model based on each of the two designs The model has been analysed with both of the two designs of the CANAPPs. The model has been analysed by means of the Design/CPN OG tool. The flowmeter system analysed is the system consisting of four CANAPPs - two in each module. Furthermore the total number of messages sent is limited in order to make the occurrence graphs finite. The analysis shows that even in this simple setting undesired properties of the two models can be found. Other configurations have been analysed. The results can be found in Sect. 4.

The internal wait point approach presents the problem, that it is not possible to access any CANAPP in a task, if one CANAPP is waiting for an asynchronous return message. This may result in deadlock situations as the following analysis shows.

The analysis reveals that the occurrence graph of the model, when the CANAPP design in Fig. 14 based on the internal wait point approach is used, has several dead markings. Not surprisingly the analysis shows that a deadlock occurs if two CANAPPs on different modules simultaneously send a request to each other. They both wait for a response but both are blocked and are not able to receive and process the request.

We will concentrate on another and more problematic kind of deadlock which is found in the analysis of the model based on the internal wait point approach. Figure 16 visualises a path to a node in the occurrence graph representing a dead marking of the CP-net.

CANAPP 2 sends a request to CANAPP 4. As the model is based on the internal wait point approach and CANAPP 1 and 2 are in the same task, both CANAPP 1 and 2 block until a response is received. CANAPP 3 sends a request to CANAPP 1. Of the same reason as before both CANAPP 3 and 4 block until a response is received. Now neither CANAPP 4 nor CANAPP 1 can receive the requests and the system is deadlocked.

All deadlocks are of course unwanted, but especially the kind of deadlock visualised in Fig. 16 is problematic in a flowmeter system because it is very hard to identify. The reason is that the CANAPPs are freely movable between the modules in the system but the concrete placement of the CANAPPs should not affect the functionality of the system. If CANAPP 4 in the example above instead is placed in a separate module (the flowmeter system consists of three modules instead of two) no deadlock would have occurred.

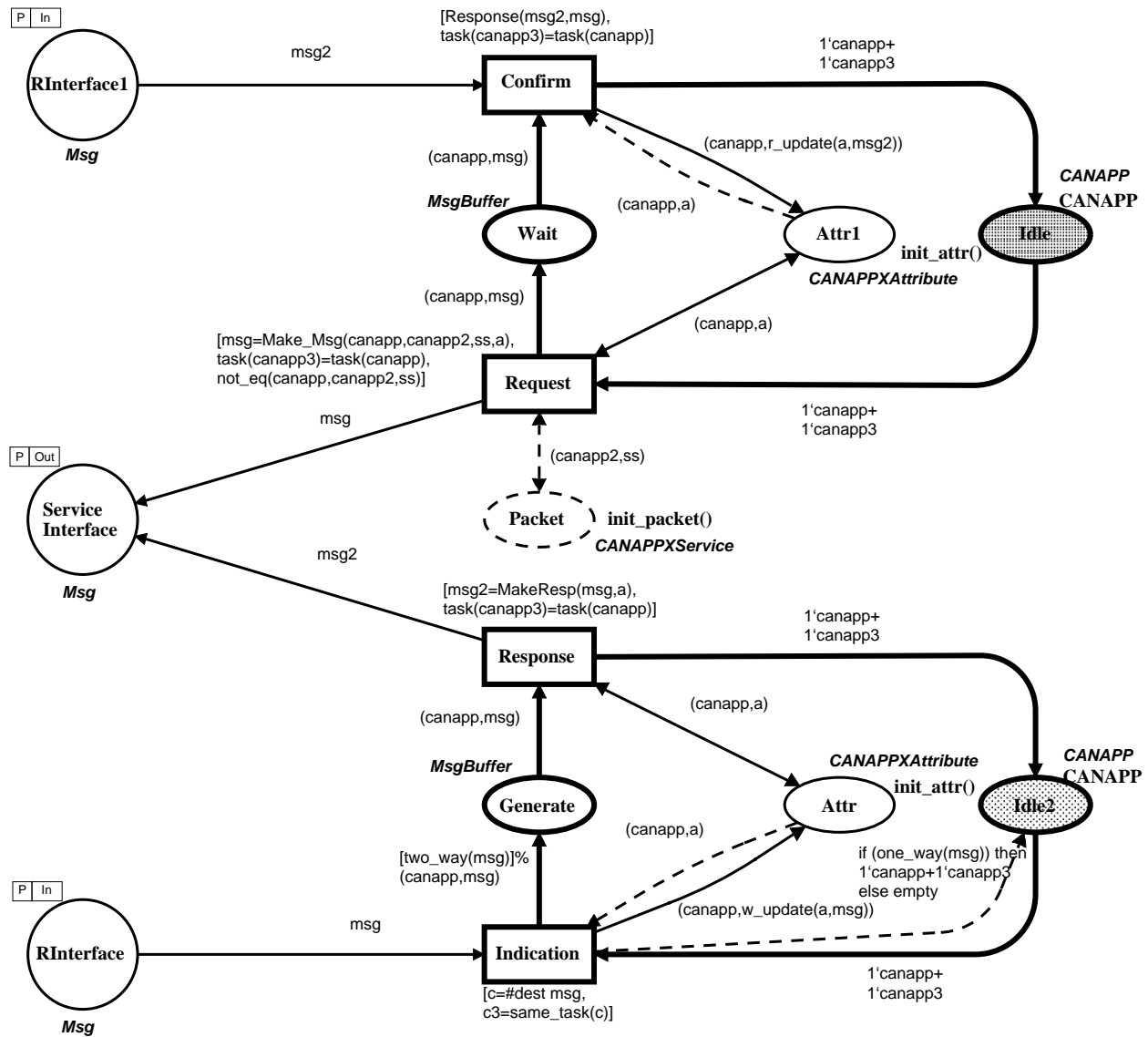


Fig. 15. Page PrimaryWaitpoint

One of the desired properties of a flowmeter system is, as listed in section 3.2, that it should not have any deadlocks - the analysis shows that the design of the CANAPPs based on the internal wait point approach does not fulfil this property.

The occurrence graph of the model, when the CANAPP design in Fig. 15 based on the primary wait point approach is used, only has dead markings due to the limitations of the number of messages sent. This means that no deadlock in the communication occurs - the design of the CANAPPs based on the primary wait point approach therefore fulfils the first desired property of the flowmeter system.

The second desired property as stated in section 3.2 is that the internal state of a CANAPP should not be corrupted while the CANAPP is involved in asynchronous communication, i.e., the attributes of a CANAPP

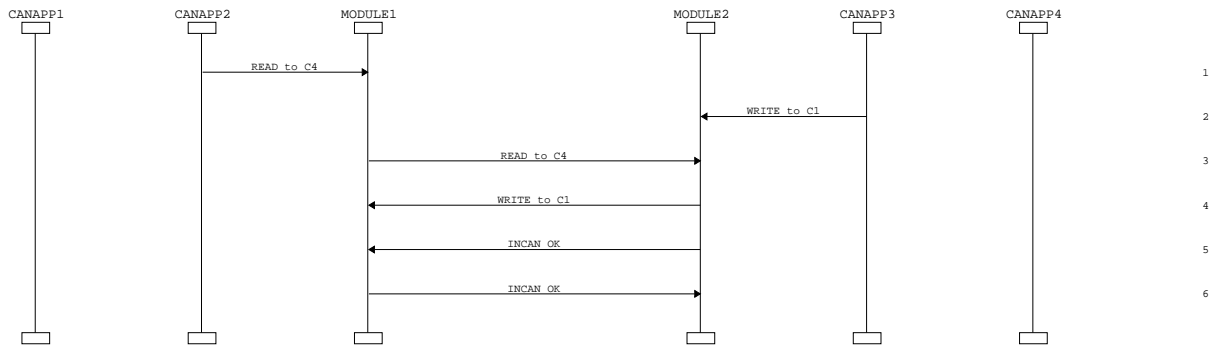


Fig. 16. Visualisation of a path to a dead marking

should not be changed while the CANAPP is waiting for a response to a previously sent request. In Fig. 17 the negation of the desired property is formulated in CPN ML [3]. All reachable markings are investigated via the predefined function `SearchAllNodes` and it is by a predicate function checked whether a CANAPP is waiting for a response in the marking and the marking has a successor marking in which the CANAPP is still waiting and the attributes have changed. If it is the case then the design does not fulfil the second desired property of the flowmeter system.

```

(*****)
(* predicate testnext: *)
(* A CANAPP is waiting in the marking and there is a successor marking *)
(* in which the CANAPP is still waiting but the attributes have changed *)
(*****)

fun testnext (n,l::li) =
  ((Mark.PrimaryWaitpoint'Wait 1 n)==(Mark.PrimaryWaitpoint'Wait 1 l))
  andalso
  ((Mark.PrimaryWaitpoint'Attr 1 n)<><>(Mark.PrimaryWaitpoint'Attr 1 l))
  orelse (testnext (n,li))
| testnext (n,[]) = false;

```

Fig. 17. The negation of the second desired property formulated in CPN ML

A search in the occurrence graph of the model in Fig. 14 based on the internal wait point approach returns the empty list - the predicate function returns false for all reachable markings, which means that the negation of the property holds in all markings; the attributes of a CANAPP are not corrupted while the CANAPP is waiting for a response to a sent request.

A search in the occurrence graph of the model in Fig. 15 based on the primary wait point approach returns a non-empty list. This means that a corruption of the attributes of a CANAPP can occur while the CANAPP is doing asynchronous communication. The design of the CANAPPs based on the primary wait point approach therefore does not fulfil the second desired property of the flowmeter system. Fig. 18 shows a sequence of events which leads to a node in the occurrence graph which represents a marking in which the attribute of a CANAPP has been corrupted.

CANAPP 2 sends a request to read an attribute of CANAPP 3. As the model is based on the primary wait point approach CANAPP 2 is now waiting for a response but is not blocked. CANAPP 3 sends a request

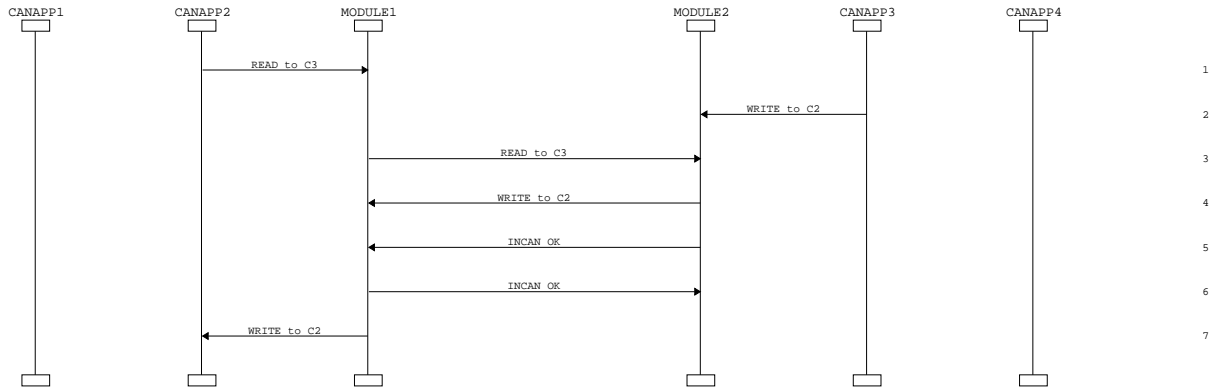


Fig. 18. Visualisation of a path to a marking where the attribute of a CANAPP has been corrupted

to write an attribute of CANAPP 2. CANAPP 2 receives the message from CANAPP 3 and changes its attribute. Now the attribute of CANAPP 2 has been corrupted while CANAPP 2 is waiting for a response to a previously sent message.

To summarise, the design based on the internal wait point approach violates the first property (no deadlocks) and fulfils the second (no corruption of attributes). The design based on the primary wait point approach on the other hand fulfils the first property but violates the second. Therefore, none of the two approaches above are suitable for the final design of the flowmeter system. In their construction of the final design the company has in fact chosen to work with a combination of the two described approaches, combining, as stated by the company, the best parts from both approaches.

4 Configurations

In Sect. 3.2 a small configuration of the flowmeter system was analysed. Even in this small setting deadlocks and corruption of attribute values could be detected thereby pointing out the problems of the, for the engineers in Danfoss, well known designs based on internal wait point and primary wait point. However the occurrence graphs get very large as the configurations analysed get bigger. This is as expected due to the asynchronous communication among the CANAPPs, which means that all possible interleavings are represented in the occurrence graph.

Below an overview of the size of the occurrence graph for different small configurations of the system is given to illustrate how the size of the occurrence graphs grows very fast as asynchronous communication over modules and more CANAPPs are introduced in the model. The results from Sect. 3.2 applies to all the configurations analysed.

Consider the flowmeter system with the configuration shown in Fig. 9, which is a system of two modules each containing two CANAPPs. The occurrence graph of this system where one message can be sent has 18,397 nodes and 48,930 arcs. If two messages can be sent, but with the restriction that only one CANAPP in each task is the sender of the messages, the occurrence graph has 45,939 nodes and 159,688 arcs. This means that the system with two messages (but the restriction that only one CANAPP in a task can be the sender of a message) instead of one message the occurrence graph grows by approximately a factor 3. The occurrence graph for the system with two messages without restrictions on the sending CANAPPs has more than 500,000 nodes and 1,500,000 arcs.

In this concrete project the use of CPNs in the design of the CANAPPs is well illustrated through small configurations. This means that the large occurrence graphs is not actually a problem. However, if a final design of a product is going to be based on a CPN model larger configurations have to be analysed to increase confidence in the system designed.

A subject for further work is therefore to use different techniques to reduce the size of the occurrence graph thereby making it possible to analyse even larger configurations. An interesting approach could be to consider the CANAPPs as symmetric in a task [6] and use stubborn sets [10] to reduce the size of the part of the occurrence graph related to the asynchronous communication among the CANAPPs.

5 Conclusions

The distributed flowmeter system studied in this paper is today a product sold by Danfoss.

The new design of a flowmeter system as a distributed system, relying on asynchronous communication among the processes in the system, raises a need for new ways of reasoning about the system and of validating the presence of the desired properties in a given flowmeter system. The project described in this paper demonstrates through concrete modelling and analysis of the constructed models by means of the Design/CPN OG tool that Coloured Petri Nets and occurrence graphs may indeed be relevant new techniques to be used in industrial settings, not only related to this product but also related to the design process of future products.

The deadlock found by Danfoss in the design phase of the product was found in a practical test of the product. The analysis performed in this project is able to detect the same deadlock and also show how this kind of deadlocks is unavoidable if the design of the CANAPPs is based on the internal wait point approach. Furthermore the project shows that the desired properties of a product as stated by the producer can be expressed and easily analysed when a CPN model of the system is constructed.

Also the graphical qualities of Coloured Petri Nets have proven to function well among engineers and between engineers and other groups involved in the design process.

6 Acknowledgements

I would like to thank Danfoss Instrumentation, especially Arne Peters, for their contributions to the project. I would also like to thank Lone Haudrum Olesen for her work on a early CPN model of the protocol.

References

- [1] M.Awad, J.Kuusela and J.Ziegler, *Object-Oriented Technology for Real-Time Systems: A Practical Approach Using OMT and Fusion*, Prentice Hall, 1996.
- [2] S.Christensen. *Design/CPN Message Sequence Charts Library Manual*. Computer Science Department, University of Aarhus, Denmark.
- [3] K.Jensen, S.Christensen, P.Huber & M.Holla. *Design/CPN Reference Manual*. Computer Science Department, University of Aarhus, Denmark. Online: <http://www.daimi.au.dk/designCPN>
- [4] J.D.Day & H.Zimmermann. *The OSI Reference Model*, Proceedings of the IEEE, vol.71, Dec.1983.
- [5] K.Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use. Vol.1, Basic Concepts*. Monographs in Theoretical Computer Science. Springer-Verlag, 1992.
- [6] K.Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use. Vol.2, Analysis Methods*. Monographs in Theoretical Computer Science. Springer-Verlag, 1994.
- [7] K.Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use. Vol.3, Practical Use*. Monographs in Theoretical Computer Science. Springer-Verlag, 1992.
- [8] K.Jensen, S.Christensen and L.M.Kristensen, *Design/CPN Occurrence Graph Manual*, Computer Science Department, University of Aarhus, 1996. Online: <http://www.daimi.au.dk/designCPN>
- [9] W.Lawrenz, *CAN Controller Area Network, Grundlagen und Praxis*, Hüttig Buch Verlag, Heidelberg, 1994.
- [10] A.Valmari, *Error Detection by Reduced Reachability Graph Generation*, Proceedings of the 9th European Workshop on Application and Theory of Petri Nets, pp.95-112.