A Multi Formalisms Prototyping Approach from Formal Description to Implementation of Distributed Systems

Une approche de prototypage multi-formalismes pour la description et la réalisation de systèmes répartis

Alioune Diagne & Fabrice Kordon,
MASI Laboratory
Blaise Pascal Institute, University P.&M. Curie
4 place Jussieu, 75252 Paris Cedex 05, France
E-mail: `Alioune.Diagne@masi.ibp.fr, Fabrice.Kordon@masi.ibp.fr`

Abstract

This paper proposes a methodology to build safe distributed systems that considers both conceptual and operational description aspects. At the conceptual level, we focus on the safety properties expected from the system. Such properties are stated and then verified. At the operational level, we focus on properties addressing the optimization of the generated code. Traceability between the two levels is managed in a satisfactory semi-automatic way. It preserves the properties proved at the first level and discards informations that are not relevant for code generation.

Keywords

Formal Specification, Validation, Verification, Prototyping, Distributed Systems.

Résumé

Cet article présente une méthodologie pour la modelisation et de prototypage de systèmes répartis considérant à la fois les aspects conceptuels et opérationnels. Au niveau conceptuel, nous insistons sur la définition et la vérification des propriétés attendues du système. Au niveau opérationnel, nous insistons sur les propriétés permettant d'optimiser le code généré à partir du modèle initial. La traçabilité entre les deux niveaux et maintenue de manière semi-automatique; elle permet d'écarter les informations contenues dans le modèle mais qui sont inutiles pour la génération de code. Les propriétés du modèle conceptuel sont préservées durant la transformation.

Mots clefs

Spécification formelle, Validation, Vérification, Prototypage, Systèmes Répartis.

1. Introduction

The complexity of distributed systems is a problem when designers want to evaluate their safety. Systematic tests of the application cannot be considered as verification because of the potentially infinite number of states due to the parallelism.

A solution should be the promotion of verification of a system at the early design phases (modeling). For the same reason that disables the systematic test, simulation of a model cannot be considered safe in off.

For this reason, formal description techniques represent a solution. While a designer describes his system, he states the expected properties and then verifies it. Formalisms such as Petri nets have been systematically improved and Colored Petri nets [Jensen 92] are now widely used for the modeling of distributed systems.

However, the main drawback of Petri nets is their poor structuring facilities. They are thus difficult to use for large scale systems. Recents studies have led to two main research areas:

- extension of Petri nets in order to add some hierarchical or structuring capabilities [Buchholz 94],
- association of Petri nets with another formalism that brings its structuration capabilities [Lakos 95].

Object Oriented (OO) techniques facilitate the decomposition of complex systems into interworking components. For that reason, such a paradigm could be a solution for the structuration of a Petri Net specification.

The design method we present in this paper deals with both conceptual and operational aspects. It merges OO techniques and formal description of a system by means of Petri nets. This should enable the construction and analysis of large distributed systems whithout having to deal with too much complexity.

In such systems, components should interact while managing local resources. Interaction does not have to corrupt the integrity of the involved components and has to be safe (i.e. free of faults like deadlocks or starvation). The reliability of the systems we aim to produce relies on the respect of such properties. So expected properties must be known and verified during the specification of the solution. Then, they have to be preserved all over the system life-cycle.

Our design method focuses on a «safe conception» of distributed systems. It relies on an adapted formal representation for such applications that enables both the definition and verification of adapted properties: Petri nets. However, while Petri nets are not easy to use (no structuration facilities, difficulties to handle large size specifications, etc.), we propose to hide their semantic. This could help a non specialist designer to elaborate a system that has «good» properties (deadlock-free, proper use of software components, etc.).

We describe our methodology in Section 2. Then, Section 3 presents the involved formalisms that are compared to depict out the similarities and differences. The Section 4 deals with the different transformations that enable to link the formalisms and manage traceabiltity all over the methodology. Analysis performed in the formal description is also presented. An example is detailed in Section 5 to illustrate the shift from the conceptual level to the operational one before conclusion.

2. The Methodology

When a designer builds a system, he first deals with its architecture. He then has to take care of ***conceptual*** aspects like the functions the system has to provide or the interoperability of its software components. Once the conceptual description has reached a satisfactory maturity, he may then think about ***operational*** or implementation related details.

These two separate aspects of the design deal with different types of information and it should be valuable to express them by means of separate representation. However, the second one is deduced from the first one and a simple procedure should be provided.

Both steps should also consider the ***properties*** of the system. Properties may be derived:

- from the requirements, like «the system has only one terminal state»;
- from conception hypothesis, like «this service must be run first to operate a subsystem»;
- from operational hypothesis, like «this variable can be duplicated in order to increase efficiency».

Verification or computation of such properties cannot be done without having a formal description of the system. Petri nets are suitable for such a role but they are not suitable to handle large specifications. Moreover their capabilities allow one to describe both conceptual and operational specifications. It is then easy to mix up both features in the same model.

This is why the methodology we propose relies on three formalisms (Figure 1):

- Well formed Petri Nets [Chiola 91] fit all the ***formal*** needs. It is a potential target used to verify or compute properties of the system model;
- OF-Class (Object Formalism Class) [Diagne 96] provides a ***conceptual description*** of the system. It provides information about the association of components, the way they behave and how they should be used. It may be simulated and transformed into the formal description;
- H-COSTAM (Hierarchical COmmunicating STAte Machine Model) [Kordon 95] allows the designer to deal with ***operational*** aspects of his system. Such a description may be derived from the conceptual description by addition of information. It can also be transformed into the formal description and enable code generation.

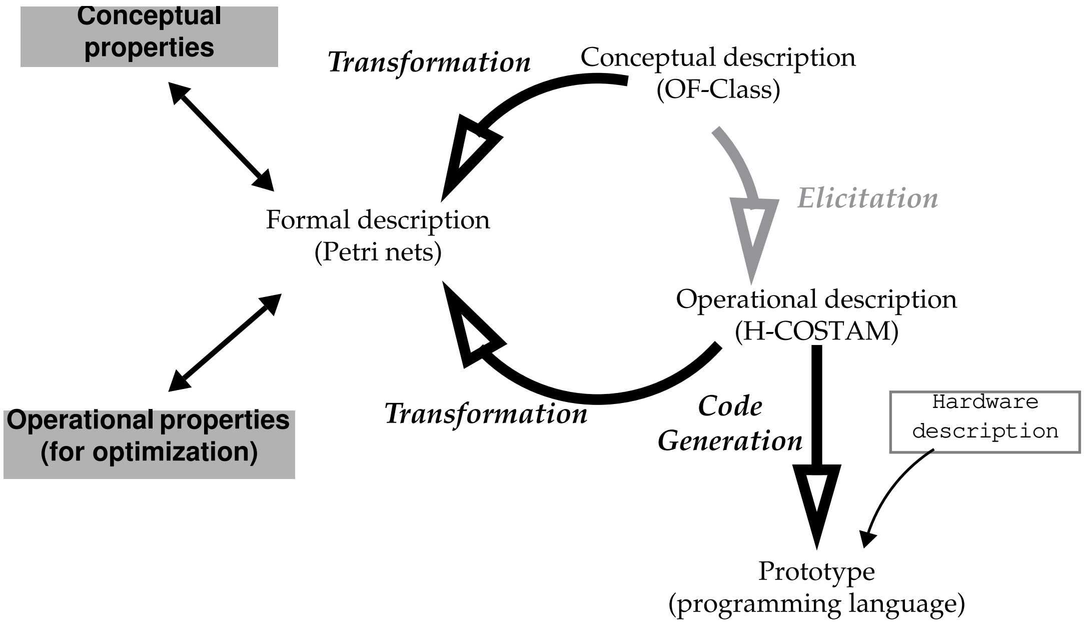


Figure 1 : Formalisms and Operations in our Methodology.

Four operations between these representations are characterized:

- Two ***transformations*** from respectively OF-Class and H-COSTAM into Petri nets enable the link with the formal representation. These transformations are different while they do preserve discrete properties. The result is a Petri net that express either functional relations (to get conceptual properties of the system) or an operational description (to extract operational properties that should lead to the optimization of the prototype);
- ***Code generation*** is performed from the operational description. It may compute and use operational properties to optimize code generation. In the context of distributed systems, this operation must produce both a compilable program and a location proposal. This location proposal is computed for a given hardware architecture description [El Kaïm 94].
- ***Elicitation*** of the system is the transformation of a conceptual description into an operational one. This step should not be automatic like the two other ones. It should be performed once when the system seems to be complete. It can be considered as a list of questions that gradually clarify all the points of the implementation.

3. The Formalisms

Our purpose is to provide a method that is dedicated to design distributed systems. The system we want to model and implement do communicate by means of messages exchanged between the application components (potentially located on discrete hosts).

As we mentioned before, our methodology involves three formalisms. This enables us to use the most appropriate formalism at each stage of the software life cycle. This is of interest because a scrupulous correspondence between these formalisms is maintained.

First we present the main characteristics of our models that are the same for each formalism (OF-Class, H-COSTAM and Petri Nets). Then, we detail OF-Class and H-COSTAM.

3.1. Main Characteristics of our Models

We consider that a model should be divided in two parts (Figure 2):

- The model of the ***execution environment***: it corresponds to already existing pieces of software for which no modification or evolution is possible. The operating system of a computer is a good example of execution environment;
- The model of the ***system*** itself that is divided in two discrete parts: parts that are reused from others applications (they already exist but can be modified if necessary) and specifically designed components.

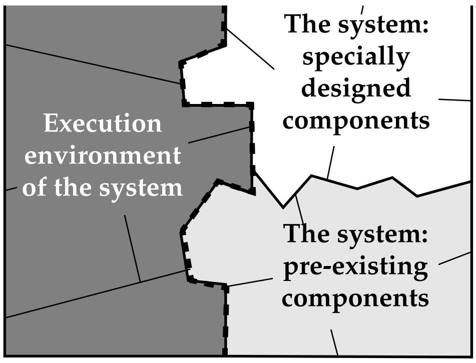


Figure 2 : Components of a Model.

Both conceptual and operational descriptions are hierarchically structured. They are divided in pages. Two granularities are considered:

- the ***macro-level*** contains pages that are more likely dedicated to the definition of relations between either elementary units or sub-systems,
- the ***micro-level*** that describes the behavior of an elementary entity.

In both cases the corresponding Petri net is flat. The hierarchy of the input description brings the structuration of the Petri net model. Such a structuration facilitates both the understanding and the verification of the model properties.

3.2. Scheduling of Formalisms

We first use OF-Class that fits the needs for validation and verification of a system obtained by composition of components. OF-Class is appropriate to all the needs pointed out at the conceptual level but is quite poor in the eyes of operational aspects necessary to enable protoyping.

This is why the Elicitation step (Figure 1) was introduced. A progressive addition of operational informations (how communication is performed, where strictly sequential parts of the system are, etc.) leads to the production of an equivalent H-COSTAM specification that is more likely oriented towards code generation of distributed systems that communicate by means of messages.

At both levels (conceptual with OF-Class and operational with H-COSTAM), an automatic transformation to Well Formed Petri Nets is possible. It enables a semantic analysis of the system.

3.3. The Conceptual Formalism: OF-Class

OF-Class is a template dedicated to the conceptual description of distributed systems. In one hand, it takes into account the main features of such systems pointed out in the Reference Model of Open Distributed Processing (RM-ODP) [ODP 95]. It provides a constrained interfacing mechanism in order to achieve most formal interactions. In another hand, it is formally associated with a modular Colored Petri Net (CPN) model [Diagne 96]. This modular CPN models the internal automaton describing the behavior of an OF-Class. It allows to undertake verification of both the structure and the dynamic of specifications.

OF-Class does not claim to support object-oriented activities such as analysis and design. However, it is as generic as possible to enable transformation from any OO model to undertake its verification.

3.3.1. Modeling a Distributed System with OF-Class

Specification and design of a distributed system consists of describing the components and their interactions. To evaluate the designed system, the expected properties of its components and their interaction must be expressed.

The execution environment of the system is considered to be a set of valid components accessed across bounded interfaces. The environment of a given component is made of both the execution environment (as defined in Section 3.1.) and the set of other components it is interacting with.

Properties that are expected from a distributed system are of two kinds:

- local properties on components concerning the managed resources or the offered services. These properties can be expressed as invariants - like in RM-ODP - on those resources or availability constraints on services. These properties are expressed on the micro-level description,
- global properties on the whole system such as safety of the interactions between its components (like deadlock and starvation free system), meaningfull reachable states (home state and reversibility). They can be expressed as assertions on the system evolution.

The model of the system may include observation facilities. They are information supplied to run the system and evaluate its behavior. They model the information necessary to simulate execution of the system.

A component has private resources to manage (it is the only one to manipulate it), offers to the environment services that handle those resources and an abstraction on its environment. To achieve these goals, there are two description levels.

3.3.2. The Micro-Level in OF-Class

The micro-level describes the local resources of a component and their possible transformations. A resource is an entity local to a component that can be accessed only through invocations of services offered by that component. Transformations of resources are done by means of elementary actions. These actions are grouped as sets called operations.

An operation is a set of actions performing a given semantical transformation on local resources. An operation can issue requests to the environment. It has input and output parameters, local variables and a return code. Two special operations handle the dynamic creation (constructor) and deletion (destructor) of instances.

A component may also trigger some sets of actions when reaching some meaningful states or when some events occur while interworking with the environment. These mechanisms are slightly different from operations because they cannot be invoked by consumers. Such actions are called ***triggers*** and have no equivalence in RM-ODP concepts. Triggers bear eventually preconditions, i.e. predicates on the resources values or input parameters specifying the state in which they are executed. Triggers can undertake interactions with the environment.

3.3.3. The Macro-Level in OF-Class

The macro-level describes the structural and dynamic links necessary for interaction of components in a distributed system. Structural links allow to compose discrete components in order to build a more complex one. Dynamic links are:

- the offered services exported by a component. An offered service is a set of operations with contractual contraints like precedence or access semantic (synchronous, rendez-vous, etc.). It is a coherent partial view on the behavior provided to the environment for access to the local resources,
- the required services from the environment show a given component the way it must use the services. They are provided by other components of the system.

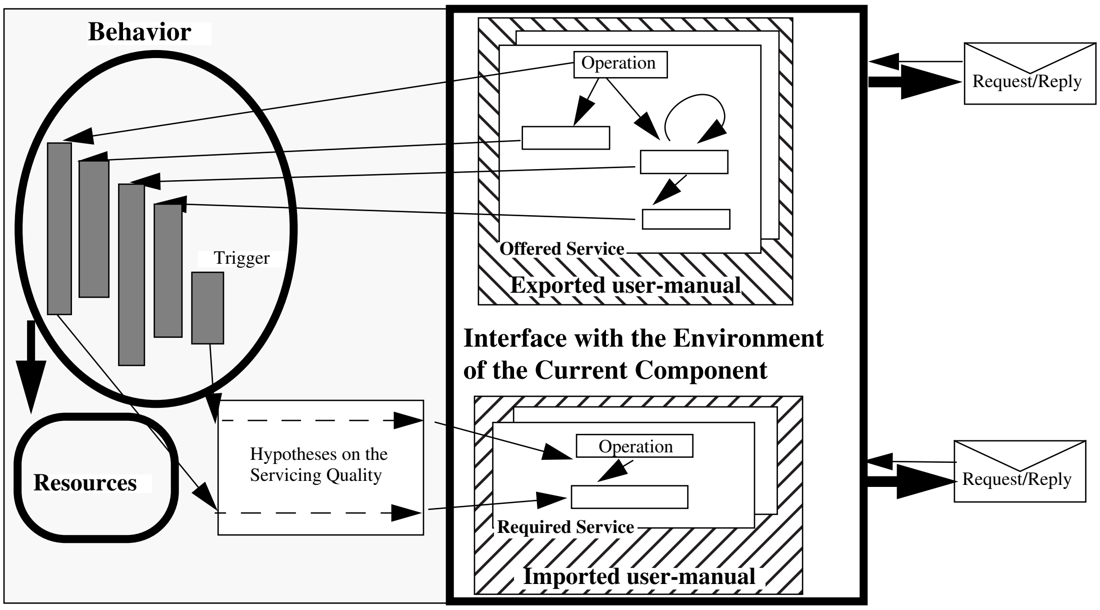


Figure 3 : The Basic Model of a Distributed System's Component.

3.4. The Operational Formalism: H-COSTAM

This section does not aim to give a full definition of H-COSTAM that can be found in [Kordon 95]. We just remind here the main principles of H-COSTAM and introduce new capabilities.

H-COSTAM is a formalism dedicated to the operational description of distributed systems that communicate by means of messages. Its features are the following:

- ***Hierarchy***: this is to support hierarchical descriptions and obtain a readable and structured specification. Translation into Petri nets preserves a «good» structure that should facilitate the computation of properties;
- ***A macro and a micro description***: entities of the system and their relations that are defined at the macro-level. Elementary components that are sequential state machines are described at the micro-level.
- ***Strongly typed communication mechanisms***: it is a communication model in the sense of CSP. However, communicating entities are sequential state machines or

subsystems instead of instructions;

- ***Integration of the system into its execution environment***: this is very valuable to enable implementation of applications that have interaction with an «outside world»;
- ***genericity management***: genericity (in the sense of the Ada language) is supported by H-COSTAM in order to facilitate the parameterization and the reuse of main components.

3.4.1. Structure of a H-COSTAM Specification

An H-COSTAM specification is composed of pages that belong either to the macro-level or the micro-level (Figure 4). A macro-level page describes the relation between entities that are either ***subsystems*** (and then a link to another macro-level page) or a ***process*** (and then a link to a micro-level page).

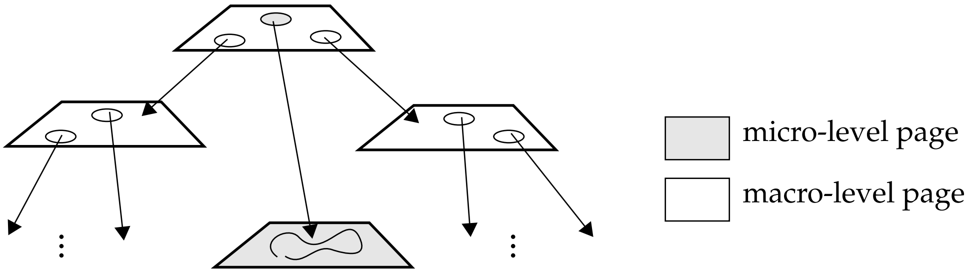


Figure 4 : Organization of a H-COSTAM Model.

Pages are tagged ***external*** if they describe the functional behavior of the execution environment. They are tagged ***internal*** if they correspond to the description of the system itself. If a macro page is tagged external, all its components should be tagged external as well.

External and internal components cannot be interpreted the same way:

- external components represent pieces of software the system will be linked to. The only necessary information is an H-COSTAM version of the user-manual introduced in OF-Class (Section 3.3.3.),
- internal components are parts of the system and must be described in detail. The code generator will use this information to produce programs that will be linked to the already existing code associated to the external components.

3.4.2. The Micro-Level in H-COSTAM

The micro-level describes one elementary component called ***process***. A process is interfaced with the outside world by means of ***media*** that define an interaction. Each medium is strongly typed by both the type of data that goes through and its behavior. There are three classes of media:

- ***multi rendez-vous*** corresponds to a synchronization between a set of N entities. The synchronization may be guarded by a condition and allows the exchange of information between the participants;
- ***links*** correspond to asynchronous transmissions that have some specific behavior. Three behaviors are defined: FIFO (the order of messages remains), LIFO (last message in is first out) or random (no order preserved). Links may be connected to an arbitrary number of entities. Each input entity may send a message that will be received by all the output entities.
- ***remote call*** relates client entities to one server entity and corresponds to a remote procedure call. On the client side, it appears like an atomic operation (equivalent to a synchronization). On server side, it behaves like two FIFO links, the first is an input that corresponds to the service invocation, the second is an output and

represents the end of the service;

- ***factories*** are entry points for special messages that enable dynamicity in the creation of sequential processes.

Media that interface a process are connected to an internal automaton that is expressed using a State/Transition model whose semantic is close to the one of Well Formed Petri Net. The internal automaton has to be a sequential state machine in the sense [Hack 74].

The automaton describes the static behavior of the elementary process. Each of them may be instanciated statically (a set of instances are predefined at the initialization of the system) of dynamically (when a special event comes from a constructor).

Each instance may have its own distinct context represented by a set of private variables. A restricted set of operations is available on these variables: Identity, successor, predecessor, product and restriction. Such a limited set is necessary because we want to maintain an equivalence with Petri nets. However, if more complex and specialized capabilities are needed, they can be introduced by means of external components.

Communication media may also be local to the entity. They are then potentially shared by all instances and cannot be accessed from outside.

3.4.3. The Macro-Level in H-COSTAM

The macro-level describes the relations between entities. So, involved objects in a macro-level page are either communication media (including constructors) or entities.

A communication medium is either the real one associated to entities interfaces or an interface with an upper level. Like in the micro-level, only interfaces media may be accessed out of the page. In the macro-level, constructors (local or interface with the upper level) cannot be defined if they are not connected to the interface of at least one contained unit.

An entity is either a process, defined in a micro-level page, or a subsystem, described in another macro-level page. In both case, media that define interfaces of contained entities are then associated to the real ones that perform the communications.

3.4.4. Genericity

Types are used to define the type of data that goes through media (both macro and micro-levels) or to declare variables in a process context (micro-level). Types may be defined :

- in the page where they are used;
- imported from an entity that is enclosed in the page.

It is also possible to define a generic entity : some types are then provided by the «upper level» (the unit that contains the generic entity). Such types are only known after some reduced characteristics :

- their name (for referencing purpose);
- some possible values. This is usefull to defined guards. Such values are then considered as constants that have to be defined when the entity is instanciated.

The instanciation in H-COSTAM works in a more restricted way than the one defined for languages like Ada. Values have to be associated to the generic parameters and each instanciated unit will be a distinct specialized copy.

3.5. Summary of Conceptual versus Operational Formalisms

Both OF-Class and H-COSTAM models deal with the main concepts underlining the specification and design of distributed systems. Thus, they do have the features that are necessary to build interacting components.

The table below summarizes these features. It may be considered as a basis for the definition of the shift from the conceptual description to the operational one.

As previously outlined, both OF-Class and H-COSTAM do have very similar capabilities. The main difference is found in the way interactions betwen entities are defined : H-COSTAM has no «user manual» capabilities that enable to verify if a component is used properly. On the contrary, communication media in H-COSTAM are closer to the implementation.

Features	OFClass	H-COSTAM
Hierarchy	yes.	yes
Component Interface	Description of the exported local views and the imported ones from the environment. Description of the structural links to other OFClasses (composition or refinement).	Descripition by means of communication media only. Constructors may be used to outline a dynamic instanciation of sequential processes.
Internal Structure of an Elementary Component	Description of local resources and their access-methods. Description of the processes (operations invocable from environment, local triggers and exceptions)	Description of a sequential state machine that is instanciated statically or dynamically. All the instances of the state machine mays share some local media
User Manual	Operations and the way they are invoked by the environment. The behavior is the sum of some of the internal structure and the expression of communication (see below).	None
Communication between Components	By means of service offer/request. The provider of a set of operations exports them with the constraints that might be observed by the consumers for their use (signature and precedence).	By means of media or constructors.
Types of Communication Supported	Synchronous RPC, Asynchronous RPC, Rendez-vous.	Links (FIFO, LIFO or random), multi-rendez-vous, remote calls (RPC) .
Partial and Global Reuse	Global reuse is supported through composition. Partial reuse is supported through refinement with enrichment or simplification.	Global reuse achieved throught composition, Generic entities (both sequential state machines or subsystems).

Table 1: OF-Class versus H-COSTAM

4. Operations in the Methodology

This section is dedicated to the description of all operations in our methodology. The operations we consider are the following :

- transformations into Petri nets (for both conceptual and operational levels),
- analysis and what information we get from it,
- elicitation of an OF-Class model.

More information about the code generation procedure may be found in [Kordon 95].

4.1. Principle of Translations

Modular aspects of Petri nets had been largely discussed in the literature as a response to the lack of compositionality which is the major weakness raised on that formalism [Bachatène 93].

The formal model associated to both the conceptual and operational descriptions is a modular colored Petri net that supports composition by means of channels [Souissi 90]. A channel is a set of two places per operation shared by its provider and the consumers. It allows to model synchronous and asynchronous communications. The rendez-vous semantic is achieved by a «serie transitions» fusion on duplicated channels [Berthelot 86].

The actions in the micro-level description are transformed into Petri net items using the method developped by Heiner [Heiner 92]. The dynamical links are transformed into channels.

A Petri net pattern replaces specific configurations like dynamical links in OF-Class or communication media in H-COSTAM. This pattern corresponds to a piece of «runtime» that handles specific semantical aspects of the formalism. These patterns may be parameterized.

For example, parameters of a FIFO link are its maximum capacity and the type of data that goes through. During the translation process, these information are used to build the appropriate instance of a pattern that models the behavior of a FIFO link.

4.2. Analysis and Exploitation of Results

Figure 5 shows the analysis cycle in our methodology. Analysis may be performed either locally (on one component only) or globally. So, the obtained properties are used to provide information either about the behavior of one component or the interactions between a set of components.

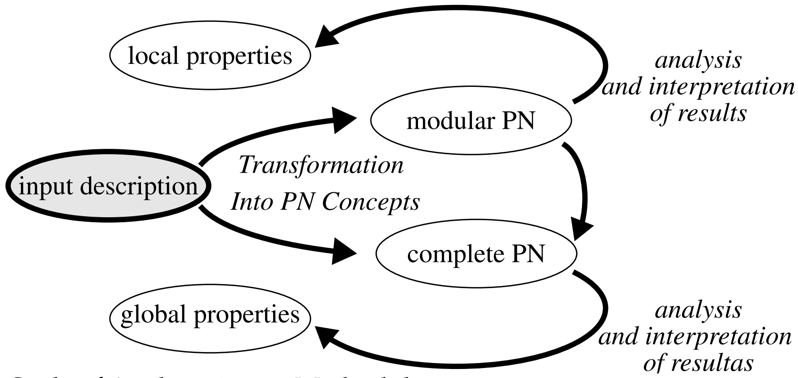


Figure 5 : The Cycle of Analysis in our Methodology.

Analysis has discrete goals according to the input of the transformation :

- if it is performed from the conceptual level, it aims at the verification of some properties of the system;
- if it is performed from the operational level, it aims at the computation of some properties that could lead to an optimization of the generated prototype.

For a given level, several transformations into Petri nets may be considered. In some case, information that bring «noise» to the analysis may be discarded, focusing on the information we are seeking. Let us provide an example. At the operational level, while checking potential replicability of communication media, their behavior is not required. On the contrary, to evaluate the bound of a link for which no maximum capacity is provided, the refined behavior of cooperating state machines is not necessary.

At the conceptual level, analysis is performed for validation purpose. Once the isolated components successfully verified, they can be simplified using the reduction rules on colored Petri nets [Haddad 88]. We then build the global net representing the interactions between components using information from both the modular nets and the conceptual or operational description (macro-level).

The major benefit we get from this formal description is the two-level verification. The mode of composition we use preserves two kinds of interesting properties:

- liveness of transitions (corresponding to the services availability constraints),
- p-flows (corresponding to the invariants on local resources).

So, these two properties may be verified locally on a component. This is much easier because the net is quite small.

The safety of interactions is evaluated on the global net. Its size had been decreased by the reduction rules. So classical tools on Petri nets can be applied without the drawback associated with the net size.

At the operational level, analysis is performed for code optimization. Local analysis mainly aims to compute bounds of local media or to agglomerate sequences of actions. Global analysis may provide information about the potential partitioning (parallelism, pipe-line, etc.) of the application, bounds of shared media, replicability of processes and communication media.

4.3. The Elicitation Operation

The shift from OF-Class to H-COSTAM is the Elicitation operation presented in Figure 1. It is a semi-formal transformation based on the correspondence established in Section 3.5.

Such an operation is too complex to be operated automatically : the system designer must provide some information that cannot be deduced from the conceptual description. Different problems are raised in this operation :

- the equivalence between OF-Class and H-COSTAM must be established. This is not complex for the macro-level description because the communication mechanisms are very similar. However, the micro-level description is quite difficult : triggers and some specific configuration have to be interpreted;
- some information has to be discarded. This is the case for the «user manual» that has no sense in H-COSTAM (while no verification is performed at this level).

In Table 2, we summarize some main rules for the elicitation. Some rules are fully automatic. Some others need some basic information from the designer. There are also some transformations that may only be guided : the designer is the only one who can handle the semantic of his system and then solve the problem.

However, if this operation cannot be completely automated, a tool should be relevant to perform the elicitation. Such a tool could run like a sort of «question game». Each question is raised at the right time considering the configuration of the studied part of the model.

	OFClass	**Equivalent in H-COSTAM**	**Auto.**
macro-level	Rendez-vous	Multi-rendez-vous (binary)	yes
	Synchronous RPC	RPC	yes
	Asynchronous RPC	Two links, one for the call, one for the return value and output parameters. If queries remain sorted, links will be FIFO. Otherwise, links will be random. The type of information that goes throug these media is deduced from the parameters of the service.	yes[a]
	User manual of a component	Discarded (no verification is performed at the operational level while the specification is supposed to be conceptually safe)	yes

Table 2: Some Translation Rules from OF-Class to H-COSTAM

	OFClass	Equivalent in H-COSTAM	Auto.
micro-level	Internal automaton	If it is a sequential automaton, the unit is one process. Otherwise, it must be divided into a H-COSTAM subsystem (a subsystem that contain several processes)	semi[b]
	Triggers	One process to handle internal events plus one process per trigger. The process that manages internal events dynamically creates instances of triggers.	semi[c]
	Services	One H-COSTAM process. It is connected by mean of a factory to a process that manages creation and destruction of OF-Class instances. More information are provided in the example.	yes
	Variables, variable manipulations and local resources	Direct translation (semantics are very close)	yes

Table 2: Some Translation Rules from OF-Class to H-COSTAM

a. A systematic question may be asked to the system designer about the type of expected communication («does messages have to remain sorted or not»).
b. It is sometimes possible to help the system designer and to propose him a set of possible decompositions.
c. There is a problem when the trigger is not strictly sequential (use of forks). However, is it sometimes possible to propose a decomposition.

If several implementation strategies are investigated, a summary of previous studied strategies could be of interest. So comparison of strategies for discrete implementations may be done.

5. Application to an Example

For sake of place, we give here an example that is only pertinent to illustrate facilities of our representations and the elicitation operation. The example is not pertinent to demonstrate verification capabilities because the system is too basic to exhibit interesting properties. In [Diagne 96], a complex example that emphasizes the validation and verification at the conceptual level is presented.

The example models the collaboration between basic producers and consumers. It contains three components:

- one «init_master» which creates a set of couple <producer, consumer> that exchange messages,
- the «prod_class» which models production of messages,
- the «cons_class» which models consumption of messages.

Each couple of producer and consumer first synchronizes after the end of initialization. Then, they communicate via a pipe (ordered communication medium). Each producer only dialogs with one consumer. The two entities of a couple share the same identifier.

The OF-Class Model

The listing below contains the macro-level description of the example. The root definition of the system is the one of the `init_master` unit. This unit corresponds to the entry point of the system. It is responsible of the dynamic instanciation of two enclosed subsystems : `prod_class` and `cons_class`. It also defines some global types and constants that will be visible in the enclosed units.

The user-manual of the offered service `comm` in `prod_class` specifies that operation `init_ok` should be performed once first. Then, it is possible to run operation get_msg as many as necessary. The required service of `cons_class` is defined according to the one offered by `prod_class`. It is this way that the dynamical links between enclosed components are specified.

```
init_master isa ofclass
macro-level
   composition { prod_class , cons_class } /* definition of enclosed units */
   types {  /* exported types toward the enclosed units *
            ident is range 1 .. 20 ,
            msg is range 1 .. 100 ,
            /* propagation to enclosed units */
            msg in cons_class is msg,
            id in cons_class is ident,
            msg in prod_class is msg
            id in prod_class is ident
         }
   constants {     /* exported constants toward the enclosed units */
                  START in msg value msg'first ,
                  STOP in msg value msg'last
               }
/* required specifications of enclosed units */
prod_class isa ofclass
macro-level
   interface
      exports    { /* offered services to the environment */
                  comm { init_ok , get_msg* }
                                 invocation-mode { synch,asynch,rendez-vous }
               }
cons_class isa ofclass
macro-level
   interface
      imports
      {
         from prod_class comm
         {
            init_ok invocation-mode rendez-vous
            get_msg invocation-mode synch
                        default-return continue
         }
     }
```

The listing below contains the micro-level description of the `prod_class` unit. It defines the actions performed in each operation referenced in the offered service of the macro-level. The local resource `cur-msg` of the unit is also specified. It is duplicated, i.e. each created instance manages one copy of the resource (keyword duplicated).

Function `oself` represents the current identifier of an instanciation. This is an attribute automatically associated in OF-Class. The type of this function is deduced from the constructor of `prod_class`.

```
prod_class isa ofclass
micro-level /* private part of the unit */
   /* local resources */
   resources { type msg m duplicated}
   process
      constructor {parameters id prod in}
         begin
            prod.m = start;
            return;
         end
      destructor
         precondition (oself.m = stop)
         begin
            delete;
            return;
         end
      init_ok {parameters id cons in}
         precondition (cons = oself)
         begin
            return;
         end
      get_msg
         returns int
```

```
begin
   m++= cur_msg;;
   return m;
end
```

The Elicitation to Get the H-COSTAM Model

The elicitation should start at the macro-level. Each unit is investigated but we only present the procedure for `prod_class`.

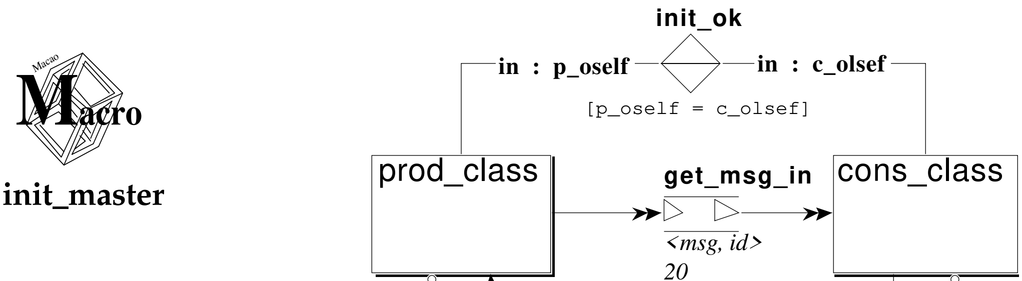
We characterize interfaces and units at the macro-level. Each OF-Class corresponds to one H-COSTAM unit (later, we will determine if it is a process or a subsystem). init_master is the root definition of the system. It is obviously a subsystem because it contains two units : `prod_class` and `cons_class`. All units are tagged «internal» because they do not describe the execution environment.

`init_master` does not export any services or entry point. It only requires the constructors of the enclosed units.

We first have to characterize the interfaces. Operations are used to deduce the corresponding set of media. Figure 6 describes the unit `init_master`. It contains three H-COSTAM entities : the two deduced form the OF-Class description plus the one that corresponds to the behavior of init_master (init_master_autom).

According to Table 2 :

- operation `init_ok` is transformed into a multi-rendez-vous. The precondition is built from the one of the `init_ok` operation;
- operation `get_msg` is transformed in two pipes. We consider here the designer says that messages remain sorted and that there is a maximum capacity of 20 pending queries. The first pipe is typed according to the parameters of the operation. The second only signals that the service is completed;
- two factories are related to `init_master_autom` because its micro-description (not presented here) is connected to OF-Class constructors of `prod_class` and `cons_class`.




```
/* declarations */
type ident is range 1 .. 20;
type msg is range 1 .. 100;
constant start : msg := msg'first;
constant stop : msg := msg'las;
/* instanciation */
in prod_class (id => ident,
               msg => msg,
               start => start,
               stop => stop);
in cons_class (id => ident,
               m => msg,
               start => start,
               stop => stop);
```

Figure 6 : The H-COSTAM Macro-Level Description of `init_master`.

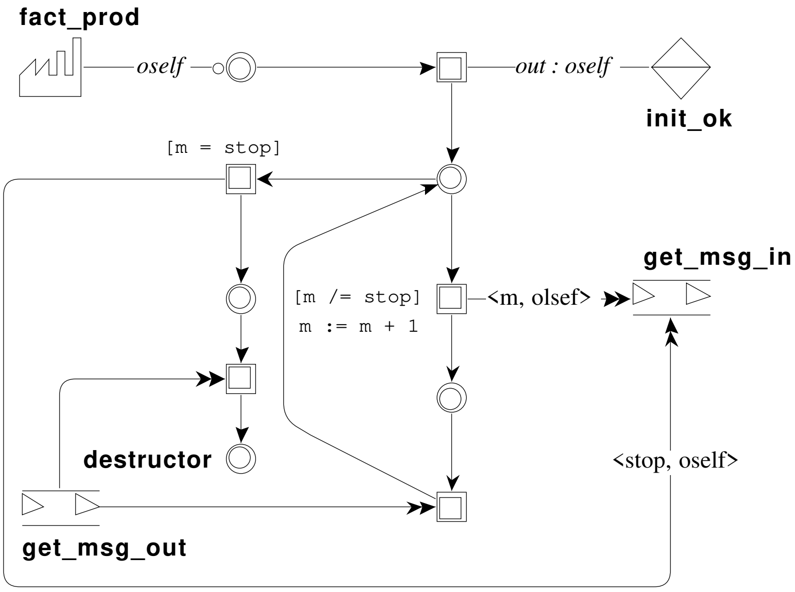
Types are directly deduced from the original definition in OF-Class. Instanciations are declared according to propagation in the OF-Class macro-page. Constants are systematically propagated when the corresponding type is.

We now focus on the micro-level transformation of the `prod_class` micro-level description. Resource `m` in the OF-Class description is duplicated so, it becomes a part of the process context. The `oself` function corresponds to a part of the context that contains the identifier of each instance. Its value is provided by the factory associated to the constructor.

There is only one service and no trigger. So, the `prod_class` unit is a sequential process. Otherwise, it would be a subsystem that contains :

- one process (service_handler) that accepts services, creates a process dedicated to this service and waits for the result that is returned,
- one process per service that is dynamically created by the service_handler;
- one process (trigger_handler) to manage trigger preconditions;
- one process per trigger that is dynamically instanciated when necessary by the trigger_handler.

The internal state machine associated to process dedicated to a service is deduced from its user manual. Here `init_ok` is performed and then, `get_msg` may be performed as many as necessary. Once the stop message is send, the destrictor is triggered. Here, it corresponds to the last state of the process. The operation `get_msg` is transformed into two sequences guarded by preconditions : this corresponds to the test on the `stop` message.



Micro
prod_class

```
generic
   types id, msg imported from up;
   constants start, stop in msg imported from up;
context
   m in msg := start;
   oself in id;
```

Figure 7 : The H-COSTAM Micro-level Description of `prod_class`.

Generic parameters are deduced from the types propagation defined in the corresponding OF-Class macro-page.

6. Conclusion

In this paper, we have proposed a prototyping methodology to build distributed systems from the conceptual level (specification) to the operational level (implementation). The multi-formalisms approach allows one to have a suitable model for each level.

Thus, a system designer may focus on the properties that are appropriate for a given level.

The conceptual level is dedicated to the explicit definition and verification of safety properties. The operational level deals more with implicit properties addressing the optimization and location of the generated prototype.

We do take care of the traceability between levels. The elicitation operation enables a coherent transformation :

- the semantic of the system remains;
- the properties proved at the conceptual level are preserved (no need for a verification at the operational level).

However, to manage large size specification and to evaluate the pertinence of our methodology, an implementation should be performed. Some experimentations have been done within the context of the universitary MARS project [MARS 94] and the industrial EURÊKA IRENA project [IRENA 95]. We are currently working on the implementation of a small CASE tool that would manage the complete methodology.

7. References

[Bachatène 93] H. Bachatène & J.M. Couvreur, "A Reference Model for Modular Colored Petri Nets", IEEE/System, Man and Cybernetics International Conference, Le Touquet, France, October 1993.

[Berthelot 86] G. Berthelot, "Transformations and Decompositions of Nets", Proceedings of Advances in Petri Nets 1986, Part I, West Germany, Bad Honnef, September 1986, Edited by W. Brauer, W. Reisig and G. Rozenberg, LNCS vol. 254, PP. 359-376.

[Buchholz 94] P. Buchholz, "Hierarchical High Level Petri Nets for Complex System Analysis", Proceedings 15th Int. Conf. on Applications and Theory of Petri Nets, Zaragoza, Spain, June 1994, LNCS vol. 815 PP. 119-138.

[Chiola 91] G. Chiola, C. Dutheillet, G. Franceschini & S. Haddad, "On Well-Formed Coloured Nets and their Symbolic Reachability Graph", High Level Petri Nets. Theory and Application. Edited by K. Jensen G.Rozenberg, Springer Verlag 1991.

[Diagne 96] A. Diagne & P. Estraillier, "Formal Specification and Design of Distributed Systems", Technical Report IBP-MASI #96-01, University Pierre & Marie Curie, France, January 1996.

[El Kaim 94] W. El Kaim & F. Kordon, "An Integrated Framework for Rapid System Prototyping And Automatic Code Distribution", Proceedings of the 5th "International Workshop on Rapid System Prototyping", N.Kanopoulos Ed, IEEE comp Soc Press, Grenoble, June 1994.

[Hack 74] M. Hack, "Extended State-Machine Allocatable Nets (ESMA), an Extension of Free Choice Petri Net Results", MIT, project MAC, Computation Structures Group, Memo 78-1, 1974.

[Haddad 88] 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 vol. 424, PP. 209-235.

[Heiner 92] M. Heiner, "Petri Net Based Software Validation, Prospects and Limitations." Technical Report TR92-022, GMD/First at Berlin Technical University, Germany, March 1992.

[IRENA 95] IRENA Consortium, "Final deliverable" of the IRENA Project, June 1995.

[Jensen 92] K. Jensen, "Coloured Petri Nets. Basic Concepts, Analysis Method and Practical Use (vol 1)", EATC Monographs on Theoretical Computer Science, Springer Verlag 1992.

[Kordon 95] F. Kordon & W.El Kaim, "H-COSTAM : a Hierarchical Communicating State-machine Model for Generic Prototyping", Proceedings of the 6th "International

Workshop on Rapid System Prototyping", N.Kanopoulos Ed, IEEE comp Soc Press 95CS8078, pp 131-138, Triangle Park Institute, June 1995.

[Lakos 95] C.A. Lakos, "From Colored Petri Nets to Object Petri Nets", Proceedings of the 16th International Conference on Application and Theory of Petri Nets 1995, Torino, Italy, June 1995, LNCS vol. 935, PP 278-297.

[MARS 94] MARS-Team, "The CPN-AMI Environment Version 1.3", MASI Laboratory, ftp://ftp.ibp.fr/ibp/softs/masi/ami/documentation, June 1994.

[ODP 95] "The Reference Model of Open Distributed Programming, Overview, and Guide to Use" Draft ITU-T, Recommendation X.901

[Souissi 90] Y. Souissi, "On liveness preservation by Composition of Nets via a Set of Places.", Proceedings of the 11th International Conference on Application and Theory of Petri Nets, Paris, France, June 1990.