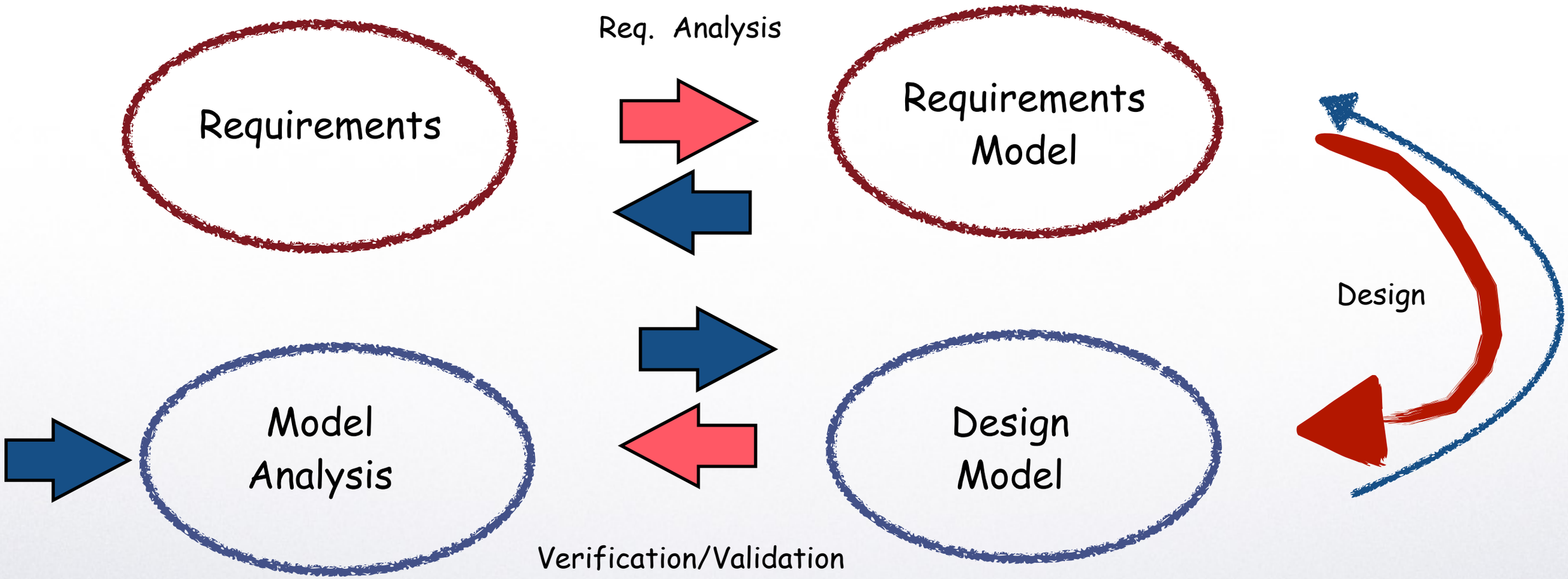
PMR 5237

Modelagem e Design de Sistemas Discretos em Redes de Petri

Aula 12 : Modelagem em redes CPN

Prof. José Reinaldo Silva
reinaldo@usp.br

Use of Petri Nets in Design



PN Basic Properties

1) *boundedness*, characterising finiteness of the state space.
2) *liveness*, related to potential fireability in all reachable markings. *Deadlock-freeness* is a weaker condition in which global infinite activity (i.e. fireability) of the net system model is guaranteed, but some parts of it may not work at all.
3) *reversibility*, characterising recoverability of the initial marking from any reachable marking.
4) *mutual exclusion*, dealing with the impossibility of simultaneous *submarkings* (p-mutex) or *firing concurrency* (t-mutex).

Fairness

Modelagem de Sistemas Discretos

- Síntese da rede;
- Procedimentos de redução;
- Análise da rede (atingibilidade, deadlock, etc.);
- Simulação.

Methods to solve these equations:

Mathematic linear programming (Simplex);

Gaus-Jordan method

Algoritmo para calcular invariantes de lugar

Tese de doutorado: Modelagem e Análise de Requisitos de Sistemas Automatizados Usando UML e Redes de Petri, Arianna Z. Olivera Salmon

Passo 1: Para obter os invariantes de lugar é preciso em primeiro lugar calcular a transposta da matriz de incidência(AT).

Passo 2: Em seguida deve-se aplicar o método de Gauss- Jordan para transformar a matriz de incidência em uma matriz diagonal. Se o posto da matriz de incidência R(AT) for menor que o numero de incógnitas (numero de lugares da rede), então ir ao passo 3. Se o posto da matriz de incidência for igual ao numero de incógnitas, então terminar, o sistema não tem invariantes de lugar.

Passo 3: Obter a solução básica do sistema representado pela matriz de incidência.

Invariants are a very important feature in CPN Design. However, we should not expect to solve the design problem by just inserting invariant analysis.

Besides those inherent problems with invariants, the difficulty to apply this approach to large systems is still present.

Boudedness in CPN

1) *boundedness*, characterising finiteness of the state space.
2) *liveness*, related to potential fireability in all reachable markings. *Deadlock-freeness* is a weaker condition in which global infinite activity (i.e. fireability) of the net system model is guaranteed, but some parts of it may not work at all.
3) *reversibility*, characterising recoverability of the initial marking from any reachable marking.
4) *mutual exclusion*, dealing with the impossibility of simultaneous *submarkings* (p-mutex) or *firing concurrency* (t-mutex).

Let be CPN net N and a generic place p in this net and $|M(p)|$ the size of the multiset for making M. The place p is said bounded iff there is an integer n which is un upper bound of p, that is,

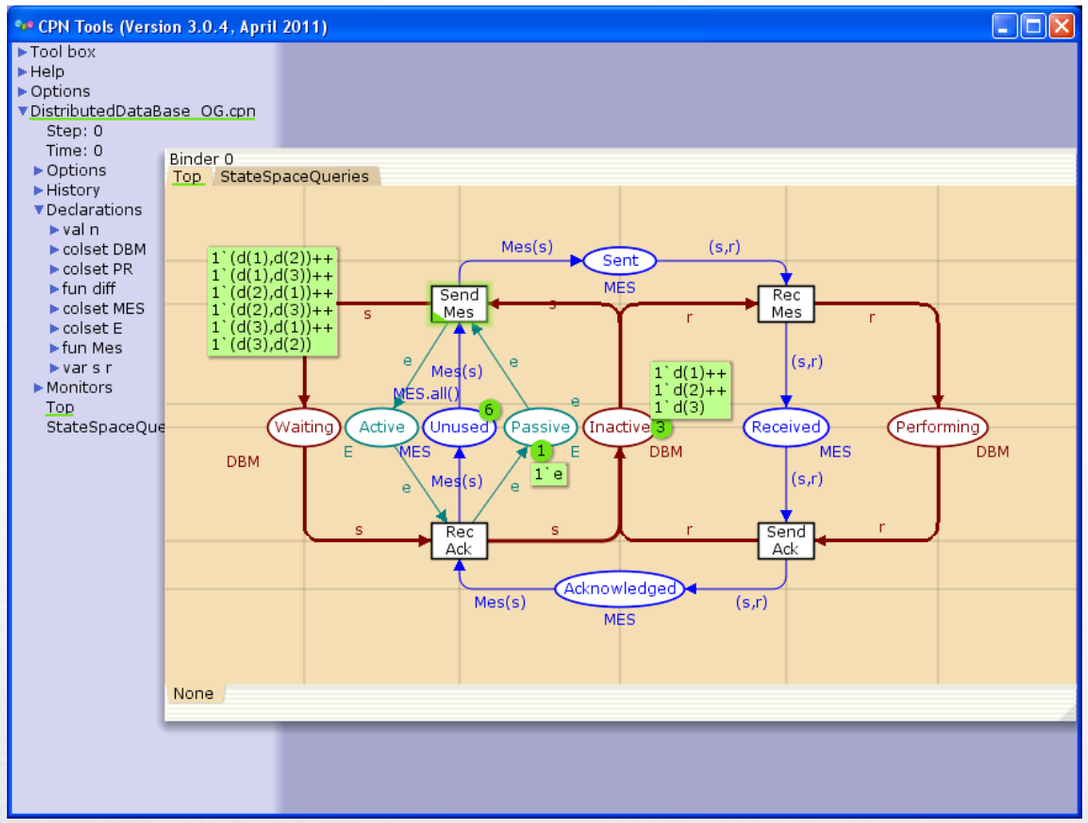
$$\forall M \epsilon \Re(M_0) \,.\, |M(p)| \leq n.$$

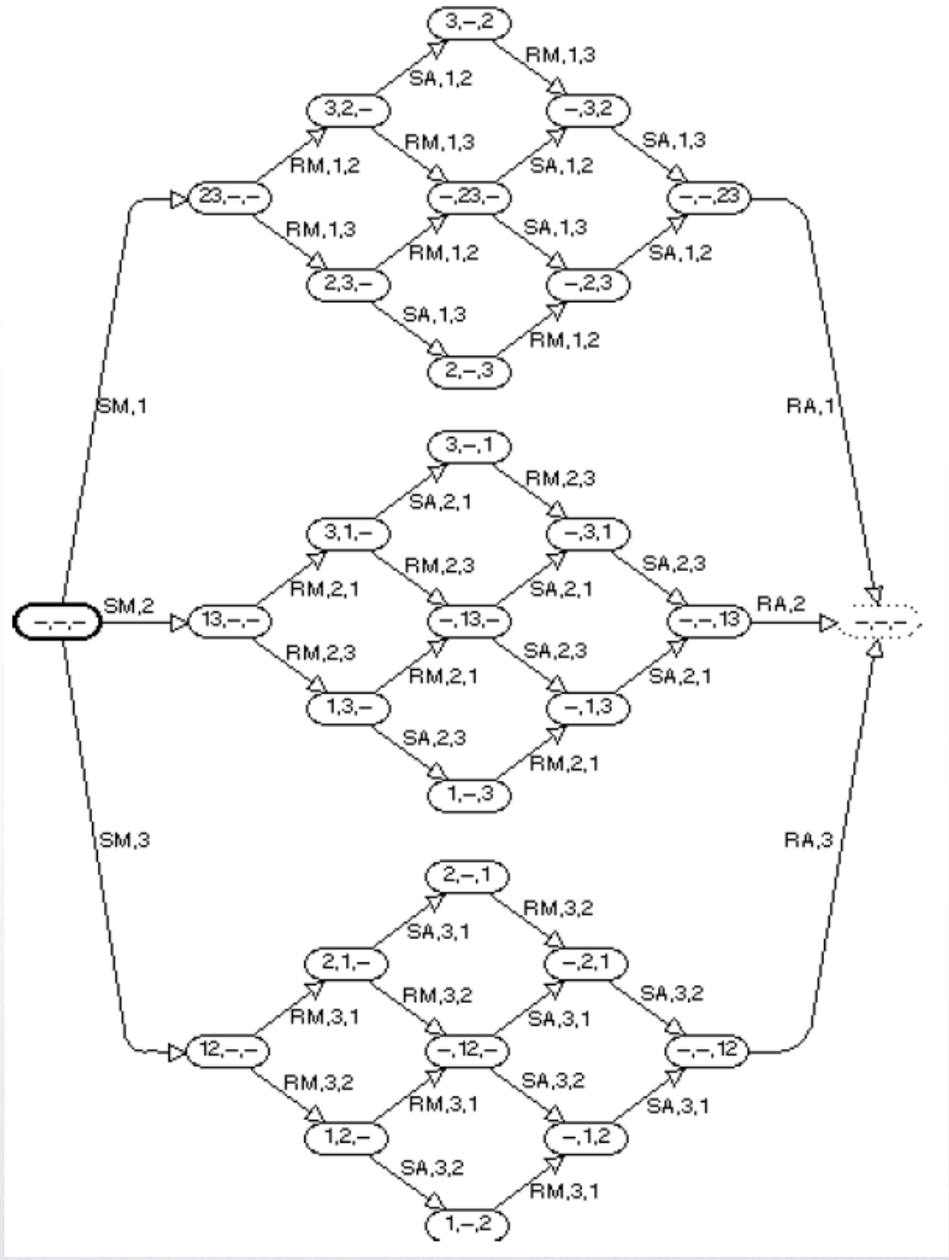
A CPN net is said to be bounded if all its place are bounded.

State space analysis in CPN

The basic idea of state spaces is to **calculate all reachable states** (markings) and **state changes** (occurring binding elements) of the CPN model and to represent these in **a directed graph** where the nodes correspond to the set of reachable markings and the arcs correspond to occurring binding elements. The state space of a CPN model can be computed fully automatically and makes it possible to automatically analyse and verify an abundance of properties concerning the behaviour of the model. Examples of such properties include the minimum and maximum numbers of tokens on a place, the states in which the system may terminate, and the system always being able to reach a certain state.

Analysis in CPN Nets





Directed Graphs

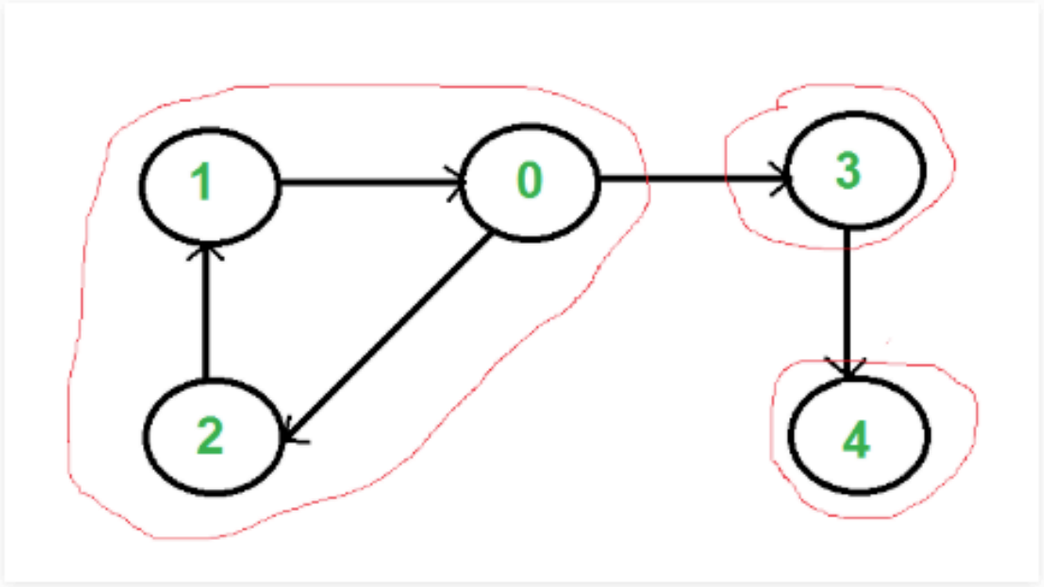
Definition 37

A direct graph is tuple $DG = (V, A, N)$ such that:
(i) V is a set of nodes or vertices;
(ii) A is a set of arcs (or edges) such that $V \cap A = \emptyset$;
(III) N is a node function or mapping $A \rightarrow V \times V$.

DG is finite if $V$and A are finite.

Strongly Connected Components

A directed graph is strongly connected if there is a path between all pairs of vertices. A strongly connected component (**SCC**) of a directed graph is a maximal strongly connected subgraph. For example, there are 3 SCCs in the following graph.



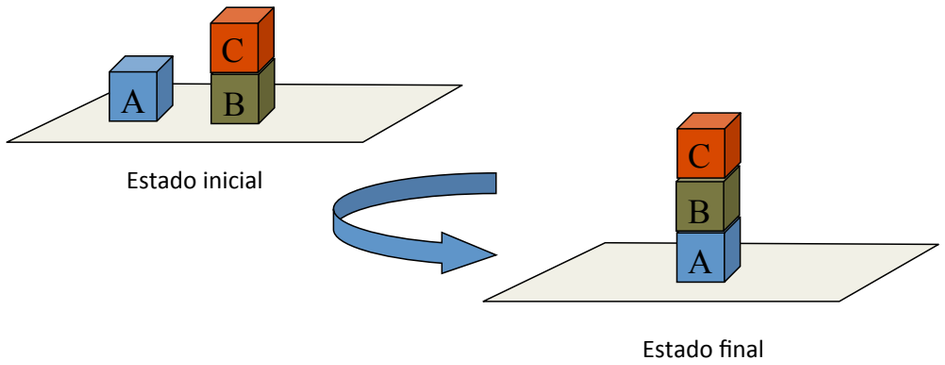
We can find all strongly connected components in O(V+E) time using Kosaraju's algorithm. Following is detailed Kosaraju's algorithm.

1) Create an empty stack 'S' and do DFS traversal of a graph. In DFS traversal, after calling recursive DFS for adjacent vertices of a vertex, push the vertex to stack. In the above graph, if we start DFS from vertex 0, we get vertices in stack as 1, 2, 4, 3, 0.

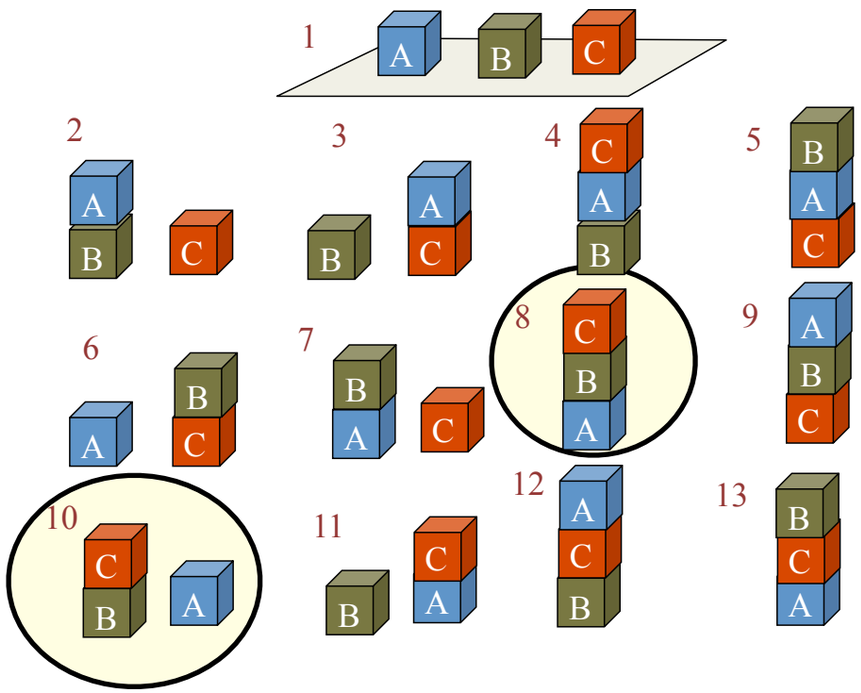
2) Reverse directions of all arcs to obtain the transpose graph.

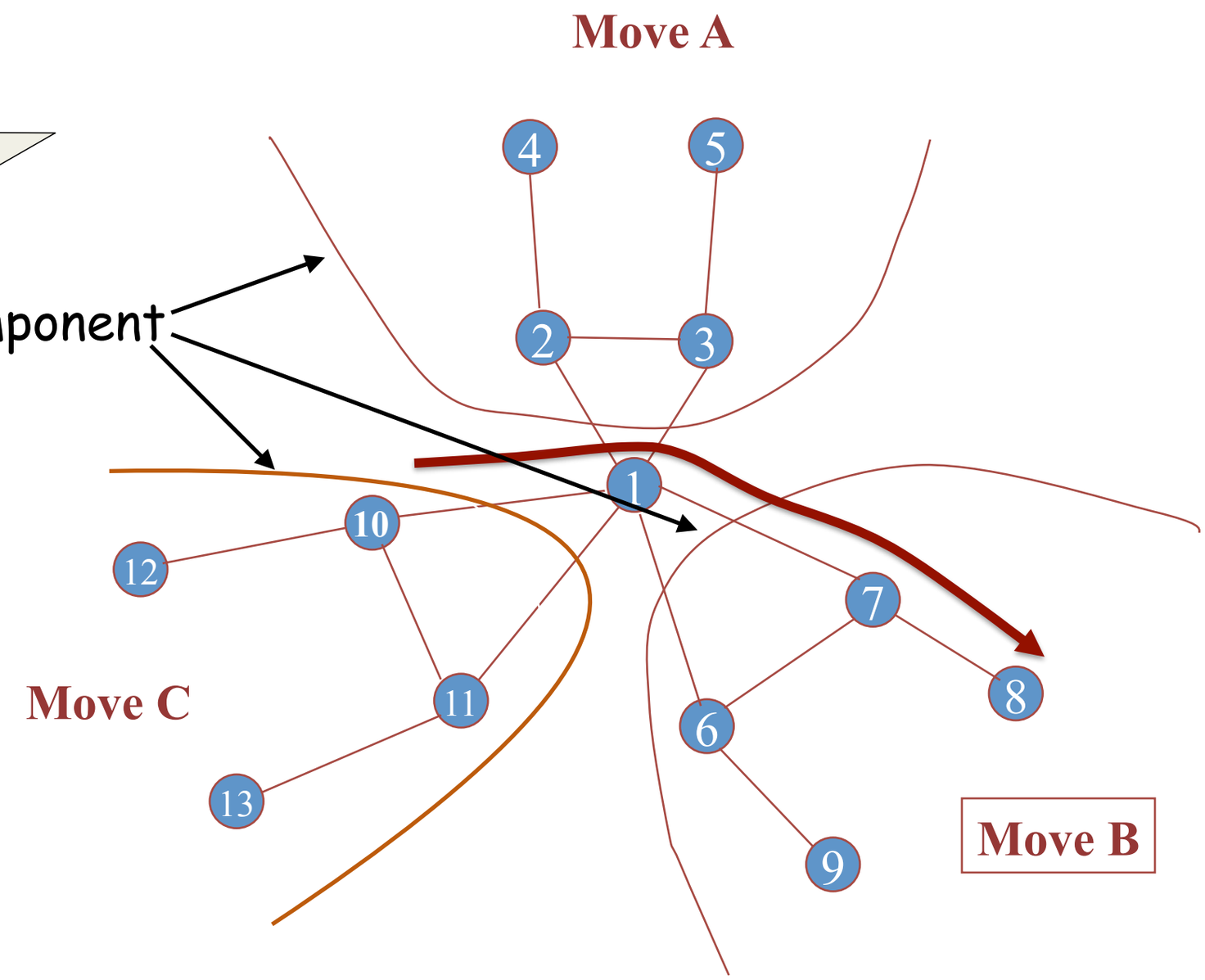
3) One by one pop a vertex from S while S is not empty. Let the popped vertex be 'v'. Take v as source and do DFS (call DFSUtil(v)). The DFS starting from v prints strongly connected component of v. In the above example, we process vertices in order 0, 3, 4, 2, 1 (One by one popped from stack).

Lembrando a Aula2...



SCC-strongly connected component





Proposition 9.8. *Let $SS = (N_{SS}, A_{SS})$ be the finite state space of a Coloured Petri Net, and let $SG = (N_{SG}, A_{SG})$ be the SCC graph. Then the following holds:*

1. *A marking M' is reachable from a marking $M \in \mathscr{R}(M_0)$ if and only if*

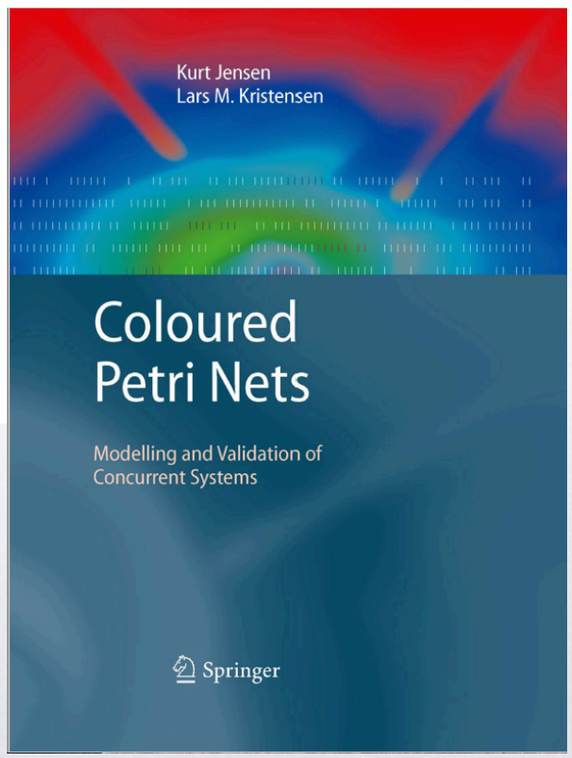
$$M' \in \mathscr{R}_{SS}(M)$$

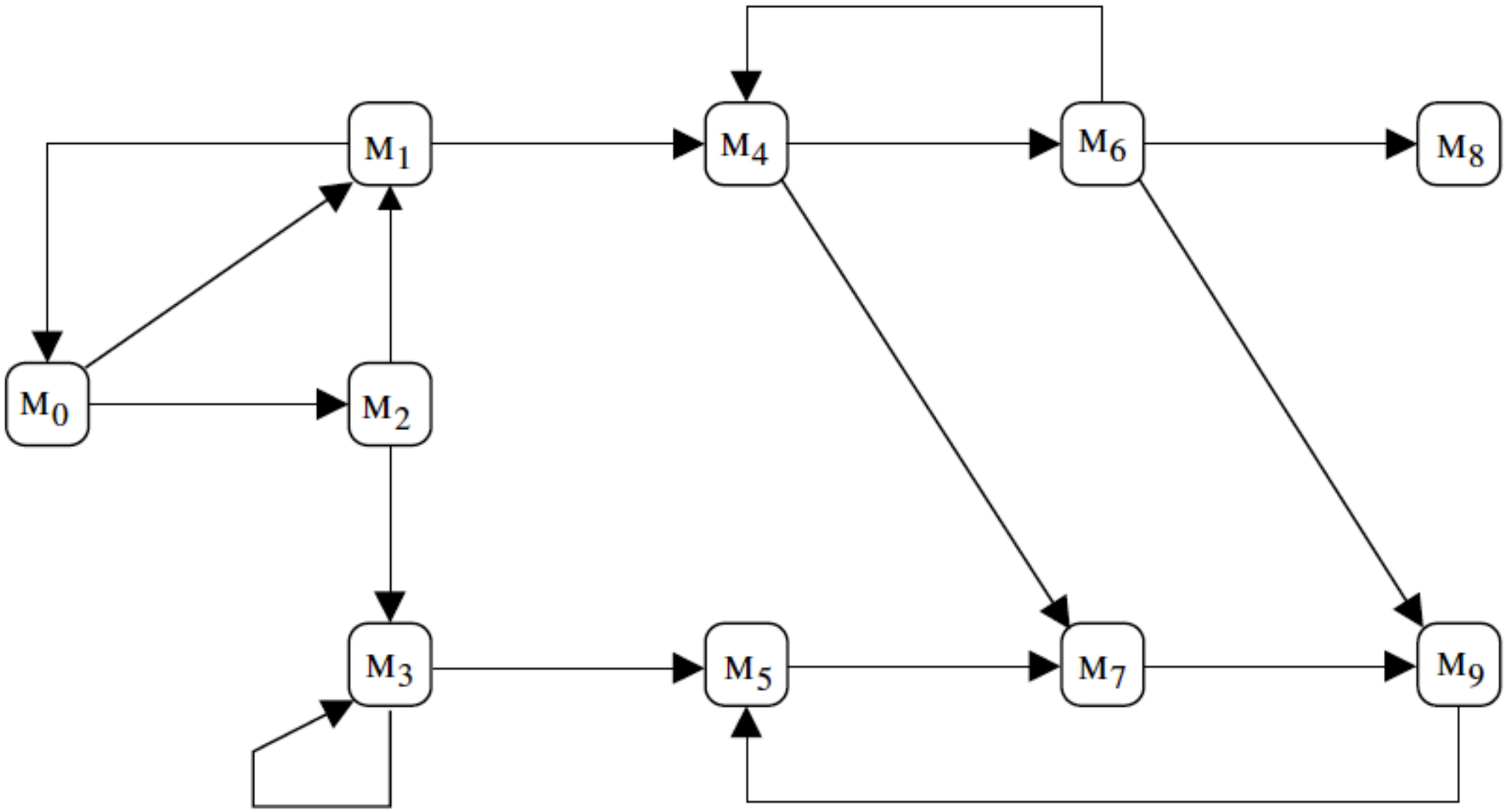
2. *A marking M' is reachable from a marking $M \in \mathscr{R}(M_0)$ if and only if*

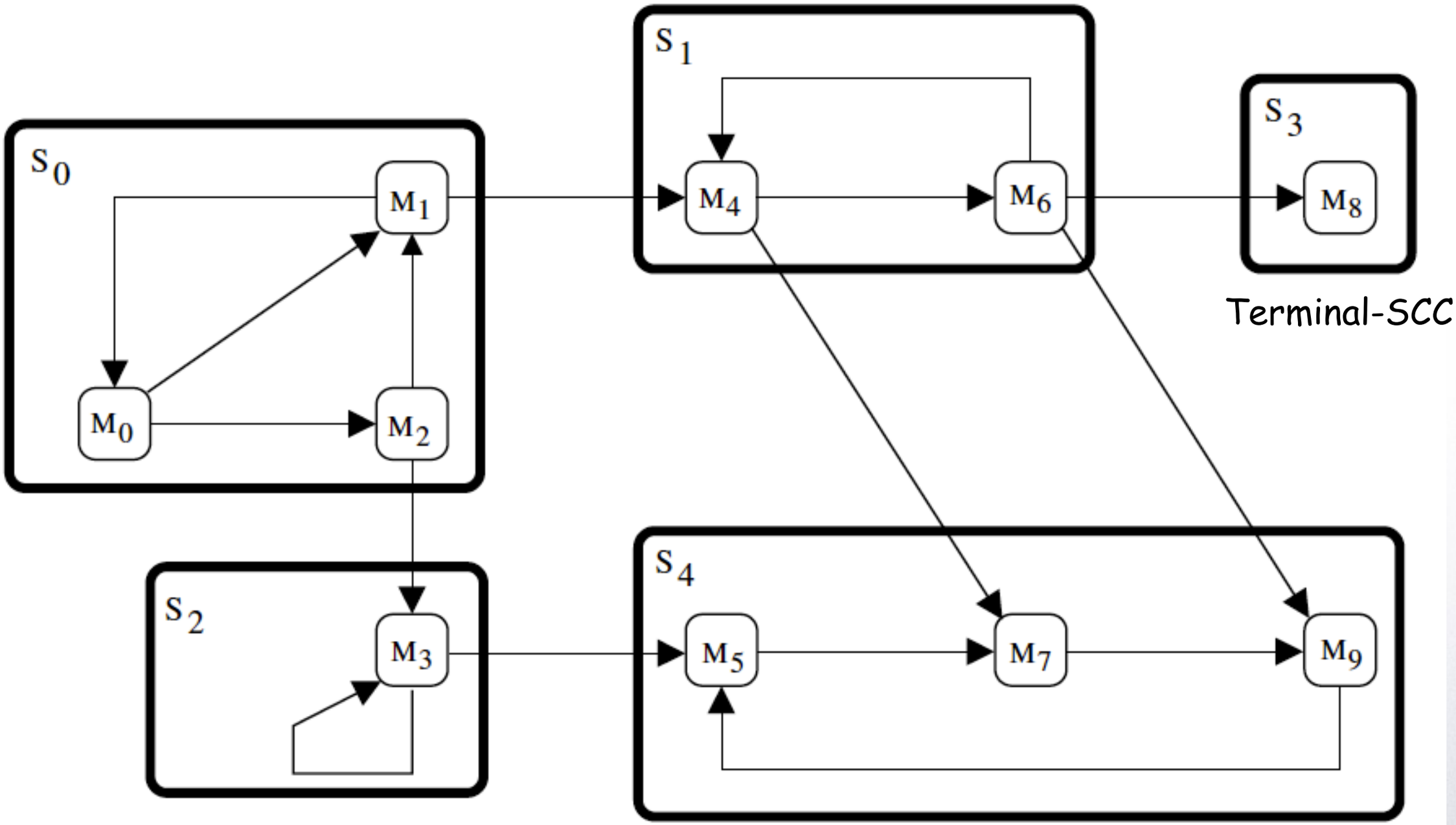
$$SCC(M') \in \mathscr{R}_{SG}(SCC(M))$$

3. *A predicate ϕ on markings is reachable if and only if*

$$\exists M \in N_{SS} : \phi(M)$$




M0
M1
M2
M3
M4
M5
M6
M7
M8
M9


S0
M1
M0
M2
S1
M4
M6
S3
M8
Terminal-SCC
S2
M3
S4
M5
M7
M9

Proposition 9.8. *Let $SS = (N_{SS}, A_{SS})$ be the finite state space of a Coloured Petri Net, and let $SG = (N_{SG}, A_{SG})$ be the SCC graph. Then the following holds:*

1. *A marking M' is reachable from a marking $M \in \mathscr{R}(M_0)$ if and only if*

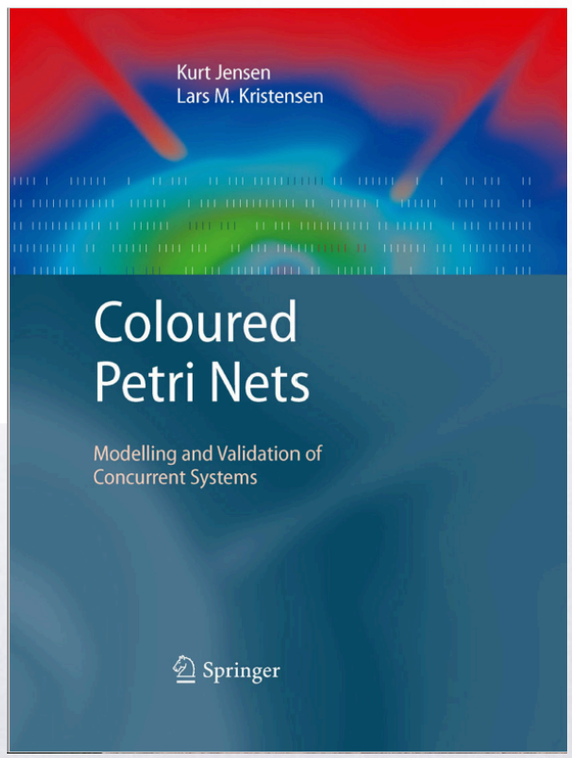
$$M' \in \mathscr{R}_{SS}(M)$$

2. *A marking M' is reachable from a marking $M \in \mathscr{R}(M_0)$ if and only if*

$$SCC(M') \in \mathscr{R}_{SG}(SCC(M))$$

3. *A predicate ϕ on markings is reachable if and only if*

$$\exists M \in N_{SS} : \phi(M)$$



1) *boundedness*, characterising finiteness of the state space.
2) *liveness*, related to potential fireability in all reachable markings. *Deadlock-freeness* is a weaker condition in which global infinite activity (i.e. fireability) of the net system model is guaranteed, but some parts of it may not work at all.
3) *reversibility*, characterising recoverability of the initial marking from any reachable marking.
4) *mutual exclusion*, dealing with the impossibility of simultaneous *submarkings* (p-mutex) or *firing concurrency* (t-mutex).

A CPN net $\mathbb{N}$ is said to be **live** iff $\forall M \exists \mid M \epsilon L(M_0)$.

A CPN net $\mathbb{N}$ is said to be strongly **live** iff $\forall M \exists \mid M \epsilon SCC(M_0)$

Definition 9.19. Let a transition $t \in T$ and a marking M be given.

1. M is a **dead marking** if and only if

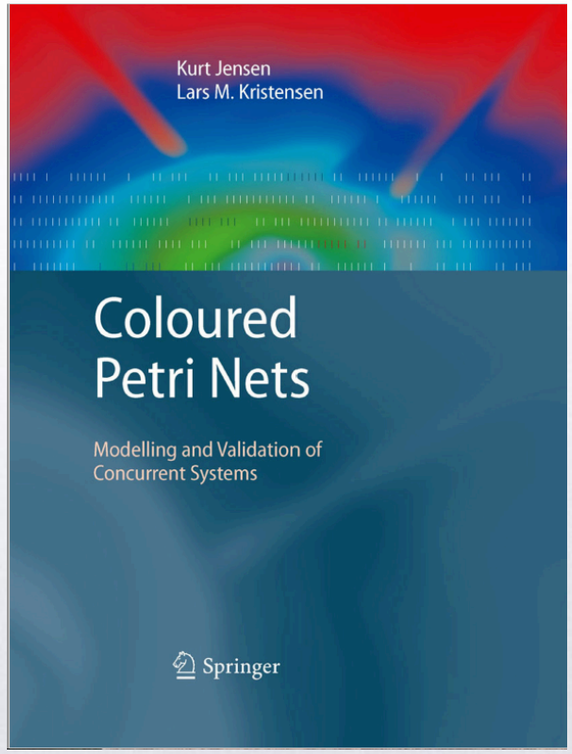
$$\forall t \in T : \neg (M \xrightarrow{t})$$

2. t is **dead in** M_0 if and only if

$$\forall M \in \mathscr{R}(M_0) : \neg (M \xrightarrow{t})$$

3. t is **live in** M_0 if and only if

$$\forall M \in \mathscr{R}(M_0)\, \exists M' \in \mathscr{R}(M) : M' \xrightarrow{t}$$



Proposition 9.20. *Let $SG = (N_{SG}, A_{SG})$ be the SCC graph for the finite state space $SS = (N_{SS}, A_{SS})$ of a CP-net. Then the following holds:*

1. *A marking $M \in \mathscr{R}(M_0)$ is dead if and only if*

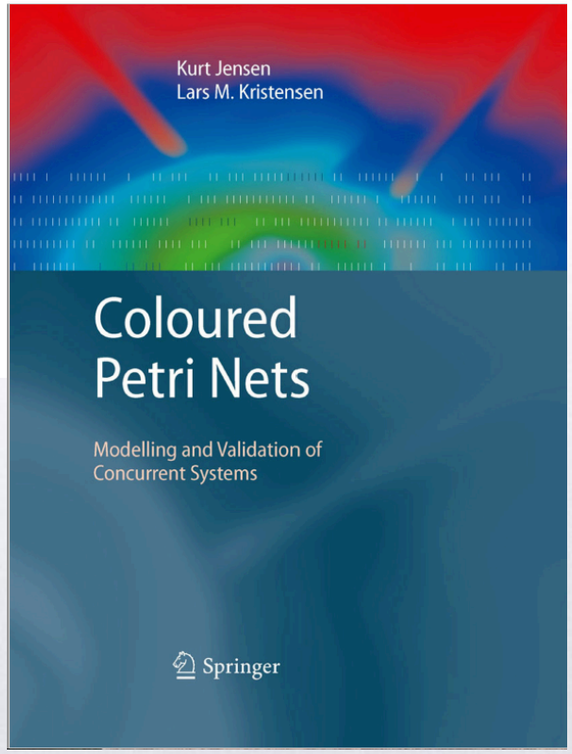
$$M \text{ is a terminal node of } SS$$

2. *A transition t is dead if and only if*

$$t \notin T(SS)$$

3. *A transition t is live if and only if*

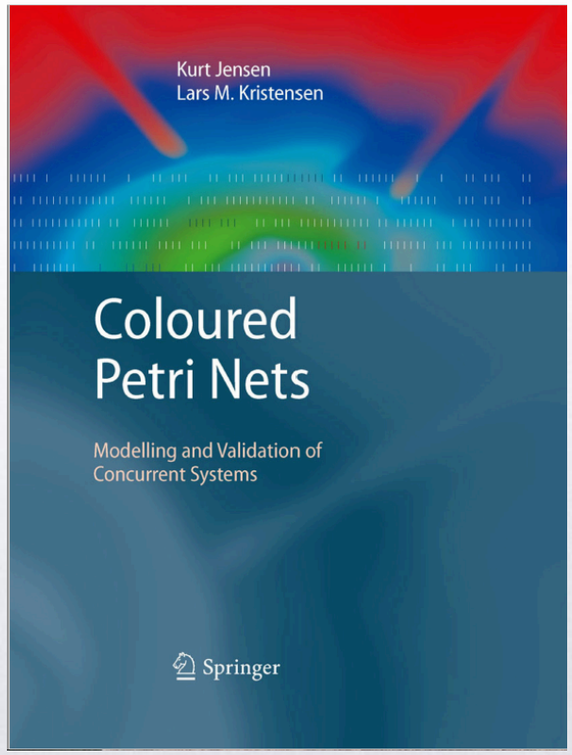
$$\forall S \in SCC_{TM} : t \in T(S)$$



1) *boundedness*, characterising finiteness of the state space.
2) *liveness*, related to potential fireability in all reachable markings. *Deadlock-freeness* is a weaker condition in which global infinite activity (i.e. fireability) of the net system model is guaranteed, but some parts of it may not work at all.
3) *reversibility*, characterising recoverability of the initial marking from any reachable marking.
4) *mutual exclusion*, dealing with the impossibility of simultaneous *submarkings* (p-mutex) or *firing concurrency* (t-mutex).

The proposition of algorithms and methods to detect home states in CPNs is still an open problem. However direct graphs can also be used to spot loops, which are essential (necessary condition) to reversibility.

The fairness properties give information about how often transitions occur in infinite occurrence sequences. We denote by OS_∞ the set of infinite occurrence sequences starting in the initial marking. For a transition $t \in T$ and an infinite occurrence sequence $\sigma \in OS_\infty$ we use $OC_t(\sigma)$ to denote the number of steps in which t occurs.



Requirement Analysis Method for Real World Systems in Automated Planning

Rosimarci Tonaco Basbaum[1] **and Tiago Stegun Vaquero**[2] and **José Reinaldo Silva**[1]
[1]Department of Mechatronic Engineering, University of São Paulo, Brazil
[2]Department of Mechanical & Industrial Engineering, University of Toronto, Canada
rosimarci@usp.br, tiago.vaquero@mie.utoronto.ca, reinaldo@usp.br

On the intelligent design field the requirement analysis phase has a fundamental role in automated planning-especially for "real life" systems - because it has the ability to identify or redesign variables which can potentially increase the model accuracy generated by the automated planner. A great effort has been made today in the area of Artificial Intelligence for defining reliable automated planning systems that can be applied for real life applications. That leads to the need for systematic design process, in which the initial phases are not neglected and where Knowledge and Requirement Engineering tools have a fundamental role for supporting designers. This paper intent to investigate design methods as well as perform a more detailed study on the adoption of UML and Petri Nets in the requirement analysis phase using the itSIMPLE framework as a KE tool.

Introduction

Planning characterizes a specific kind of design problem where the purpose is to find a set of admissible actions to

eling language and the place/transition Petri net to derive the UML diagrams. Throught the Petri net the requirement analysis will be performed using the available techniques, such as invariant analysis and equation state matrix (Murata 1989b). The first step detects contradictions and conflicts between the requirements, or the diferent view points in the diagrams. The second step identifies deep inconsistencies, that are hidden in the dinamic perspective of the plans, as well as additional information about the model that could be interpreted for the planners. Such informaton includes, new constraints, invariants, partial solution strategies, characteristics that could help to improve the planner aperformance.

The tool we choosed to use is the itSIMPLE framework (Vaquero et al. 2007b), which currently is not a 100% ready to performe the UML/Petri net translation, but it will be during the devopment of this project.

In the section 2 it will be discussed the advantages os UML in automated planning, followed by a brief description of Petri Nets, after that it will be presented a study case, the results and discussions and the conclusion.

UML for Design of Real Life Problems

Proceedings of the International MultiConference of Engineers and Computer Scientists 2016 Vol I,
IMECS 2016, March 16 - 18, 2016, Hong Kong

Formal Verification of Web Service Orchestration Using Colored Petri Net

C. Dechsupa, W. Vatanawood, and A. Thongtak

***Abstract*—Composite web service is typically comprised of a main orchestration description written in BPEL and its associated set of simple and individual web services. A common way to test the composite web service is to execute the BPEL along with its fully implemented version of the individual web services. Unfortunately, a particular web services orchestration in BPEL is difficult to be tested beforehand alone, without the availability of the related web services. In this paper, we propose an alternative mean to verify the web services orchestration in the early stage of the high level design process using the formal verification of the BPEL and its imitated stubs, called dummy web services. In our approach, a relevant dummy web service is generated to cope with its defined WSDL and the equivalence class partitioning technique is specifically focused. Thus, the valid and invalid classes of the invocation of the web services are simulated. The simple transformation rules of the BPEL alone, without the detailed of its associated individual web services, into Colored Petri Net are proposed. The resulting Colored Petri Net of a BPEL is correctly and consistently transformed and verifiable in CPN Tool using the equivalence class partitioning technique.**

***Index Terms*—Formal Verification, Dummy Service, Composite Web Service**

I. Introduction

PARADIGM for distributed computing, Service-Oriented Computing (SOC)**[1]** provides a manner to create new application architecture for cooperating services loosely connected, creating dynamic business processes and agile applications. The SOC promise requires the design of Service-Oriented Architectures (SOAs) that allow the established from several individual web services. Generally, these applications have a lot of sub-processes, data paths, input vectors, and state transitions that affect application complexity.

The standard for assembling a set of discrete services, the Web Services Business Process Execution Language (WS-BPEL) is used to model the behavior of both executable and abstract processes. All paths of service component architecture need to be aware of the business process, operations to execute, messages to exchange, and the timing of message exchanges. To ensure design, formal verification is the methodology for model checking that is the algorithmic analysis of programs to prove the properties of their executions.

Model checking tools are distinguished formal verification methods that can be ensured the correctness of application. They are used to detect and diagnose the faults in software and hardware. Currently, various verification tools have been used to verify a model, which they support automatic checking.

We propose methods for model abstraction and verification of composite web service application described as WS-BPEL. The WS-BPEL would be transformed into Colored Petri-Net model and we also introduce the dummy web service generation technique. The Colored Petri-Net tool is used to draw the resulting model and check the desired properties. The remaining of this paper is organized as follows: Section II describes background for web service verification. Section III reviews the related researches. Section IV discusses the proposed approach and Section V illustrates our implementation with a simple case study. Section VI is our conclusion.

II. Background

HOW TO FIND INVARIANTS FOR COLOURED PETRI NETS

Kurt Jensen
Computer Science Department
Aarhus University, Ny Munkegade
DK - 8000 Aarhus C, Denmark

Abstract: This paper shows how invariants can be found for coloured Petri Nets. We define a set of transformation rules, which can be used to transform the incidence matrix, without changing the set of invariants.

1. INTRODUCTION

In [2] coloured Petri Nets are defined as a generalization of place/transition-nets, and it is shown how to generalize the invariant-concept, [4], to coloured Petri nets. The elements in the involved matrices are no longer integers but functions, and matrix multiplication is generalized to involve composition/application of these functions. In [2] it is shown how to use invariants when proving various properties for the considered systems. In the present paper it will be shown how to find invariants by a sequence of transformations mapping the incidence matrix into gradually simpler matrices with the same set of invariants. The present paper is a continuation of [2], and it will use the definitions and notations from [2] without further explanation.

In section 2 we define four transformation rules, which can be used to transform the incidence matrix of a coloured Petri net. The four transformation rules are inspired by the method af Gauss-elimination, which is used for matrices, where all elements belong to a field. We prove that the transformation rules are sound, i.e. they do not change the set of invariants. The matrix elements for coloured Petri nets are not contained in a field, but only in a non-commutative ring, and thus division of two elements may be impossible. For this situation no general algorithm is known to solve homogeneous matrix equations. Thus we cannot expect our set of transformation rules to be complete, i.e. it is in general not possible to find all invariants only by means of the rules.

An Introduction to the Theoretical Aspects of Coloured Petri Nets

Kurt Jensen
Computer Science Department, Aarhus University
Ny Munkegade, Bldg. 540
DK-8000 Aarhus C, Denmark

Phone: +45 89 42 32 34
Telefax: +45 89 42 32 55
E-mail: kjensen@daimi.aau.dk

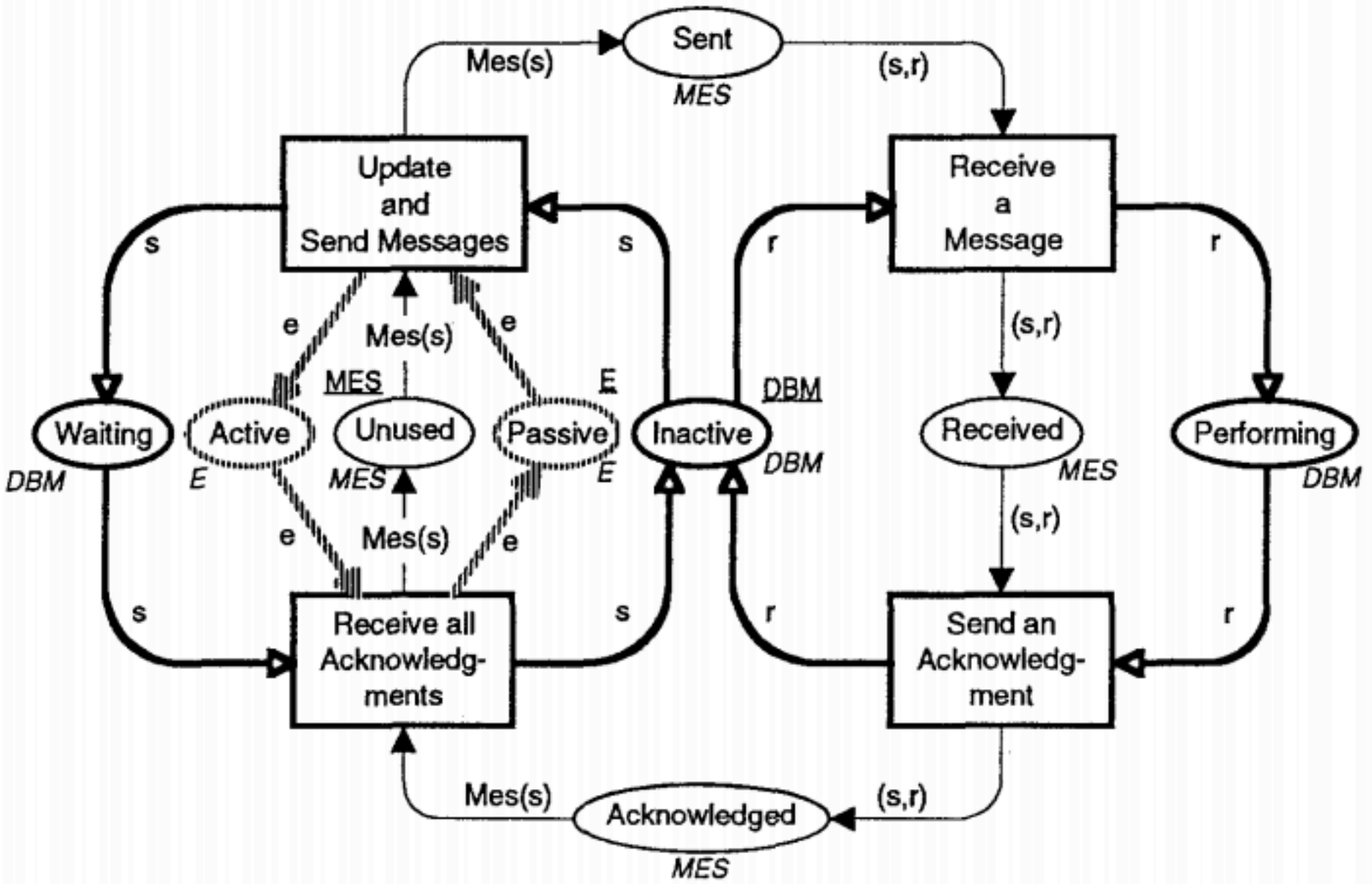
Abstract: This paper presents the basic theoretical aspects of Coloured Petri Nets (CP-nets or CPN). CP-nets have been developed, from being a promising theoretical model, to being a full-fledged language for the design, specification, simulation, validation and implementation of large software systems (and other systems in which human beings and/or computers communicate by means of some more or less formal rules). The paper contains the formal definition of CP-nets and their basic concepts (e.g., the different dynamic properties such as liveness and fairness). The paper also contains a short introduction to the analysis methods, in particular occurrence graphs and place invariants.

The development of CP-nets has been driven by the desire to develop a modelling language – at the same time theoretically well-founded and versatile enough to be used in practice for systems of the size and complexity that we find in typical industrial projects. To achieve this, we have combined the strength of Petri nets with the strength of programming languages. Petri nets provide the primitives for the description of the synchronisation of concurrent processes, while programming languages provide the primitives for the definition of data types and the manipulation of their data values.

The paper does not assume that the reader has any prior knowledge of Petri nets – although such knowledge will, of course, be a help.

Keywords: Petri Nets, High-level Petri Nets, Coloured Petri Nets.

```
val n = 5;
color DBM = index d with 1..n  declare ms;
color PR = product DBM * DBM  declare mult;
fun diff(x,y) = (x<>y);
color MES = subset PR by diff  declare ms;
color E = with e;
fun Mes(s) = mult'PR(1`s,DBM-1`s);
var s, r : DBM;
```



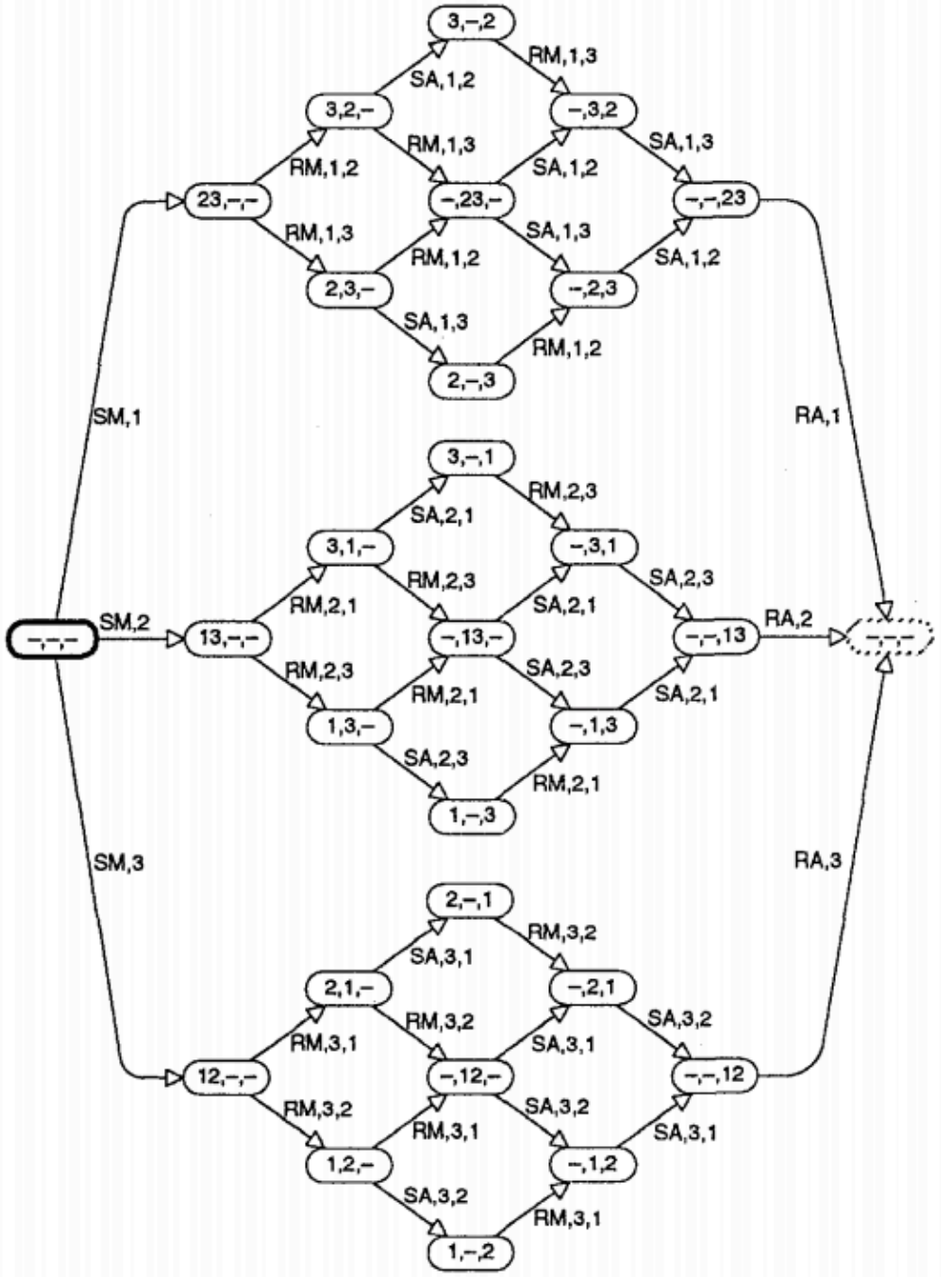


Fig. 2. Occurrence graph for data base system with 3 managers

The basic idea behind place invariants is to construct equations which are satisfied for all reachable markings. In the data base system we expect each manager to be either Inactive, Waiting or Performing.

This is expressed by the following equation satisfied for all reachable markings

M: M(Inactive) + M(Waiting) + M(Performing) = DBM.

If we now look at the messages properties...

M(Unused) + M(Sent) + M(Received) + M(Acknowledged) = MES

M(Active) + M(Passive) = l'e.

Each of the above equations can be written on the form:

$$W_{p_1}(M(p_1)) + W_{p_2}(M(p_2)) + \ldots + W_{p_n}(M(p_n)) = m_{inv}$$

where $\{p_1, p_2, \ldots, p_n\} \subseteq P$. Each **weight** W_p is a function mapping from the type of p into some common type $A \in \Sigma$ shared by all weights. Finally m_{inv} is a multi-set. It can be determined by evaluating the left-hand side of the equation in the initial marking (or in any other reachable marking).

Definition 7.1: Let $A \in \Sigma$ be a type and let $W = \{W_p\}_{p \in P}$ be a set of linear functions such that $W_p \in [C(p)_{ws} \rightarrow A_{ws}]$ for all $p \in P$.

(i) W is a **place flow** iff:

$$\forall (t,b) \in BE: \sum_{p \in P} W_p(E(p,t)<b>) = \sum_{p \in P} W_p(E(t,p)<b>).$$

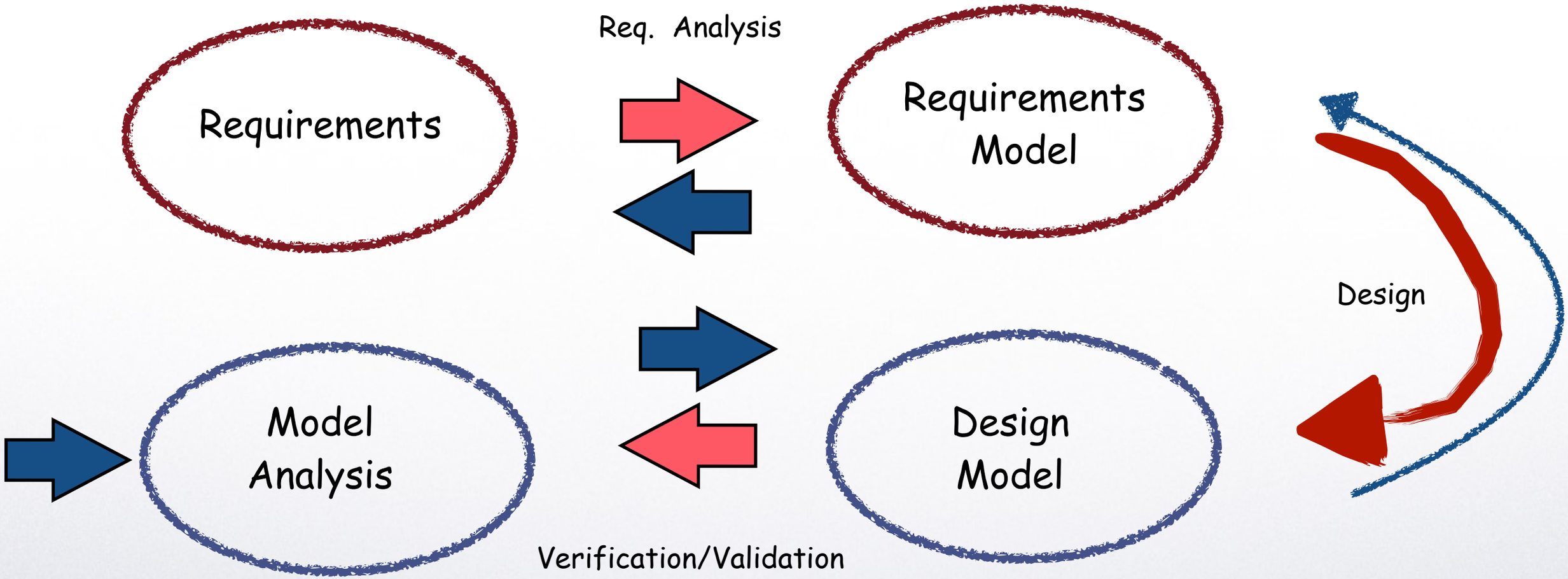
(ii) W determines a **place invariant** iff:

$$\forall M \in [M_0>: \sum_{p \in P} W_p(M(p)) = \sum_{p \in P} W_p(M_0(p)).$$

Invariants now mens to preserve the type of marks, or to establish a common type to be preserved in different places.

Still, there is no well accepted "algorithm" to perform invariant calculus in Coloured Petri Nets.

Use of Petri Nets in Design



There are invariants that could be inferred at the very beginning, it does not matter is we are using P/T or CPN. We will call it "**design invariants**".

Other invariants can be only calculated (specially in large projects) and then there is the problem of synthesising a P/T net - or unfolding the net - before doing it. We will call that **operational invariants**.

Tradeoff

- **More information in tokens**
  - color sets, functions, etc.
  - behavior may be hidden in “code”
  - extreme case: all behavior folded into one place and one transition
- **More information in network**
  - possibly spaghetti networks to encode simple things
  - behavior may be incomprehensible
  - cannot be parameterized
  - extreme case: (infinite) classical Petri net

What is a model?

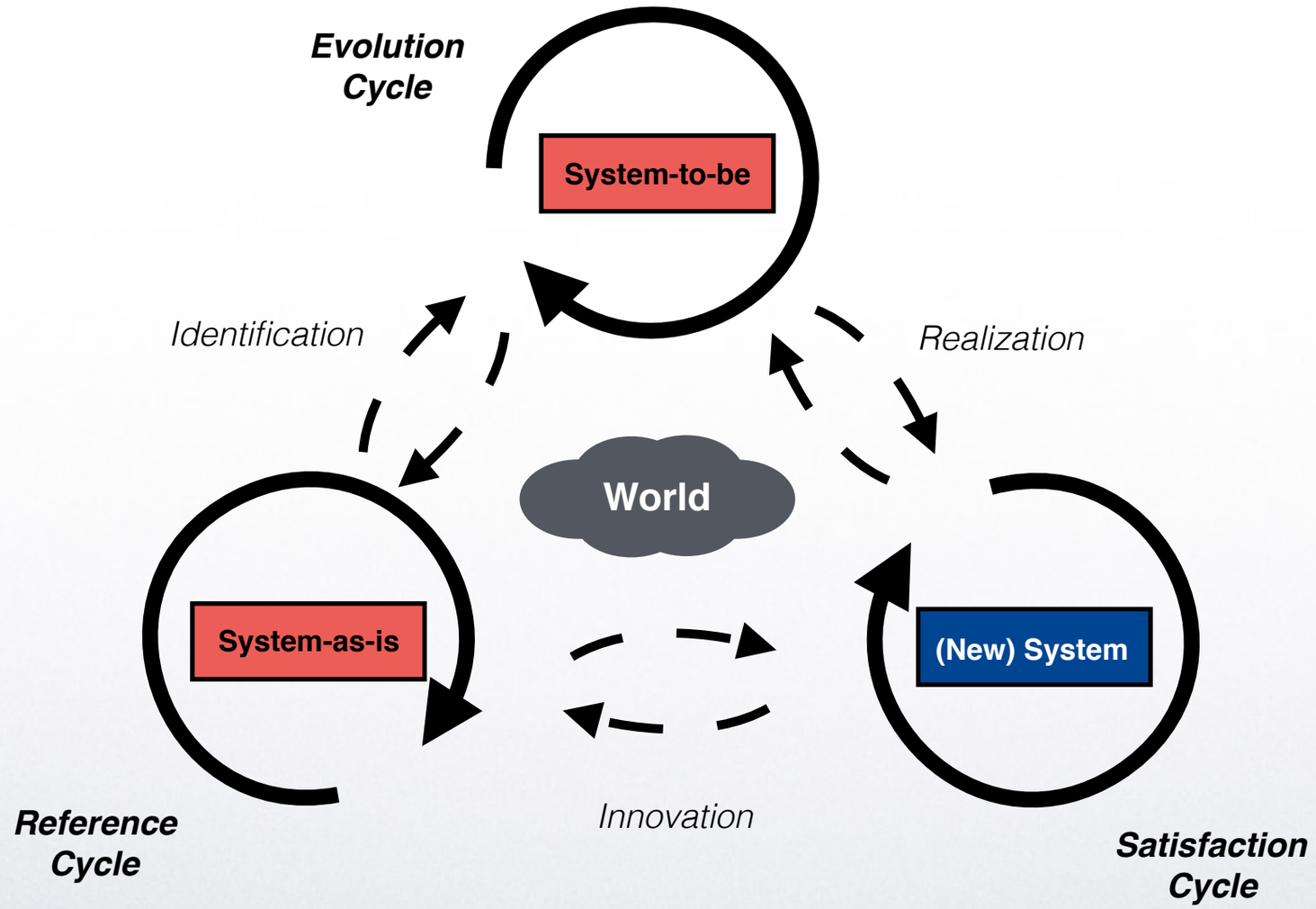
model and modelling
in painting, the *use of light and shade to simulate volume in the representation of solids*. In sculpture the terms denote a technique involving the use of a pliable material such as clay or wax. As opposed to carving, modelling permits addition as well as subtraction of material and lends itself to freer handling and change of intention. The technique is exemplified also by those works in cast metal and plaster that are made from the mold of a clay original. The mold is made by the process of cire perdue. The noun model is used to describe such an original and also *any three-dimensional scale model for a larger or more elaborate project in architecture, landscaping, or industry*. It also denotes a person or object used as an aid to representation in painting.

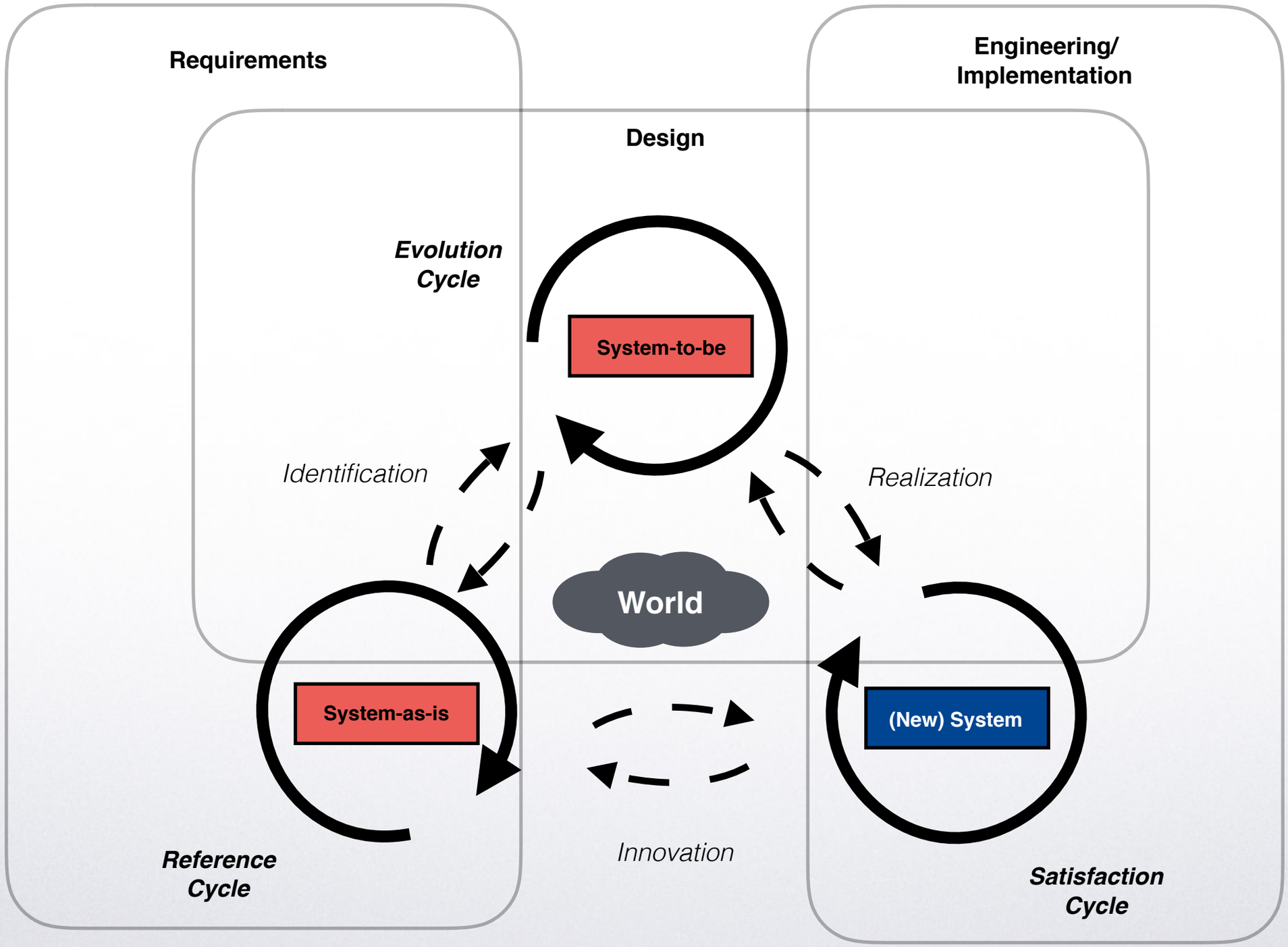
The Columbia Encyclopaedia, Sixth Edition. 2001.

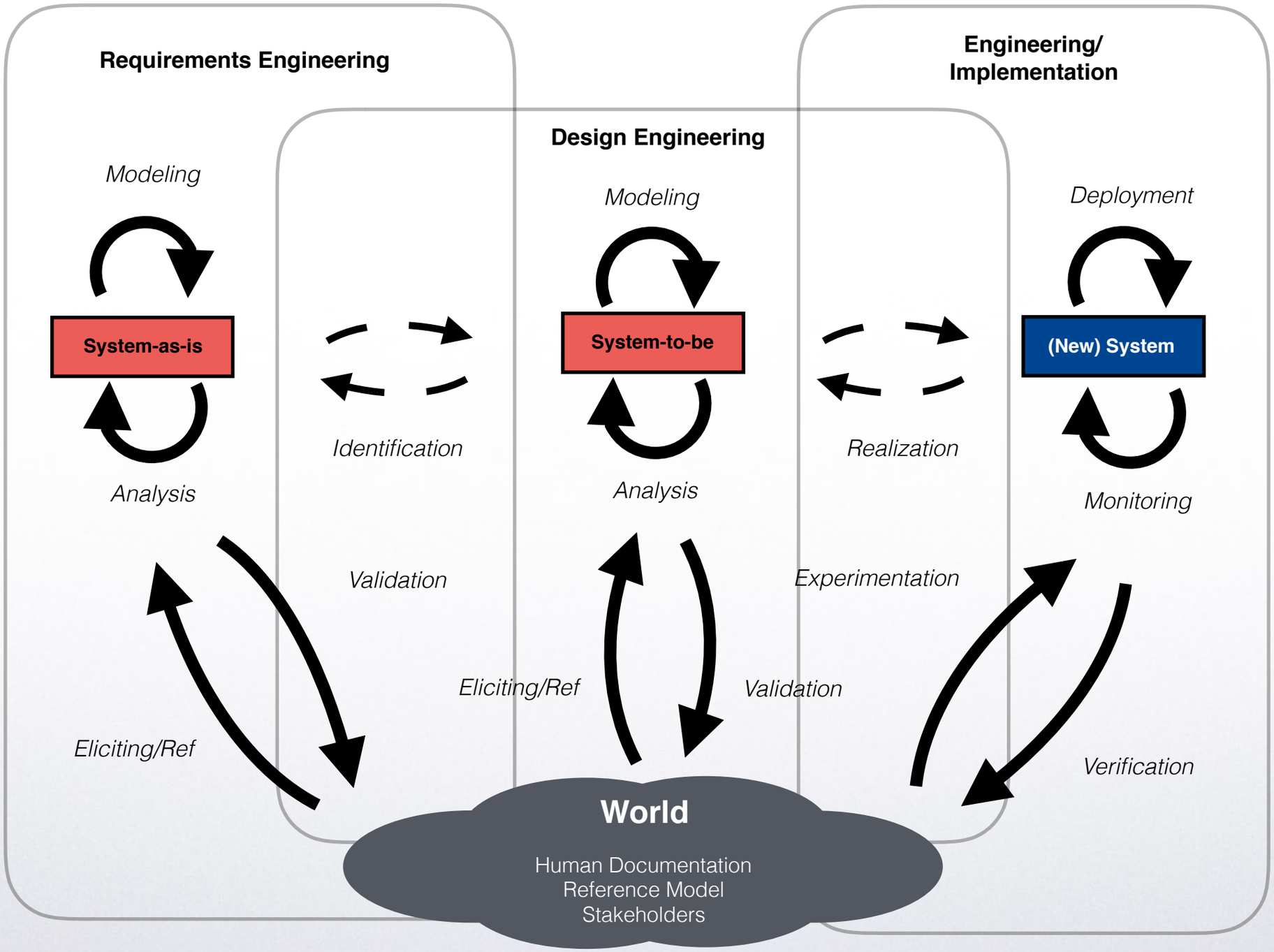
Abstract representation, scale model of future design

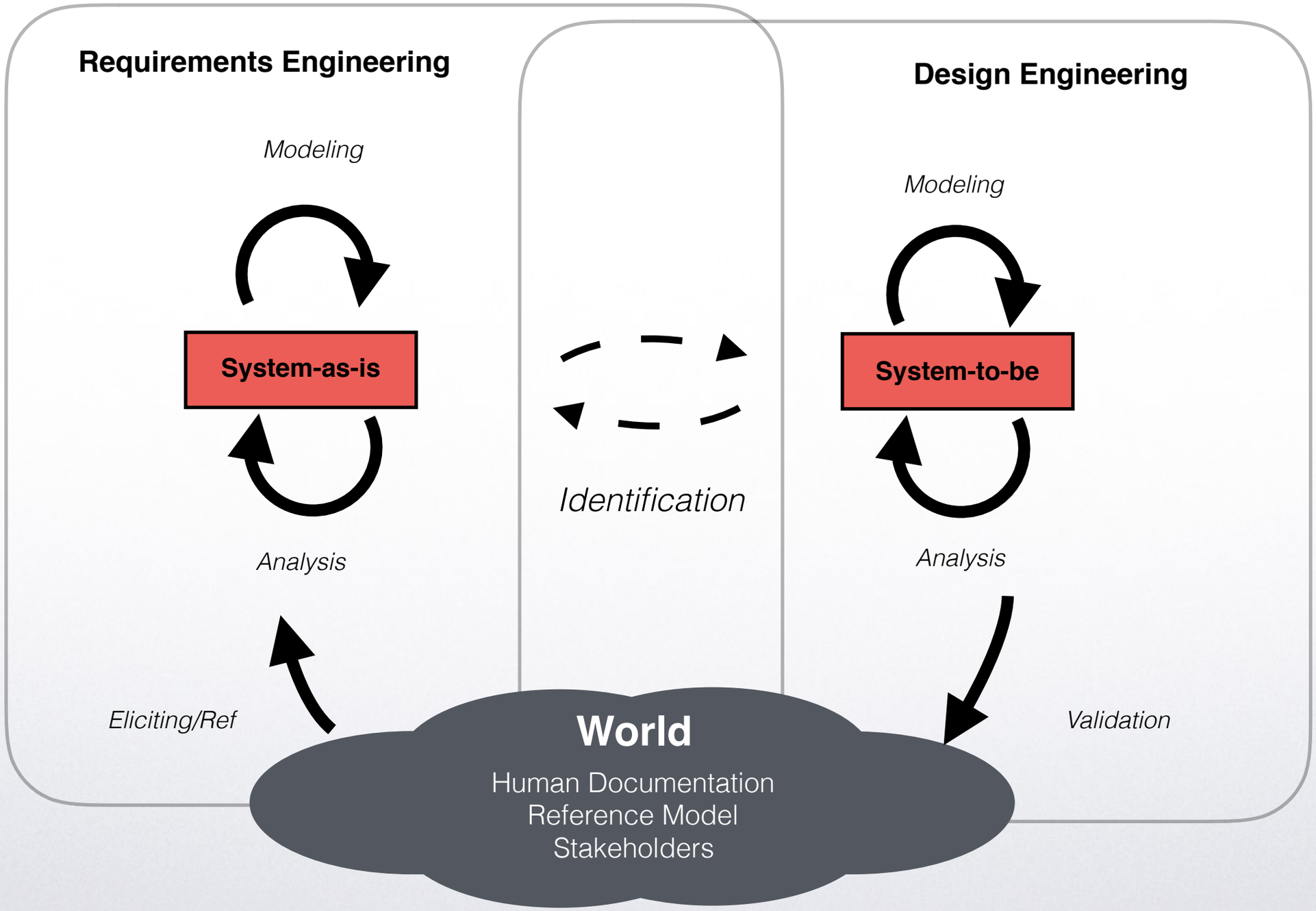
August 27, 2002 Søren Christensen, MOCA'02 7

Systems Design and Petri Nets




Requirements
Engineering/
Implementation
Design
Evolution
Cycle
System-to-be
Identification
Realization
World
System-as-is
(New) System
Reference
Cycle
Innovation
Satisfaction
Cycle


Requirements Engineering
Design Engineering
Engineering/
Implementation
Modeling
System-as-is
Analysis
Identification
Validation
Eliciting/Ref
Modeling
System-to-be
Analysis
Eliciting/Ref
Validation
Realization
Experimentation
Deployment
(New) System
Monitoring
Verification
World
Human Documentation
Reference Model
Stakeholders


Requirements Engineering
Modeling
System-as-is
Analysis
Identification
Design Engineering
Modeling
System-to-be
Analysis
Eliciting/Ref
Validation
World
Human Documentation
Reference Model
Stakeholders

Next class will discuss High Level Petri nets, or HLPN, stressing CPN as a special case of that.

Theoretical aspects must be developed in HLPN basis (and eventually applied to CPN) to fit the standard ISO/IEC 15.909.

Fim