PA160: Net-Centric Computing II.

Specification and Verification of Network Protocols

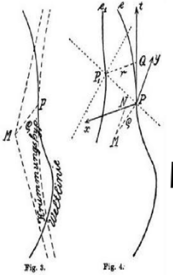
Vojtěch Řehák

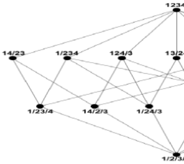
Faculty of Informatics, Masaryk University

Spring 2017

Theory Definition and Usage

Theory





Theoretical Results as Tools for Users

Formal Models - what are they good for

The basic concept is a *model* of system (i.e. the object we work with).

thought handling

- individual approach (intra-brain) - to grasp it
- documentation (inter-brain) - to pass it on

automatic/computer processing (comparing model to specification)

- testing
- simulation
- symbolic execution
- static analysis
- model checking
- equivalence checking
- theorem proving

Formal Models in Specification

natural language **vs.** formal language

natural language			formal language
freedom in writing	+	−	restrictions in writing
human resources	−	+	automatic methods

nothing $\leq$ text $\leq$ structured text $\leq$ text with formal “pictures” $\leq$
formal description with informal comments $\leq$ complete formal description

Goal: Find an appropriate level of abstraction and keep it.

“What will be the model used for?”

Map - Abstraction Example

Find Pardubice or directions from Brno to Liberec.

source: www.mapy.cz

Outline

Models we will talk about:

- Message Sequence Charts (MSC)
- Specification and Description Language (SDL)
- Petri nets
- Queueing theory

What they can be used for?

- modelling
- specification
- analysis
- simulation
- testing
- partial implementation

Distributed Systems

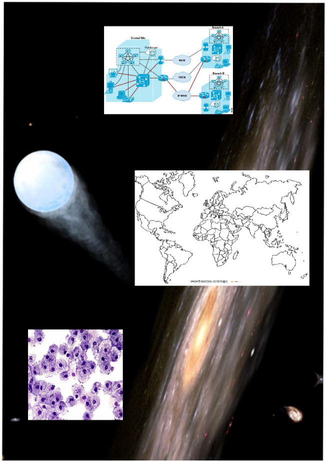
“What is the problem of distributed systems?”

Distributed Systems – Key characteristics (Déjà vu)

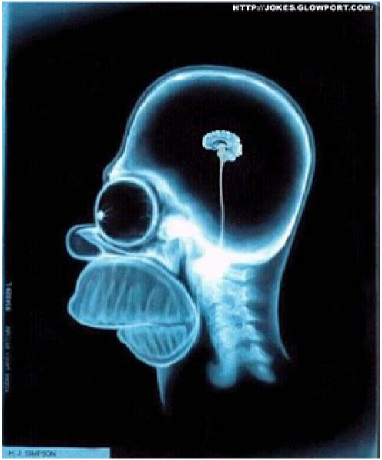
Key Characteristics of Distributed Systems:

- **Autonomicity** – there are several autonomous computational entities, each of which has its own local memory
- **Heterogeneity** – the entities may differ in many ways
  - computer HW (different data types' representation), network interconnection, operating systems (different APIs), programming languages (different data structures), implementations by different developers, etc.
- **Concurrency** – concurrent (distributed) program execution and resource access
- **No global clock** – programs (distributed components) coordinate their actions by exchanging messages
  - message communication can be affected by delays, can suffer from variety of failures, and is vulnerable to security attacks
- **Independent failures** – each component of the system can fail independently, leaving the others still running (and possibly not informed about the failure)
  - How to know/differ the states when a network has failed or became unusually slow?
  - How to know if a remote server crashed immediately?

Distributed Systems



World is distributed



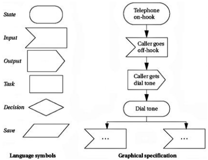
Human way of thinking is sequential

Distributed vs. Local

SDL

Specification Description Language

ITU-T Z.100

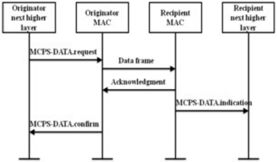


models of components

MSC

Message Sequence Chart

ITU-T Z.120



communication model

Message Sequence Chart (MSC)

Message Sequence Chart (MSC)

international standard of ITU-T, Z.120

- 1993 - first version of Z.120 recommendation
- ...
- 2011 - current version of Z.120 recommendation
- all documents of the current version:
  - Z.120 - Message Sequence Chart (MSC)
  - Z.120 Annex B - Formal semantics of message sequence charts
  - Z.121 - Specification and Description Language (SDL) data binding to Message Sequence Charts (MSC)

It formally defines both textual and graphical form.

MSC is a similar concept to UML Sequence Charts.

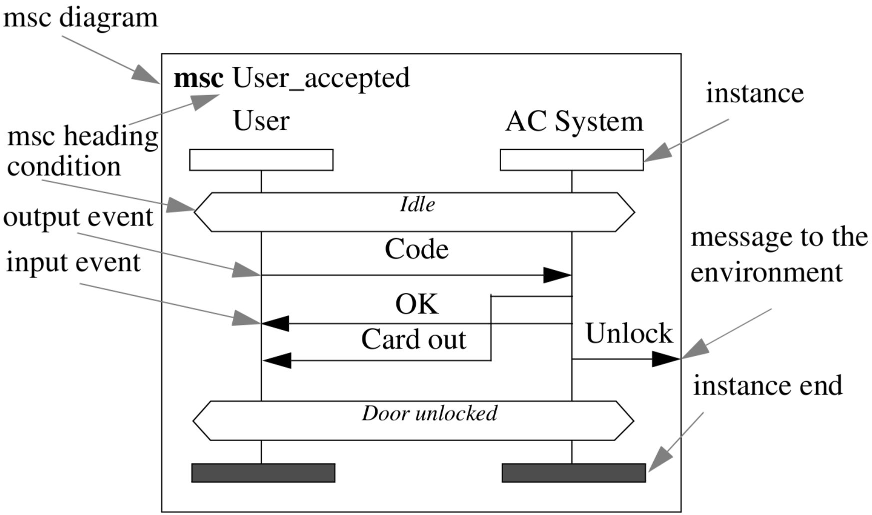
Message Sequence Chart (MSC)

A **trace language** for the specification and description of the **communication behaviour** by means of **message exchange**.

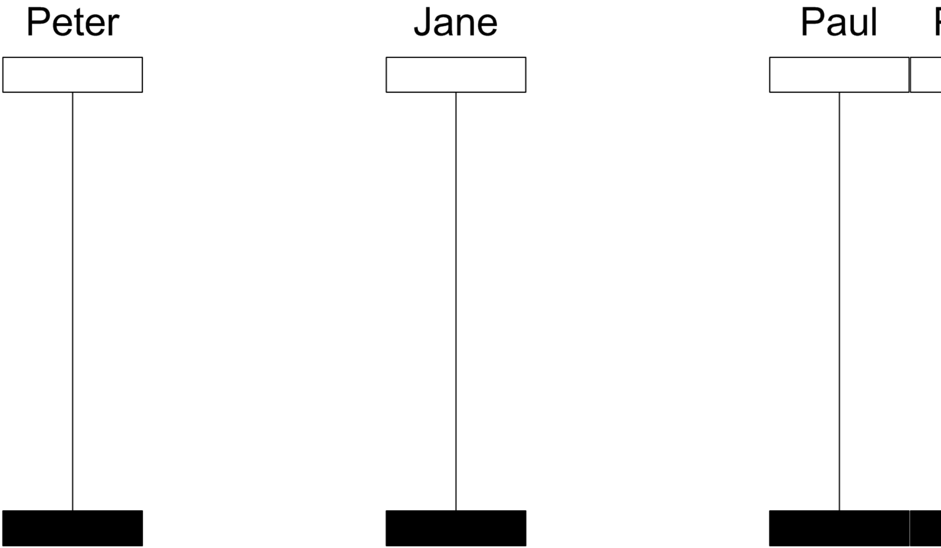
Describes

- communicating processes,
- communication traces,
- message order,
- time information (timeouts, constraints),
- high-level form for set of traces.

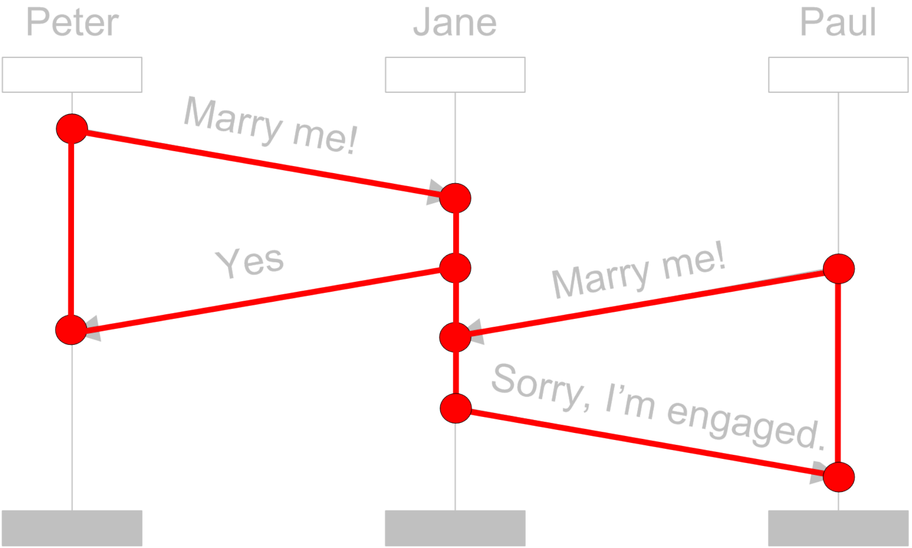
Message Sequence Chart (MSC)



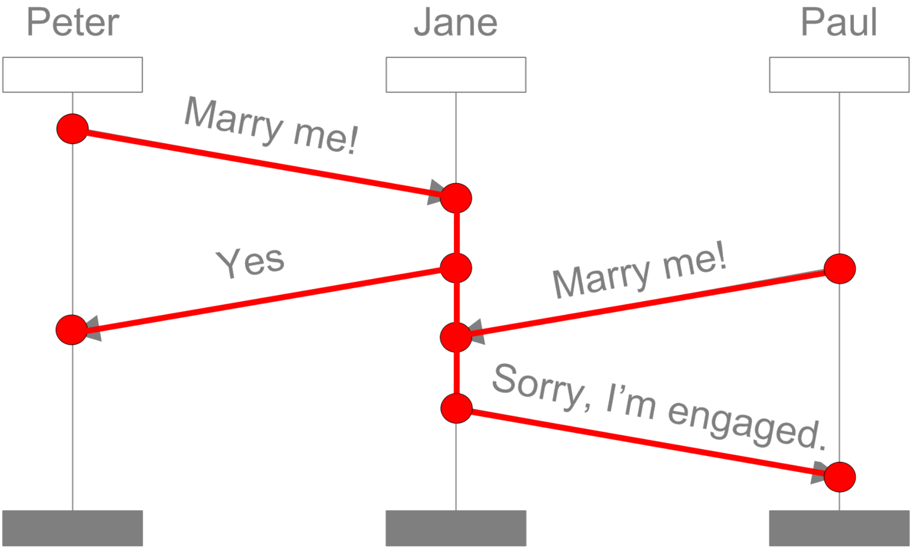
Message Sequence Chart (MSC) - semantics



MSC - Visual Order



MSC - Visual Order - Hasse Diagram



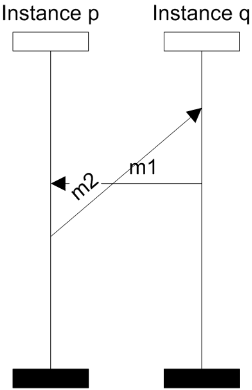
MSC Properties

What is an unwanted behaviour/property?

Fundamental problems in the specified model, e.g. an implementation of the model does not exist in the given environment.

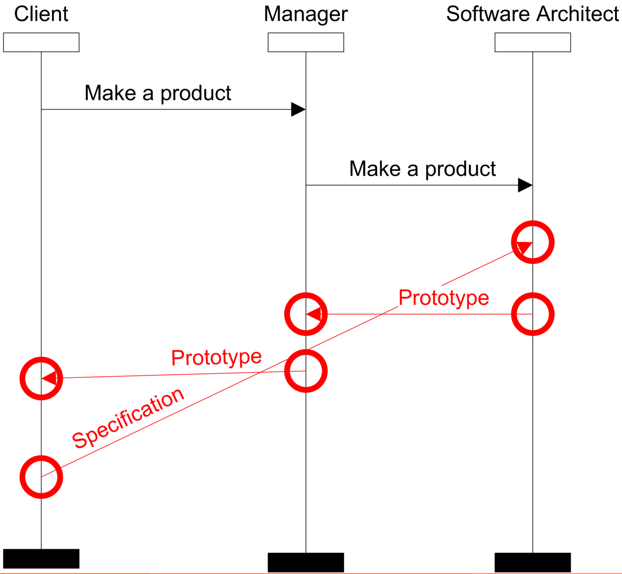
Acyclic/Cyclic property

cyclic dependency among events



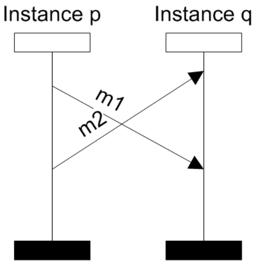
unrealizable in any environment

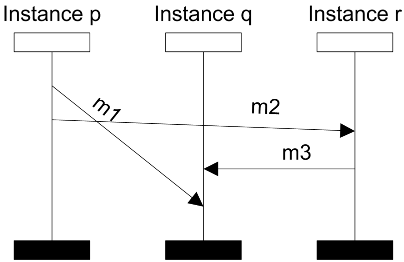
Acyclic/Cyclic property



FIFO/non-FIFO property

overleaping messages

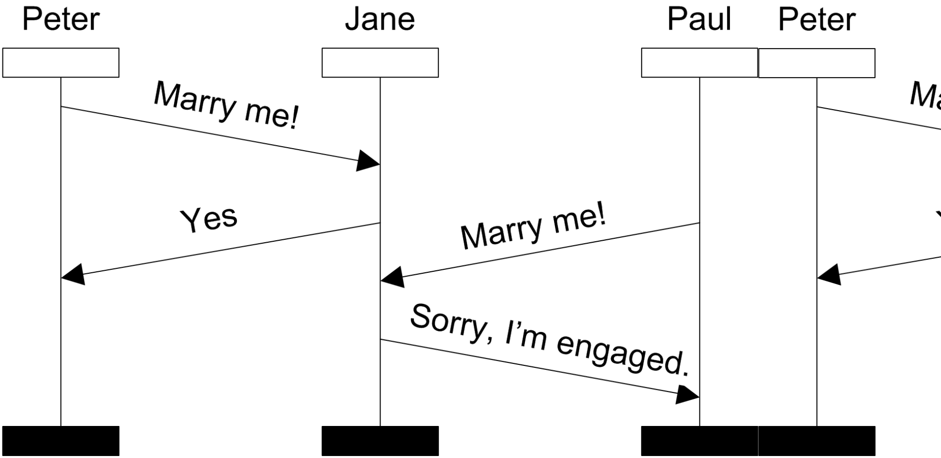




unrealizable in an environment preserving message order

realizable in an environment with P2P channels but unrealizable in case of one global channel

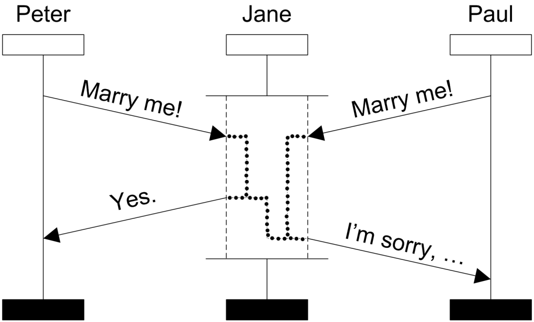
Race Condition



Informally, **race** is when some receive event can come earlier.

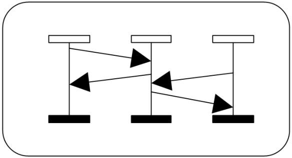
Solution #1 - Coregion Construction

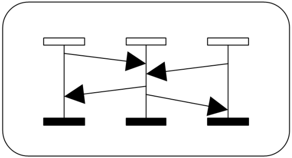
Let us demonstrate that some events are not ordered.

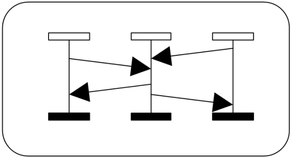


Events in a *coregion* are not order;
except for the event related by *general ordering*.

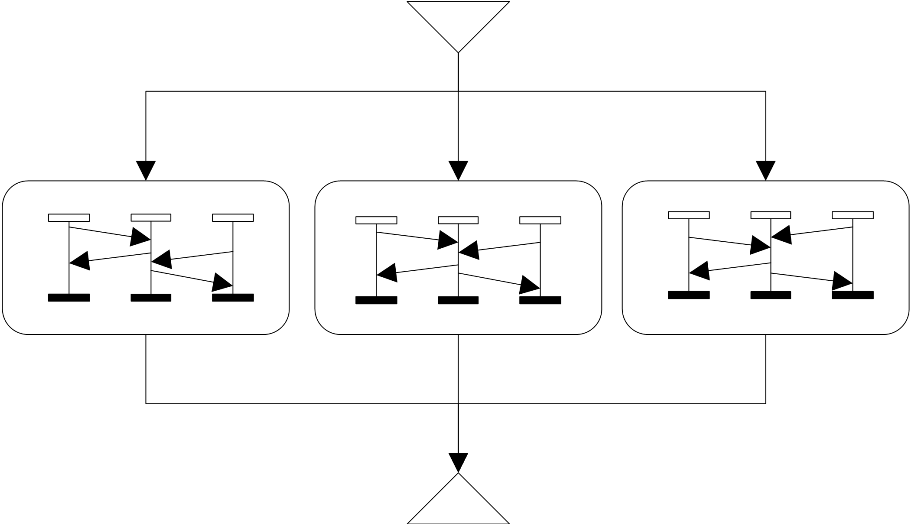
Solution #2 - List/set of all possibilities



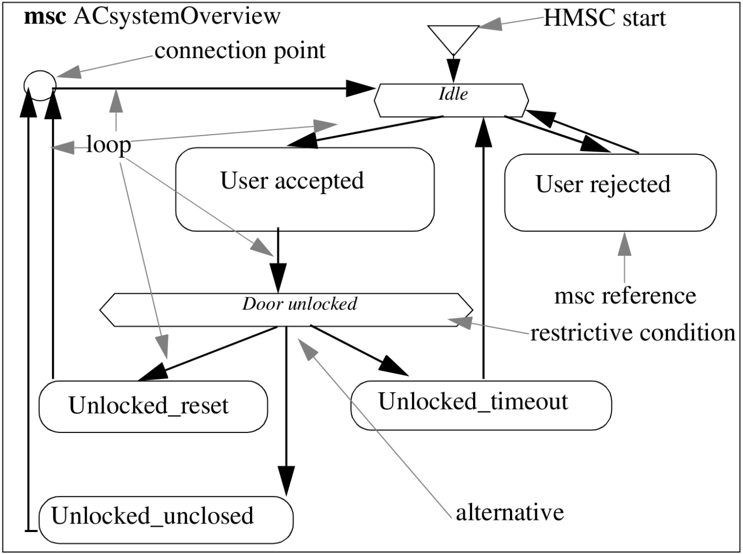




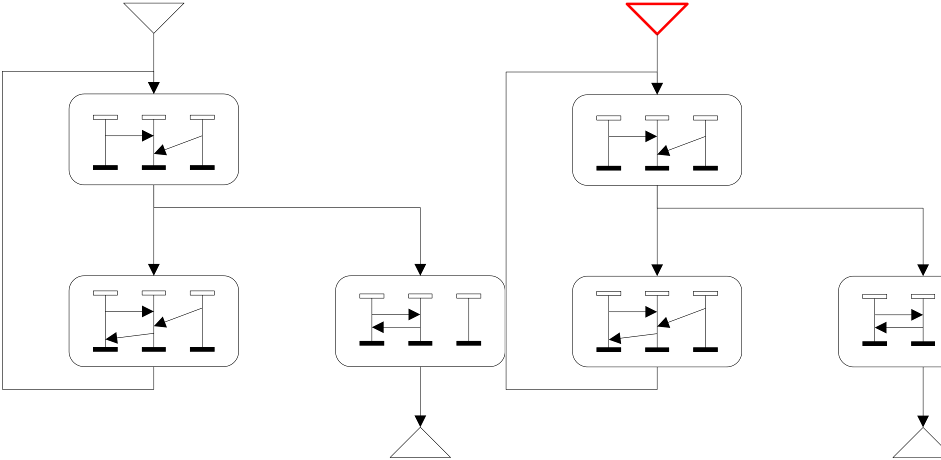
High-Level MSC (HMSC)



High-Level MSC (HMSC) - ITU-T Z.120

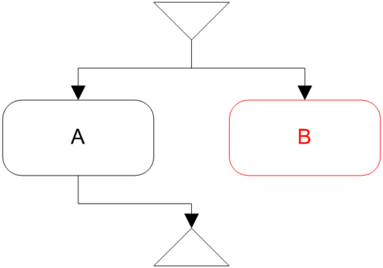


High-Level MSC (HMSC) - ITU-T Z.120

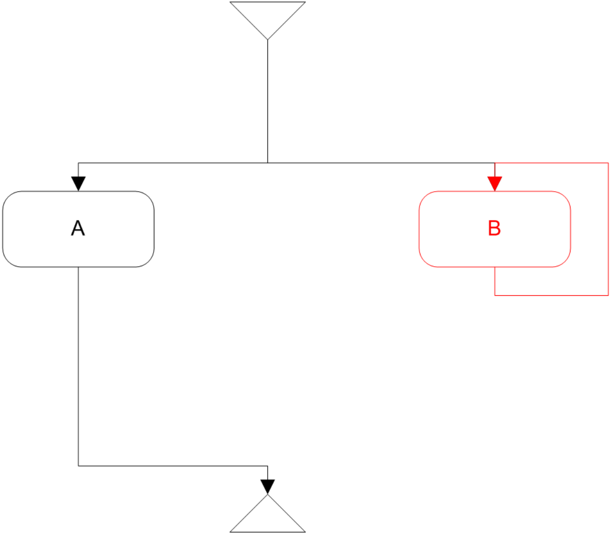


these events are not ordered!

Deadlock Property

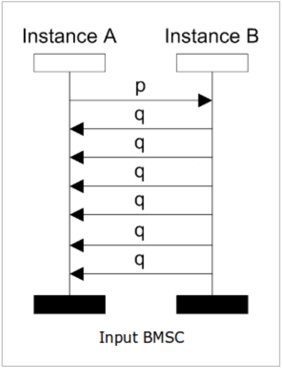


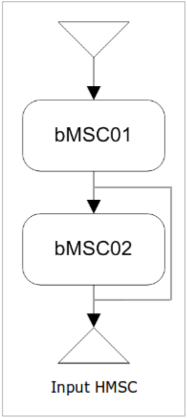
Livelock Property

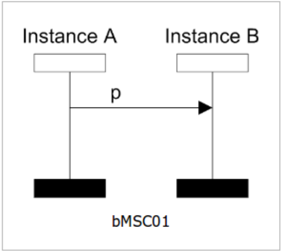


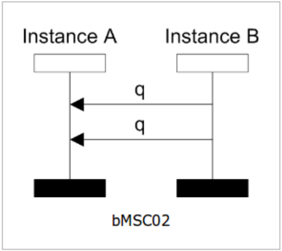
Membership

Is a given MSC included in a given HMSC?

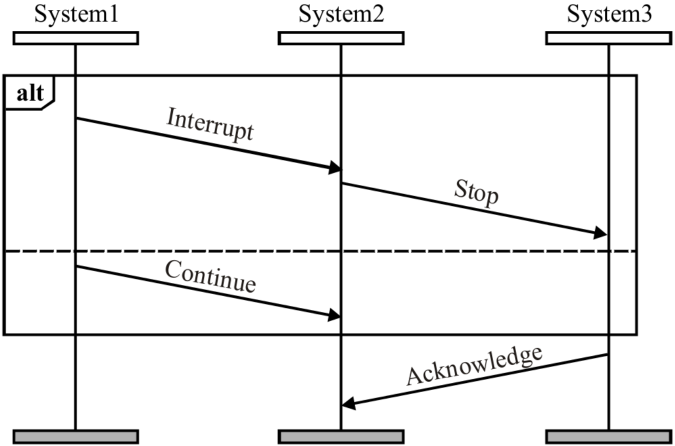






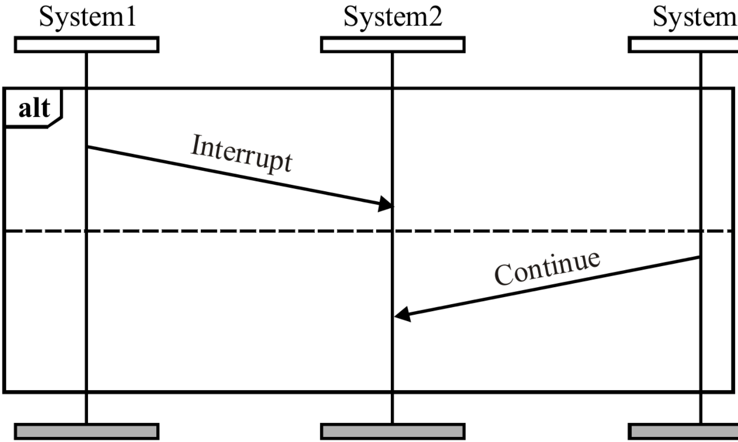


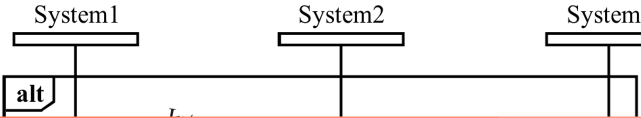
Inline Expressions



Other inline expression types: **opt**, **loop**$\langle m, n \rangle$, **exc**, **seq**, **par**.

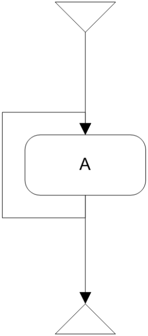
Non-local Choice

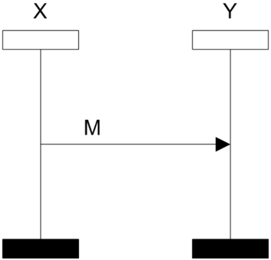




Universal Boundedness

What is the size of input buffer of Y that will never overflow?

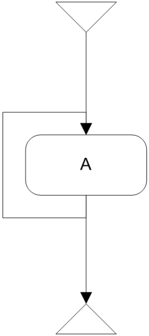


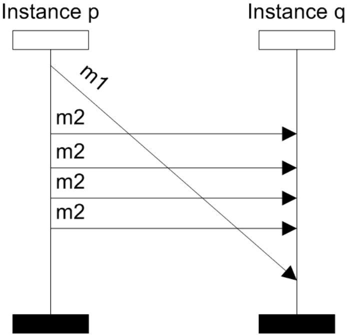


Every finite input buffer of Y can overflow.

Existential Boundedness

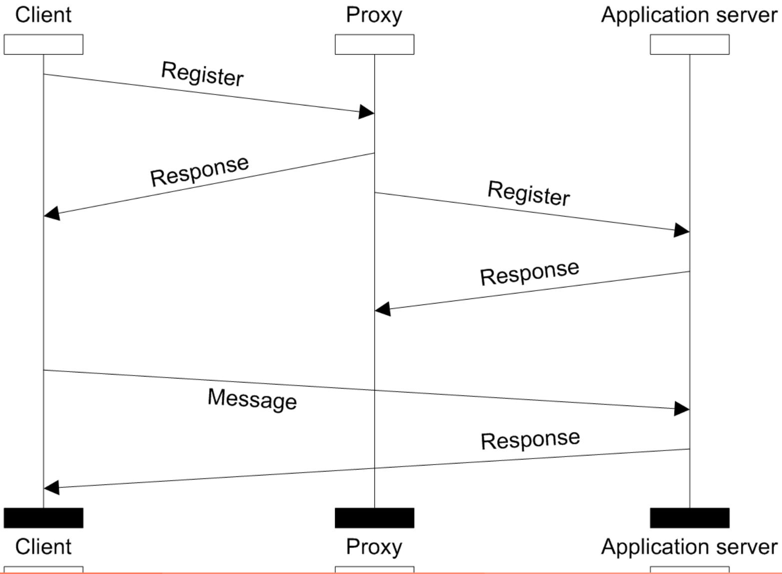
The system deadlocks in case of FIFO channels (and FIFO buffers).
What is the size of non-FIFO buffer needed to avoid deadlock (in case of FIFO channels)?





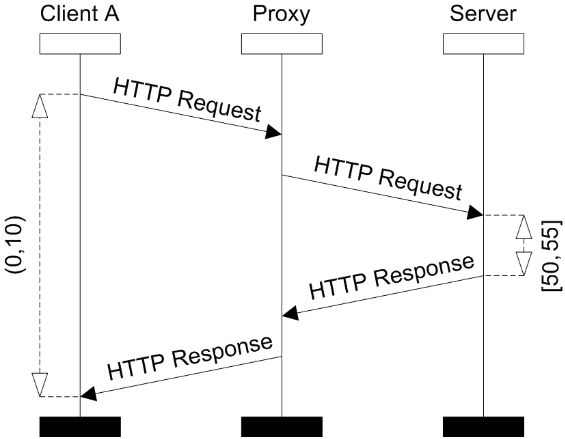
Buffer of size 2 suffices to avoid deadlock.
Or one buffer for each message label (type).

Race Condition - Solution #3 - Time Constraints



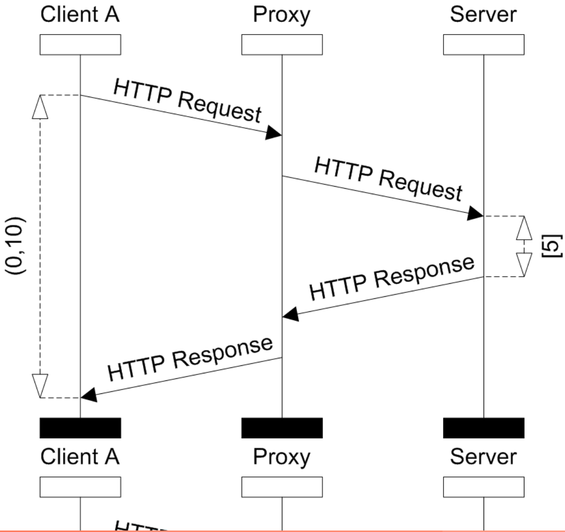
Time Consistency

Are the given time conditions consistent?

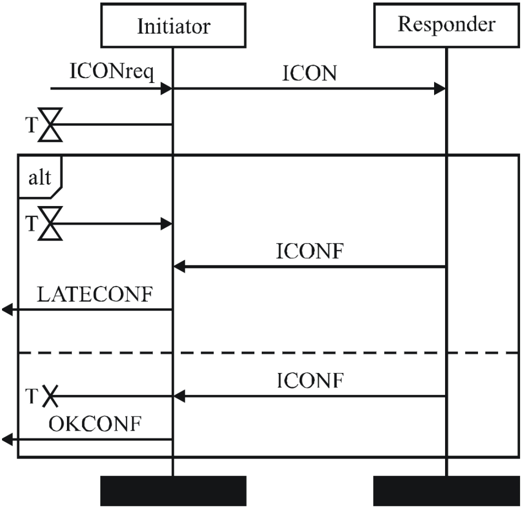


Time Tightening

Some time conditions can be tightened.



Timers


Initiator
Responder
ICONreq
ICON
T
alt
T
ICONF
LATECONF
ICONF
T
OKCONF

MSC - Summary

Basic MSC

- instances
- messages
- send events
- receive events
- conditions
- coregions
- general ordering
- inline expressions
- time constraints
- timers

High-level MSC (HMSC)

- start node
- end node
- reference nodes
- connection points
- lines
- conditions
- time constraints

MSC - Properties

- Acyclic property
- FIFO property
- Race Condition

- Deadlock
- Livelock
- Membership
- Nonlocal Choice
- Universal Boundedness
- Existential Boundedness

- Time Race Condition
- Time Consistency
- Tighten Time

MSC - Goals

What MSC is good for?

Both **human** and computer readable formalizm for:

- basic behaviour demonstration (use cases),
- high level system behaviour description,
- test case specification, and
- (test) log visualization.

What MSC is NOT good for?

detailed specification (before implementation), hierarchical structure of communicating entities, implementation details (primitives for communication, detailed data manipulation), etc.

MSC - Tools

Mesa

- academic tool
- local choice and time checkers

MSCan

- academic tool
- only textual input
- some checkers

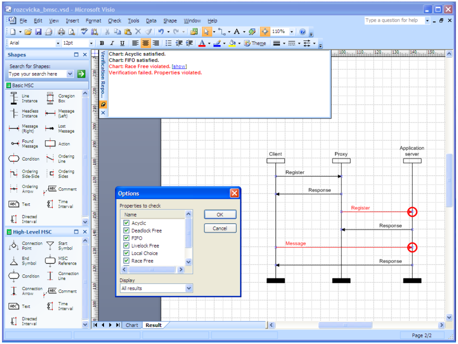
IBM Rational, SanDriLa SDL, Cinderella SDL

Sequence Chart Studio (SCStudio)

- MS Visio addon
- drawing, import, export
- checkers for all the mentioned properties

Sequence Chart Studio

MSC drawing and verification tool developed at FI MU.



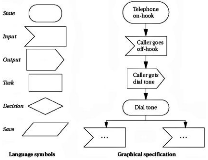
http://scstudio.sourceforge.net

Distributed vs. Local

SDL

Specification Description Language

ITU-T Z.100

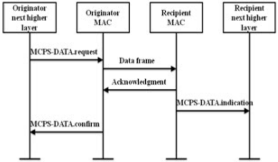


models of components

MSC

Message Sequence Chart

ITU-T Z.120



communication model

Specification Description Language (SDL)

Specification Description Language (SDL)

international standard of ITU-T, Z.100

- 1972 - Establishment of a working group for SDL
- 1976 - first version of Z.100 recommendation
- ...
- 12/2011 - current version of Z.100 recommendation
- all documents of the current version:
  - Z.100 - Specification and Description Language (SDL)
  - Z.100 Supplement 1 - SDL+ methodology: Use of MSC and SDL
  - Z.Imp100 - SDL implementer's guide
  - Z.101 - SDL - Basic SDL-2010
  - Z.102 - SDL - Comprehensive SDL-2010
  - Z.103 - SDL - Shorthand notation and annotation in SDL-2010
  - Z.104 - SDL - Data and action language in SDL-2010
  - Z.105 - SDL - SDL-2010 combined with ASN.1 modules
  - Z.106 - SDL - Common interchange format for SDL
  - Z.108 - SDL - Object-oriented data in SDL-2010
  - Z.109 - SDL - UML profile for SDL-2010

SDL - Specification Description Language

An **object oriented** languages for specification of applications that are

- heterogeneous,
- distributed (concurrent),
- interactive (event-driven, discrete signals), and
- real-time dependent (with delays, timeouts).

Describes

- structure (distributed components of the system),
- behaviour (instructions within the components), and
- data

of distributed systems in **real-time** environments.

SDL - representations

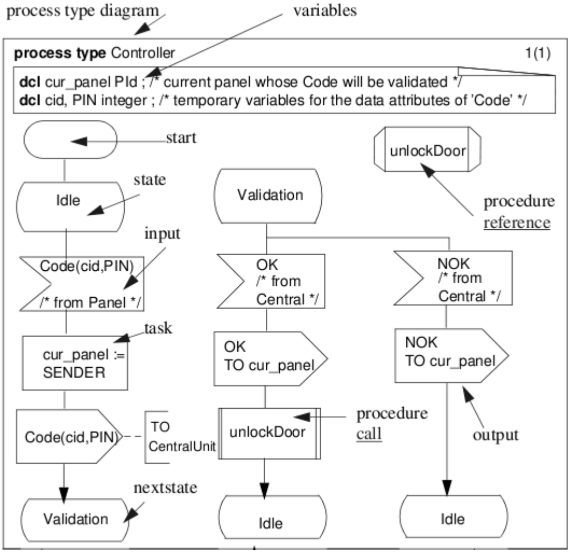
Three representations:

- **SDL/GR** graphical representation (human readable)
- **SDL/PR** textual phrase representation (machine readable)
- **SDL/CIF** common interchange format (SDL/PR with graphical information)

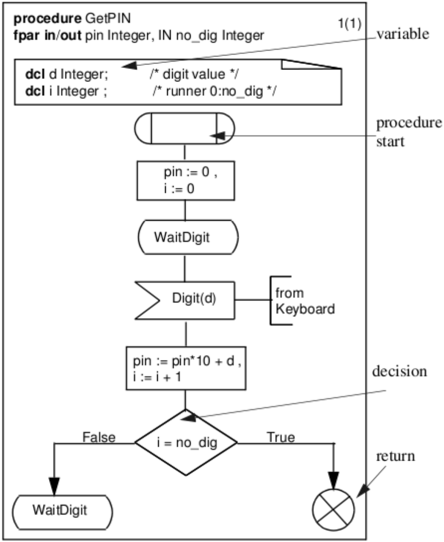
In what follows, we focus on the graphical representation (SDL-GR).
Basic SDL components

- system and blocks (structure)
- processes and procedures (behaviour)

SDL/GR - Process

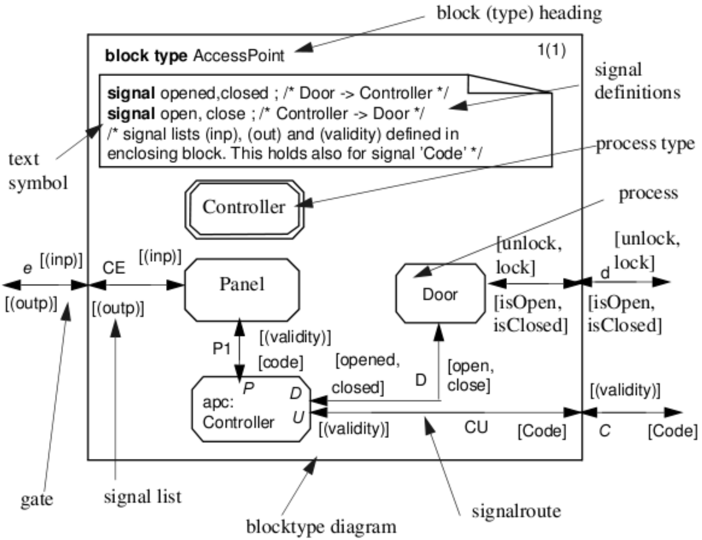


SDL/GR - Procedure



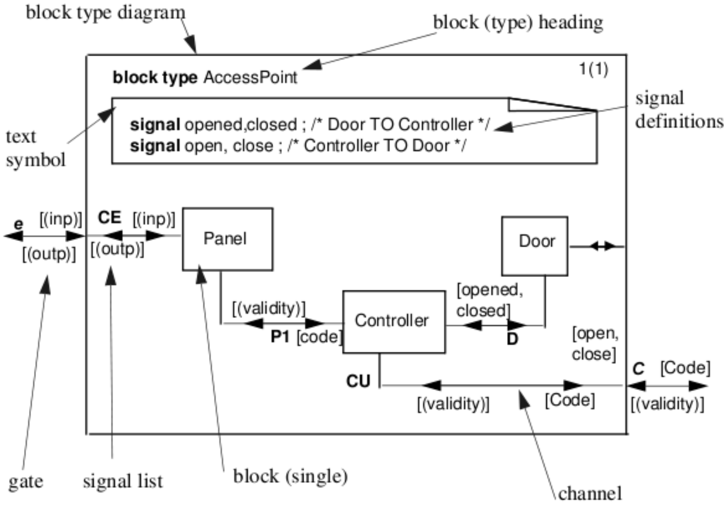
source: TIMe Electronic Textbook v4.0

SDL/GR - Block



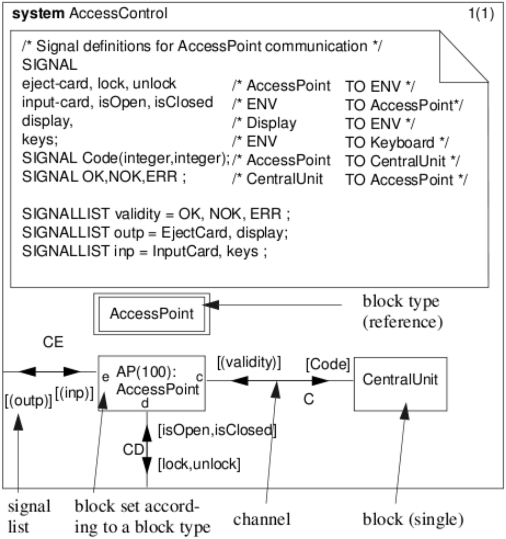
source: TIMe Electronic Textbook v4.0

SDL/GR - Block with block structure



source: TIMe Electronic Textbook v4.0

SDL/GR - System (the top most block)



source: TIMe Electronic Textbook v4.0

SDL/GR - Channels

Nondelaying channels for “immediate” message delivery (e.g., between processes within a computer).

Delaying channels for “time consuming” message delivery (e.g., between dislocated blocks).

Channels can also be one-directional.

Summary of SDL Basics

System - is the top most block surrounded by environment.

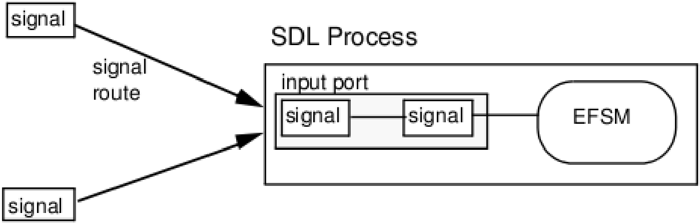
Block - consists of blocks or processes that are connected by channels.
- expresses the hierarchical structure of the system.
- its names are references to other objects.

Process - sends and receives messages.
- stays in states.
- can call procedures.

Procedure - is a subroutine that can finish.
- does not return any value (only in variables or sent messages).

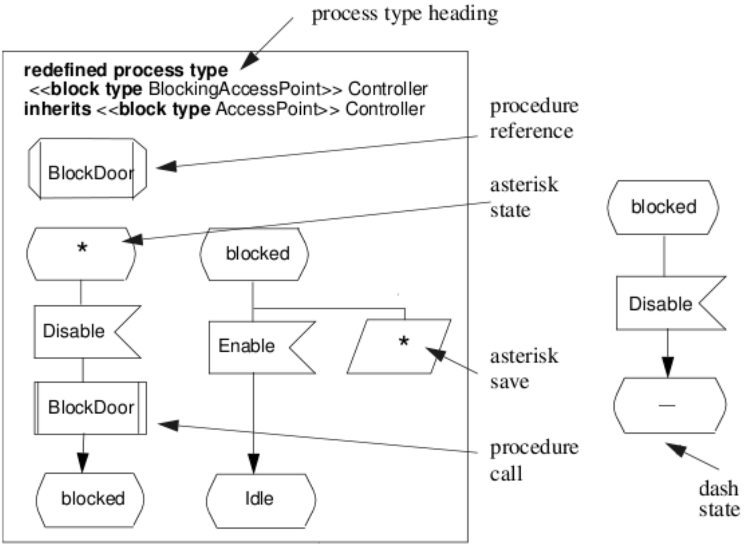
Message Exchange - Operational Semantics

- one input buffer for a process
- FIFO behaviour
- no priority queues
- signal which is unspecified in the current state is **discarded**



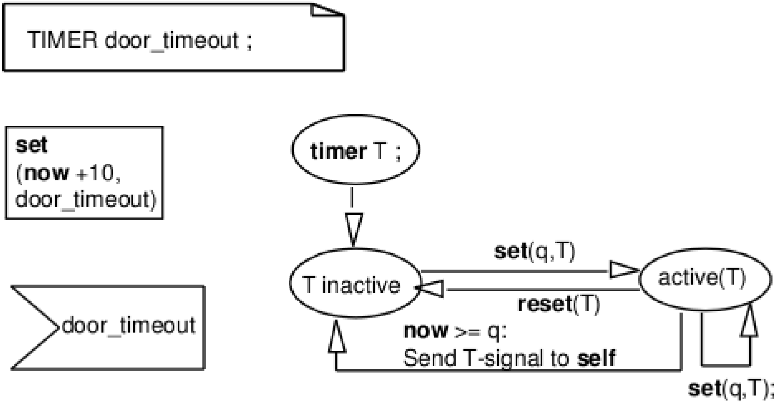
source: TIMe Electronic Textbook v4.0

Asterisk Save, Asterisk State, and Dash State



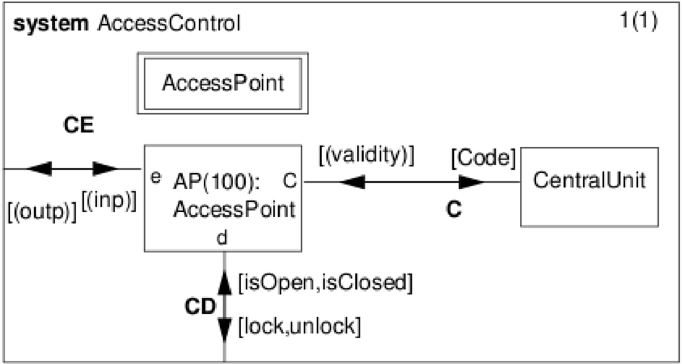
source: TIMe Electronic Textbook v4.0

Timer Construction



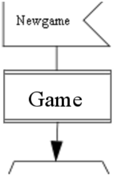
source: TIMe Electronic Textbook v4.0

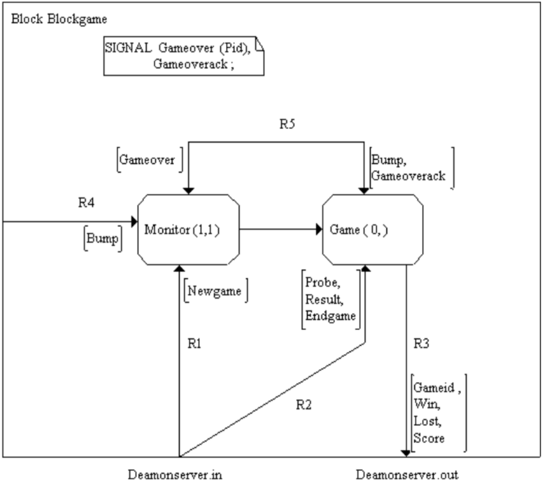
Multiple Instances of a Block



source: TIMe Electronic Textbook v4.0

Multiple Instances of a Process



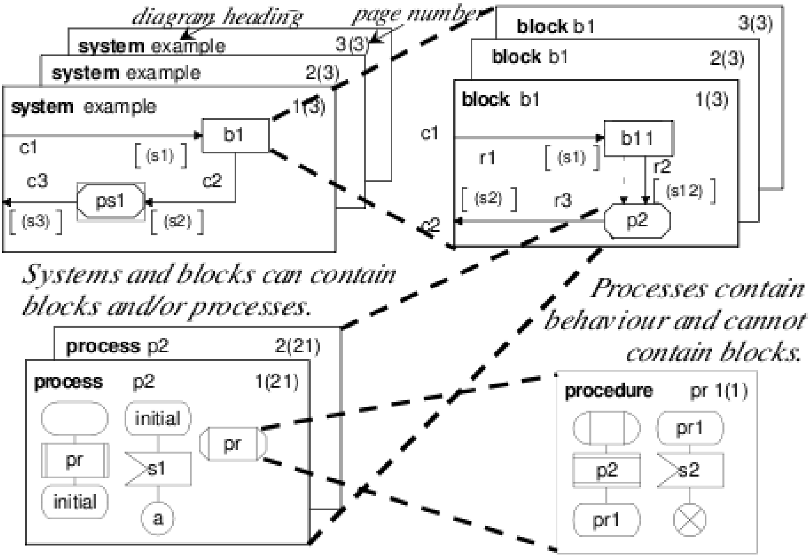


Additional SDL Constructs

- **Asterisk save**, **asterisk state**, and **dash state**
- **Timer** construction
- **Multiple block instances** (no dynamic creation)
- **Multiple process instances** (with dynamic creation and limit)

- **Packages** collections of related types and definitions (library)
- **Subtypes**
- **Virtual processes**
- **Process type redefinition** and **finalization**
- **Inherited blocks**

SDL - Overview



SDL - Goals

What SDL is good for?

SDL is designed for unambiguous **specification** of requirements and **description** of implementation of the normative requirements of **telecommunication protocol** standards.

For computer based tools to improve the process of

- specification (create, maintain, and analyze), and
- implementation (automatic code generation).

What SDL is NOT good for?

high level system description (what the system serves for), demonstration of good or wrong behaviour, test trace specification, implementation details (primitives for communication, detailed data manipulation), etc.

MSC and SDL in Workflow

- typical/optimal communication sequences (MSC)
- error sequences (MSC)
- optionally - full specification in (HMSC)
- distributed specification (SDL)

Formal model benefits

- (H)MSC to SDL transformation (realization)
- SDL to source code transformation (implementation)
- MSC to test case transformation
- simulation to MSC transformation (membership checking)
- ...

SDL - Tools

IBM Rational

- from tools of Telelogic (SDT, Geode, Tau)
- drawing, import, export
- automatic implementation in C++
- simulation support

SanDriLa SDL

- MS Visio stencil
- drawing, import, export
- analyses of states in process diagrams
- open for addons

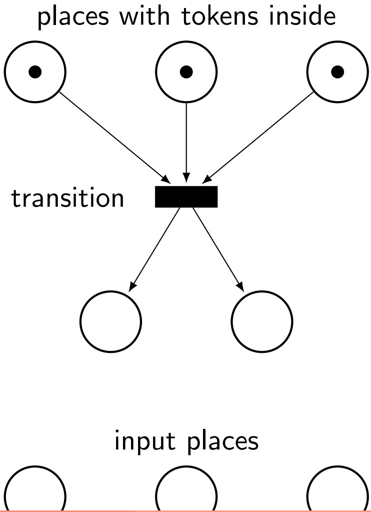
Cinderella SDL

- modelling, import, export
- analyses and simulation

Petri Nets

Petri Nets

C. A. Petri: Kommunikation mit automaten, 1962

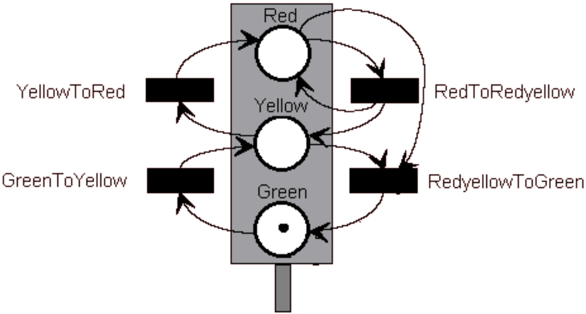


Basic components:

- places
- transitions
- tokens
- arcs

Marking = configuration
= distribution of tokens

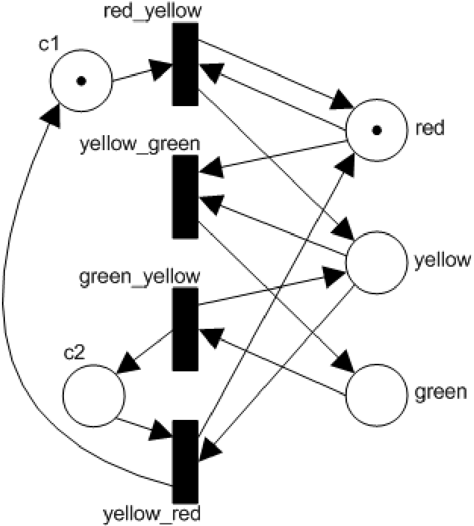
Demonstration Example #1



What is wrong in this example?

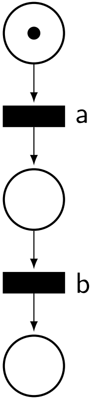
Demonstration Example #2

Better and a bit more complicated example.

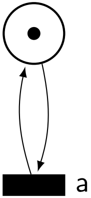


Basic Constructions

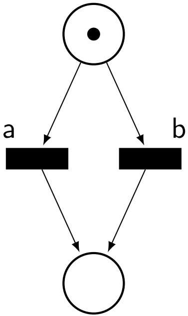
Sequential execution



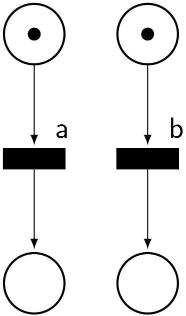
Iteration



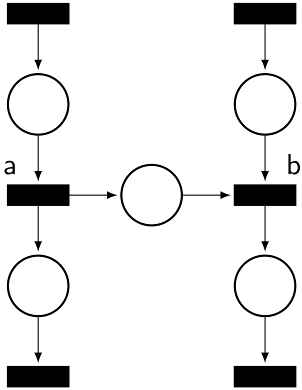
Alternative



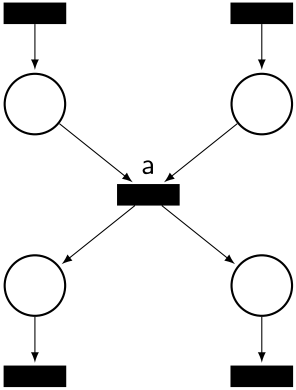
Parallel execution



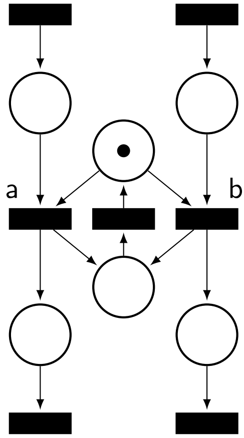
Semaphore



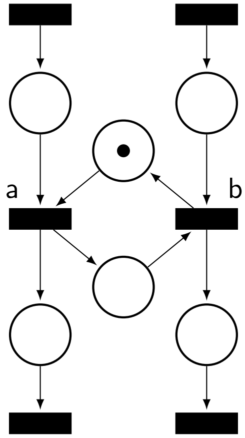
Rende-vous



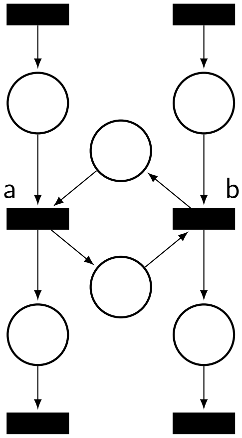
Basic Constructions



Critical section



Alternation



Deadlock

Different Modifications/Extensions of Petri Nets

- Condition-Event Petri nets (C-E PN)
- Place-Transition Petri nets (P-T PN)
- Coloured Petri nets
- Hierarchical Petri nets
- Timed Petri nets
- Time Petri nets
- Stochastic Petri nets

Condition-Event Petri Nets

In this case:

- places = conditions
- transitions = event

An event/transition is *enabled* if and only if

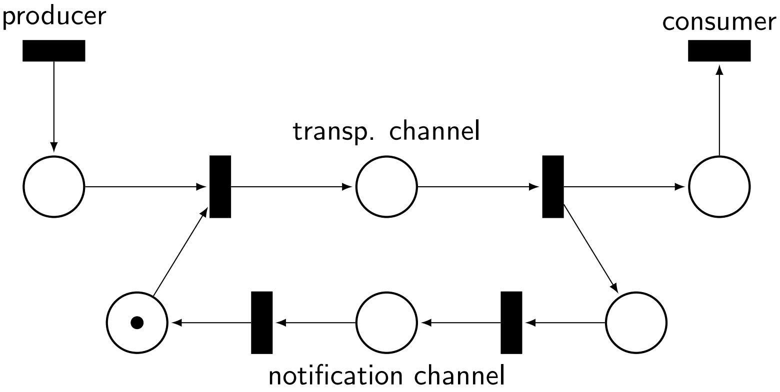
- all its pre-conditions are true and
- all its post-conditions are false.

I.e., an event occurrence negates its pre- and post-conditions.

Therefore, there is one or none token in each place.

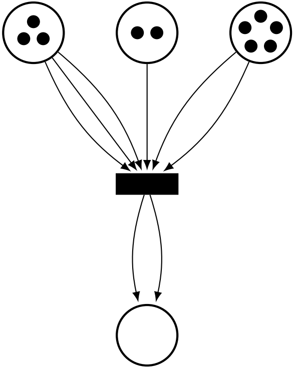
Transition-Place Petri Nets

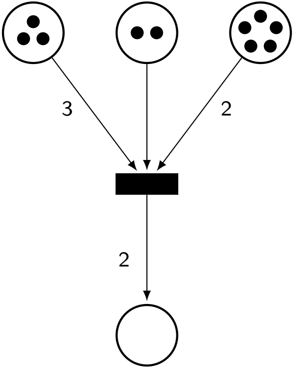
An arbitrary number of tokens in each place.

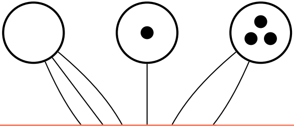


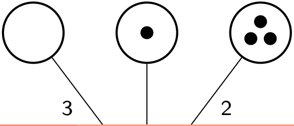
Producer-consumer model for bounded transport channel.

Additional Constructs - Arc Multiplicity

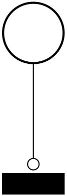




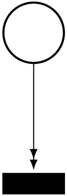




Additional Constructs - Inhibitor and Reset Arcs



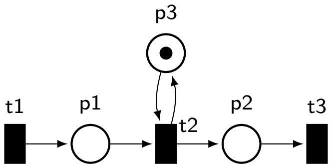
An **inhibitor arc** imposes the precondition that the transition may only fire when the place is empty.

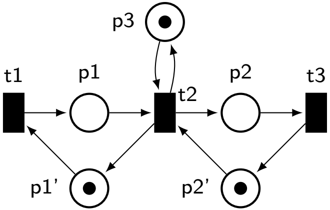


A **reset arc** does not impose a precondition on firing, and empties the place when the transition fires.

Properties of Petri nets

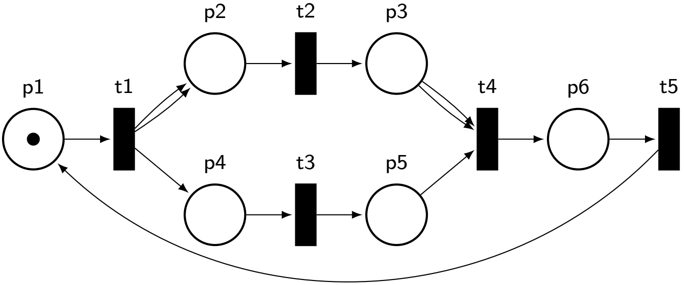
- **reachability** - reachability tree or coverability tree
- **bounded (safe) places**
  - a place with a **bound** on the number of its tokens in all reachable markings
  - a place is **safe** if the number of its tokens ≤ 1 in all reachable markings
- **liveness**
  - a transition is **live** if, from every marking, one can reach a marking where the transition is enabled
  - a net is **live** if all its transitions are live





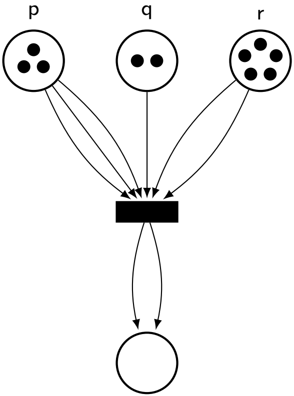
Properties of Petri nets

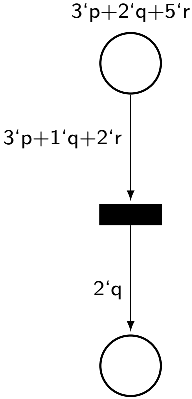
- **p-invariant**
  - an invariant vector on places, i.e. a multiset of places representing weighting such that any such weighted marking remains invariant by any firing, e.g. $3 * p_1 + p_2 + p_3 + p_4 + p_5 + 3 * p_6$.
- **t-invariant**
  - an invariant vector on transitions, i.e. a multiset of transitions whose firing leave invariant any marking, e.g. $t_1 + 2 * t_2 + t_3 + t_4 + t_5$.



Coloured Petri Nets

Different colours (classes) of tokens.

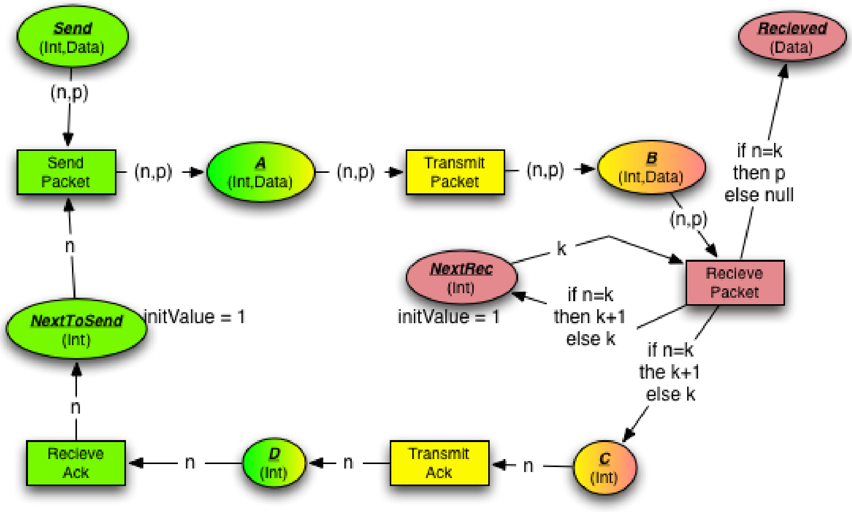




marking expression, arc expression, transition guard (next slide)

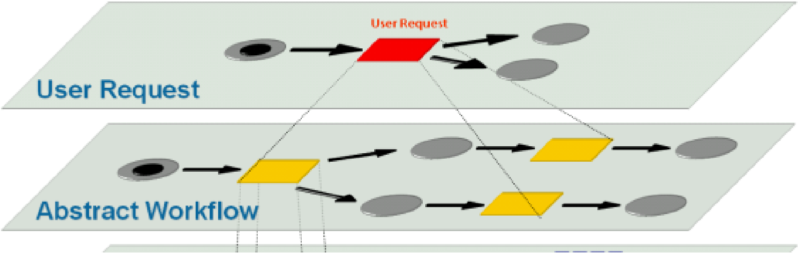
Colours usually serves for data type representation.

Coloured Petri Net Example



source: http://scienceblogs.com/goodmath/2007/10/colored_petri_nets.php

Hierarchical Petri Nets



source: http://www.gridworkflow.org/kwfgrid/gwes-web/

Time PN, Timed PN, Stochastic PN, ...

priority nets

- priorities of concurrent transitions

time (or timed-arc) nets

- tokens has its lifetime, arcs to transitions are labeled by time intervals of required ages of tokens

timed nets

- firing starts when a transition is enabled but it takes some specified time to produce output

stochastic nets

- probability distribution on time to fire (exponential, deterministic, or general distributions)

PN Tools

CPN Tools

- Coloured Petri Nets (with prioritized transitions and real time support)
- editor, simulation, analyses

Tapall

- Timed-Arc Petri Nets (with real time support)
- editor, simulation, compositional models, TCTL logic checker

TimeNET

- Coloured and Stochastic PN with non-exponential distributions
- editor, simulation, analyses (p-invariant, performance analyses)

SNOOPY, TINA - TIme petri Net Analyzer, Roméo, ...

And many more tools and use cases, see, e.g.

http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html
http://cs.au.dk/cpnets/industrial-use/

Queueing theory

Queueing Theory

In 1909 A.K. Erlang, a danish telephone engineer, was asked:

> "What the queue capacity should be
> of the central telephone switch in Copenhagen?"

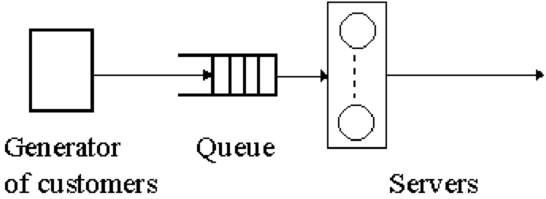
Our motivation example:

Example

30 customers will visit the cash machine in an hour.
Each will use it for 1.5 minute on average.

How busy is the cash machine?
For how long time does a customer wait (on average)?

Queues and Thier Parameters



- inter-arrival time distribution (type of the distribution, rate λ, or mean inter-arrival time $1/\lambda$, other moments ...)
- service time distribution (type of the distribution, rate μ, or mean service time $1/\mu$, other moments ...)
- number of servers
- maximal queue length
- ...

Queue Parameters - Kendall Notation

$A/S/n/B/K/SD$

A - inter-arrival time distribution

G - general, **M** - Poisson, **D** - deterministic...

S - service time distribution

G - general, **M** - exponential, **D** - deterministic...

n - number of servers

$\mathbf{1}, \mathbf{2}, \dots, \infty$

B - buffer size (the max. number of waiting and served requests)

$\mathbf{1}, \mathbf{2}, \dots, \infty$

K - population size

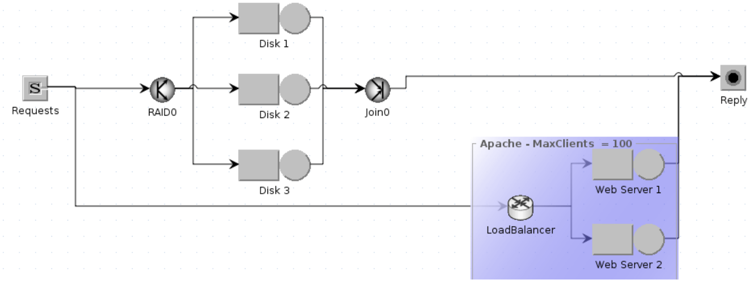
$\mathbf{1}, \mathbf{2}, \dots, \infty$

SD - service discipline

FIFO, **LIFO**, **Random**, **RR** - Round Robin

E.g., $M/G/1/\infty$

Queueing Networks



- open and closed networks
- system dependences
- traffic intensity
- occupancy (on different servers), bottleneck detection, ...

- very similar to Stochastic Petri Nets

Questions about Queues

- What is the utilization factor ρ, probability of being not empty?
- What is the mean number N of waiting (or being served) requests?
- What is the mean waiting and service time, i.e. the time T the requests spends in the system?
- And so, how many servers do I need to ...

General solution:

- simulation

For specific types of queues:

- analytical results

Analytical Solutions for M/M/1 Queues

For M/M/1/∞ queue with arival rate λ and service time rate μ:

- The mean inter-arrival time is $1/\lambda$.
- The mean service time is $1/\mu$.
- The utilization factor $\rho = \lambda/\mu$.
- The queue is stable if $\rho < 1$, i.e. $\lambda < \mu$.
- The (stable) queue is empty with probability $P_0 = 1 - \rho$.
- The mean number of requests (waiting or being served) in a stable system $N = \rho/(1-\rho)$. It is also usually denoted by L as it is the length of the queue.
- The average time spend in a stable system $T = 1/(\mu - \lambda) = 1/(\mu(1-\rho))$.
- The rate of the trafic carried out by the queue is $\mu\rho = \mu(1 - P_0)$.

Our Motivation Example Solved as $M/M/1/\infty$

Example

30 customers will visit our cash machine in an hour.
Each will use it for 1.5 minute on average.
How busy is the cash machine? What is the average waiting time?

- The mean inter-arrival time is 2 minutes.
- The rate of the inter-arrival time λ is $1/2 = 0.5$.
- The mean service time is 1.5 minute.
- The rate of the service time μ is $2/3 \approx 0.666667$.
- The queue is stable and the utilization factor $\rho = 3/4 = 0.75$.
- The mean number of requests (waiting or being served) N is 3.
- The average time spend in the system T is 6 minutes.

I.e., the utilization of the cash machine is 45 minutes in an hour, i.e. 75%.
Average number of customers at the cash machine is 3.
The average waiting of a customer is 4.5 minutes + 1.5 min for service.

Little's Law

Theorem

Let L be the long-term **average number of customers** *in a stable system, λ be the long-term average effective* **arrival rate**, *and W be* **the average time a customer spends** *in the system. Then it holds that*

$$L = \lambda \cdot W$$

for a queue of any type.

Although it looks intuitively reasonable, it is quite a remarkable result, as the relationship is "not influenced by the arrival process distribution, the service distribution, the service order, or practically anything else."

Tools for Queueing Systems

G/M/c-like queue

- online steady-state solution of a G/M/c-like queue
- http://queueing-systems.ens-lyon.fr/formGMC.php

Queueing Simulation

- online queueing network analyzer
- http://www.stat.auckland.ac.nz/~stats255/qsim/qsim.html

JMT - Java Modelling Tools

- framework for model simulation and workload analysis
- http://jmt.sourceforge.net/

SimEvents

- simulation engine and component library for Simulink (MATLAB)
- http://www.mathworks.com/products/simevents/

Up-to-date List of relevant Queueing theory based tools:

http://web2.uwindsor.ca/math/hlynka/qsoft.html

Relevant Lectures

IV113 Úvod do validace a verifikace (Barnat)

IA169 System Verification and Assurance (Barnat, Řehák, Matyáš)

IV109 Modelování a simulace (Pelánek)

IA159 Formal verification methods (Strejček)

IA158 Real Time Systems (Brázdil)

References

- ITU-T recommendation Z.120: Message Sequence Charts (MSC). 2011.
- ITU-T recommendation Z.100: Specification Description Language (SDL). 2011.
- S. Mullender. *Distributed Systems.* ACM, 1993.
- D. Peled. *Software Reliability Methods.* Springer, 2001.
- R. Bræk at al. *TIMe: The Integrated Method.* SINTEF, 1999.
- L. Doldi. *Validation of Communications Systems with SDL.* Wiley, 2003.
- M. Broy and K. Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement.* Springer, 2001.
- W. Jia and W. Zhou. *Distributed Network Systems: From Concepts to Implementation.* Springer, 2005.
- M. Češka. *Petriho sítě.* Akademické nakladatelství CERM Brno, 1994.
- J. Markl. *Petriho sítě.* VŠB - Technická univerzita Ostrava, 1996.
- G. Giambene. *Queuing Theory and Communications - Networks and Applications.* Springer, 2005.
- D. Gross at al. *Fundamentals of Queuing Theory.* Wiley, 2008.