32

VERIFICATION OF AN INTERACTIVE SOFTWARE BY ANALYSIS OF ITS FORMAL SPECIFICATION

*Ph. Palanque *† and R. Bastide *†*

* LIS-IHM, University of Toulouse I,
1, Place Anatole France
31042 Toulouse cedex FRANCE
Email: {palanque,bastide}@cict.fr

† Centre d'Études de la Navigation Aérienne,
7 avenue Edouard Belin
31055 Toulouse cedex
FRANCE

KEY WORDS: Formal specification, verification, UI design, Petri nets, tasks models

ABSTRACT: While designing interactive software, the use of a formal specification technique is of great help by providing non-ambiguous, complete and concise notations. The advantages of using such a formalism is widened if it is provided by formal analysis techniques that allow to prove properties about the design, thus giving an early verification to the designer before the application is actually implemented. This paper presents such a formalism, called Interactive Cooperative Objects. The paper mainly focuses on the formal analysis of the design, describing the calculation of its properties and their interpretation in terms of the software behaviour.

1. INTRODUCTION

Much current interface design is characterised by the use of event-driven direct manipulation interfaces. Despite the widespread use of such interfaces, the design methods and tools are surprisingly weak for handling the most critical aspect: the design of dialogue structure. Focus has often been devoted to the external appearance of the interface and several powerful interface presentation editors, e.g. Microsoft Visual Basic™, are now available. Although a lot of work has recently been devoted to formal specification of interfaces (Harrison & al. 1990, Paterno 1994), few references explain the practical use of formal analysis results in the design of an interactive application, and even fewer detail the mathematical analysis techniques to be applied to those formalisms.

In (Bastide & al. 1990) we proposed a Petri-net and object based formalism (PNO) that supports the specification of the dialogue structure of event-driven interfaces. In this paper we present a more complete formalism called Interactive Cooperative Objects (ICO). ICO extends PNO to support more of the software design cycle including specification, design (Palanque & al. 1993), verification and implementation (Palanque & al. 1994b). As with the PNO formalism, ICO is mathematically founded and the description of the mathematical analysis of a specification is the main goal of this paper. The verification of models is achieved by using the techniques provided by Petri Net theory. The analysis gives results concerning the dialogue and its influence on the interface behaviour. Hereafter, the first kind of results and their interpretation are fully detailed and their calculation is presented.

The next section presents the ICO formalism, in which a High-Level Petri Net (HLPN) model is used for modelling the object's behaviour. Section III uses a case study to demonstrate the use of the formalism. Section IV fully details the formal analysis results that can be obtained, and their interpretation in terms of the interface behaviour. Due to space reasons we are assuming from readers basic knowledge of Petri nets that can otherwise be found in (Peterson 1981).

2. INTERACTIVE COOPERATIVE OBJECTS

Interactive Cooperative Objects (ICO) is an object-oriented formalism intended for modelling interactive software where concurrency takes an important place (Palanque & al. 1994a). In this formalism, an object is an entity featuring four components: a data structure (a set of attributes), a set of operations (services offered to its environment or internal operations), an Object Control Structure (ObCS), and a presentation.

The ObCS of an object fully defines its behaviour: the availability of its services, how it processes service requests, the operations it performs on its own behalf, and the services it requires from other objects as a client. The ObCS is defined by a high-level Petri net. In a Petri net, places stand for state variables (and are depicted by ellipses) and transitions for state changing operators (and are depicted by rectangles). Each service of an object is associated with one or several ObCS's transitions using a function called Disp. A service request may be accepted only when one of its associated transitions is enabled, and it is performed by the occurrence of such a transition. Graphically, the name of the service associated with a transition is written inside it (and

K. Nordby et al. (eds.), *Human-Computer Interaction*

corresponds to the graphical representation of the function Disp), while the service input (resp. output) parameters label a broken arrow incoming to (resp. outgoing from) the transition. Services of an ICO which are at the user's disposal, called **user services**, are pointed out by a small ellipse at the beginning of the incoming broken arrow. An ObCS may include transitions which are not associated to any service: they correspond to the object's spontaneous activity. The action performed when a transition occurs is written in it. It consists either in performing an internal operation of the object, or in applying for a service offered by another object (if the transition is moreover related to a service, those two texts are separated by a line in the transition as for example the transition T3 of the left part of Figure 4) . Thus, the communications between objects are not buried in the code of the internal operations; they are defined in an explicit way, and this enable to check more easily whether the object cooperation satisfies the intended properties. Objects communicate according to a **client/server** relationship which formal semantics is defined in term of Petri nets in (Bastide 1992).

The Presentation of an object states its external look and is structured as a set of **widgets** (or interactors), such as Pushbutton, List-Box, etc. The user / software interaction will only take place through those components. Each user action on a widget may trigger one of the ICO's user service, and the user services of an ICO are precisely the ones which may be activated by the user through a widget. The relationship between user services and widgets is fully stated by the **activation function** which associates to each couple (widget, user's action) the service to be triggered.

In an interactive system each window is modelled by an ICO. The sequencing and synchronisation constraints for the availability of user services are expressed in the ObCS (as for the other services). Transitions relate to the object's user services, stating their availability, and user services are related to widgets through the activation function. Thus the active or inactive state of the widgets can be determined from the ObCS's current state: if no transition associated to a user service is enabled by the current state means that this user service is not currently available to the user. This must be shown to the user by shading out or otherwise inactivating the related widgets (Dix 1991).

3. MODELLING AN APPLICATION

This section demonstrates how to model an Automated Teller Machine (ATM), using the ICO formalism. Actually, we will not model the physical device but a software featuring the same functions as the real device. User's input will take place through classical interactors using direct manipulation input device as for example mouse (representing real user's actions on the physical device).

3.1. Informal Specification

In order to exemplify the particular features of the ICO formalism, we will add and remove several features to the classical functioning of an ATM. Our ATM allows the user to change the amount selected if it is greater than the amount allowed by uses. The entering of the pin number has been omitted to simplify the presentation. We abstract away from the actions performed by the user to enter the amount of cash he/she wants to withdraw, modelling this by an atomic action Select. Some actions may be performed in any order e.g. the withdrawing of the cash and the card.

3.2. Modelling the ATM

The ATM class is a full fledged ICO, featuring attributes, services, an ObCS and a presentation. The description of the class may be seen in Figure 1. The presentation, which consists of the widgets list and the activation function, is not entirely detailed here as the layout of an ATM is quite common.

```
Class ATM
Attributes
 a, b: Real; c: Card; r: {CANCEL, RETRY};
Methods
 Ok<a,b:real>:Boolean; Avail<c:Card>:Real;
Services
 InsertCard <c:Card>; Select <a: Real>;
 GetCash; GetCard;
ObCS (see Figure 2)
Presentation
      Layout  -- Not presented
      Activation function:
```

Widget	User's actions	Service
Pushbutton InsertCard	Click	InsertCard
Pushbutton Select	Click	Select
Pushbutton Cash	click	GetCash
Pushbutton Card	click	GetCard

end

Figure 1 : The Class ATM

In the ATM specification, the ObCS is modelled by a high level Petri net (HLPN). The main difference from basic Petri nets is that HLPNs incorporate the data structure and the data values in the models. (e.g. the precondition in transition T7). In HLPNs tokens are no more undifferentiated items, but may be references to other ICOs of the interactive software. This formalism provides a concise, yet formal and complete specification for the control structure of the software as each facet of an ICO is formally described using the mathematical foundations of Petri nets.
The functioning of a HLPN is quite the same as the one of a basic Petri net. However, the firing rule and the fireability of transitions is somehow different. A transition of a HLPN is fireable (may occur) if and

only if each of its input places holds at least one token (so that each variable labelling an input arc may be bound to an object). Moreover, if the transition features a precondition (graphically represented by a pentagon; see for example transition T7 in Figure 2) the values of the tokens in the input places of the transition must make the boolean expression of the precondition true in order for the transition to be fireable. When a transition is fired (or occurs), the objects bound to the input variables are removed from the input places and their values are processed by the transition's action which may also generate or delete objects. The new or modified objects are finally set into the output places, according to the variables labelling the arcs and thus fully stating the flow of references of objects in the net.

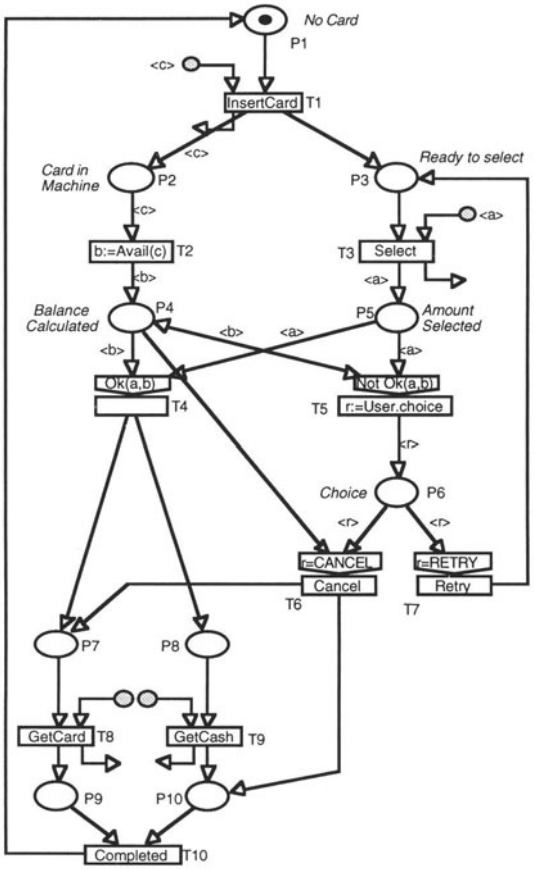

Figure 2 : ObCS of the ATM Class

The ObCS of the ATM (Figure 2) must be read in the following way:

• *Initialisation:*
The initial marking of the ObCS net is shown on Figure 2 by a black dot in the place *No Card*. This dot represents a basic token, describing the current idle state of the ATM.

• *Processing*
From this initial state, only the InsertCard service can be performed by the user. Indeed, the only transition that is fireable from the initial marking is the transition T1 which is related to the user service InsertCard. If the InsertCard service is requested by the user, the transition T1 will be fired. Its occurrence removes the token from the place *No Card*, and puts one token in place P2 and another in place P3. The token put in place P2 is not a basic one. Actually it is a token of the type Card, and is a reference to the Card inserted by the user. At that moment two transitions (T2 and T3) are enabled in the model as there is one token in each of their input places. T3 is related to the pushbutton Select so it will be fired only when the user clicks on this widget. At the opposite T2 is not related with a service and thus shows how internal behaviour as well as spontaneous activity can be modelled. This transition is fired as soon as it is enabled.

After the occurrence of the Select service (the transition T2) the ATM will perform one of the transitions T4 and T5, as both of them are enabled. Only one of then can occur as they are related to the same input places. The choice among them is directed by the result of the Ok internal operation of the ATM stating if the amount selected by the user is compatible with the card he/she has introduced. This is modelled by a precondition that has to be checked by the ATM before firing the transition. So, if the result of the Ok internal operation is TRUE, the ATM will fire T4, otherwise it will fire T5. If T4 is fired, the tokens in the places P5 and P4 are removed and one token will be put in each output place of the transition T4, that is to say P7 and P8. At this point, two user services are available: GetCash and GetCard, because their associated transitions (T8 and T9) are enabled within the ObCS. This illustrates the description of concurrent multi-threaded dialogues in ICO. At the end of the multi-threaded dialogue the transition T10 enforces synchronisation since it demands a token in both of its input places in order to fire. When transition T10 is fired, the system comes back to its initial state i.e. one basic token in the *No Card* place and no tokens in all the other places of the model.

4. VERIFICATION OF THE DIALOGUE

In this section we fill in the gap between the mathematical tools available for studying nets properties and the validation of a concrete system, by showing how Petri net theory may be used to prove properties on the behaviour of an interactive system modelled by ICOs. We show that proofs on a Petri net model can be performed in an automated way or by people with limited knowledge of mathematics unlike other formalisms (such as Z or temporal logic) where proof is made by theorem proving.

4.1. Verification of models

The validation of a model may be done in three different steps:

Unitary validation studies the behaviour of a single ICO, making a "good cooperation" assumption upon its relationships with other objects, that is: the servers used by the ICO are always able to process all its requests, and its clients send as many requests for its services as it may process. The dialogue properties we want to check are those specified in the application's interface requirements. These properties may relate: either to sequences of commands e.g. each update command may be followed either by a save command or by a cancel command; or to states that must or must not be reachable e.g. a list box may be able to contain any number of items above a given minimum. The main properties that can be proved using Petri net theory are: the absence of deadlock, the predictability of a command, the reinitiability, the availability of a command, among others (see (Peterson 1981) for more details on the properties themselves). When modelling an interface, unitary validation enables to prove that the dialogue of each window is designed in such a way that it meets the basics requirements in HCI design (for example no deadlock, user-driven style of dialogue, etc.).

Cooperation validation studies the behaviour of a set of ICOs according to the way they cooperates through the client/server relationship. It shows whether the objects cooperate in such a way that each server fulfils the needs of its clients, that is whether the cooperation preserves the objects' behaviour properties set out by the unitary validation. Cooperation validation concerns ICO cooperation that can be seen in two different way: cooperation with the user (through user services) and cooperation between ICOs. The former enables to ensure that the user's requests may be satisfied by the application's functional kernel and that the ICO's presentation reflects the state of the functional kernel, the latter is needed if an application may open several windows that can cooperate together (for example a user's action in a window can affect the availability of interactors in another window).

User cooperation validation relates the model of the inner behaviour of the interactive software to the typical behaviour of users interacting with that software. This requires to build one or more task models, and to checks whether those tasks are actually compatible with the application model.

The formal aspects of user cooperation validation are identical to those of cooperation validation. The key to our verification practice is that the users themselves may be considered as ICOs, interacting with the application in a client/server relationship. In modern event-driven software, the user mainly acts as a client, driving the interaction and requesting services from those offered by the application. The user is considered as a server only when an imperative confirmation is requested by the application (e.g. a modal dialogue box such as the one modelled in transition T5, Figure 2). The task models are thus taken into account in our formalism by embedding them into objects, and by checking their cooperation with the application's objects.

We shall illustrate this approach by giving two different task models within the ICO formalism in section 4.3.

4.2. Example of analysis: the ATM

Although this analysis is shown as done manually it may be **automated**, and several tools are actually available for this purpose (Feldbrudge & al. 1986). The ObCS of the ATM is live as is most often the case with "well designed" interfaces, except for some particular transitions (e.g. those associated with a *quit* command or initialising a window). The analysis of the structure of the net is based on both calculation of conservative and repetitive components in the ObCS of the ATM.

A set of places is called a conservative component if the number of tokens in this set of places remains the same whatever transitions are fired in the net. A sequence S of transitions is called a repetitive component if, given a marking M (that enables this sequence), the fireing of this sequence brings the net back in the marking M.

The structure of a Petri net N can be formally defined by N = <P, T, Pre, Post> (Peterson 1981) where P is the set of places of the Petri net, T the set of transition ($P \cap T = \emptyset$) and Pre and Post two functions such as:

Pre : P x T $\rightarrow$ $\mathbb{N}$; is called the input function

Post : P x T $\rightarrow$ $\mathbb{N}$; is called the output function.

Informally, Post represents the number of arcs from each transition to each place, and Pre represents the number of arcs from each place to each transition, and of course, both Post and Pre can be represented by matrices where rows represent places and columns transitions. More precisely, the element Pre(i,j) of the matrix Pre holds the number n if there exists n arcs from the place Pi to the transition Tj. The element Post(i,j) of the matrix Post holds the number n if there exists n arcs from the transition Ti to the place Pj. As the matrix C= Post-Pre, each element C(i,j) of C is such as : C(i,j) = Post(i,j) - Pre(i,j).

Calculation of conservative components.

The ObCS of the ATM is defined as follow:

P={P1, P2, P3, P4, P5, P6, P7, P8, P9, P10}

T={T1, T2, T3, T4, T5, T6, T7, T8, T9, T10}

The Pre and Post matrixes are not given for space reasons, however the matrix C is shown in Figure 3.

Finding the conservative components of the ATM consist in looking for all the positive solutions of

the equation: $\mathbf{f}^T.\mathbf{C}=\mathbf{0}$ where $\mathbf{f}^T$ is the transpose of a vector of places **f**. This consists in finding all the linear combinations of rows that lead to a null row.

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
P1	-1	0	0	0	0	0	0	0	0	1
P2	1	-1	0	0	0	0	0	0	0	0
P3	1	0	-1	0	0	0	1	0	0	0
P4	0	1	0	-1	0	-1	0	0	0	0
P5	0	0	1	-1	-1	0	0	0	0	0
P6	0	0	0	0	1	-1	-1	0	0	0
P7	0	0	0	1	0	1	0	-1	0	0
P8	0	0	0	1	0	0	0	0	-1	0
P9	0	0	0	0	0	0	0	1	0	-1
P10	0	0	0	0	0	1	0	0	1	-1

Figure 3. Matrix C=Post-Pre

It is trivial to understand that this kind of calculus can be automated, so we will only give here the result of this process. In this case four vectors are solution of the previous equation, and the conservative components for the ATM are:
P1+P3+P5+P6+P8+P10 ; P1+P3+P5+P6+P7+P9
P1+P2+P4+P7+P9 ; P1+P2+P4+P8+P10

Calculation of repetitive components. The calculus of repetitive components follows the same process, but it is applied to columns instead of rows. Finding the repetitive components of the ATM consist in looking for the positive solutions of the equation: $\mathbf{C.s}=\mathbf{0}$ where **s** is a vector of transitions. Two vectors are solutions of the previous equation and the repetitive components of the ATM are:
T1+T2+T3+T4+T8+T9+T10 and T3+T5+T7
It results from the calculation of the invariants that each place of the net belongs to at least one conservative component (the net is bounded) and each transition of the net belongs to at least one repetitive component (which is necessary for the net to be live).

Let's examine what can be proved:
- There is absence of deadlock (at least one transition is available at anytime) because the net is live (this has been ensured by the calculation of the marking graph (not presented here);
- The reinitiability of the ATM results from the fact that the net is live and bounded (Brauer 1986) whatever the state the ATM is in, there exists a sequence of transitions that will put the ATM back in its initial state. This is an important property since it ensures that each user will find the ATM in the same state as when he/she starts to use it.
- Availability of commands, as for example:
  - InsertCard is only available when there is one token in the place *No Card*. That means that it will be impossible for the user to insert several cards in the ATM.
  - GetCash is available only if the P8 place contains a token, i.e. after transition T4 has occurred. T4 can occur only if the result of the Ok operation is TRUE which means that the user will be able to withdraw cash only in that case.

4.3. Verification of cooperation with users

Once the unitary validation has proved that each object has the appropriate properties, it remains to verify that they cooperate suitably. The criterion for "suitable cooperation" between objects is that it doesn't change the possibilities of the behaviour of each object. In that case, if each of the objects interacting with the environment satisfies individually the functional requirements, then the same will hold when they cooperate. This cooperation verification is available because the client server protocol used is formally described in terms of Petri nets. As these constructs are very technical to describe, we will only use the results here, and the theoretical foundation of the protocol can be found in (Bastide 92) refined in (Sibertin 1993).

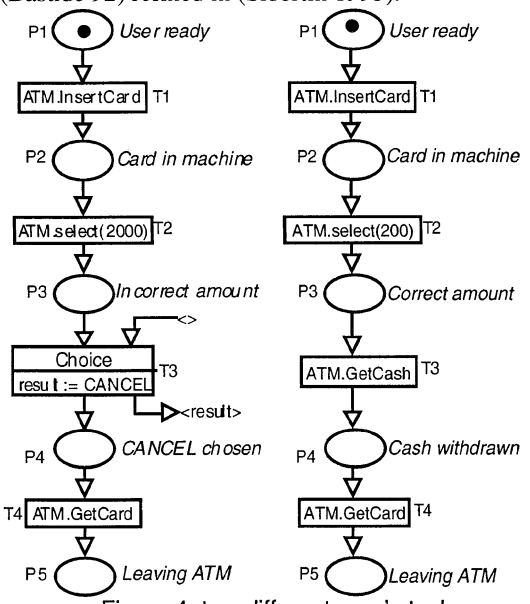

Figure 4. two different user's tasks

The ObCS in Figure 2 represents the behaviour of the ATM object as a server, i.e. to precisely state the sequence of services it is able to perform, and which operations it permits according to its internal state. Figure 2 also states the behaviour of the ATM as a client, i.e. how it may request information from other entities during its evolution. The behaviour as a client is stated by user transitions that are enabled, while the behaviour as a server is stated by invocation of other objects' services inside the actions of the transitions . Actually, at this rather coarse level of detail, the ATM is mainly described as a server. The only time it acts as a client is in transition T5: this transition contains the call `r := User.choice()` which models a modal prompt for a user response, blocking the evolution of the whole system until an answer has been provided.
Figure 4 describes two possible user's tasks for the use of the ATM. The model on the right describes the typical behaviour of a customer fronting the ATM, as might be gathered by observation. The task is a sequence of interactions between the user and the

system, consisting only in the user requesting services among those offered by the ATM. The question to ask is whether this particular sequence of service requests might actually be performed by the ATM, and this question may be answered by Petri net analysis. The model on the left describes a typical "incident" which may occur : the user requests an amount exceeding his/hers withdrawing capabilities. The sequence of interaction thus involves a step where the user acts as a server towards the ATM : the ATM requests the user to state whether he/she will try another amount or cancel his cash request altogether. In this case, the user chooses to cancel, and only removes his card from the machine.

The formal analysis of those models is done by comparing the language of the task models and the language of the system model. As both of those Petri nets are bounded, the language is regular, and thus can be compared automatically. We do not show the analysis here (only for space reasons) but it can be easily proved that the language of the ATM includes the language of the tasks models.

Moreover, it is possible to make unitary validation on the user's tasks models. The expected properties of "good" user's tasks models are exactly the opposite to the one expected for the system model. Indeed, we want to ensure that these models cannot be reinitialisable and that they do not feature infinite cycles. The analysis of the marking graph of the net allows to ensure the ending of a task (it actually occurs when the terminal marking of the graph represents the final state of the task).

5. CONCLUSION

In most interactive software that are nowadays developed, concurrency takes a very important part either because of their User-Driven nature or because of the interactions between several simultaneous users. Since it is difficult to cope with concurrency, specification, verification and implementation of this kind of software raise problems.
We have presented an approach based upon the Interactive Cooperative Objects formalism to solve these problems. The ICO formalism relies on Petri nets for modelling the dynamics of an interactive software. Indeed, PNs enable to cope with concurrency in an explicit and formal way, while they are quite natural and may be used by non concurrency-specialists. Moreover, due to the object oriented structuring of the ICO formalism, the models are well structured, easy to read, and have a well defined semantics. We have shown in this paper how the results of PN theory may be used for the verification of the design of an application, and what kind of properties can be checked through this analysis. The implementation stage is also addressed using the ICO formalism as the models of the interactive software can be directly executed by an ICO interpreter (currently under construction), or the code for an object-oriented language and an UIMS can be generated in an automated way (Palanque & al. 1994b). Moreover, several other interesting features (for example contextual help) can be ensured by exploiting the formal specification at run time (Palanque & al. 1994b).

ACKNOWLEDGMENTS

The development of the environment of the ICO UIMS is made possible by the use of high-level UIMS and graphical tools kindly donated by the ILOG company. The authors would like to thanks the HCI group of the Department of Computer Science of the University of York (U.K.) (and precisely A. Dearden for his useful comments) where Ph. Palanque was a visiting researcher during the development of this paper.

REFERENCES

Bastide R. (1992). Cooperative Objects: a formalism for modeling concurrent systems. PhD Dissertation. (In French).
Bastide R., Palanque P. (1990). Petri net with objects for the design, validation and prototyping of user-driven interfaces. In proceedings Interact'90, Elsevier, p.625-631.
Brauer W. (Editor) (1986). LNCS 254 & 255, Springer Verlag.
Dix A. (1991). Formal methods for interactive systems. Academic Press.
Feldbrudge F., Jensen K. (1986). Petri net tool overview in (Brauer 1986).
Harrison M. & Thimbelby H. (eds.) (1990). Formal Methods in HCI. Cambridge University Press.
Palanque P., Bastide R. (1994a). Petri net based Design of User-driven Interfaces Using Interactive Cooperative Object Formalism ; In (Paterno 1994).
Palanque P., Bastide R., Senges V. (1994b). Automatic code generation from a high-level Petri net based specification; In proceedings of EWHCI'94.
Palanque P., Bastide R., Sibertin C., Dourte L. (1993). Design of User-Driven Interfaces using Petri nets and Objects, In proceedings of CAISE'93. LNCS 685, Springer-Verlag.
Paterno F. (Editor) (1994). Proceedings of EG Workshop on Design, Specification and Verification of Interactive Systems.
Peterson J.L. (1981). Petri net theory and modeling of systems. Prentice-hall.
Sibertin-blanc C. (1993). A client server protocol for the composition of Petri nets. In proceedings of Application and Theory of Petri nets LNCS 691, Chicago.