24

Formal Specification and Design of Distributed Systems

A. Diagne[(*)] *& P. Estraillier*
Institut Blaise Pascal - Laboratoire MASI - CNRS UA 818
4, place Jussieu, 75252 Paris Cedex 05, FRANCE
Phone Number: (33 1) 44 27 73 65, Fax Number : (33 1) 44 27 62 86
e-mail : {Alioune.Diagne,Pascal.Estraillier}@masi.ibp.fr

Abstract

To design Distributed Systems (DS), the advantages of integration of Petri Nets (PN) and Object Oriented (OO) concepts had been widely discussed in the literature. Some integrations are mainly concerned with providing a formal basis for object oriented languages. Others focus on combining the abstract data type of the object technology with Petri nets. In both approaches the use of Petri nets is explicit so the resulting model diverges from the classical OO technology. The aim of this work is to make use of the both paradigms in a multi-formalism approach. We have a formal correspondence between a «pure» OO model derived from the Reference Model of Open Distributed Processing (RM-ODP) and a Petri net modular model. So specification, design and validation of DS can be considered in the OO paradigm which is most appropriate while verification is done in the PN paradigm for

Keywords

Distributed Systems, Specification, Design, RM-ODP, Petri Nets, Validation, Verification, Simulation.

1 INTRODUCTION

Specification and design of complex Distributed Systems (DS) requires to undertake validation and formal verification of such systems to manage their Quality Assurance (QA) [McG92]. Validation of a system consists of verifying the global coherence of its specification while verification consists of formal proof of structural and behavioral expected properties. The both activities are necessary to build reliable systems and to perform tasks such as debugging and testing at a symbolic level. They also allow one to enhance reuse of previously validated components.

OO paradigm offers a seamless process along the DS life-cycle with a good correspondence between its entities (class and objects) and the system's components. It gains also effective legitimacy from standardizations such as RM-ODP to federate architectural needs [X901]. But

(*) This work is partially supported by the IRENA project (EU 389), involving MASI Laboratory as academic partner and Alcatel ISR, Emeraude and SLIGOS as industrial partners. IRENA has developed formal methods for quality engineering in distributed systems with added values Petri nets.

it lacks of formal verification tools.

Our claim is not to define another OO toolset supporting all phases from analysis to implementation. We aim to define an OO template called OFClass. So we can shift from a classic OO model to OFClass in order to undertake validation and verification (Figure 1).

The OFClass template supports the modeling of both structural and dynamical information. Structural information is mainly the managed resources and dynamical information is local transformations on those resources and interactions in the whole system. The integration of these two conceptual modeling features allows smooth shift from many Object Oriented Analysis tool used for analyzing the domain requirements.

The model is solution oriented because first we do not deal with requirements analysis problems and second we aim to involve end-users as well as domain experts in the evaluation of the proposed solution by early simulation.

The proposed object template - OFClass - supports design and specification of DS with an integration of the main concepts outlined in the RM-ODP framework. The requirement analysis is supposed to have been done previously with another object model. OFClass is mapped to a modular PN model called OF-CPN which supports the formal method and theory of the Petri nets field. The OFC-PN model will be fully defined in Section 3. Properties are expressed on the designed system and once the validation phase run successfully, they are formally verified.

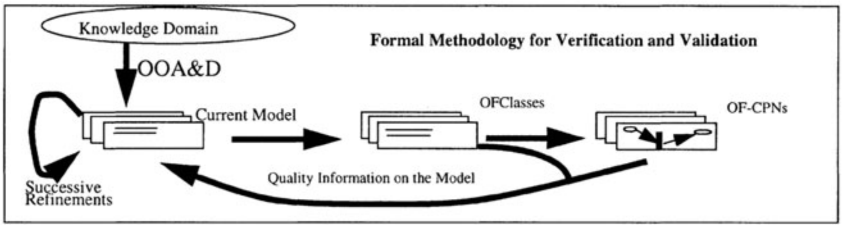
Knowledge Domain
Formal Methodology for Verification and Validation
OOA&D
Current Model
OFClasses
OF-CPNs
Quality Information on the Model
Successive Refinements

Figure 1 The Go To and From between the OFClass Template and a given OO Model

DS specification and design consist of describing the components, the interactions between them and the properties expected from the system. A component is an entity managing resources and performing processings which are its behavior. The components interact with each other to exhibit the global behavior of the system. The properties we need to verify on distributed systems are of two kinds:

- structural properties on the system components (resource conservation, service availability),
- behavioral properties on the whole system (deadlock-freeness, unreachable states, etc.).

These properties can be expressed as p-flows, liveness expectations and temporal logic assertions on the underlying OF-CPNs and hence be formally proved using the PN theory and tools. PN paradigm offers a powerful toolset dedicated to establish and prove properties dealing with DS reliability and safety.

The formal specification of DS by means of PN is tackled by many authors:

• Sibertin-Blanc has defined a client-server model in [Sib93]. He points out in properties that might be expected from interworking components such as honesty, discretion and reliability, a class of concepts that suits particularly to DS components. These properties can be verified on our OF-CPN model and will be conserved through their composition into a global PN.

• Lakos has also defined a generic modular PN model which supports the main concepts of OO technology [Lak95]. The model has a descriptive power comparable to the OO models. It supports inheritance, polymorphism and dynamic binding. We choose to put those OO concepts on the OF-Class template but not on the PN model because they do not enhance the verification phase.

In both cases, the global PN remains the sum of components and can have a very big size to prevent using classical PN tools (the composition mode does not preserve local properties of components except liveness of transitions in [Sib93]). We overcome this problem by the two-level validation. In our approach, the OFC-PNs are first verified to check the structural and some of behavioral expected properties. One this phase succeeded, we undertake simplifications on them by applying reduction rules [Had88]. Reduction is controlled to avoid loss of necessary structural information. We then compose them by place fusion, so the previously proved properties are conserved through all those transformations. The second level of verification is achieved on the resulting global net to ensure the safety of interactions.

This paper demonstrates the suitability of building a specification framework based on the OO and PN formalisms. The advantage of the multi-formalisms approach is to use in every phase of the methodology the most appropriate paradigm. We specially target DS interworking by demand-driven service invocation because of the class of interesting problems - directly related to the quality of the design - it sets up but the methodology is still valid for any object oriented specification and design for some adaptations.

Once the specification and design phases completed, we obtain a model of the DS as OFClasses. The methodology (Figure 2) we propose is based on a semi-formal validation of the global coherence an the completeness of the model in terms of fixed objectives and a formal verification of expected properties after transformation in a dedicated PN formalism.

Refinements of the model can be led by the information produced by our methodology (Figure 1). In fact, the quality information combined with refinement on the knowledge domain can introduce changes in some of the viewpoints and consequently in the model. The analysis phase is not in our concerns but it can be restarted if the initial model is too poor in the eyes of the quality information supplied on the model by our methodology.

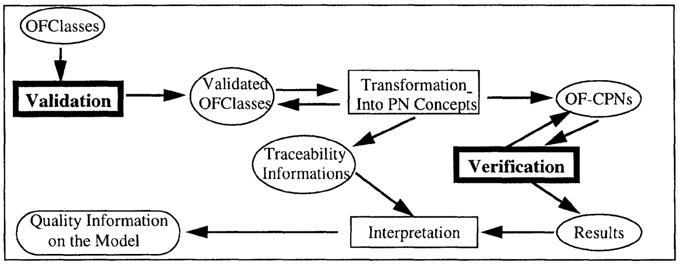
OFClasses
Validation
Validated OFClasses
Transformation_ Into PN Concepts
OF-CPNs
Traceability Informations
Verification
Quality Information on the Model
Interpretation
Results

Figure 2 The Distributed System Validation and Verification Formal Methodology

The OFClass template is presented in Section 2 through the DS specification needs while Section 3 presents the PN formalism underlying the methodology. In Section 3 we also introduce the formal correspondence between the two formalisms and the way properties on the OFClass template are mapped on the OF-CPN model to be formally verified. The Section 4 presents the validation an verification processes before conclusions in Section 5.

2 THE OO SPECIFICATION AND DESIGN OF DISTRIBUTED SYSTEMS

Object Oriented technology is now widely used to specify and design DS because of the tight similitudes between DS components and OO entities (classes/objects) and the fact that OO exhibits a seamless process along the whole life-cycle of the system. It is then possible to trace requirements through the different phases of the DS life-cycle from analysis to maintenance [Hay91]. Tracing the requirements allows an incremental evaluation and a smooth and safe move from phase to phase. Several proposals had addressed providing better formality to OO technology [Lop92], [Lin94] but none proposed a full support of QA in the earliest stages of the life-cycle.

ITU-T and ISO had proposed a RM-ODP as a framework for standardization and providing an architecture for the support of distribution and interworking among other features [X901], [902], [X903], [X904]. In [X903] It is stated that «The RM-ODP is generic, that is, independent of, and equally applicable to, arbitrary application domains making use of, or requiring distributed systems technology. Specific application domains may consider refining and specializing the RM-ODP to suit their particular needs, resulting in a model and standards for the realization of functions and components identified in the specific reference model».

The OFClass template is derived from the RM-ODP while conserving a good level of generality to stay a generic model for the DS specification. It is specially design to support the early evaluation of expected properties from the designed systems. We will show how the RM-ODP concepts are adapted to the OFClass as and when required in the remainder of the paper.

2.1 Specification and Design of a DS

Specification and design of a DS consists of describing the components one by one and the properties expected from them and from the whole system. Some of the components of a DS may already exist and then they are imported and make up the environment of the system. They are accessed across bounded interfaces and are considered to be valid. The environment of the current component (Figure 3) is the one of the system and the other system components it is interworking with.

Each component manages resources, offers services to the environment and has an abstraction on the others. This abstraction tells the way the remote resources might be accessed. It is defined through offered services by the environment. Each component defines for that purpose its **user-manuals**, i.e. the allowed sequences of its operations as services it offers. Each user-manual gives a coherent partial view on the component behavior and determines some of its usage constraints. It is first a compromise between the **Enterprise viewpoint** and the **Information viewpoint** of the RM-ODP in the sense that it is determined by the global scope of the DS to be designed and the semantics of informations handling in the DS.

In addition, a component must enhance the encapsulation towards a maximum locality on specification and design but also on verification and validation. So it must have locally a truthful abstraction of the interworking ones in order to proceed local evaluation by making hypotheses on the environment. The hypotheses are made on the safety and reliability of the mechanisms allowing to support distribution with the right technological choices. In other words, the local abstraction also includes the **Engineering** and **Technology** viewpoints of the RM-ODP.

The fact of localizing the main viewpoint at the component level is an arbitrary choice we make in order to validate and verify components one by one before the whole DS. The choice is warranted by the following statement in [X903]: «Viewpoints can be applied, at an appropriate level of abstraction, to a complete ODP system, in which case environment defines the context in which the ODP system operate. Viewpoints can also be applied to individual components of and ODP system, in which case the component environment will include some abstraction of both the system environment and other system components». The component correspond to a **class** - in the OO paradigm sense - of **basic computational objects** of RM-ODP.

A DS can have integrity constraints - the properties it must exhibit - which might be observed along its life. Properties expected from a DS can be expressed as **invariants** - like in RM-ODP [X903]. Some information is added to the model of the DS in order to observe the properties and functionalities it exhibits. This information is used to initiate interactions with the system. They model the observation facilities (Figure 3).

A component in a DS is a set of isolated entities - called **instances** - managing similar resources and offering similar services. Resources are the informations held by the component whose values determine the state of that component. Services are allowed processings performed on the resources on behalf of interworking environment with the usage constraints on them (Figure 4).

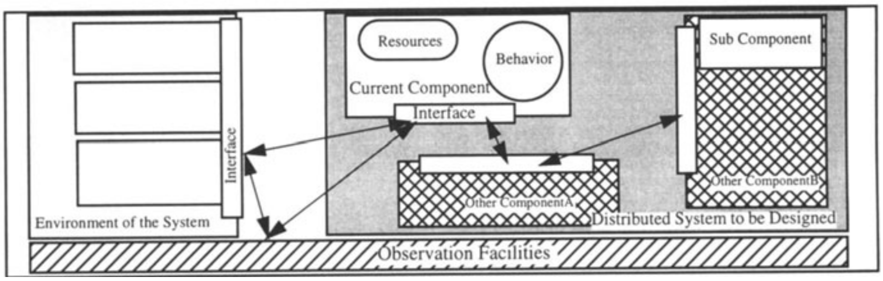

Figure 3 The General Model of a Distributed System.

Processings on the resources are called **operations** like in RM-ODP and consist of a set of elementary actions - basically resources accesses and modifications - executing a given semantical transformation. Precedence between operations in a service allows to obey the integrity constraints on the values of the resources and then make the component move safely from state to state. A service is held by a component called **provider** (the RM-ODP **server object**) and the components accessing it are called **consumers** (the RM-ODP **clients objects**). The service concept is not depicted in RM-ODP but we need it to have more formal interfacing and hence more reliable interworking.

2.2 Specification and Design of a Component

A component can also trigger some processings when reaching some meaningful states or when some events occur during interworking with the environment. These processings are slightly different from operations in the sense that they are not involved in services and can not be invoked by consumers. They model the autonomy of a given component. They are called **triggers** and **exceptions** and have no corresponding in RM-ODP concepts. Triggers bear eventually preconditions i.e. predicate on the resources values specifying the state in which they are executed. Triggers and exceptions can undertake interworking with the environment. Operations, triggers and exceptions are called processes in short.

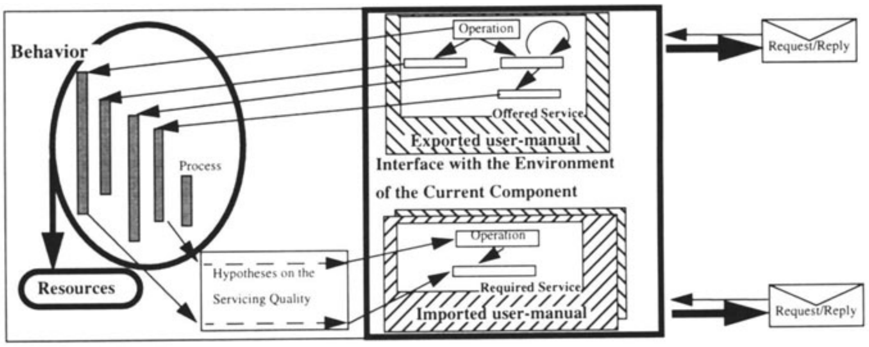

Figure 4 The Basic Model of a Distributed System Component.

2.2.1 Specification and Design of the Resources managed by a Component.

The resources managed by component are simple - even composite - informational entities with zero or many variables and the methods for their access and modification. The methods attached to a given resource can issue invocations and announcements to the environment to access other resources external to the component they belong to. They can also invoke directly the access methods of other resources of the same component. These methods are the only access points to the corresponding resources and are encapsulated in the operations of offered services. They are accessible from the environment through the operations.

A resource can be shared by the community of instances of the managing component or duplicated, each instance managing one copy. Unless otherwise stated, the resources are duplicated.

2.2.2 Specification and Design of the Behavior of a Component

The behavior of a component is given by the structuring of its processes and services. The processes are the sequences of elementary actions performed by the component on its local resources and interactions with the environment. There are some differences between processes according to their aimed purpose:

- the **operations** are processes accessible from the interface of the component. They can be invoked by the environment and have a **return code**. They can issue calls to access methods of the local resources and terminations to the environment. The remote resources are accessed through the operations of their managers,
- the **exceptions** are processes executed by the component on some terminations of invoked external operations. They are local to the component and can not be invoked by the environment but they can issue invocations and announcements to it. They have no return code,
- the **triggers** are processes automatically executed whenever the component reaches some state. As for exceptions, they are not invoked by the environment but they can issue invocations and announcements to it. They have no return code. Moreover they bear a precondition which is a predicate on the values of the component resources and which specifies the triggering state.

Operations have one entry point and one or many exit points according to the execution context at the provider side. The entry points of operations are accessible from the environment by consumers of services encapsulating it. On the contrary of operations, exceptions and triggers have one entry points but no exit points and their entry point are not accessible from the environment.

Processes are specified by full description of:

- **input/output parameters** and **local-variables** (return code is an output parameter),
- for triggers, the eventual precondition guarding the availability,
- the sequence of elementary actions realizing the body of the process.

Parameters represent informations supplied by a consumer in order to invoke an operation of a service while local variables are useful in actions description. Parameters and variables are described by giving their name and the type of values they must bear.

Precondition is a predicate on values of resources. Precondition enforces integrity constraints of a component. A trigger is executed if and only if holder is in a given state.

The description of the body for a process allows sequential and parallel compositions of elementary actions as well as loops and conditionals.

Service specification consists of giving the control-automata that describes the precedence constraints between operations [Kat95] and their signature. The control-automata describes all the parallel and sequential composition of operation-calls accepted by the provider. The consumer must issue requests according to the allowed sequence only. The sequencing of operations is the way we put control on the integrity of resources managed by the component. The accepted sequences of operations make the resources move between safe states.

A service can be attached to some instances which perform it; this is a way to design cooperating sub-systems such as replicated servers. The dedicated instances - if any - must then appear in the description of the service. Otherwise, the policy of cooperation used by the provider instances to handle incoming invocations must be defined at the provider side. The provider of a service also gives the invocation semantics (see Section 2.2.3) that are allowed to the consumers.

2.2.3 Specification and Design of the Interworking

Interworking is achieved by means of **specification proxies** which are the adaptation of the

stub-objects of RM-ODP. A specification proxy is an abstraction of a provider local to a consumer. Interactions with a given provider are addressed to its proxy. The major interest of specification proxies is to have a truthful abstraction of environment local to each component. At the design level local proxies can be overloaded in order to process invocations as if they were forwarded to the provider (without ensuring any constraint at the provider side) and return terminations, so stand-alone validation can be run with this abstraction respecting some given hypotheses on environment.

The provider of a service creates and exports a specification proxy carrying information about the service such as:

- the signature of the involved operations and their return code type to indicate their invocation constraints,
- eventually instances to address for performing the operations of the service.

A consumer of a service imports the right specification proxy and add information on return code treatment indicating processing that can be triggered after a call and the mode of invocation it would use (cf. the remainder of this Section). Some return codes can trigger exceptions at the consumer side. An exception is an internal method that restores some local context or performs some processing. The concept of **interceptor** which links the binding objects in RM-ODP is not used in our model. It is assumed to correspond to the message passing mechanism of the OO technology.

The design of interworking includes also the definition of different semantics of invocation between objects. The main problem one faces in designing DS is the asynchrony of events occurring in it. It is then very important to fully specify the way occurring events, mainly interactions, are handled. Interactions are basically **invocations** sent by consumers and **terminations** sent by provider once the operation performed. Some invocations do not require any termination and are called **announcement**. Many terminations may correspond to one invocation for instance in the case of a consumer subscribing to an event that is delivered each time it occurs. Invocation, terminations and announcement concepts are those of the RM-ODP.

The interworking is under respect of some invocation semantics:

- **Synchronous RPC**: the blocking of the consumer between the invocation and the termination allows one to model the synchronism at the consumer side,
- **Asynchronous RPC**: the consumer can keep on processing after sending an invocation and while waiting for the termination; it realizes then asychronism,
- **Synchronous Rendez-vous**: such a need is pointed out in DS design in order to model a strong synchronization between two components or instances to compute some «partial global state». It is achieved by a «rendez-vous» which requires the involved components to be in given states to get place.

The invocation semantics attached to a service by its provider are **contractual constraints** [Hel90] the consumers must observe. It is the **environment contract** in RM-ODP.

2.3 The Object Model: OFClass Definition Template

The OFClass template integrates the main concepts of OO paradigm. It allows one to build composite components from simple ones or to refine existing components. Hereafter is the **template** for the OFClass(*):

```
<identifier> isa OFClass
  interface
    [refines <refined OFClass identifier>
      [drop-processes {<selector of operation to drop>}]
      [drop-services {<selector of service to drop>}]
      [drop-resources {<selector of attribute to drop>}]]
    [includes {<identifier of OFClass involved in composition> instances {instances of the component}}]
```

(*) [...] means that the enclosed item is mandatory. {...} means that the enclosed item can be repeated once or many times. <...> means that the enclosed item is user-supplied.

```
    [uses
       {<provider identifier> service <service-identifier>
          {operation <requested operation selector> invocation-mode <synch|asynch|rendez-vous>
             [{accept-return <return-value expected> do <continue/exception>}]
             default -return do <continue/exception>}
       }
    ]
    [exports
       {service <service-identifier> operations <involved operations signature> automata <control-automata>
          [performed-by {<identifier of instance that performs the service's operations>}]
             invocation-mode <synch|asynch|rendez-vous>
       }
    ]
 structure
    instances {<identifier of instance>}
    resources { type <resource type> selector <resource selector> default < value> shared |duplicated
                    [composition {<previously defined resources of the current component>}]
                    {access-methods <access-methods-selector> begin <access-method-body> end}
              }
    [constructor begin {<actions for initialization>} end]
    [destructor begin {<actions for destruction>} end]
    processes
       [{<process selector>
          [triggered-on <predicate guarding the process execution in case of a trigger>]
          [returns <return-code type of the process>]
          [local-variables {type <local-variable type> selector <local-variable name>}]
          [input-parameters {type <parameter type> selector <parameter name>}]
          [output-parameters {type <parameter type>
                                   selector <parameter name> default <default returned value>}]
          begin [{<actions realizing the body of the process>}] end
       }]
    invariants
       [{<equational statements on resources>}
       [{<availability constraints on operations>}]
```

3 THE OF-CPN FORMALISM

The OFClass template is dedicated to design components of a DS and allows loose and tight coupling between them. We now propose a method for validation and verification of OFClasses based on the PN paradigm. It is then necessary to map formally the OFClass on a modular PN. The structure of the used modular PN will allows us to model the main aspects of the OFClass template (interface and structure). The invariants are assumed to be the expected properties on the model. They are expressed as flows and livelocks the OF-CPN must exhibit.

3.1 Some Modular Approaches in the colored Petri nets paradigm

Hierarchy and modularity supporting are two growing fields in the PN literature. These features provide framework for modular specification of complex systems in both top-down and bottom-up approaches. Their main objective is to fulfil lack of compositionality which is a weakness raised against PN formalisms [Jen90]. Several formalisms are proposed such as:

- Modular Net [Bac93] is a version of the colored PN model dedicated to modeling behaviors of active objects. Modules model internal states with attributes and communicate by interfacing transitions.
- Cooperating and Communicating Nets [Sib93], [Sib94] connect loosely-coupled components with formal protocol. OO concepts such as localization and communications across bounded interfaces are implicit. Asynchronous semantic of objects interactions are the only ones to be supported.
- The Object Petri Nets formalism [Lak95] supports complete integration of OO concepts including inheritance, polymorphism and dynamic binding to PN. Concepts of super places and super transitions enable the support of synchronous and asynchronous communications.

For the scope of this work, we will choose an hybrid approach of those depicted below. We will use both places merging and series transitions fusion (see Section 3.5) to achieve synchronous, asynchronous and rendez-vous interactions between OFClasses for the convenience of distributed systems needs. Tokens color domain allows an instance of an OFClass to choose dynamically partners for interactions. OF-CPN is the modular colored Petri net we use to model OFClass behavior.

Analyses can be performed on local sub-nets for almost all kind of composition in an more efficient way than on the global nets [Buc94] , [Mur94], [Chr95]. The main results that can be proved on sub-nets such as liveness and flows are preserved by the kind of composition we use (places merging and series transitions fusion). So state explosion in complex systems modeled by very big nets can be overcame by the «divide and rule» principle.

3.2 OF-CPN: Definition and General properties

We give here some notations on colored Petri(*) nets used throughout the remainder of the paper.Given a set E, E^* is the set $\bigcup_{n \in N} E^n$ where N is the set of natural. For two sets E and F, $E \times F$ is the cartesian product. A **color class** is a finite non empty set whose elements are called **colors**. There is one special color class which is an empty set called NULL. A **domain** is one color class or a cartesian product of a finite number of color classes, its elements are also called colors. If E is a set and f: $E \rightarrow N$ a function such $f^{-1}(N \backslash \{0\}) \neq \emptyset$ and is finite, f is called a multi-set and noted MSet(f) or $\sum_{e \in E} f(e) \cdot e$. Bag(E) is the set of all finite multi-sets over E.

A **colored Petri net** (CPN) is a **8-uple** <P, T, Dom, Type, Pre, Post, Guard, M_0> where

- P is a finite and non empty set whose elements are called **places**,
- T is a finite and non empty set whose elements are called **transitions** with $(P \cap T) = \emptyset$,
- Dom is a finite and non empty set whose elements are **domains**,
- Type: $P \cup T \rightarrow \mathrm{Dom}$ where Type(x) is the **color domain** of x,
- for each $(p,t) \in P \times T$, Pre(p,t), Post(p,t): $\mathrm{Type}(t) \rightarrow \mathrm{Bag}(\mathrm{Type}(p))$ the **input** and **output functions** if they exist,
- for each $t \in T$, Guard(t): $\mathrm{Type}(t) \rightarrow \{\mathrm{true}, \mathrm{false}\}$ the **predicate** associated with transition t. The default value is $\mathrm{Guard}(t) = \mathrm{true}$,
- for each $p \in P$, $M_0(p) \in \mathrm{Bag}(\mathrm{Type}(p))$, M_0 is the **initial marking** of the net.

A **marking** of the net is an injection $M: P \rightarrow \bigcup_{p \in P} \mathrm{Bag}(\mathrm{Type}(p))$ which verifies: $M(p) \in \mathrm{Bag}(\mathrm{Type}(p))$ for each p in P.

Firing rule: a transition is enabled for a color c in marking M - what we note M[t,c> - iff $\mathrm{Guard}(t)(c) \wedge \forall (p \in P), M(p) \geq \mathrm{Pre}(p,t)(c)$. The firing leads to a new marking N = M[t,c> defined by $N(p) = M(p) + \mathrm{Post}(p,t)(c) - \mathrm{Pre}(p,t)(c)$. The color c is often omitted in the notation.

A **sequence** of transitions is an element of T^*. Given two sequences σ_1 and σ_2, σ_2 is a **substring** of σ_1 if there is σ_3 and σ_4 such that $\sigma_1 = \sigma_3\sigma_2\sigma_4$. For a sequence σ and a subset T'' of T, $\sigma_{/T'}$ is the set of substrings of σ which are elements of T'^*. For $t \in T$, and σ a sequence, $\sigma(t)$ is the number of occurrences of t in σ. For two markings M and M', we note $\Sigma(M,N) = \{\sigma | M[\sigma > N]\}$ i.e. the set of sequences leading from M to N. A sequence s is allowed iff there is M and N markings such that $\sigma \in \Sigma(M,N)$.

Places and transitions are also called **nodes**. For a given node x, $\tilde{x}$ denotes the set

(*) For the Petri nets vocabulary and basic notations see respectively [Mur89] and [Sib93].

$\{y \in P, Pre(y,x) exists\}$ which is the set of its **predecessors** the and $\underset{\sim}{x}$ the set $\{y \in P, Post(x,y) exists\}$ which is the set of its **successors**.

<u>*Definition 1:*</u> ***The OF-CPN Model.***
*An **OF-CPN** is a **7-uple** O = <Net, AcP, DeP, ReP, GeP, BeT, ReT> where:*

- *Net is a Petri net,*
- ***AcP** is a subset of the set of places of Net whose elements are called the **accept-places** and for each p in AcP, $\tilde{p}$ is a single included in BeT and $\underset{\sim}{p} = \emptyset$,*
- ***DeP** is a subset of the set of places of Net whose elements are called the **deliver-places** and for each p in DeP, $\underset{\sim}{p}$ is included in ReT and $\tilde{p} = \emptyset$,*
- ***ReP** is a subset of the set of places of Net whose elements are called the **request-places** and for each p in ReP, $\tilde{p} = \emptyset$,*
- ***GeP** is a subset of the set of places of Net whose elements are called the **get-places** and for each p in GeP, $\underset{\sim}{p} = \emptyset$ and $\tilde{p}$ is included in ReT,*
- ***BeT** is a subset of the set of transitions of Net whose elements are called the **begin-transitions** and for each t in BeT, $\underset{\sim}{t}$ is a single included in AcP,*
- ***ReT** is a subset of the set of transitions of Net whose elements are called the **return-transitions** and for each t in ReT, $\tilde{t}$ is a single included in DeP,*
- *there is a bijection S:* $AcP \rightarrow DeP$,
- *there is a bijection R:* $ReP \rightarrow GeP$,

AcP, DeP, ReP and GeP are pairewise disjoints as well as BeT and ReT.

The OF-CPNs are individual components interfaced with the environment (other OF-CPNs) by sink (i.e. without successors) and source (i.e. without predecessors) places, and transitions connected to those places. Source places model incoming informations (invocations and announcements) while sink places model outgoing informations (invocations and terminations).

Places merging and series transitions fusion is the way we make components interwork because it preserves local properties on components [Mur89], [Sou90], [Val94]. Merging sink places at the consumers side with right source places at the provider side achieve the interactions between them by message passing (cf. Section 3.5).

There is a one to one correspondence between the accept-places and the deliver-places called S and another one between the request-places and the get places called R. The inverse of these bijections are equally called with the same names. Their signification will be highlighted later in Section 3.4. The set of all OF-CPNs is noted Θ.

We are now able to deal with the properties Sibertin-Blanc points out on components in [Sib93]. We must before establish the correspondence with specification concepts on the OFClass model.

3.3 Tracing Component Properties from OFClass to OF-CPN

Reliability properties are defined on components and influence the global interworking in the system. They determine the correctness of interworking sequences.

<u>*Definition 2:*</u> *Consumer Basic Properties*
*An OFClass acting as consumer is **honest** if each expected termination of an operation call from a provider corresponds to one and only one previous invocation of that operation. An OFClass acting as consumer is **discreet** if any termination sent by a provider corresponds to one and only one expectation of that termination.*

A honest consumer only expects terminations for the previous invocations it sends to its providers. The discretion property enforces consumers to request all the terminations issued by the providers in response to their invocations.

Definition 3: *Provider Basic Properties*
An OFClass acting as a provider is ***honest*** *if each outgoing termination corresponds to one and only one previous invocation of that operation. An OFClass acting as a provider is* ***reliable*** *if each accepted invocation of an operation from a consumer corresponds at least to one later termination.*

A reliable provider gives at least one termination for each accepted invocation from its consumers. The reliability of providers is a vital property to the non-blocking of the whole DS. A honest provider does not issue undue terminations.

We now define the honesty, discretion and reliability on the OF-CPN model.

Definition 4: ***Consumer Basic Properties in OF-CPN***
An OF-CPN modeling a consumer is honest iff for each allowed σ *in* T^*, σ *verifies:*
$\forall (p \in GeP)\, \forall (t_1 \in \tilde{p}) \left(\sigma(t_1) \le \sum_{t_2 \in S(\tilde{p})} \sigma(t_2) \right)$. *In other words, the allowed sequences of transitions contain less terminations scans than invocations.*
An OF-CPN modeling a consumer is discreet iff:
$\forall (p_1 \in ReP)\, \forall \left(t_1 \in \tilde{p_1} \right) (\exists (\sigma \in T^*))\ (\exists (t_2 \in R(p)))\ (t_1 \sigma t_2)$ *is an allowed sequence. In other words, each invocation corresponds to a termination scan.*

Definition 5: ***Provider Basic Properties in OF-CPN***
An OF-CPN modeling a provider is honest iff for each allowed σ *in* T^*, σ *verifies:*
$\forall (p_1 \in AcP)\, \forall \left(t_1 \in \tilde{p_1} \right) \left(\sigma(t_1) \le \sum_{t_2 \in S(\tilde{p})} \sigma(t_2) \right)$. *In other words, the allowed sequences of transitions contain less return-transitions than begin-transitions.*
A OF-CPN modeling a provider is reliable iff:
$\forall (p_1 \in AcP)\, \forall \left(t_1 \in \tilde{p_1} \right) (\exists (\sigma \in T^*))\ (\exists (t_2 \in S(p)))\ (t_1 \sigma t_2)$ *is an allowed sequence. In other words, each accepted invocation can be processed.*

Honesty is a very strong property for instance in case of subscribing to an event which is notified whenever it occurs. In such case there are many terminations sent for one invocation at the provider side and the consumer must scan them all, so it has many termination scans for the same invocation.

3.4 Modeling the Dynamics of an OFClass with OF-CPN

In [Hei92], Heiner studies thoroughly the transformation of code statements into PN concepts. The correspondence between basic code statements (like sequencing, loops and conditionals) and interworking semantics (like many variations of synchronism and asychronism) in one hand and PN concepts in the other is shown. We will use those correspondences to transform the statements in the language of our template into PN concepts.

Transformations into PN Concepts

From the interface of an OFClass, we can extract the contracts established between components. Each exported service is associated with a non-empty set of access semantics. Each user of that service declares the access-semantic chosen in that set for each of the operations. So we know for each operation the set of clients and how they use it.

Each operation at the server side is associated to an access-channel. The access-channel is exported to the community of clients. At client side we state the following rule:
• for synchronous and asynchronous accesses, the access-channel is shared by the community of clients. Invocations are distinguished by the color of circulating tokens (invocations carry the identity of sender and terminations carry the identity of the recipient).

• for rendez-vous accesses the access-channel is duplicated as many as there is such clients. For each client, the channel is again duplicated as many as it is invoked. The rendez-vous semantic is realized by a series-transitions fusion [Had88].

Places of access-channels have color domains determined by the parameters of the corresponding operation. The automaton for an offered service is transformed into a free-choice Petri net and exported to the clients.

The structure transformation is based on the method developed by Heiner in [Hei92]. This method studies thoroughly the transformation of the main of code statements into Petri net items.

Resources of a component, parameters and variables of its operations and triggers are modeled by places. The color domains of such places are determined by their types.

The elementary local actions of operations and triggers are transformed to transitions. Places are connected to transitions according to transformations performed. Arcs are valuated by color functions that traduce those transformations.

The transitions corresponding to invocation issues and termination scans are connected to places of access-channels imported from the right server. They are also connected to places modeling parameters of the interactions. These transitions are merged with the ones modeling the corresponding operation (or its duplicate in case of multiple invocations) in the Petri net describing the automaton of the enclosing service. So the component respects the contractual constraints on the interworking.

For each operation, a begin-transition is added. It has the input-place of the access-channel as input place and the places corresponding to input parameters as output places to perform their initialization. The arcs are correctly valuated to propagate the values of parameters. Each transition corresponding to the action return that ends an operation is connected to the output place of the right access-channel.

Coloration

Once the structure of the OF-CPN built, we take into account the features related to circulating information. For instance, many requests issued by different clients must be distinguished at the server side. For that reason, we apply the coloration procedure. It is useful to give to circulating tokens a color that indicates their origin and destination. The coloration procedure has three constraints to fulfil:

- duplicated local resources are colored to know their managing instance,
- parameters are colored to know which client had sent them in an invocation,
- variables are colored to know to which operation or trigger they are attached.

The coloration procedure modifies color domains of places according to these three constraints above. It allows one to make the many instances of a given OF-CPN to share the same structural skeleton.

3.5 Modeling Interworking with OF-CPNs

Interworking is achieved by merging the access-channels of providers and consumers with respect of the typing (color domains for places). This composing procedure is called **interworking-composition**. Source places at one side are merged with sink places at the other and vice versa.

Definition 6: *Interworking in the OF-CPN Model*
*Interworking is defined by the binary relation **Uses** on* $\Theta \times \Theta$ *as following:*
$O_1, O_2 \in \Theta, \quad O_1 \mathrm{Uses} O_2 \Leftrightarrow \exists (u)\, \mathrm{ReP}(O_1) \rightarrow \mathrm{Acp}(O_2)\, \forall \left(p \in u^{-1} (\mathrm{AcP}(O_2)) \right),$
we have: $(\mathrm{Type}(p) = \mathrm{Type}(u(p))) \wedge (\mathrm{Type}(R(p)) = \mathrm{Type}(S(u(p))))$.

The binary relation ***Uses*** maps access-channels of consumers on those of providers in way that enables the merging. The correspondence between request-places of the consumer with accept-places of the provider is extended to their associated get-places and deliver-places respectively. The type matching associated with the application u through the bijections R and S enables to merge the places.

For a given OF-CPN O_1, we note $\mathrm{Uses}(O_1, \Theta) = \{O_2 \in \Theta | \mathrm{Uses}(O_1, O_2)\}$ the set of providers for O_1, and $\mathrm{Uses}(\Theta, O_2) = \{O_1 \in \Theta | \mathrm{Uses}(O_1, O_2)\}$ the set of consumers for O_2. The resulting OF-CPN after interworking-composition of O_1 and O_2 is noted $U(O_1, O_2)$.

3.6 Local Properties Preservation by Interworking

We now have to show that the local properties of components are conserved by the place fusion we use to compose them into clusters of interworking components. An OF-CPN modeling a provider (resp. a consumer) is told «well behaving» if it is honest and reliable (resp. honest and discreet).

Proposition 1: ***Conservation of the Basic Properties of a Consumer through Interworking***

For an OF-CPN modeling a consumer, honesty and discretion are conserved through interworking-composition with one or many well behaving providers.

Proof

We will give the proof for one consumer composed with one provider and the generalization is straightforward.

Let cons be an OF-CPN modeling a consumer, we consider here that ***Uses***(cons, Θ) is not empty and let $\mathrm{prov} \in \mathrm{Uses}(\mathrm{cons}, \Theta)$.

Let us prove first the honesty conservation. We have ***Uses***(cons, prov), hence there is an application

u: $\mathrm{ReP(cons)} \rightarrow \mathrm{AcP(prov)}$ that defines the merging.

We consider the sets:

$P_0 = (\mathrm{ReP(cons)} \cup \mathrm{AcP(prov)}) - u^{-1}(\mathrm{AcP(prov)})$ and

$P_1 = (\mathrm{GeP(cons)} \cup \mathrm{DeP(prov)}) - R(u^{-1}(\mathrm{AcP(prov)}))$ where u is the application defined above. We must prove the honesty on the sets P_0 and P_1.

Let $p \in P_1$, $t \in \tilde{p}$ and $\sigma \in T^*$ where T is the set of transitions on the resulting OF-CPN after interworking composition, we consider $\sigma_1 = \sigma_{/T(\mathrm{prov})}$ and $\sigma_2 = \sigma_{/T(\mathrm{cons})}$:

σ_1 respects honesty on prov and σ_2 respects it on cons. Hence σ respects honesty.

Let us prove now the discretion always on P_0 and P_1.

Let $p \in P_1$ and $t \in \tilde{p}$, discretion is respected with some sequence in T(prov) or in T(cons) which are subsets of T. Hence discretion is respected.

The generalization to ***Uses***(cons, Θ) is possible because if prov_1 and prov_2 are providers for cons then prov_2 is still a provider for $U(\mathrm{cons}, \mathrm{prov}_1)$.

Proposition 2: *Conservation of the Basic Properties of a Provider through Interworking*

For an OF-CPN modeling a provider, reliability and honesty are conserved through interworking-composition with one or many well behaving consumers.

Proof

As for the previous proposition, the proof is given for one provider composed with one consumer. The proof here is exactly the same that in Proposition 7.

4 THE VALIDATION AND VERIFICATION OF A DS

4.1 Validation of a DS

Validation of a DS is run on the OFClasses to check the global coherence and completeness of the model. It is done part with non PN oriented tools and is semi-formal. It consists of:

- check the coherence of interfaces to see if the model is fully specified,
- compute the graph of invocations to detect in the earliest faults like trivial deadlocks.

Validation gives a first information about the quality of the model. It is not worth to undertake formal proving activities on a model that does not fulfil expectations like completeness and global coherence. This first step is non-formal but it is a previous and necessary step to the more formal ones.

4.2 Verification of a DS

The verification of DS in run using PN tools. Its objective is to prove properties expected from the target system such as resources conservation, existence of some interesting states for the DS and safety of interworking. Verification of specifications consists of:

- prove formally local properties such as resources conservation, availability of offered services and respect of directions of use enforced by providers on required services,
- prove formally global properties such as safety of interactions (i.e. detect non trivial deadlocks caused by concurrence) and also find the meaningful states,
- perform simulation on the model to make sure it matches the logical expectations of its end-users.

The major drawback of PN tools is the combinatorial explosion for complex systems modeled by big nets. To shape it, we will undertake a two-level verification. OFClasses carry sufficient informations to have a local and truthful abstraction of the whole system for some postulates. It is hence interesting to use such informations in order to verify properties on a component that are conserved on the whole system. This level is called fine-grained verification. Once those properties verified, simplifications can be made on components. For instance, it is no longer worthy to deal with the detailed design of processes because the main concern now is the interworking between components, we can therefore consider interactions between components as the atomic verification step. This second level is the coarse-grained verification.

4.2.1 Fine-Grained Verification

At this first level, we deal with properties local to a given component. We can compute p-flows [Cou93] that model resources conservation and reachability graph for liveness properties. The p-flows are conserved through interworking composition [Sou90]. The graph local to a component carries information about liveness of local transitions and reachability of states of the component.

We can complete the local graph by an abstraction of the environment, so we obtain a self contained component for the postulate that the environment is well-behaving. We sketch the information expected from environment by overloading the specification proxies but we do not ensure how this information is produced. This postulate allows one to enforce continuity among operations according to defined exported services. Further information can be deduced from the completed component like some new reachable states introduced by the interactions. Moreover, the basic properties of components (honesty, reliability and discretion) [Sib93] can be proved at this level on the reachability graph.We can detect local deadlocks in a component. The local home state, if any, predict the ability of a given component to restart and make some services always available.

4.2.2 Coarse-Grained Verification

Once the fine grained level verification achieved, behaviors of components can be shrunk to simplify local features. For instance it is no longer worthy to deal with detailed operations execution including elementary actions as processing steps. The OF-CPNs can then be simplified.

Simplification consists of applying the reduction rules [Hadd88] while respecting the following principles:

• we avoid the reduction rules which skip away shared places or transitions. This allows to continue to clearly identify the interworking,

• we avoid the reductions rules which skip away transitions with guards because they model the control on components and hence on the application.

Here the major concern is safety of interactions between components. In [Chr95], Petrucci & Christensen have proposed a way to compute global properties from the local reachability graphs. We prefer this method to the ones exposed by Murata & Notomi in [Mur94] and by Valmari in [Val94] because it shapes infinite state spaces and it handles place fusion.

Global properties worth to be studied are:

- home state for the whole system (i.e. state interesting for restarting),
- deadlock freeness of interactions (i.e. no global deadlocks caused by concurrence),
- fairness of resources access in the whole system (i.e. accesses do not suffer from starvation),
- liveness properties through interworking (i.e. allowed sequence of service execution).

Simulation of the whole system can stand for model animation. It is a way to involve end-users in evaluating the behavior of the specified system. Eventual faults in the design can then be detected and corrected in the earliest by refining the model.

5 CONCLUSIONS AND FUTURE WORKS

This work has enhanced the RM-ODP with a formal basis for validation and verification. The multi-formalisms approach undertakes each of the processes in the most appropriate field of study. So specification, design and validation are achieved in the OO domain while verification is formally done in the PN domain. Specification must be oriented towards the expected results i.e. that concepts like safety and reliability must be expressed as properties and verified in the earliest phases of the life-cycle. These concepts are often known from the analysis of the requirements domain as expected properties on the target system. We guarantee that the built systems will not diverge from those properties. The use of PN is implicit and the correspondence with OO is made formal. So engineers involved in the specification and design activities do not need to win up the PN theory and practise.

Instead of modeling complex systems by big nets, we use the two-level approach in verification to manage them, what was not possible before when using the analysis tools - reachability graph for instance - on the raw net. This two-level approach enhances the possibility to store validated components as reusable libraries. Enhancing reuse in DS specification and design is way to involve more experts working independently from each other at this stage.

Components structure and dynamics are modeled in a solution-oriented way that allows via simulation to involve end-users in early phases of analysis and modeling. Components validation with abstraction of it's environment is a way towards managing libraries of validated entities and enforcing of reuse in specification and modeling.

This model is under stand-alone development and test before integration in the **MARS** Methodology [Est92] supported by the **AMI** framework. Prototyping can be achieved in a satisfactory way because of the formal semantic of the template and will be one of our next concerns. We therefore need to transform OFClass into a more operational formalism called H-COSTAM (Hierarchical COmmunicating STAte Machine) developed in our laboratory and which is better oriented towards code generation [Kor94].

Our approach is led by the RM-ODP one. We do not have any expectations neither from the domain of the target systems nor from the software engineering methodology. We can therefore prove properties suited to any DS in the limits of the semantic of PN structural and dynamic properties. We are foreseeing integration of other formal methods (theorem provers) for validation of elaborated collaboration in DS like negociation.

The current model does not take in account the stream and signal interactions outlined in RM-ODP. It is because we only target DS interworking by service invocations. We are now extending the model toward these features to support specification of applications like telecommunication services that handle them.

6 REFERENCES

[Bac93] H. Bachatene & J.M. Couvreur, «*A Reference Model for Modular Colored Petri Nets*», In Proc. IEEE/System Man and Cybernetics Int. Conf., Le Touquet, France, October 1993.

[Buc94] P. Buchholz, «*Hierarchical High Level Petri Nets for Complex System Analysis*», In Proc. 15th Int. Conf. on Applications and Theory of Petri Nets, Spain, June 1994, LNCS 815 PP. 119-138.

[Chr95] S. Christensen & L. Petrucci, «*Modular State Space Analysis of Colored Petri Nets*», In Proc. 16th Int. Conf. on Application and Theory of Petri Nets 1995, Italy, June 1995, LNCS 935, PP 201-217.

[Cou93] J.M. Couvreur, S. Haddad & J.F. Peyre, «*Generative Families of Positive Invariants in colored Nets Sub-Classes*», In Advances in Petri Nets 1993, G. Rozenberg Ed. LNCS 674, PP. 51-70.

[Est92] P. Estraillier & C. Girault, «*Applying Petri Net Theory to the Modeling, Analysis and Prototyping of Distributed Systems*», In Proc. IEEE/SICE Int. Workshop on Emerging Technology for Factory Automation, Australia, August 1992.

[Had88] S. Haddad, «*A Reduction Theory for Colored Petri Nets*», In Proc. 9th European Workshop on Application and Theory of Petri Nets, Venice, Italy, June 1988, LNCS 424, PP. 209-235.

[Hay91] F. Hayes & D. Coleman, «*Coherent Models for Object Oriented Analysis*», In Proc. OOPSLA'1991, Phoenix, Arizona, USA, PP. 171-183.

[Hei92] M. Heiner, «*Petri Net Based Software Validation, Prospects and Limitations*», Technical Report TR92-022, GMD/First at Berlin Technical University, Germany, March 1992.

[Hel90] R. Helm, I.M. Holland & D. Gangopadhyay, «*Contracts: Specifying Behavourial Composition In Object Oriented Systems*», In Proc. OOPSLA'1990, Ottawa, Canada, October 1990, PP.169-180.

[Hen90] B. Henderson-Sellers & J.M. Edwards, «*The Object Oriented Systems Life Cycle*», In Communications ACM, 33(9), September 1990, PP. 142-159.

[Jen90] K. Jensen, «*Colored Petri Nets: A High Level Language for System Design and Analysis*», In Proc. Advances in Petri Nets 1990, New York, 1990, LNCS 483, PP. 342-416.

[Kat95] J. P. Katoen, «*Causal Behaviours and Nets*», In Proc. 16th Int. Conf. on Application and Theory of Petri Nets 1995, Torino, Italy, June 1995, LNCS 935, PP 258-277.

[Kor95] F. Kordon & W. El-Kaim, «*H-COSTAM, a Hierarchical COmmunicating STAte-Machine Model for Generic Prototyping*», In Proc. of the 6th Int. Workshop on Rapid System Prototyping, Triangle Park Institute, N. Kanopoulos Ed., IEEE Comp. Soc. Press 95CS8078, PP 131-138.

[Lak95] C.A. Lakos, «*From Colored Petri Nets to Object Petri Nets*», In Proc. 16th Int. Conf. on Application and Theory of Petri Nets 1995, Torino, Italy, June 1995, LNCS 935, PP 278-297.

[Lop92] O.P. Lopez, F. Hayes & S. Bear, «*OASIS: An Object Oriented Specification Language*», In Proc. 4th Conf. CAiSE'92, Manchester, UK, May 1992, LNCS 593, PP.348-363.

[Lin94] F.J. van der Linden, «*Formal Methods: From Object-Based to Object-Oriented*», In ACM Sigplan Notices, 29(7), PP. 29-38, July 1994.

[McG92] S. McGinnes, «*How Objective Is Object Oriented Analysis*», In Proc. 4th Conf. CAiSE'92, Manchester, UK, May 1992, LNCS 593, PP.1-16.

[Mur89] T. Murata, «*Petri Nets: Properties, Analysis and Applications*», In Proc. IEEE, 77(4), April 1989, PP.541-580.

[Mur94] T. Murata & M. Notomi, «*Hierarchical Reachability Graph of Bounded Nets For Concurrent Software Analysis*», In Transactions IEEE on Software Engineering, 20(5), May 1994, PP.325-336.

[Sib93] C. Sibertin-Blanc, «*A Client-Server Protocol for the Composition of Petri Nets*», In Proc. 14th Int. Conf. on Application and Theory of Petri Nets 1993, Chicago, June 1993, LNCS 691, PP. 377-396.

[Sib94] C. Sibertin-Blanc, «*Cooperative Nets*», In Proc. 15th Int. Conf. on Application and Theory of Petri Nets 1994, Zaragoza, Spain, June 1994, LNCS 815, PP. 471-490.

[Sou90] Y. Souissi, G. Memmi, «*Composition of Nets via a Communication Medium*», In Proc. Advances in Petri Net 1990, LNCS 483, G. Rozenberg Ed. PP. 457-470.

[Val94] A. Valmari, «*Compositional Analysis with Place-Bordered Subnets*», In Proc. 15th Int. Conf. on Application and Theory of Petri Nets 1994, Zaragoza, Spain, June 1994, LNCS 815, PP. 531-547.

[X901] ITU X.901 & ISO/IEC 10746-1, «*Basic Reference Model of Open Distributed Processing, Part 1: Overview and Guide to Use*», Committee Draft , July 1994.

[X902] ITU X.902 & ISO/IEC 10746-2, «*Basic Reference Model of Open Distributed Processing, Part 2: Descriptive Model*», Committee Draft , April 1994.

[X903] ITU X.903 & ISO/IEC 10746-3, «*Basic Reference Model of Open Distributed Processing, Part 3: Prescriptive Model*», Committee Draft, February 1994.

[X904] ITU & ISO/IEC JTC1/SC 21/WG 7, «*Basic Reference Model of Open Distributed Processing, Part 4: Architectural Semantics, Specification Techniques and Formalisms*», Committee Draft , July 1994.