From Formal Specification to Optimized Implementation of Distributed Systems : A Multi-Formalisms Approach

Alioune Diagne & Fabrice Kordon,
Laboratoire d'Informatique de Paris 6
Université P.&M. Curie
4 place Jussieu, 75252 Paris Cedex 05, France
E-mail: Alioune.Diagne@lip6.fr, Fabrice.Kordon@lip6.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 and liveness 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 information that are not relevant for code generation. The paper presents the general methodology and proposes an application to a simple example.

Keywords

Formal Specification, Petri Nets, Validation, Verification, Design Traceability, Prototyping, Distributed Systems.

1. Introduction

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

Such systems are made of cooparating components that should interact while managing local resources. Interaction must not corrupt the integrity of involved components and has to be safe (i.e. free of faults like deadlocks or starvation). 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.

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

Thus, formal description techniques represent a solution. While a designer describes his system, he/she states the expected properties and then verifies them. Formalisms such as Petri nets have been systematically improved and Colored Petri nets [Jensen 92] are now widely used in research for modeling distributed systems. Their use in a potentially industrial context has also began (like in the PARSE Software engineering environment for the production of Distributed systems [Jelly 96]).

However, the main drawback of Petri nets is their poor structuring facilities. They are thus difficult to use for large scale systems. Recent 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 95a].

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 enables the construction and analysis of large distributed systems without having to deal with too much complexity at the design level. Real-time systems are not specifically addressed by our method because the Petri nets class we use do not handle time.

The paper is organized as follows. 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 traceability 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 level before a conclusion.

2. The Methodology

Building distributed systems enforces a couple of needs :

1) formal specification techniques allowing verification against quality criteria. The quality expected from the system should be stated and verified. In this work, we are mainly interested in safety and liveness properties [Valmari 94]. Safety properties state *what should **always** or **never** happen in the system*. Liveness properties state *what is a correct behavior for the system.* Violation of safety properties leads to hazardous components while violation of liveness properties leads to deadlocking;
2) optimized implementation allowing traceability of the confidence level attained in a specification. Code generation is a valuable way to deduce implementation from specification without any drift. Optimization techniques make the code efficient and can be used as long as we maintain the traceability of the quality.

To complete these needs, the use of ***Formal Description Techniques*** (such as Petri nets, algebraic specification, etc.) is of interest. They enable the computation of properties (structural, behavioral), without having to consider the entire state space of the system and also support model checking. However, due to the underlying mathematical theory, the description of the system is more complex. On an other hand, it is easier to handle system specification using ***high-level description techniques*** (for example Object oriented based modeling). Our approach tries to mix both type of description and to maintain traceability between them. Its goal is clearly to take benefits from Petri nets without having to manipulate them.

Moreover, the two discrete phases of the design procedure have been identified : Conception and implementation. When a designer builds a system, he/she first deals with its architecture. He/she then has to take care of ***conceptual*** aspects like the functions the system has to provide or the interoperability of its software/hardware components. Once the conceptual description has reached a satisfactory maturity, the designer may then think about ***operational*** or implementation related details.

Code generation is more efficient if it states from the operational description. It can lead to an executable prototype; this is then quite different from an animation/simulation approach. On the contrary, the conceptual description can only be animated for debug purposes.

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 assumptions, 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.

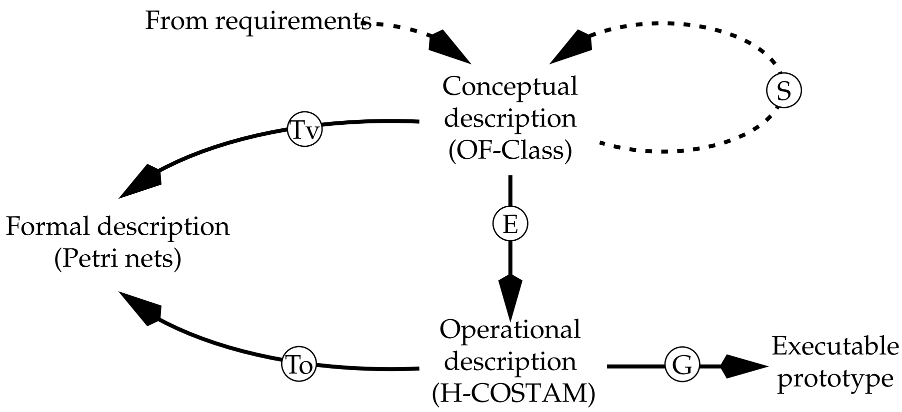

Figure 1 : Formalisms and Operations in our Methodology.

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 animated and transformed into a 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 enables ***code generation***.

Three operations between these representations are characterized:

- Two ***transformations*** from respectively OF-Class into Petri nets (Tv in Figure 1) and H-COSTAM into Petri nets (To in Figure 1) 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 implementation characteristics that should lead to the optimization of the prototype). Transformation Tv aims to provide information about the safety and liveness of the system while transformation To focuses on the computation of characteristics for optimization purpose;
- ***Elicitation*** of the system is the transformation of a conceptual description into an operational one (E in Figure 1). This step should not be automatic like the two other ones. It should be performed once when the system attains a satisfactory level of confidenceregard of properties. It can be considered as a list of questions that gradually clarify all the points of the implementation.
- ***Code generation*** is performed from the operational description (G in Figure 1). 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].

3. The Formalisms

Our purpose is to provide a method that is dedicated to design distributed systems. The systems we want to model and implement 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 the use of 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 ongoing works that go in similar direction than ours. Then, we present the main characteristics of our formalisms and we detail OF-Class and H-COSTAM.

3.1. A view on some similar approaches

The idea to associate a formal method with higher level formalisms is being investigated since the late 80's. Some first studies, like [Di Giovanni 90] or [Paludetto 91] have proposed an association of Petri Nets with HOOD. HOOD brings the structuring capability and Petri Nets its verification strength. Such methodologies mainly focus on the structuration of Petri nets to ensure the readability of large size specification. Some prototyping characteristics are enabled in [Paludetto 91] while it is possible to deduce some implementation directives from the specification.

Some later works, like [Bruno 94] and [Lakos 95b] propose a closer association between high level concepts like instanciation, encapsulation and even inheritance. These studies basically focus on the executability of the specification. The model can be animated and executed. This point is important but the high-level concepts introduced may disable major verification capabilities.

Among the couple of approaches depicted so far, the first one requires a system designer to have a valuable knowledge on both the high level formalism and the formal description technique while the second one generally lacks in verification capabilities. So, a third approach has to be considered : Petri net encapsulation. The idea is to completely hide the complexity of Petri nets and the underlying theory to system designers.

This kind of solution is investigated in the PEP environment [Grahlmann 96] designed at Hildesheim University. A language : B(PN)2 (Basic Petri Net programming) [Fleinschhack 97], is used to model systems. It is an imperative language including procedures in its last version. Petri nets are synthesized from this high level formalism in order to enable formal verification of the system. The environment also has enhanced simulation capabilities to execute the specification. Thus, it is a prototyping approach, more likely oriented to simulation and verification than to code generation.

In the same line, but more likely adapted to Software/Hardware systems, the SEA environment (System Engineering and Animation), designed at Paderborn University, also takes aim at providing a similar encapsulation [Heitbreder 97]. The input specification level is therefore a hierarchical graphical representation whose semantics is an extension of the one of Petri Nets [Kleinjohann 96]. This high level language appears to be a good media for the elaboration of heterogeneous systems (Codesign). The kernel of the SEA environment is a Predicate/Transition net simulator that is in charge of the execution and animation of the upper specification.

Our approach also encapsulates the Petri net model (and its theory). However, by providing two discrete description levels, our goal is to reasonably cover both the validation/ verification aspect and the production of an independent application : i.e. which may run out of the environment that produced it. Our formalisms have been designed especially for distributed cooperative software systems.

3.2. Main Characteristics of the high level Formalisms

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

- The sub-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. This submodel is tagged ***external***;
- The sub-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. These submodels are tagged ***internal***.

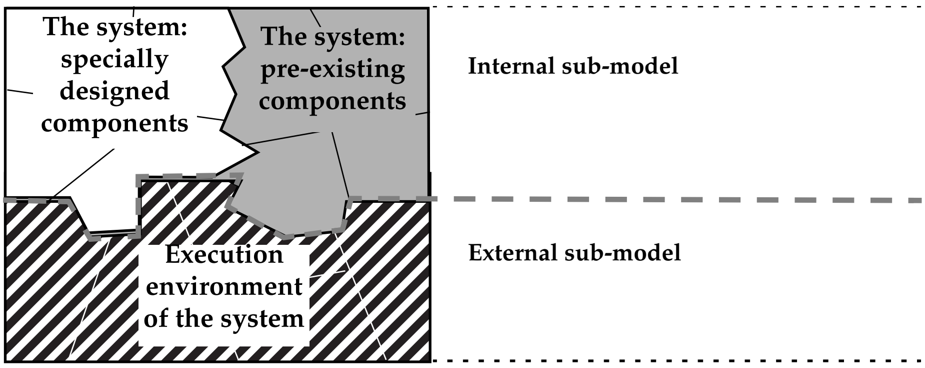

Figure 2 : Components of a Model.

Both conceptual and operational descriptions are hierarchically divided. The ***macro-level*** describes the relation between components. The ***micro-level*** contains the description of an elementary component.

3.3. Scheduling the 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 for the operational aspects necessary to enable prototyping.

This is why the Elicitation step (Figure 1) was introduced. A progressive addition of operational information (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 enables a semantics analysis of the system.

3.4. 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 conception. However, it is as generic as possible to enable transformation from any object-oriented model to undertake its verification.

3.4.1. Modeling a Distributed System with OF-Class

Specification and design of a distributed system consist 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 3.2.) 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), meaningful reachable states (like home state and reversibility). They can be expressed as Linear temporal Logic 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) and an abstraction on its environment. It offers to the environment services that handle those resources. To achieve these goals, there are two description levels.

3.4.2. 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 (see next section). Transformations of resources are done by means of elementary actions. These actions are grouped to build 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. The interactions by means of operations are subject to fault propagation. We alleviate this by a notion of ***exceptions***. Each component can state expectations on the results coming from its environment and execute exceptional behavior when they are not met (think about testing the results of Unix system calls).

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 from the environment. Such actions are called ***triggers*** which 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 are executed automatically and can undertake interactions with the environment.

3.4.3. The Macro-Level in OF-Class

The macro-level describes the structural and dynamic links necessary for interaction of components in an OF-Class specification (Figure 3). 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 constraints like precedence or access semantics (asynchronous, rendez-vous, etc.). It is a coherent partial view on the behavior provided to the environment for access to the local resources. The set of offered services is the exported usage pattern of the component;
- 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.

Components are functionally assembled by using exported and imported usage pattern. The semantics of interactions is therefore attached to these usage patterns. An exported service specifies a communication mechanism (synchronous, asynchronous or client driven). The available operation behaviors are the following :

- ***rendez-vous*** operations which model hard synchronization between components,
- ***synchronous RPC*** operations which model blocking requests,
- ***asynchronous RPC*** operations which model non-blocking requests.

A component may have expectations on the service quality provided by other components. These expectations must be used to check results provided when an interaction occurs. This is important because, associated with the notion of external components, this notion allows us to integrate already existing pieces of software in a formally validation design approach.

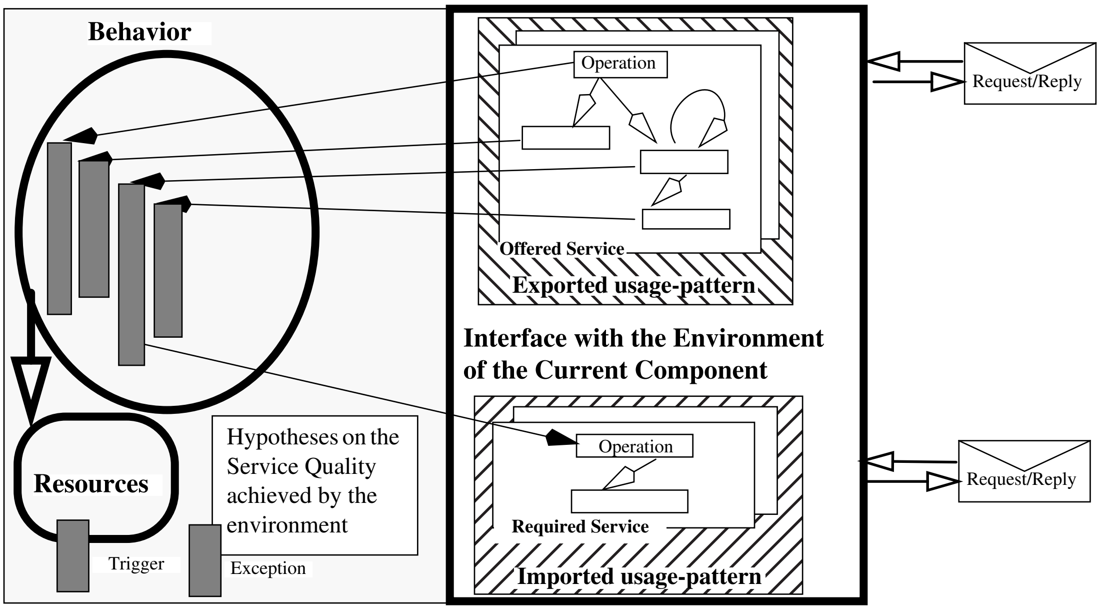

Figure 3 : The Basic Model of an OF-Class Component.

3.5. 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;
- ***A macro and a micro description***: entities of the system and their relations 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.

3.5.1. 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 presented in Section 3.5.2.).

According to the rules identified in Section 3.1, pages are tagged ***external*** if they describe the functional behavior of the execution environment or ***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 in the same way:

- external components represent pieces of software the system will be linked with. The only necessary information is an H-COSTAM version of the usage-manual introduced in OF-Class (Section 3.4.3.),
- 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 with the already existing code associated to the external components.

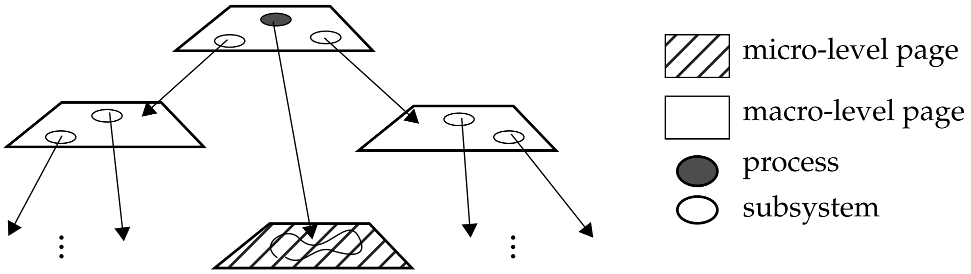

Figure 4 : Organization of a H-COSTAM Model.

3.5.2. 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 go through and its behavior. There are four 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 is preserved), 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 procedure call*** relates client entities to one server entity. On the client side, it appears like an atomic operation (equivalent to a synchronization). On the server side, it behaves like two FIFO links, the first is an input that corresponds to the service invocation, the second is an output that corresponds to the service termination;
- ***factories*** are entry points for special messages that enable dynamicity in the creation of sequential processes.

Media interfacing a process are connected to an internal automaton 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 of [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) or dynamically (when a special event comes from a factory).

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.

3.5.3. 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 factory) or entities.

A communication medium is either the real one associated with 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, factories (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 with the real ones that perform the communications.

3.5.4. Typing and Genericity

Types are used to define the format of data that go 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 contains 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 useful to define 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 with the generic parameters and each instanciated unit will be a distinct customized copy.

3.6. 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 common basic features that are necessary to build interacting components.

Table 1 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 between entities are defined : H-COSTAM has no «usage-manual» capabilities that enable to verify if a component is properly used. On the contrary, communication media in H-COSTAM are closer to the implementation.

Features	OF-Class	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 OF-Classes (composition or refinement).	Description 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)	Description of a sequential state machine that is instanciated statically or dynamically. All the instances of the state machine may share some local media
Usage-Manual	Operations and the way they are invoked by the environment. Interoperability constraints at specification level.	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) .

Table 1: OF-Class versus H-COSTAM.

Features	OF-Class	H-COSTAM
Partial and Global Reuse	Global reuse is supported through composition. Partial reuse is supported through refinement with enrichment or simplification.	Global reuse achieved through 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 into an H-COSTAM model,
- code generation from an H-COSTAM model.

4.1. Principle of Transformations

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 with 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 developed by Heiner [Heiner 92]. Dynamic 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 go through. During the transformation 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 about the interactions between a set of components.

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

- when it is performed from the conceptual level, it aims at the verification of some properties of the system;
- when 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.

At the conceptual level, a modular colored Petri net is obtained from each component. These modular nets can be analyzed using structural invariants techniques (arrow 2 in Figure 5) to check for flows for instance. Such flows can give information about the way the resources in the module evolve. The nets can be composed by place fusion to have a net modeling the whole system.

Modular nets can have their interface places overloaded with some abstraction of the environment and then a reachability graph is computed (arrow 3 in Figure 5). Such a graph supports verification of properties related to safety and/or liveness (arrow 5 in Figure 5)

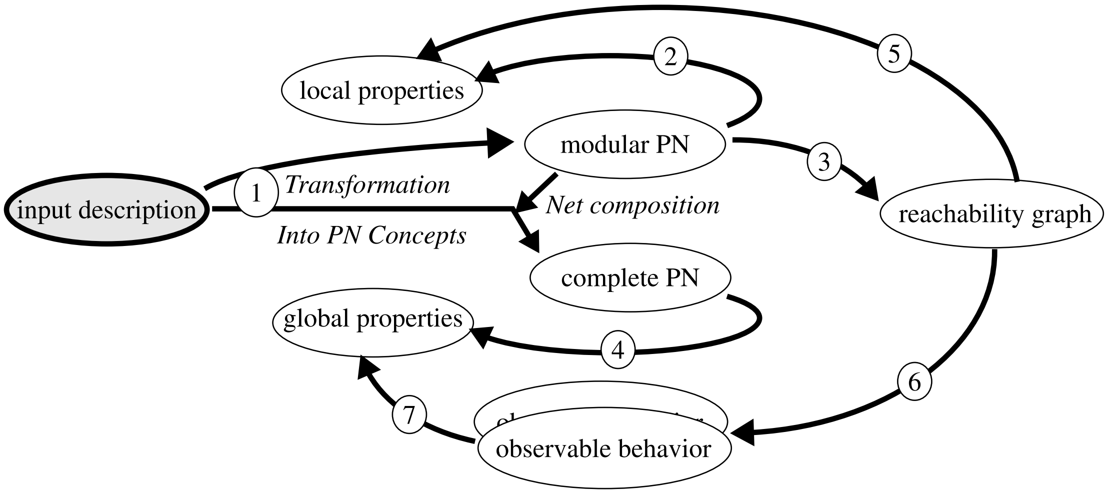

Figure 5 : The Cycle of Analysis in our Methodology.

At the operational level, we focus more on the production of complete Petri nets that are the composition of submodels that represent the behavior of H-COSTAM processes (communication by means of channel places). We are mainly interested in the communication aspects of the system to be prototyped. So, the behavior of processes may be some times reduced to a single transition or a more complex submodel if the process has some characteristics (for example loops) that may interfere with communication aspects. On such models, the properties we exploit so far are boundness and structural P-invariants. We also study how the net can be partially unfolded according to discrete color domains in order to detect symmetries from which replicability of software components can be deduced.

For both levels, several transformations into Petri nets may be considered. It is useful, to ease the verification process, to discard information that brings «noise» (i.e. is not relevant). For example, the behavior of communication media is not required when checking their potential replicability.

4.2.1. Conceptual Level

At the conceptual level, analysis is performed for verification purpose. We focus on safety and reliability properties. The analysis is based on the following principles:

- for each OF-CPN, we compute a version of *Chaos-Free Failures Divergence model* (CFFD-model) [Valmari 94]. This model is a reduction of the reachability graph of a given component to observable actions (i.e. performing interactions with the environment), divergent actions (i.e. that make the components enter endless loop), deadlocking states and interactions faults (arrow 6 in Figure 5). For the purpose of the CFFD-model computation, we overload some interface places with the information that can be expected from the environment. Such a model gives a good point of view on the quality of an isolated component,
- Once all the CFFD-models computed, we can deduce from them smaller Petri nets which are equivalents to the components for the observable actions. Such nets can be composed with the whole net of a given component as an abstraction of its environment. We can then undertake further evaluation on such an extended component. The validation step is done either for components we found faulty on the previous step or for components on which we want state-oriented information. The CFDD models can be synchronized to check for deadlocks in the whole system coming from interactions between components (arrow 7 in Figure 5).

The CFFD-model computation allows one to distinguish actions modifying the interfaces from actions that do not. We abstract those last actions considered as unobservable actions. Divergences need to be analyzed precisely because they can lead a component to commit interaction faults.

The major benefit we get from this formal description is the two-level verification. The CFFD-models give a first level of verification which can be refined at the following step. Observation equivalence based on the CFFD-model allows to reduce the abstraction of the environment in order to consider further state-oriented information for a given component.

4.2.2. Operational Level

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. An example of the involved techniques will be illustrated in Section 5.4.

Moreover, H-COSTAM can also takes benefits from operational research studies about distributed location of tasks [Norman 93]. Most of these studies consider a simple application structure composed of tasks that consume inputs to produce outputs (sort of batch-like jobs). Tasks are then related using precedence constraints. Communication are also considered in some algorithms.

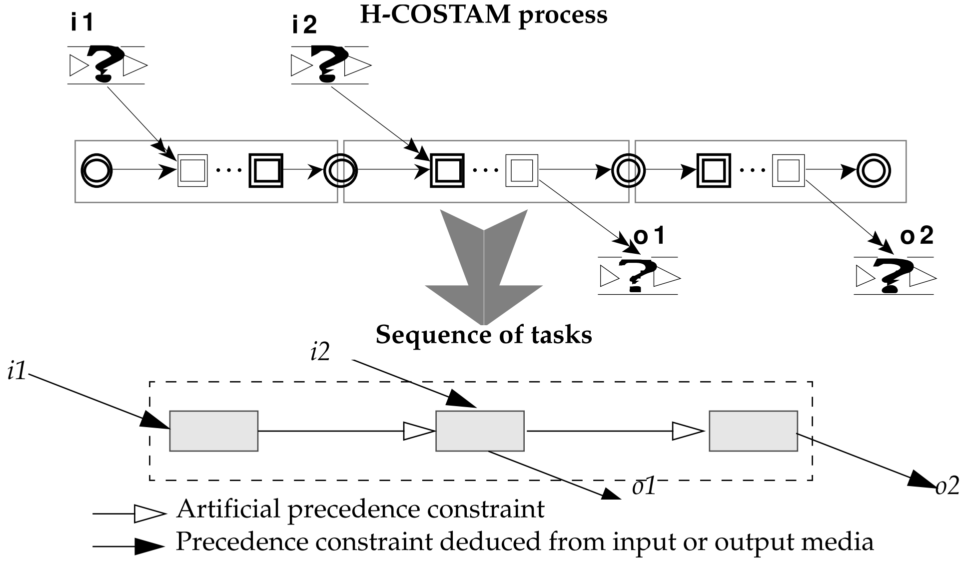

Figure 6 : From H-COSTAM sequences to the task model used in operational research.

The granularity of H-COSTAM is higher than the one of such application structures but it can be transformed into it. Figure 6 illustrates such a transformation. The upper H-COSTAM process automata is related to input and output communication media (here, random links). When a sequential process is split into a sequence of elementary tasks, precedence constraints are inserted where links existed with a media and between tasks issued from the same H-COSTAM process (artificial precedence constraints in the Figure).

To avoid the computation of absurd placement (e.g. an H-COSTAM process spread on discrete processors), a infinite valuation (or communication cost) is affected to artificial precedence if both involved tasks are not located in the same processor. Multi-rendezvous (not represented in Figure 6) can be associated with a bilateral constraint.

The main problem still being investigated are loops. So far, we unfold them when the imported heuristic does not handle them if an estimation of the number of loops is provided. Such an information can be deduced from previous executions of the generated prototype.

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 3.6.

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, at the micro-level description, the equivalence is quite difficult : triggers and some specific configuration have to be interpreted;
- some information has to be discarded. This is the case for the «usage-manual» that has no sense in H-COSTAM (while no verification is performed at this level).

	OF-Class	Equivalent in H-COSTAM	Auto.
macro-level	Rendez-vous operations	Multi-rendez-vous (binary)	yes
	Synchronous RPC operations	RPC	yes
	Asynchronous RPC operations	Two links, one for the call, one for the return value and output parameters. If the queries order is preserved, links will be FIFO. Otherwise, links will be random. The type of information that go through these media is deduced from the parameters of the service.	yes[a]
	Usage manual of a component	Discarded (no verification is performed at the operational level while the specification is supposed to be conceptually safe)	yes
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 contains 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 means 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 Unix-like forks). However, is it sometimes possible to propose a decomposition.

In Table 4, 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 semantics of his system and then solve the problem.

However, even 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.

4.4. Code Generation from H-COSTAM

H-COSTAM is dedicated to code generation. So, its entities can be translated into code patterns. Thus, a generic architecture (Figure 7) can be formulated and used as a frame for code generation. This architecture contains dedicated modules that are especially generated from the H-COSTAM processes [Kordon 95].

Generic modules perform the main operations in H-COSTAM :

- The type manager handles all types manipulation and operations (it is a library);
- Both passive and active media managers take care of communications. They should be implemented as sets of concurrent processes;
- The prototype manager is in charge of initialization and termination of the prototype execution. It also takes care of dynamic process creation (by means of factories). If we consider a distributed implementation, there should be one prototype manager per target CPU.

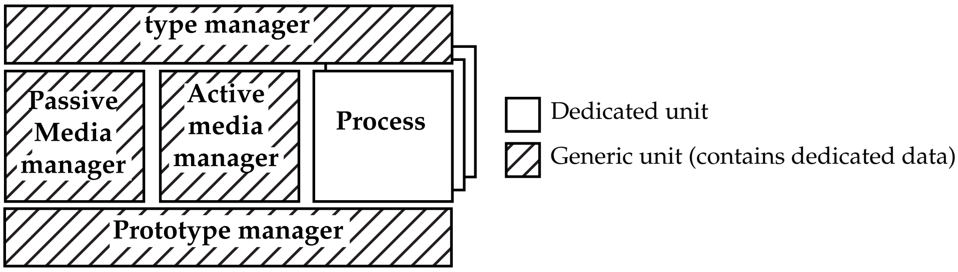

Figure 7 : Generic Architecture of a Prototype Produced from an H-COSTAM Description.

H_COSTAM genericity can be supported using the corresponding feature in the target language (templates in C++, genericity in Ada, etc.). Otherwise, a rewriting mechanism similar to macros has to be implemented in the code generator.

Communication between components of the generated prototype can be done via a standard message passing interface like PVM [Geist 94] or MPI [MPI 94], according to the target execution architecture.

The code generator only considers specification components that correspond to the system (i.e. tagged internal in Section 3.2.). Specification components that represent the execution environment are «glued» with the prototype during code generation. It is not really a linking procedure while the implementation of such components must respect convention regarding interfaces (connection between H-COSTAM processes are done using communication media only). Procedures having no side effect (otherwise, the validation of the system has no more signification) may be connected to H-COSTAM actions.

Then, it is of interest to use the software architecture of the prototype (issued from the H-COSTAM structure) to compute an appropriate location of components over a set of networked processors. Information we can extract from either Petri net synthesized models or operational research heuristics (see Section 4.2.2.) is valuable to help the system designer to get a location of its software components. Such an application partitioning should provide «good» performances and reasonably exploit the target execution architecture.

5. Application to an Example

We give here a small example to briefly illustrate capabilities of our representations (expression, analysis), the elicitation operation and code generation. The example models the collaboration between basic producers and consumers. It contains three components:

- one «init_master» which creates a set of couples <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 communicates with one consumer. The two entities of a couple share the same identifier.

5.1. 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 usage-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 is 1 ,
                  STOP in msg is 100
               }
/* 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 with an OF-Class component. 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}
   instances { 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19}
   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
```

5.2. Properties of the OF-Class model

According to Table 4, we transform the OF-Class specification into a Petri net for each component and compose them to a global net (Figure 8). In this model <ident.ALL> represents the diffusion function (the action is performed for all the elements of the *ident* color class).

	OF-Class	Petri net equivalent
macro-level	Offered service	One medium per operation. If the access is a rendez-vous, the medium is a shared transition. Otherwise, it is a set of shared places (one for an asynchronous operation, two for a synchronous one).
	required service	Arcs connected to the corresponding offered service.
	enclosed components	A sub-Petri net obtained after computation.
micro-level	resources, variables and parameters	A place whose color domain is deduced from the type of the item.
	Actions	One transition connected to the places modeling involved resources variables and service parameters.
	Operations and triggers	Set of places to sequence the actions.

Table 3: Some Translation Rules from OF-Class to Petri nets.

In this example, the components are simple, thus we do not need to compute CFFD-models. We directly use the reachability graph of the net modeling the global system.

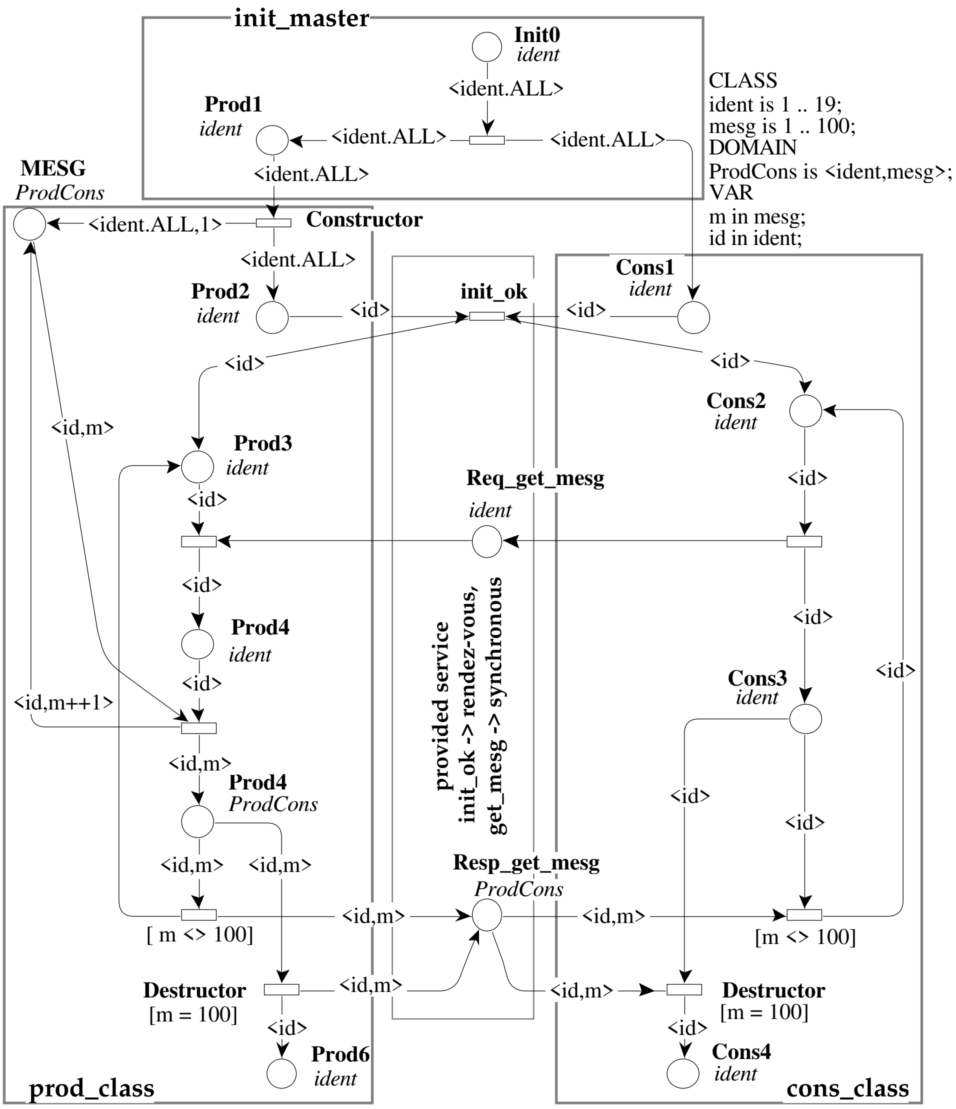

Figure 8 : The Petri net derived from the OF-Class model.

We can show that the model presents a deadlocked state, the one in which the places DeadCons and DeadProd are marked by all processes. On the reachability graph, this state appears as a deadlock but we know it is a correct final state.

We have the following place invariant :

(<ident.x1>)*Req_get_mesg*+(<ident.x1>)*Resp_get_mesg*+(<ident.x1>)*Cons2*+ (<ident.x1>)*Cons3*+(<ident.x1>)*Prod4*+(<ident.x1>)*Prod5*+ (<ident.x1>)*Cons4* = (<ident.x1>)*Cons3*+(<ident.x1>)*Cons2*+(<ident.x1>)*Cons4*.

where *ident.x1* corresponds to a variable that takes any value in the color class *ident*.

Its projection on the communication channel shows that the channel is safe (no message is neither lost nor generated). The other information we deduce from that place invariant is that the components (producer and consumer) are honest and reliable. Each request corresponds to one and only one reply and vice versa.

5.3. 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 9 describes the unit `init_master`. It contains three H-COSTAM entities : two deduced from the OF-Class description plus one corresponding to the behavior of `init_master` (`init_master_autom`).

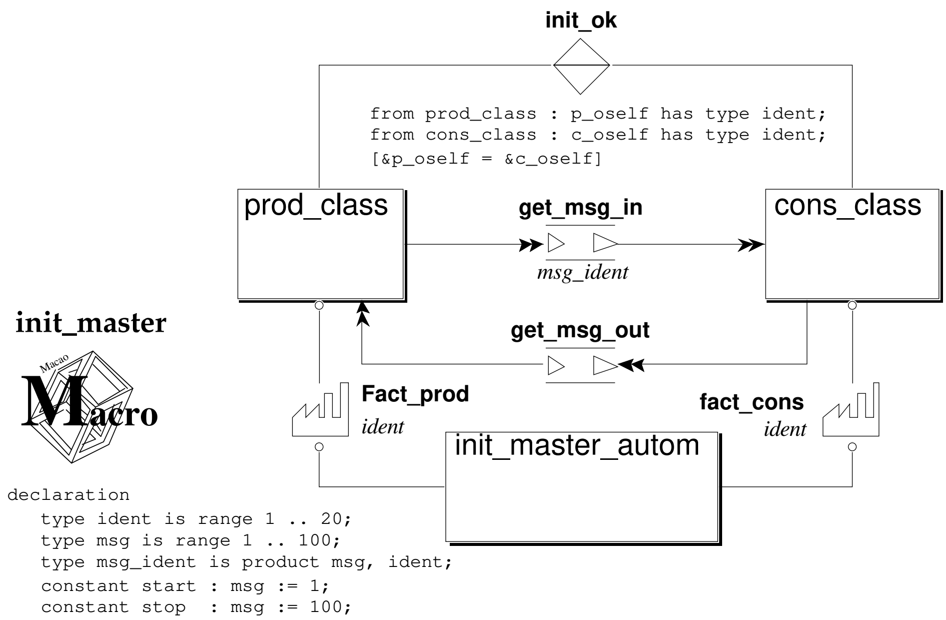

Figure 9 : The H-COSTAM Macro-Level Description of `init_master`.

According to Table 2, page 13 :

- 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 into a couple of pipes. The designer considers here 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`.

Types are directly deduced from the original definition in OF-Class. Instanciations are declared according to propagation in the OF-Class macro-page.

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 with the constructor.

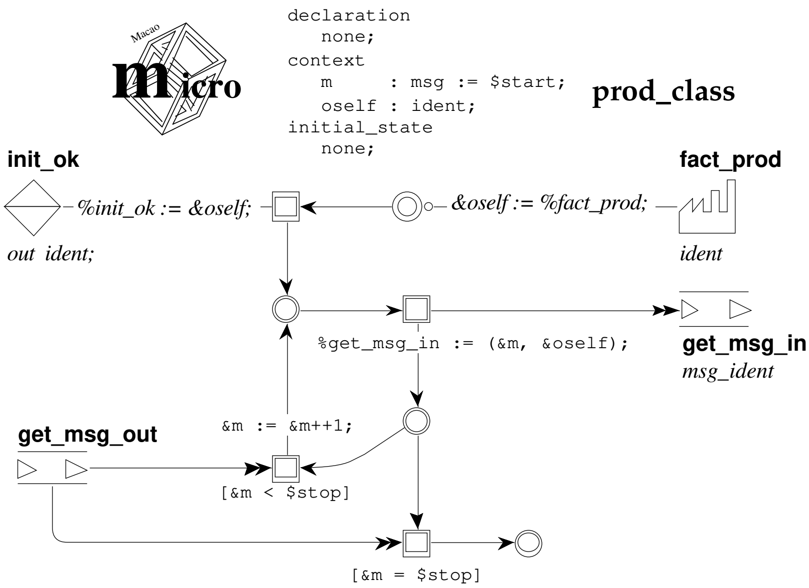

Figure 10 : The H-COSTAM Micro-level Description of `prod_class`.

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 with process dedicated to a service is deduced from its usage-manual. Here `init_ok` is performed and then, `get_msg` may be performed as many as necessary. Once the stop message is send, the destructor 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.

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

5.4. Some deduced characteristics of the H-COSTAM model

We propose in [Kordon 95] some translation rules from H-COSTAM to Petri nets. Similar rules, presented in Table 4, are used here to get a Petri net that is equivalent to the H-COSTAM specification.

The objective of such a transformation is to get a «view» that is suitable to extract some interesting characteristics. In the present example, we are interested by some distribution capabilities that are discussed in the next section.

	H-COSTAM	Petri net equivalent
macro-level	Multi-rendez-vous	If it is the «representative»(it is the corresponding object found in the page closest to the root page), it is a transition, otherwise, it is fusioned with the «representant».
	RPC	Two places (one for the query, one for the answer).
	Factory	If it is the «representative», an arc from all the input actions (transition) to all the output states (place). otherwise, nothing.
	Communication media	One place only. This is to reduce the size of the model and to focus on the media itself, not on its behavior (transformation is not for validation purpose)..
	Enclosed components (subsystems and processes)	a sub-Petri net obtained after computation.
micro-level	Process context	A color domain that contains all the involved variables. This color domain is the one of all transitions issued from places.
	States	One place.
	Actions	One transition if it is not connected to a multi-rendez-vous. Otherwise, it is fusionned with the «representative» of this multi rendez-vous.
	Interfaces (media, RPC, factories and multi-rendez-vous)	Fusionned with the «representative».

Table 4: Some Translation Rules from H-COSTAM to Petri nets.

Figure 11 corresponds to the Petri net obtained when the transformation is completed. The macro-level description defines the structure of the model (the three processes are outlined).

From this Petri net model, we can first deduce that the number of message stored in either `GET_MSG_IN` or `GET_MSG_OUT` cannot exceed 19 (the number of couple `prod_class/cons_class` created by init_master_autom). This information is derived from the bound of places associated with these FIFO media. Thus, a fixed array implementation is sufficient and induces no extra task blocking.

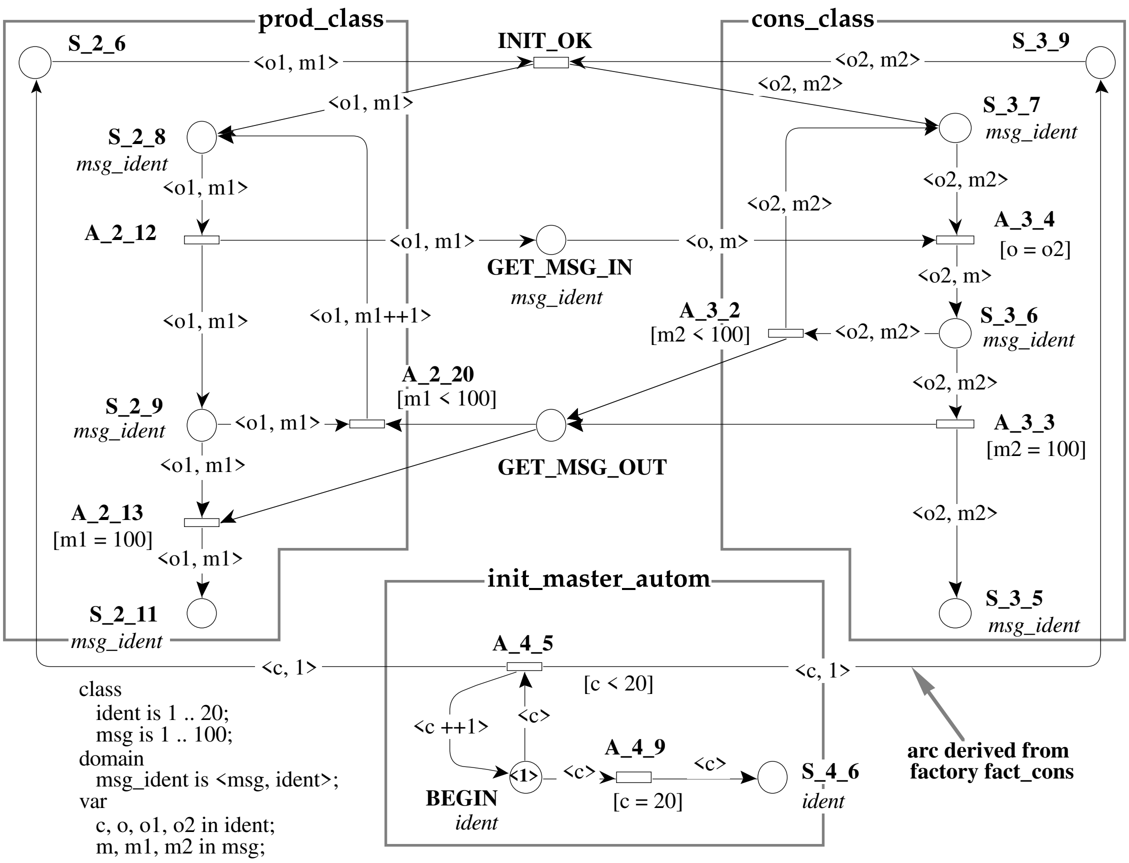

Figure 11 : The Petri net derived from the H-COSTAM specification.

We may find out that the couple `prod_class/cons_class` can be replicated without risks if the following rule is respected : ∀ *oself* ∈ *[1.. 19] prod_class(oself) on host x* ⇒ *prod_class(oself) on host x*. Such an information can be deduced from one of the following observations :

i. the partial unfolding of the Petri net model after color class `ident` produces a model in which there are many distinct `prod_class + cons_class + GET_MSG_IN` + `GET_MSG_OUT + INIT_OK` non connected components;
ii. the predicate of transition `A_3_2` forces a `prod_class` instance identified by a value of `oself` to communicate with the instance of `cond_class` having the same value. The predicate of shared transition `init_ok` also respects this rule.

So, groups <`prod_class`, `cons_class`, `GET_MSG_IN`, `GET_MSG_OUT`, `init_ok`> can be duplicated on several hosts.

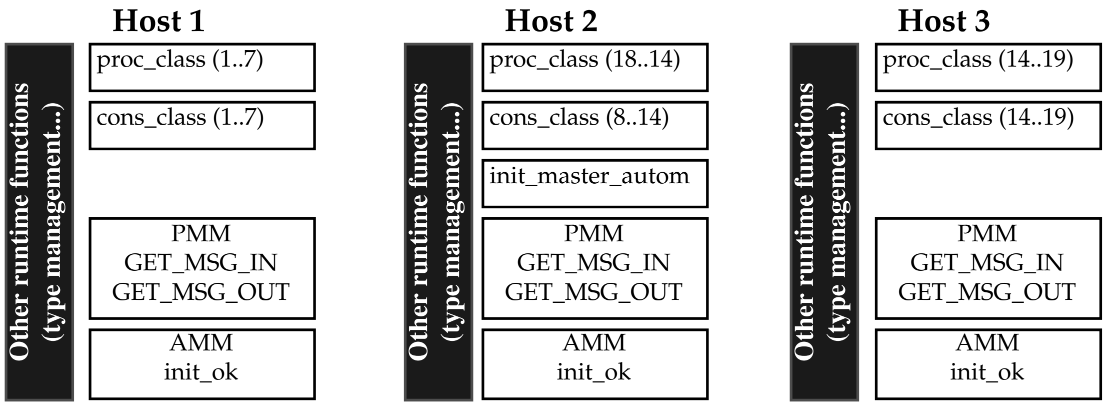

Figure 12 : Task Allocation Strategy over a Set of Processors.

Figure 12 proposes an example of task allocation over three processors. Process `init_master_autom` may be located on any CPU (in this case, it is the second one). Each host runs its own copy of `prod_class`, `cons_class`, `GET_MSG_IN`, `GET_MSG_OUT` and `init_ok`. This execution remains correct (and faster) if the condition computed from the Petri net is respected.

6. Conclusion

In this paper, we have proposed a methodology to build distributed systems from the ***conceptual*** level (specification) to the ***operational*** one (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 and liveness properties. The operational level is more likely dedicated to implicit properties addressing the optimization and location of the generated prototype. Code generation is achieved from the operational level.

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

- the semantics of the system,
- the properties proved at the conceptual level (no need for a verification at the operational level).

Strongs links with the Petri nets theory allows us to derive formal specification from both the conceptual and the operational levels (potentially checked properties are summarized in Table 5). H-COSTAM can also take benefits from operational research studies about distributed location of tasks [Norman 93]. Properties evaluated at the conceptual level are related to safety and liveness of the system. Properties evaluated at the operational level are exploited to optimize key points like task allocation in the generated prototype.

	Property
Conceptual Level	Modular verification of safety and liveness properties. For each module extended with an abstraction of its environment, we look for terminal states. Such a state tells that the module is blocked for ever. We also look for safe termination of the operations.
	The reachability graphs for the modular verification are abstracted while hiding the internal actions. Theses abstractions are synchronized to look for deadlocks caused by the interactions.
Operational Level	Pipe-line detection: pipe-lines are outlined by some structural invariants (P-semi-flows). Then, pipe steps may be located on discrete machines.
	Duplication detection: resources or communication mechanisms that can be duplicated on a target architecture can be detected analyzing partial unfolding of the colored Petri net. The system designer may then get both the replicable element and how to duplicate it.

Table 5: Summary of the properties we can validation using Petri nets in our methodology.

Some aspects of our approach have been considered in other works. [Di Giovanni 90] focuses on the specification aspects. [Paludetto 91], [Bruno 94] and [Lakos 95b] also support prototyping either by execution or implementation directives. However, they are less concerned with tracing properties from conceptual to operational level. Our methodology also enables optimization on distribution of the generated code.

In order to manage large size specification and to evaluate the pertinence of our methodology, we are currently working on the implementation of an experimental CASE tool. Previous experimentation where also performed within the university project MARS [MARS 94] and the industrial EURÊKA IRENA project [IRENA 95]. The example presented in this paper was partially computed using the current implementation.

7. References

[Bachatène 93] H. Bachatène & J.M. Couvreur, "A Reference Model for Modular Colored Petri Net", 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.

[Bruno 94] G. Bruno, A. Castella, R. Agarwal & M.P. Pescarmona, "CAB: An Environment for Developping Concurrent Applications", Proceedings of the 15th International Conference on Application and Theory of Petri Nets, Zaragoza, Spain, June 1994, LNCS vol. 815 PP. 141-160.

[Buchholz 94] P. Buchholz, "Hierarchical High Level Petri Nets for Complex System Analysis", Proceedings of the 15th International Conference on Application and Theory of Petri Nets (LNCS, spinger Verlag), Zaragoza, Spain, June 1994, LNCS vol. 815 PP. 119-138.

[Chiola 91] G. Chiola, C. Dutheillet, G. Franceschini & S. Haddad, "On Well-Formed Colored 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", International Workshop FMOODS'96, Paris, Mars 1996.

[Di Giovanni 90] R. Di Giovanni, "Petri Nets and Software Engineering : HOOD Nets", Proceedings of the 11th International Conference on Application and Theory of Petri Nets, Paris, June 1990.

[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 Prototypin", N. Kanopoulos Ed, IEEE comp. Soc. Press, Grenoble, June 1994.

[Fleinschhack 97] H.Fleinschhack & B.Grahlmann, "A Petri Net Semantics for B(PN)2 with procedures", in proceedings of PDSE'97, 1997.

[Geist 94] A.Geist, A.Beguelin, J.Dongarra, W.Jiang, R.Manchek & V.Sunderam, "PVM: Parralel Virtual Machine, A Users' Guide and Tutorial for Networked Parrallel Computing", MIT Press, 1994

[Grahlmann 96] B.Grahlmann & E.Best, "PEP - More than a Petri Net Tool", in proceedings of TACAS'96, LNCS vol 1055, Springer Verlag, April 1996.

[Grahlmann 97] B.Grahlman, "The PEP Tool", tool presentation at the 18th International Conference on Application and Theory of Petri Nets, Toulouse, France, June 1997

[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", Proceedings of the 9th International Conference 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, International Computer Science Institute, Berkeley, California, USA, March 1992.

[Heitbreder 97] O.Heitbreder, B.Kleinjohann, E.Kleinjohann & J.Tacken, "Intelligent Design Assistance with SEA", IEEE International Symposium and Workshop on Systems Engineering of Computer Based Systems (ECBS '97), Monterey, CA (USA), March 1997

[Jelly 96] I. Jelly & I. Gordon, "The PARSE Project", in proceedings of the IFIP International Workshop on Software Engineering for Parallel and Distributed Systems, Berlin, Germany, Chapman and Hall, March 1996

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

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

[Kleinjohann 96] B.Kleinjohann, E.Kleinjohann & J.Tacken, "The SEA Language for System Engineering and Animation", 17th International Conference on Application and Theory of Petri Nets, Osaka, Japan, LNCS 1091, Springer Verlag, pages 307-326, 1996

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

[Lakos 95b] C.A. Lakos, "An Open Software Engineering Environment based on Object Petri Nets", Technical Report, University of Tasmania, May 1995.

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

[MPI 94] MPI Forum, "MPI: A message-passing interface standard. International", in Journal of Supercomputer Application, 8 (3/4), pp165-416, 1994

[Norman 93] M. Norman & P. Tanisch, "Models of machines and computation for mapping in multicomputers", in ACM computing Surveys, vol 25, n°3, pp 263-302, September 1993

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

[Paludetto 91] M. Paludetto, "Sur la commande de procédés industriels: une Méthodologie basée objets et réseaux de Petri", PhD thesis from Université Paul Sabatier de Toulouse, Décembre 1991

[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.

[Valmari 94] A. Valmari, "Compositional Analysis with Place-bordered Subnets", Proceedings of the 15th International Conference on Application and Theory of Petri Nets (LNCS, spinger Verlag), Zaragoza, Spain, June 1994.