1 PETRI NET

PETRI NET system (completely specified initial marking)

S={P, T, I, O, H, M_0}

WHERE

P places

T transitions, $T \cap P = \varnothing$

I, O, H : T → Bag(P)

M_0: P → ℵ initial marking

Graphically, places are circles, transitions are boxes and arcs are arrows between them. The value of M_0 is written inside the circle or indicated by an appropriate number of black dots, so-called tokens, inside the circle.

If we model a system by a petri net, then:

- the transitions model the active part of the system
- the places model the passive parts, and
- the markings describe the system states

The arcs of a graph are classified as:

- Input arcs: arrow-headed arcs from places to transitions
- Output arcs: arrow-headed arcs from transitions to places
- Inhibitor arcs: circle-headed arcs from places to transition

1.1 Colored Petri Nets (CPNs)

The notation of CPNs introduces the notion of token types, namely tokens are differentiated by colors, which may be arbitrary data values.

Each place has an associated type determining the kind of data that the place may contain.

1.2 Hierarchical Colored Petri Nets

Hierarchical Colored Petri Nets (HCPNs) introduce a facility for building a PN out of subnets or modules. The idea behind HCPNs theory is to allow the construction of a large model by using a number of small PNs, which are related to each other in a well-defined way.

2 AN EXAMPLE OF PETRI NET

2.1 Description of the problem: A harbor

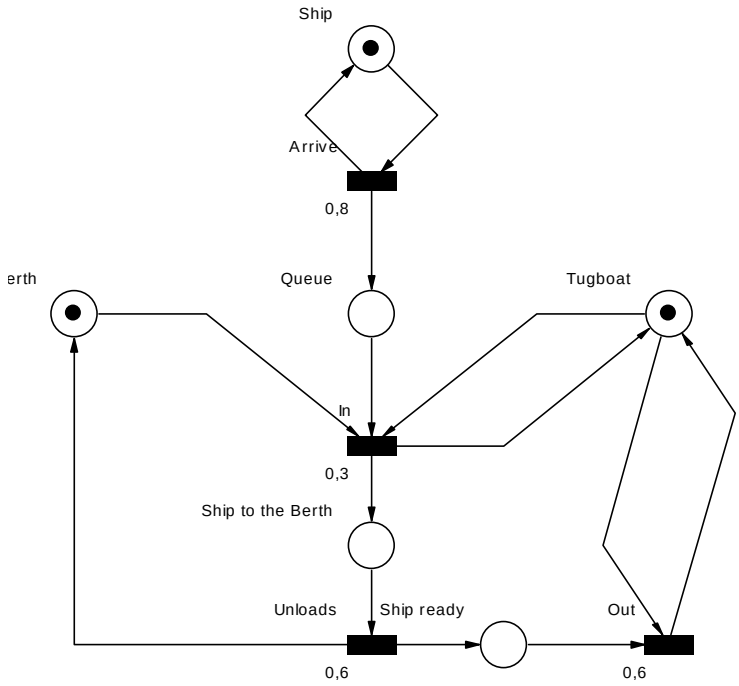
A harbor has one berth and one tugboat. Ships arrive outside the harbor every 80 min.

A ship is transferred from outside the harbor to the berth when the tugboat and the berth are available. The transfer from outside the harbor to the berth (with the help of the tugboat) takes 30 min. At the end of transfer the tugboat becomes available.

The ship unloads at the berth for 60 min. Then, if the tugboat is available gets the ship (the berth becomes available) and it transfers it outside the harbor. This takes 60 min.

Model the system using Petri Nets.

2.2 Simulation of the system with a PN



3 OBJECT ORIENTED PETRI NETS

The term object-oriented is generally used to describe a system that deals primarily with different types of objects, and where the actions one can take depend on what type of object are manipulated. The methods are based on simple mathematical models of abstraction and classification.

In order to deal with large specifications, several formal specification languages have adopted the object-oriented paradigm. Petri Net has at least, two-object-oriented extensions:

1. LOOPN (Language for OO Petri-Nets) and LOOPN++
2. CO-OPN (Concurrent OO Petri-Nets) and CO-OPN/2.

3.1 Object Petri Nets

Object Petri Nets (OPN) are presented as an extension of colored Petri nets made in a similar way to Hierarchical colored Petri nets, but in an object-oriented perspective[1].

An OPN is a tuple OPN=(T, C, C_0) where:

a) T is a set of basic types that determines the underlying component values, operators and functions
b) C is an OPN class hierarchy with inheritance relations based on sub typing of data fields
c) The class $C_0 \in C$ is a designated root class

OPNs support a complete integration of object-oriented concepts into petri nets, including inheritance and the associated polymorphism and dynamic binding. A class is

[1] Object and states, encapsulation, synchronous requests, asynchronous request, data abstraction, data structures of objects, object identity, intra-object concurrency, inter-object concurrency, class as templates, inheritance, sub typing, multiple inheritance, multiple sub typing, static object instantiation, generic classes, semantics, calculus.

defined as a Petri net, which can be, as usual, instantiated. In addition to places and transitions, a class contains data fields and functions. Data fields have types that may be simple (integer, real Boolean), class, or multi-set, which generalizes classical Petri net, places. New functions can be defined assuming predefined types and functions.

3.1.1 LOOPN Specification Language

LOOPN is a specification language used to describe Object Petri net models. It allows the decomposition of a Petri net into subnets or modules. Each module can be effectively replaced by a transition called super transition. Connection to a module is done via the input and output interface place nodes. LOOPN descriptions are mapped into C code. Executing the code after compilation can simulate systems specified in this language.

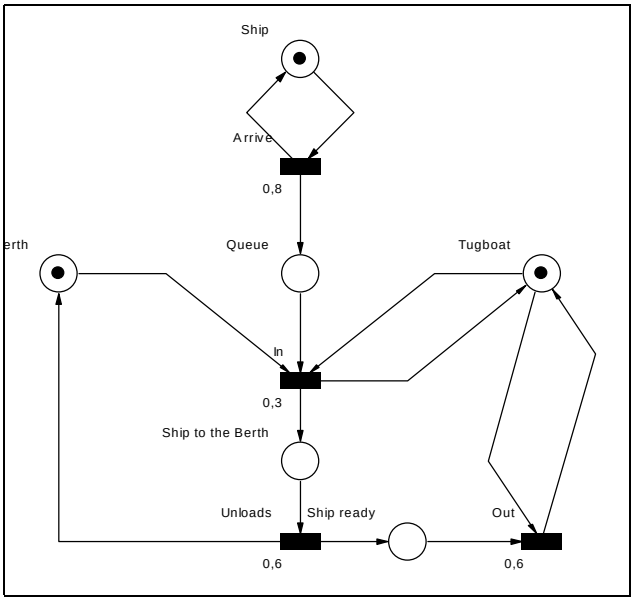
TOKENS: LOOPN supports the declaration of token types. A token type can be declared as a subtype of another. In this case, it inherits the parent's data fields and functions. A token type consists of a set of data fields and a set of associated functions and defines the values that tokens of that type may assume. The data fields determine the range of possible color combinations (values) that tokens of this type may assume. The function defined for a token type allows the evaluation of conditions pertaining to a particular token or to the associated list of tokens resident in a particular place. The most elementary token type, which all others ultimately inherit, is called null. While it has no data fields, it includes definitions of the functions first, last, delay, empty, tail, which are available to every other token type.

PLACES: Places are declared to be of a specific token type, thus limiting the tokens, which may be resident at that place.

MODULES: A petri net is coded as one or more modules, each of which consists of constant, type, place, and transition declarations. Syntactically it is similar to a transition but semantically it is fired only once at start up. A module may be self-contained, with only constant parameters to identify the particular instantiation. It may provide status information to other modules through access functions or transfer tokens via interface places.

LOOPN++ derives from LOOPN and it introduces new object-oriented notion such as class.

3.1.2 Example



```
MODULE Harbor()
CONST
TYPE  B_T = TOKEN null WITH
                Available: Boolean;
        S = TOKEN null WITH
                Priority: Integer;
PLACE           Ship, Queue, Ship_to_the_Berth, Ship_ready:S;
                Berth, Tugboat:B_T;
INITIALISATION
        OUTPUT      Ship ← i1 =[Priority: identifier]
                    Berth ← i2 =[Available: true]
                    Tugboat ← i3 =[Available: true]
TRANSITION Arrive;
INPUT           in1 ← Ship | in1.delay(8);
OUTPUT          Ship ← in1;
                Queue ← in1;
ACTION          printf("New arrival");

TRANSITION In;
INPUT           in3 ← Berth | in3.delay(3);
                in4 ← Queue | in4.delay(3);
                in5 ← Tugboat | in5.delay(3);
OUTPUT          Ship_to_the_Berth ← in4;
                Tugboat ← in5;
ACTION          printf("Ship to the Berth");
. . .
. . .
END MODULE
```

3.1.3 COOPN Specification Language

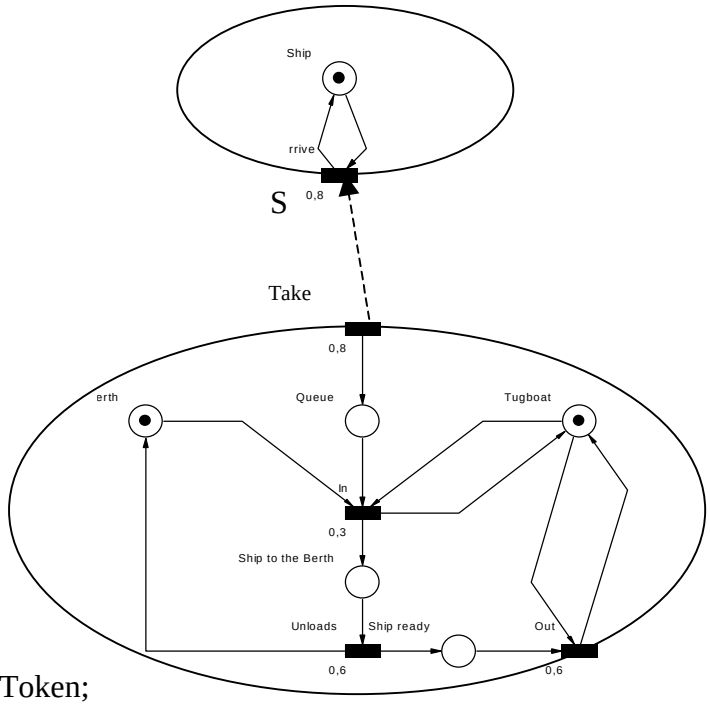
CO-OPN specification consists of a collection of autonomous entities called objects, which execute concurrently and interact with each other synchronously. But, CO-OPN does not support some object-oriented features such as the notion of class and dynamic creation of objects, inheritance, sub typing and object mobility.

CO_OPN/2 is a formalism devised for the specification of concurrent systems that integrates object-oriented features missing or poorly integrated in its predecessor CO-OPN. CO-OPN/2 has adopted the object-oriented paradigm and according to this technique, a system is considered as a collection of autonomous objects grouped into classes and which use data structures. Thus a CO-OPN/2 specification is composed of two kinds of modules, namely the abstract data type (ADT) modules and the class modules. The ADT module concern the data structure involved in the specification. As for the class modules, they represent a collection of encapsulated entities that possess an internal state and provide the exterior with various services (description of behavioral aspects).

Object: is an encapsulated modular algebraic net in which the places compose its internal state and transitions (invisible events) represent the concurrent events of the object. Interaction with an object can be made by request of its services through its methods (observable events - external object stimuli).

Class: an encapsulated modular algebraic net used as a template from which objects are statically or dynamically instantiated.

3.2 Example



Adt BlackToken;
Interface
Sort blacktoken;
Generator @: → blacktoken;
Body
Axioms
Def @;
End BlackToken;

Class Harbor;
Inherit
Interface
Use BlackToken;
Methods Take, Out;
Transitions In, Unloads;
Body
Places
Berth _, Queue _, Tugboat _, Ship_to_the_Berth _,
Ship_ready _:BlackToken;
Initial
Ship @, Berth @, Tugboat @;
Axioms
Take with S.Arrive :: → Queue @;
In :: Berth @, Queue @, Tugboat @ → Ship_to_the_Berth @;
Unloads :: Ship_to_the_Berth @ → Berth @, Ship_ready @;
Out :: Ship_ready @, Tugboat @ → Tugboat @;
End Harbor;