Some Important Concepts Related to State Machine Modeling

Gregor v. Bochmann

School of Electrical Engineering and Computer Science (EECS)

University of Ottawa
Canada

uOttawa

L'Université canadienne
Canada's university

SCARE Seminar- Universität Oldenburg

22. Oktober 2013

Originally an invited paper at the SDL Forum (Montréal, June 2013)

Abstract

After a personal view of the history of state-machine modeling, we will concentrate on three concepts that are important for state-machine modeling and have been taken into account (in one way or another) in the design of related modeling languages: (1) conflict resolution between several system components, (2) message buffering, and (3) the refinement relation used for object-oriented inheritance. We will discuss some historical milestones that contributed to clarifying the related issues and providing solutions to the problems that arise in the context of these three concepts. We will also discuss how these concepts impact dependability engineering, and how they are accommodated in SDL and other specification languages.

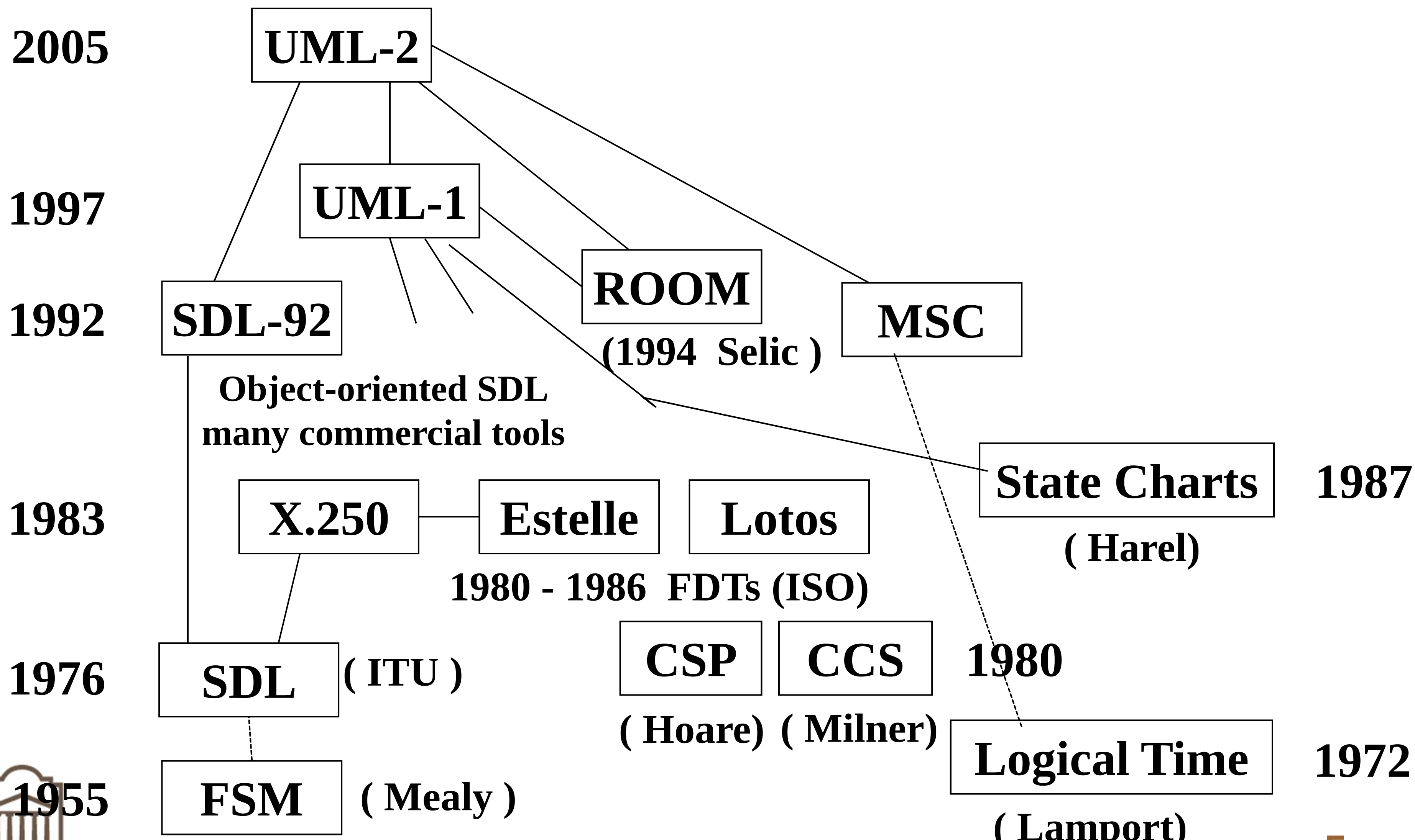
Perspective on state machine modeling

- State machine modeling is old
  - More than 50 years for hardware
  - More than 35 years for software
  - I have been involved in modeling of communication protocols since 1975
- Basic concepts are well known, but certain aspects are often not well understood:
  - The impact of different types of message buffering vs. rendezvous communication
  - Conflict resolution in a distributed environment
  - Inheritance and the substitution principle

Outline of talk

- History of modeling languages, concepts and tools
- Similarity of notations
- Distributed system design – an example
- Systematic design of distributed systems
- Substitution principle (inheritance) for state machines
- Concluding remarks

History: modeling languages



History: Verification of concurrent system designs

2005 UML-2

Current UML tools
have less verification support

1997 UML-1

Commercial SDL tool (Telelogic)
including Bit-State verification

1992 SDL-92

1987 Bit-State representation
for SPIN tool (Holzmann)

1983

1975 - 80 First protocol verification
based on FSM specifications
(Bochmann; West, prototype tools at IBM Zurich)

1976 SDL

1955 FSM

History: Code generation from model specifications

2005 UML-2 — Umple: a linear representation of UML embedded into an implementation language (Lethbridge)

1997 UML-1 — Similarly for UML

1992 SDL-92 — Commercial SDL tool with code generation

1987 Paper on code generation from Estelle models (Bochmann)

1983

1982 First paper on code generation from extended FSM models (Blumer & Tenney)

1976 SDL

1955 FSM

History:
Object orientation - inheritance

2005 **UML-2**

1995 **Substitution principle with assumptions (Abadi & Lamport)**

1993 **Conformance relations for behavior (Bochmann, Petrenko, Dssouli)**

1992 **SDL-92**

1991 **Behavior specifications with assumptions (Misra & Chandy)**

1987 **Substitution principle based on dynamic behavior (Liskov)**

Structural subtyping based on interfaces (Emerald language)

1976 **SDL**

1967 **Object orientation, inheritance (Simula)**

1955 **FSM**

History: Other interesting concepts

2005 UML-2

1992 SDL-92

1976 SDL

1955 FSM

2005 Avoiding race conditions by considering message consumption (Mooij et al.)

2004 Stuckfreeness (Fournet et al.)

2000 Implied scenarios (Alur et al.)

1996 Weak sequencing (for MSC)

1984 Conflict resolution by priority (Gouda)

1983 Service primitives (ISO)

1980 Rendezvous (Milner, Hoare)

1972 Partial order (Lamport)

Outline of talk

- History of modeling languages, concepts and tools
- Similarity of notations
- Distributed system design – an example
- Systematic design of distributed systems
- Substitution principle (inheritance) for state machines
- Concluding remarks

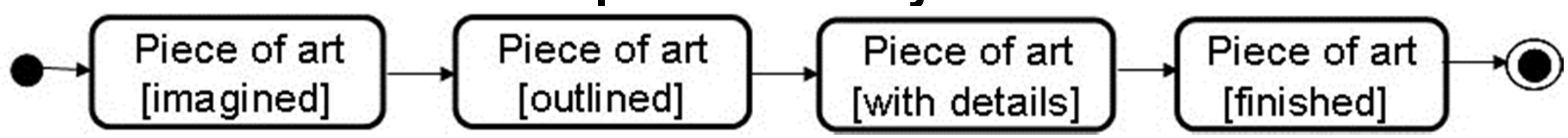
Two perspectives for state machines

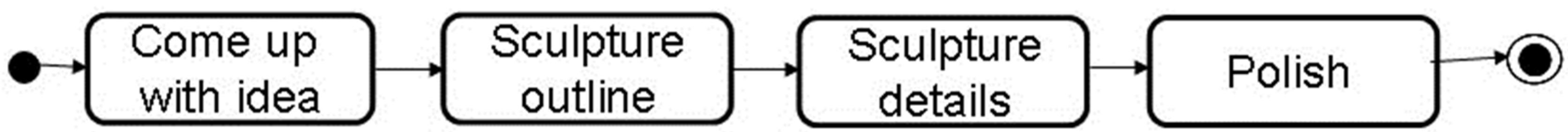
A. **Actions during transitions**

B. **Actions in states**

Example: The making of a sculpture

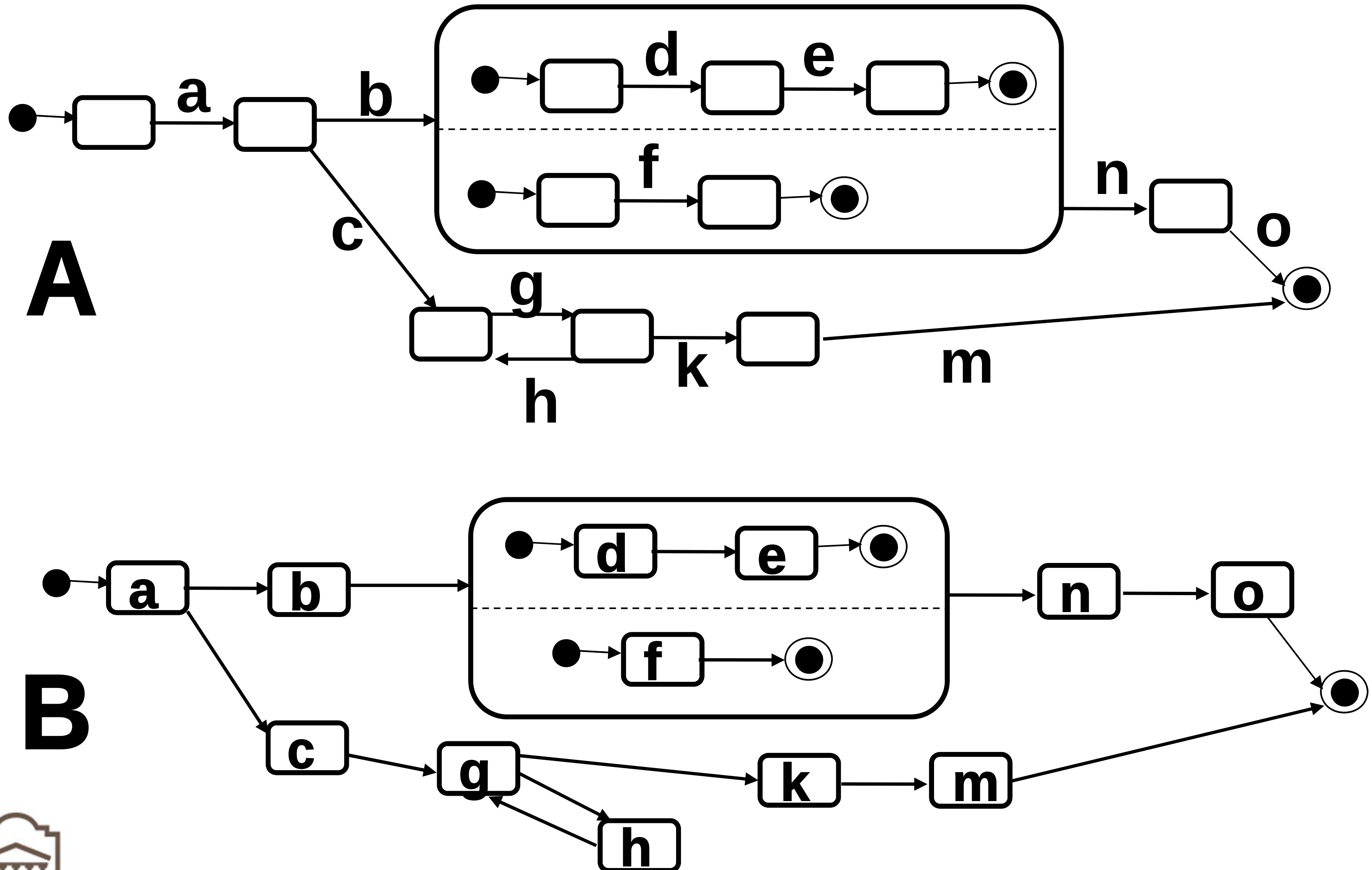
A states of the sculptured object



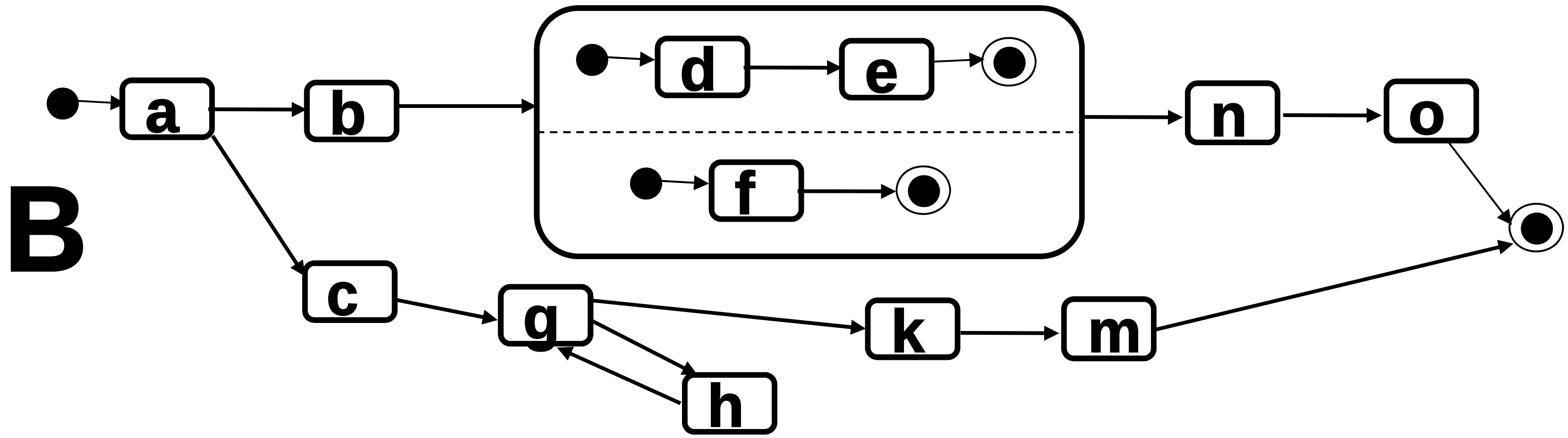


B states of the artist

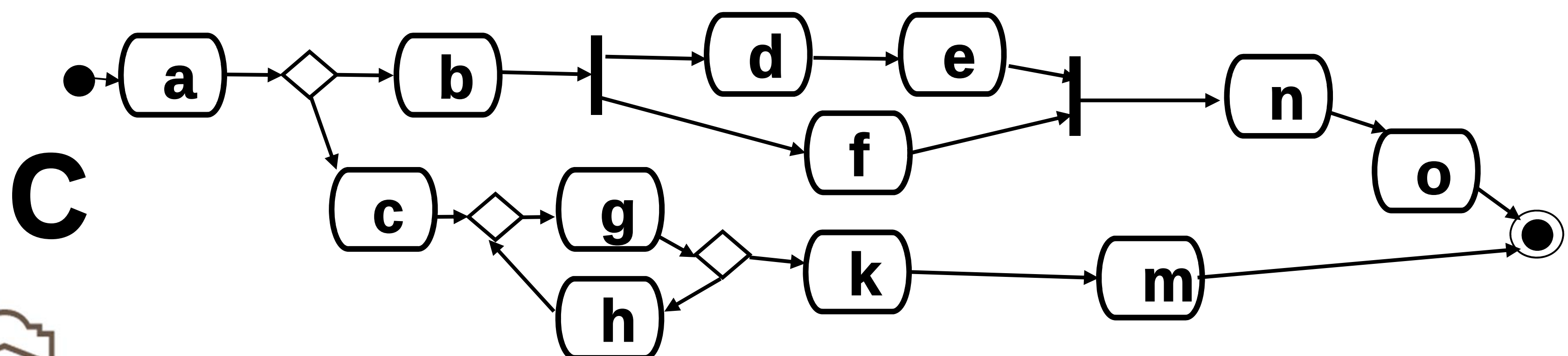
A more complex example



Similarity of different notations

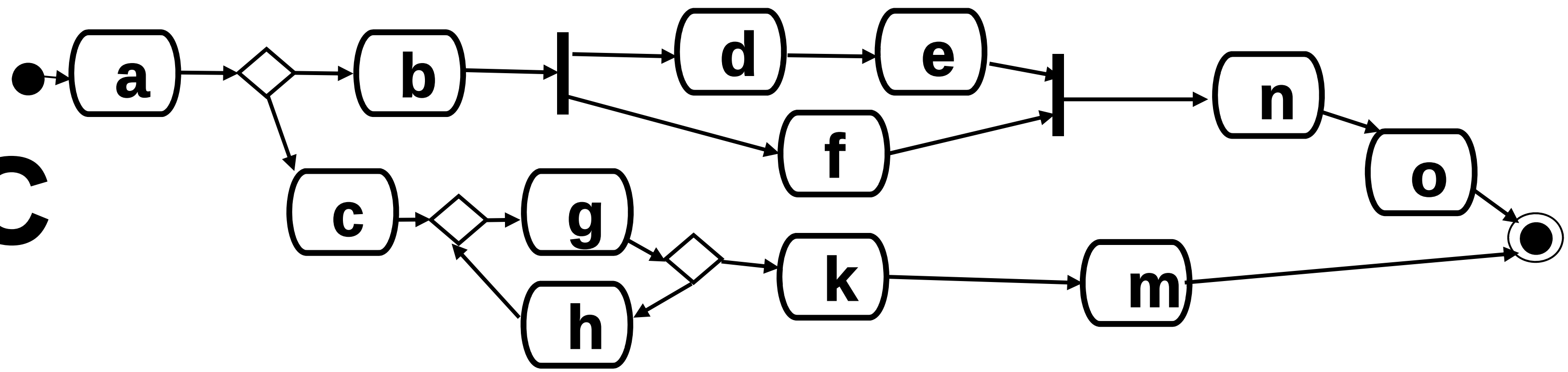


The above UML State Machine corresponds exactly to the following UML Activity Diagram



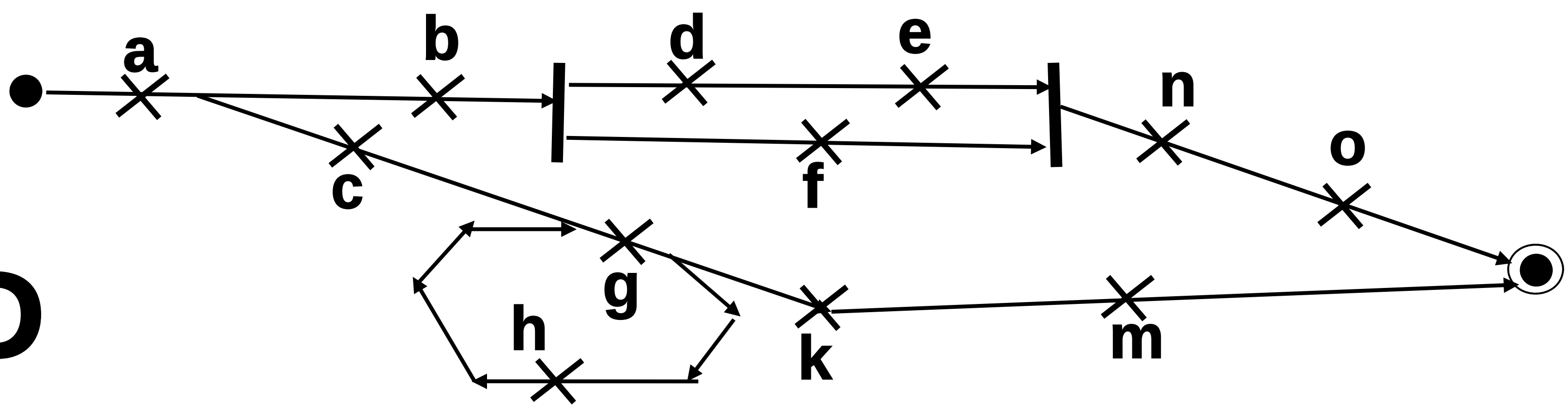
Similarity of different notations (ii)

C



The above UML Activity Diagram corresponds exactly to the following Use Case Map

D



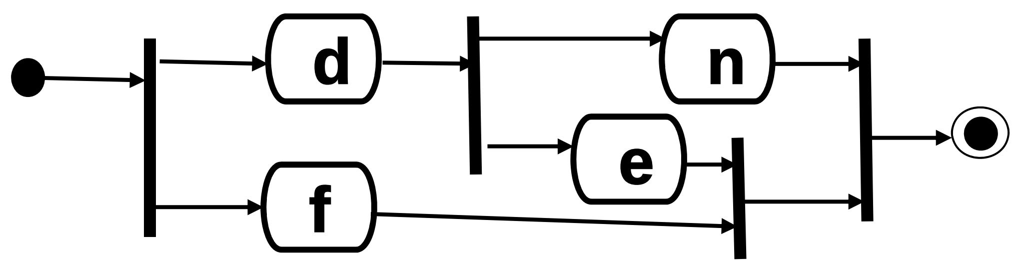
Note: An arrow represents **control + data flow**

Conclusion (1)

- **Do we really need so many different notations ??**

- **Hierarchical State Machines may be used to represent “well-structured” Activity Diagrams or Use Case Maps.**

Example of non well-structured Activity Diagram:

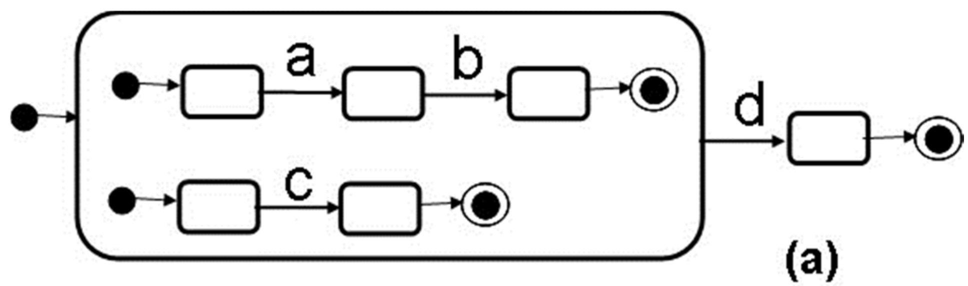


Outline of talk

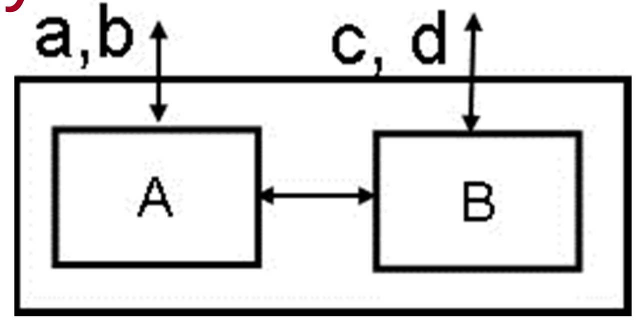
- History of modeling languages, concepts and tools
- Similarity of notations
- Distributed system design – an example
- Systematic design of distributed systems
- Substitution principle (inheritance) for state machines
- Concluding remarks

Distributed system design – an example

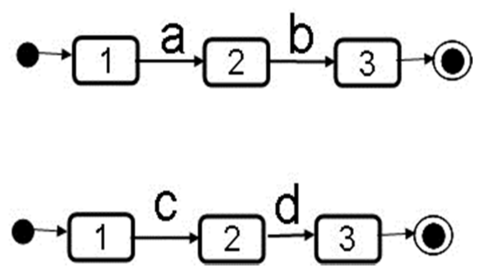
Global behavior spec



System architecture

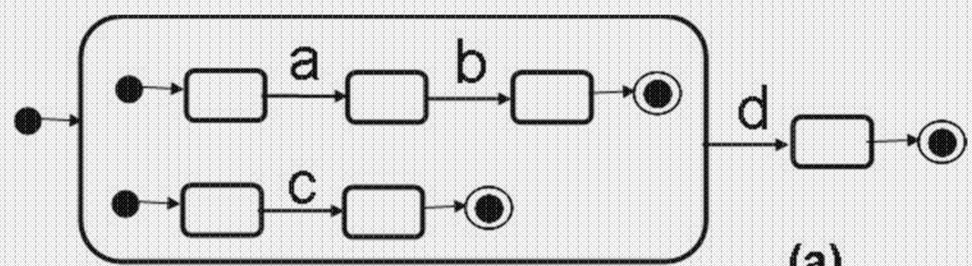


A draft solution **by projection**:



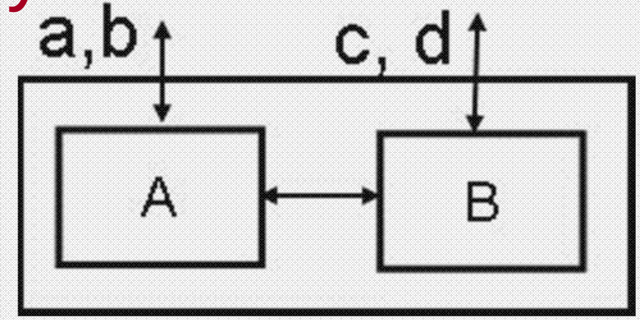
Distributed system design – an example

Global behavior spec

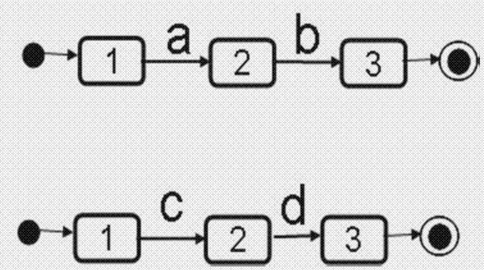


(a)

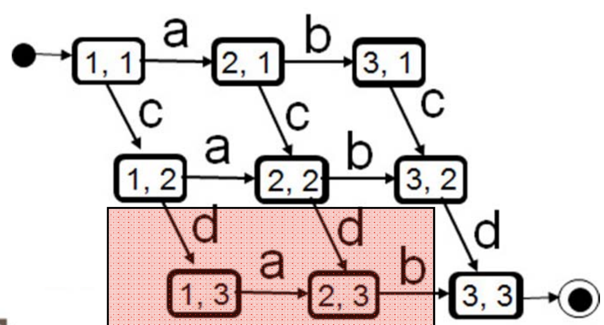
System architecture



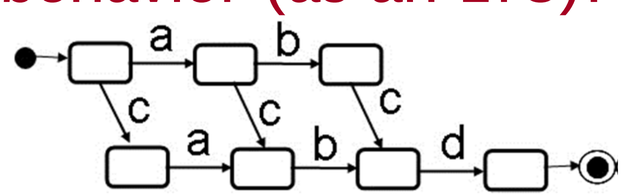
A draft solution **by projection**:



Leads to invalid behavior (red)



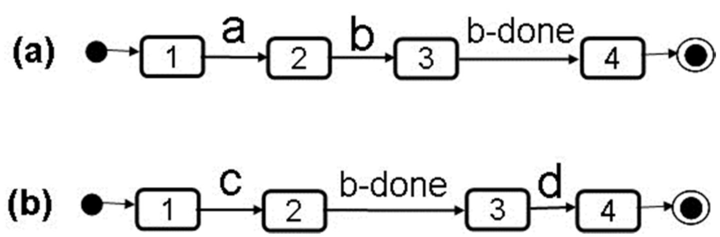
Desired global behavior (as an LTS):

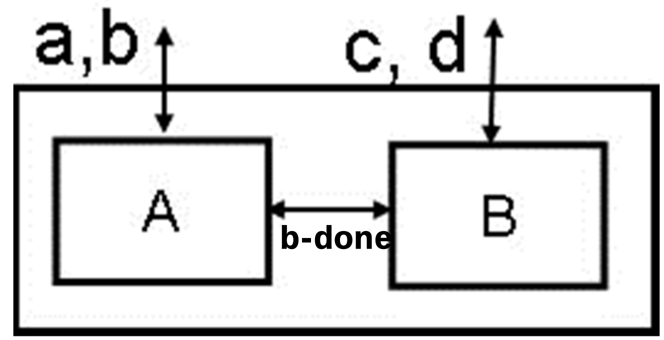


Rendezvous communication

- **Note: Rendezvous communication is more abstract than message passing**
  - Synchronous communication

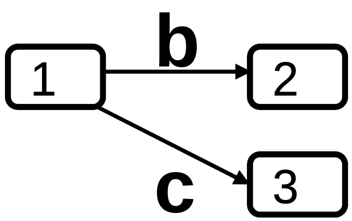
    Here is a solution to the example:





  - Avoids cross-over of messages over the interface
    - *This is important for competing initiatives*

Competing initiatives



Different cases:

- Actions b and c initiated by the same system component: “local choice”
- Actions b and c initiated by different system component: “competing initiatives” or “non-local choice”
  - Example: Telephone call collision (simultaneous incoming and outgoing call)

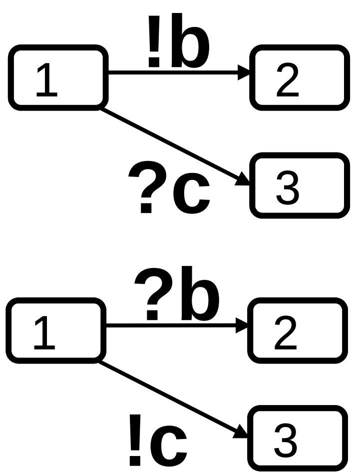
Rendezvous communication:

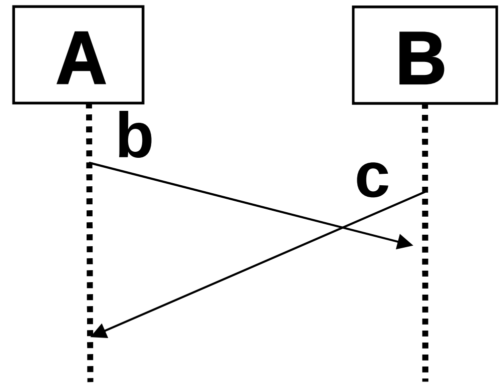
- No problem at the level of behavior specification
- However, non-local choice requires some protocol between the different parties at the implementation level (e.g. circulating token)

Competing initiatives and message passing

Component **A**

Component **B**





Possible joint behavior

- Both components will be in different states
- Non-specified receptions

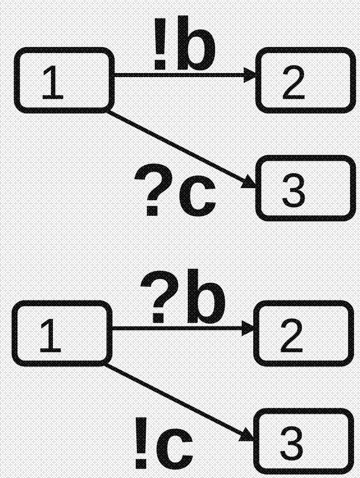
Concept: Non-specified reception (a process receives a message for which there is no explicit behavior specified) – three different interpretations:

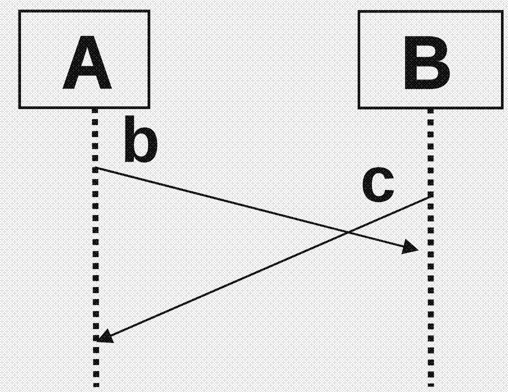
1. Undefined behavior (design error if it occurs)
2. Machine goes into an error state (design error if it occurs)
3. Message will be dropped (SDL)

Competing initiatives and message passing

Component **A**

Component **B**



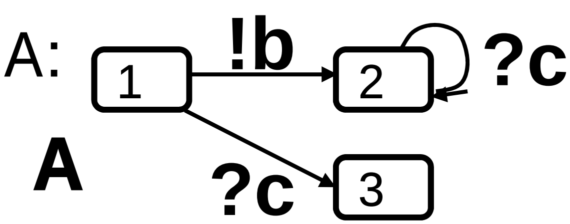


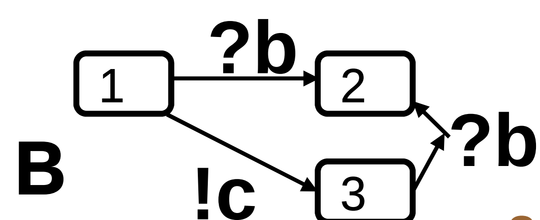
Possible joint behavior

- Both components will be in different states
- Non-specified receptions

Gouda's solution (1984):

- One side obtains priority **(a design choice)**
- e.g. priority for A:



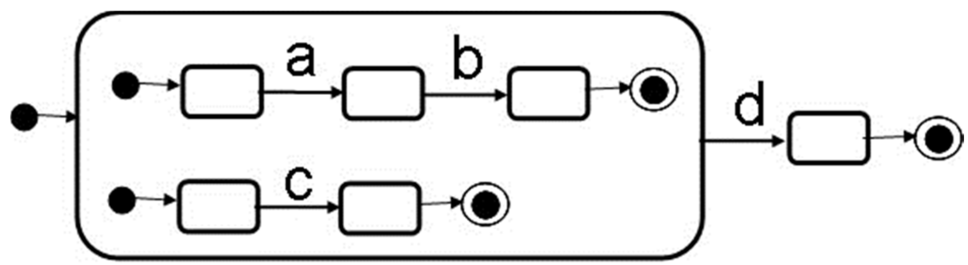


Important concept: Service primitives
(synchronous message passing, like in CSP - 1980)

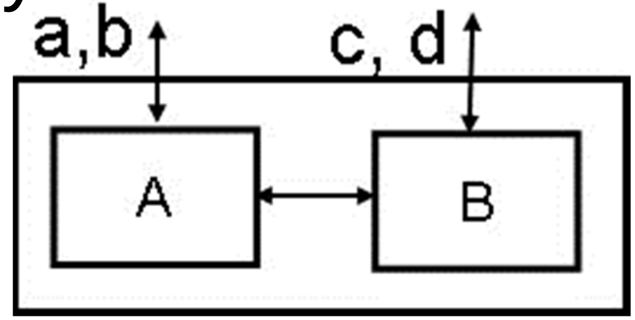
- Layered protocol architecture (Pouzin et al. 1973)
- Formalization of protocol architecture (Bochmann 1978)
  - zero-queue communication at service interfaces, called *direct coupling* (like in CSP or in Input-Output Automata (IOA), Lynch 1989)
  - asynchronous message passing for communication through network
- OSI: Service Primitives (early 1980ies)
- Spin tool with zero-queue option (Holzmann)

Distributed system design **with message passing** – the same example

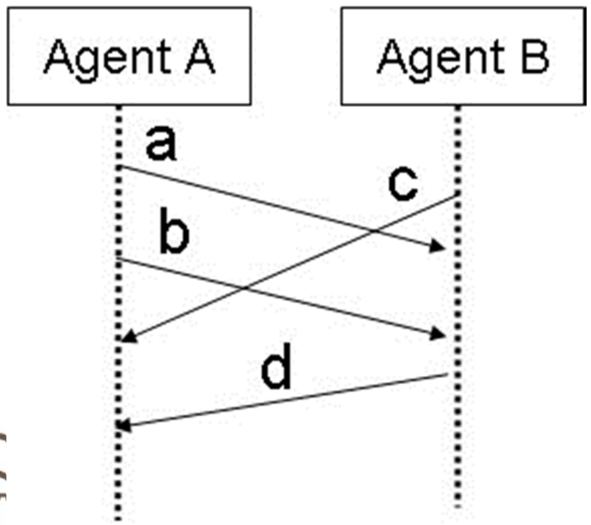
Global behavior spec



System architecture

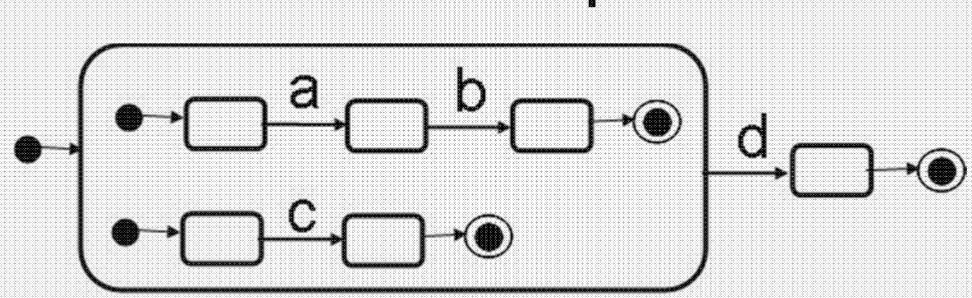


Assumption: Each action (a, b, c, d) implies a message sent to the other party. We assume the following sequence diagram:

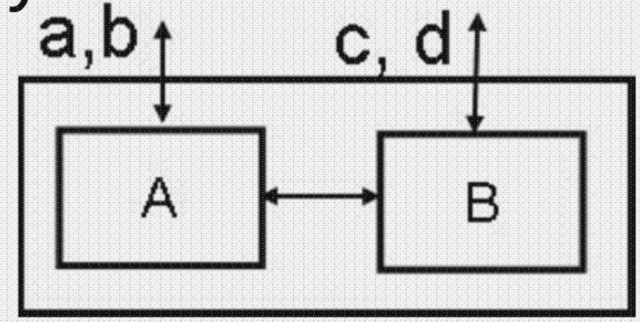


Distributed system design **with message passing** – the same example

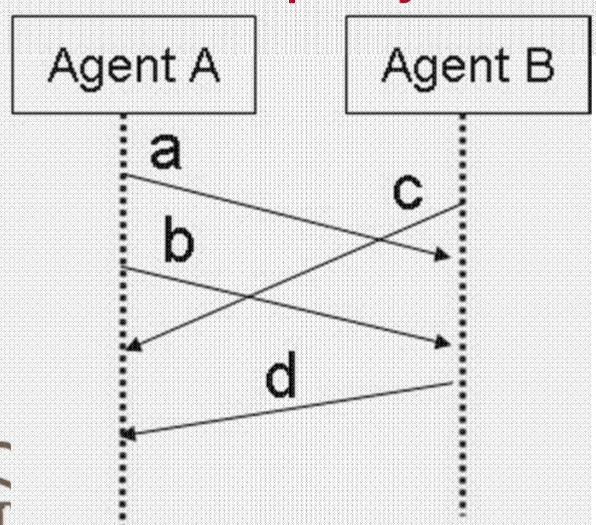
Global behavior spec



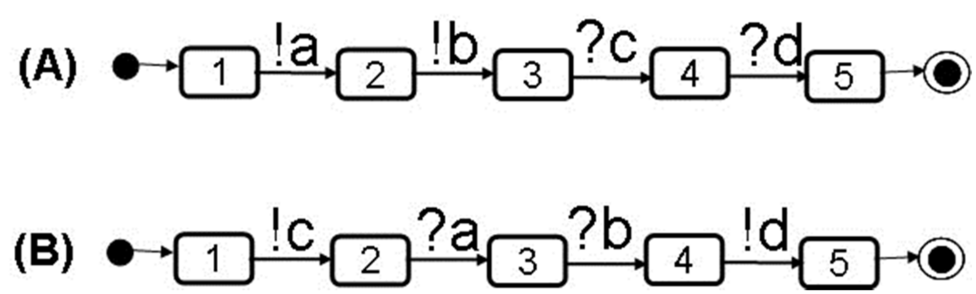
System architecture



Assumption: Each action (a, b, c, d) implies a message sent to the other party. We assume the following sequence diagram:

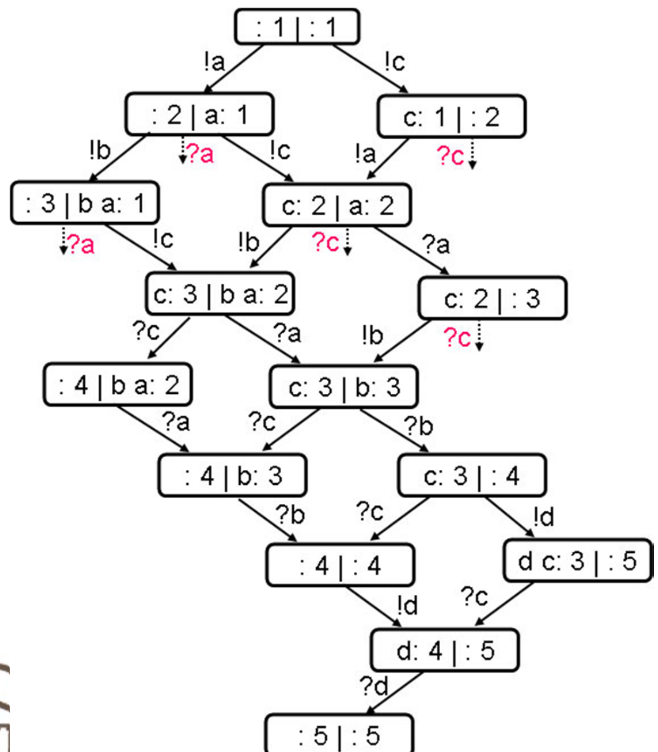


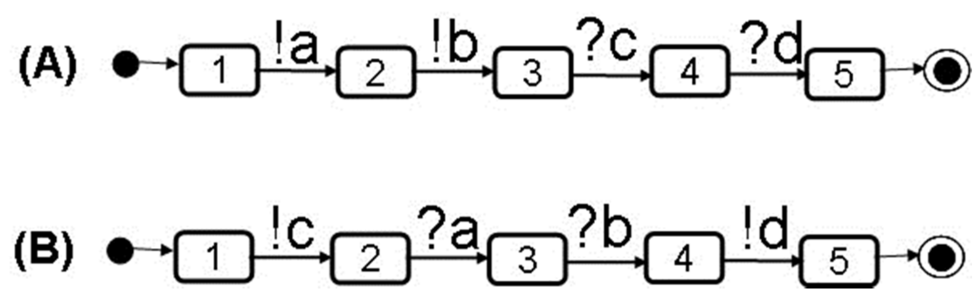
Design method: by projection, we obtain the following behaviors for A and B:



Verifying the design

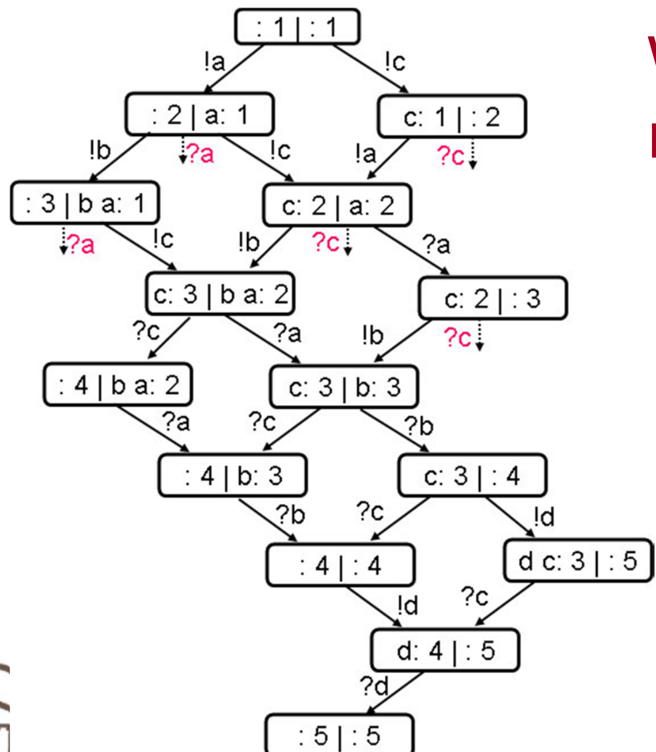
Verifying the obtained agent behaviors by reachability analysis (Bochmann or West, 1978), one finds the following global reachability graph which shows non-specified receptions (in rose):



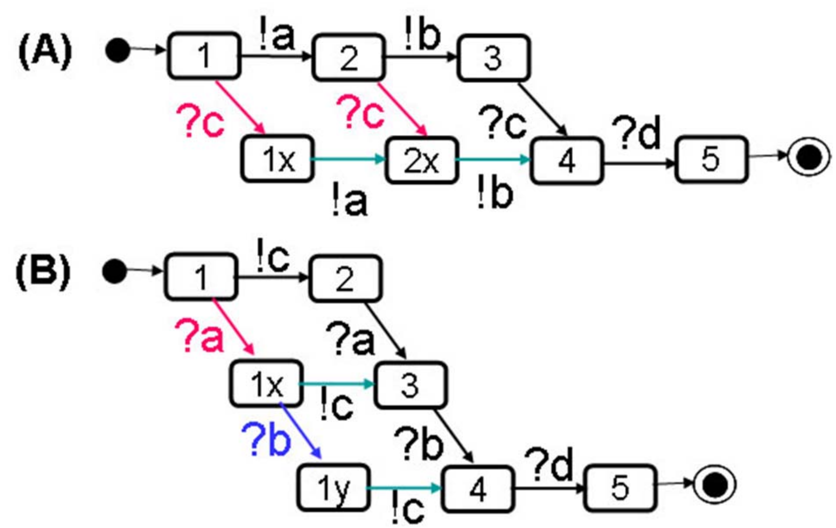


Verifying the design

Verifying the obtained agent behaviors by reachability analysis (Bochmann or West, 1978), one finds the following global reachability graph which shows non-specified receptions (in rose):

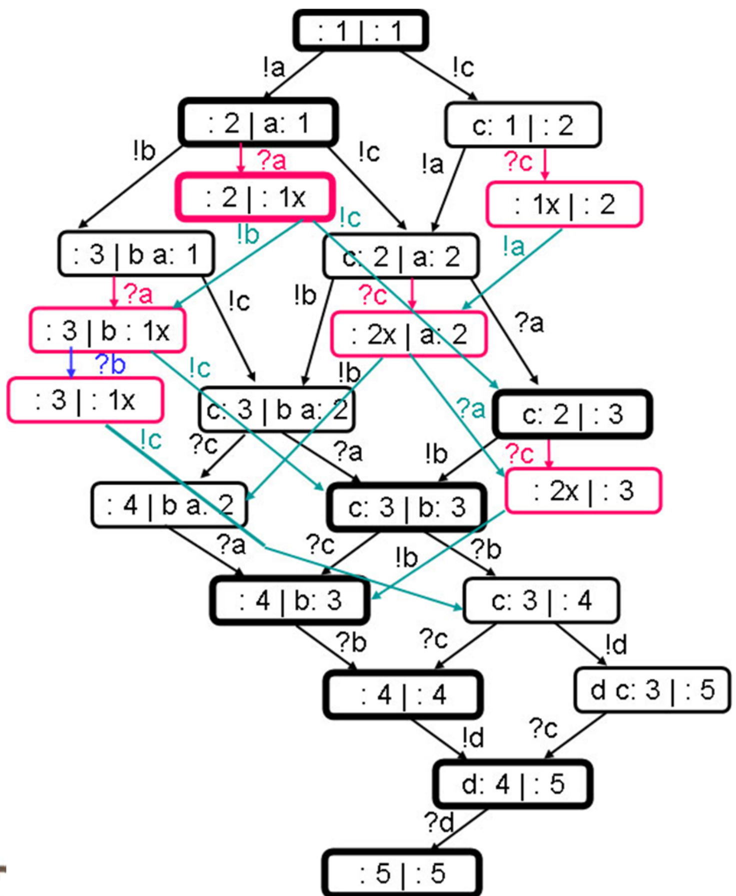


We extend the behaviors to allow for the non-specified receptions, e.g. as follows:



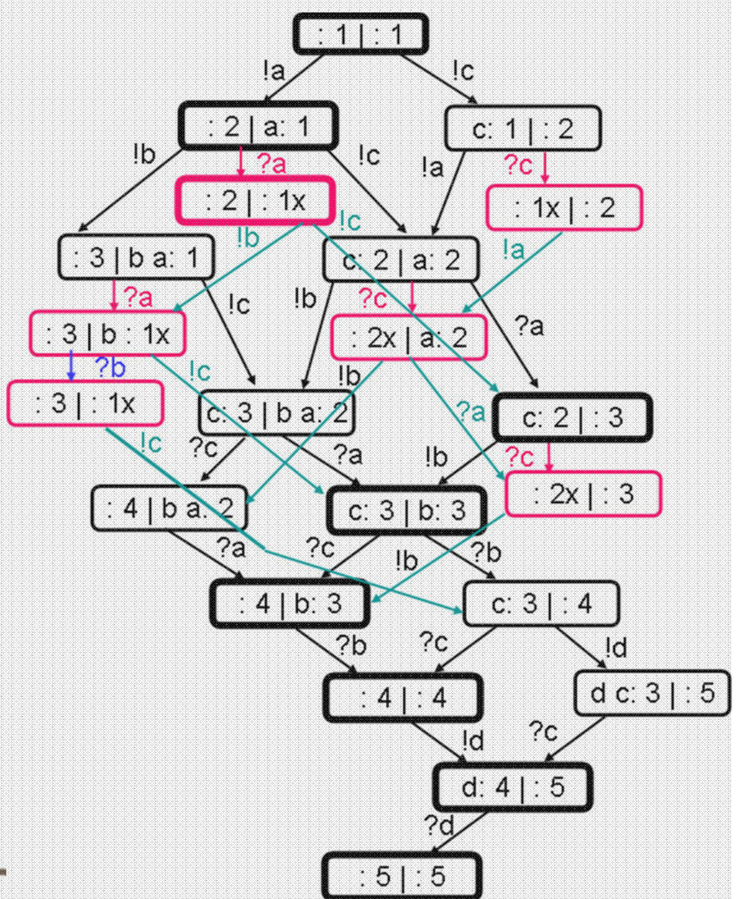
Implied scenarios

- The revised design results in the following graph:

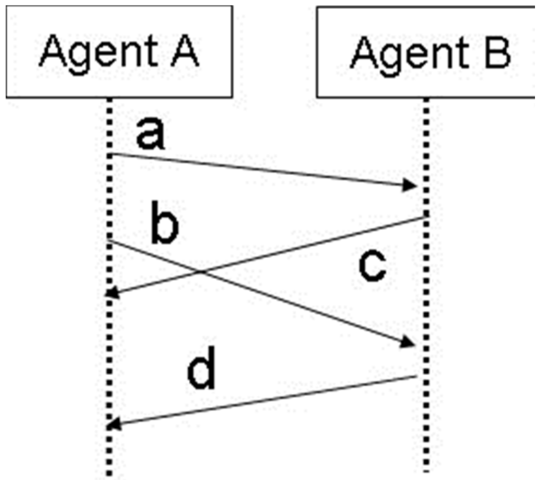


Implied scenarios

- The revised design results in the following graph:



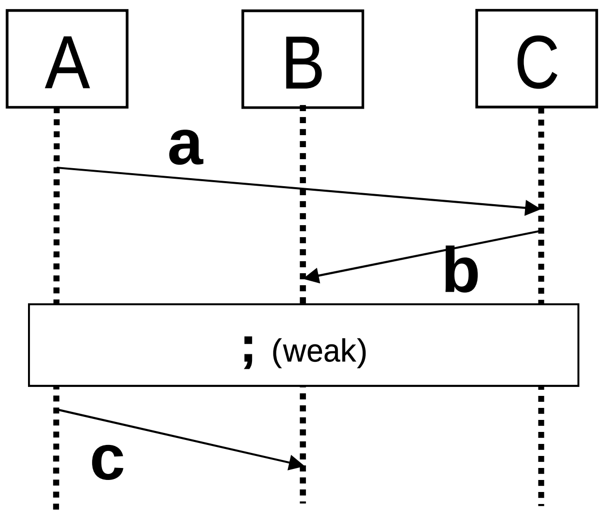
This global behavior includes other sequence diagrams, such as:



Observation (Alur et al., 2000): The minimal protocol behavior required to realize a given scenario (sequence diagram) will often also realize other scenarios, so-called "**implied scenarios**".

Weak sequencing

Weak sequencing is the natural sequencing operator for sequence diagrams

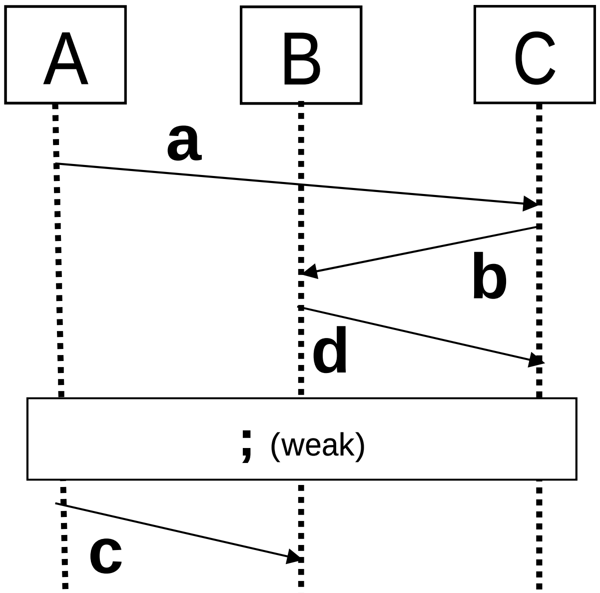


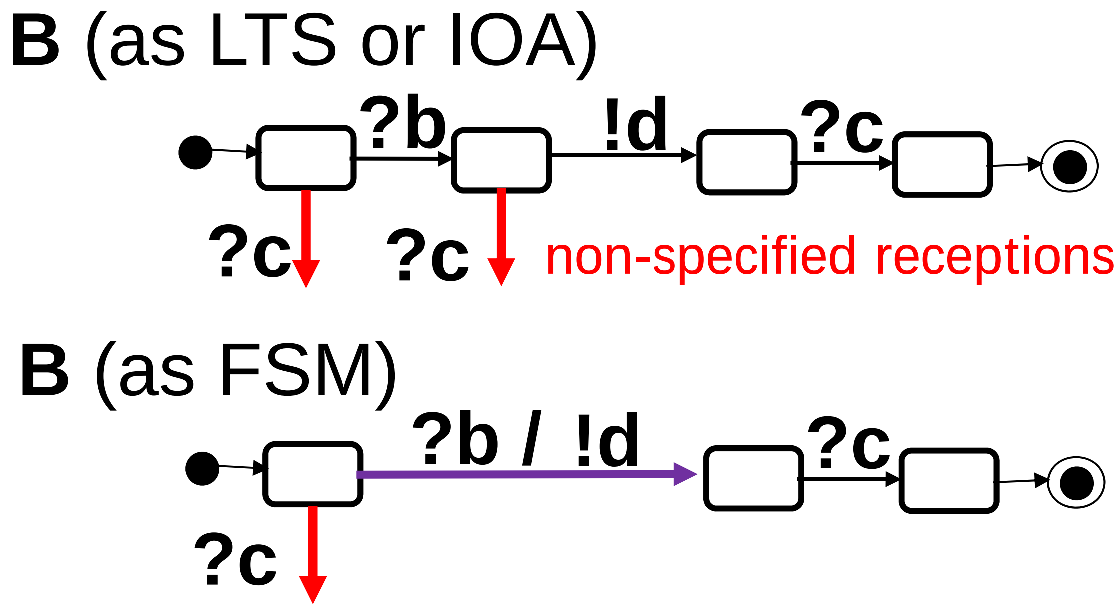
Ordering is enforced only locally

It often leads to so-called race conditions

In this example, A may send c immediately after sending a – therefore B may receive message c before b

Different semantics of state machines





- FSM: Input/output transitions – larger atomic actions – simpler understanding
- Common queue or separate queues
- Different interpretations of non-specified reception

Difficulty of dealing with race conditions

Hypothesis for traditional reachability analysis (definition of non-specified reception) and for race conditions:

- A message must be consumed when it is received.

This is the cause for race conditions !

There has been much work on dealing with race conditions (including ours) – no simple solution !

Message **consumption** vs. reception

Hypothesis for traditional reachability analysis (definition of non-specified reception) and for race conditions:

- A message must be consumed when it is received.

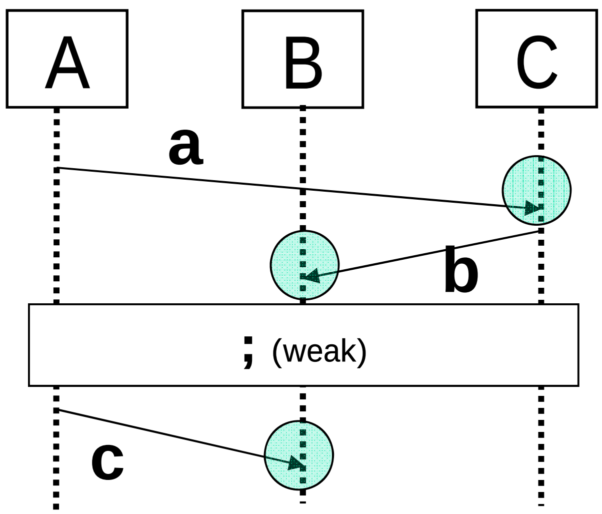
But: here is a solution to race conditions:

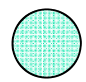
A message is consumed when the receiver is ready [Mooij – 2005] - This avoids race conditions

- Distinguish between message reception and consumption
- Hypothesis: There is a message pool where received messages are stored until they are consumed.

Specifying message consumption

- Hypothesis: There is a message pool
- Notation: “Message reception” means consumption

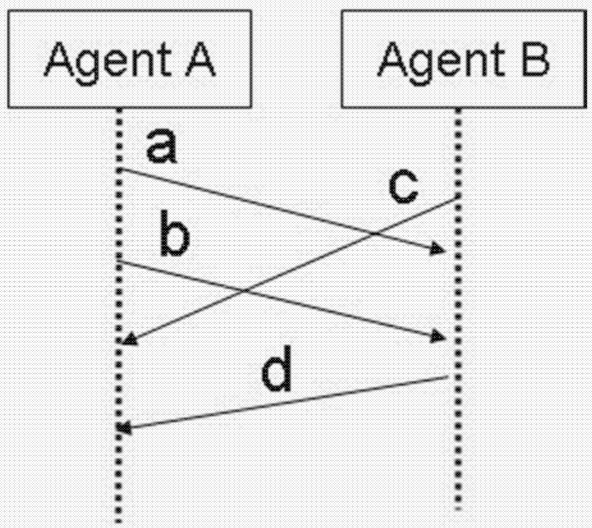




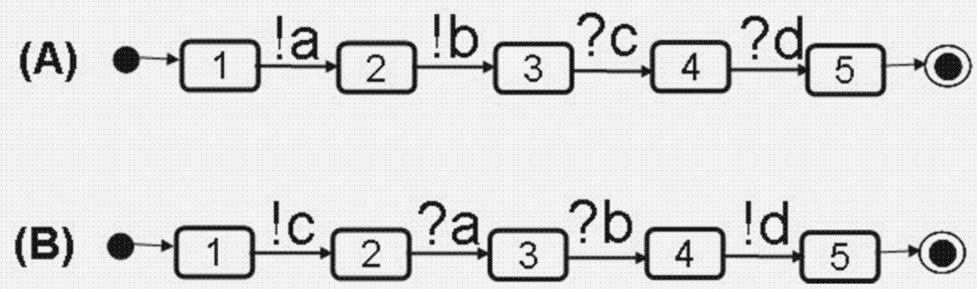
This means the consumption of the message, it may have been received earlier

Example: B will consume message b before message c, although c may arrive before b.

Our earlier example



Design method: By projection, we obtain the following behaviors for A and B:



Assuming message pools - "?" means consumption:

- There are no non-specified receptions, and no implied scenarios.
- **The design method appears to be perfect.**
    - Note: Khendek (2005) used a similar method to obtain SDL component specifications from sequence diagrams. He used the SDL **Save** construct to obtain the equivalent of a message pool.

Can these things be modeled with rendezvous communication ?

Process algebra languages usually use rendezvous communication.

- **How to model message passing communication ?**
  - In the context of LOTOS (developed in the 1980ies), explicit input queues were proposed (similar to SDL).
  - In the context of “stuck-free conformance”, Fournet (2005) models a message pool.
    - See next slide

Modeling a message pool in process algebra

- Rendezvous communication à la CSP:
  - A rendezvous takes place when an action a and a complementary action ã are executed jointly.
  - If **a** represents sending message "a" and **ã** represents the consumption of message "a" we may write as follows:
    - Sender process: a || P *where P are the next actions*
    - Receiver process: ã ; Q *where Q are the next actions*
      - *The rendezvous between a and ã will occur when the receiver is ready to execute ã (it represents the consumption of the message), while the sender has already continued with P.*

Decision power of the receiver

Different cases

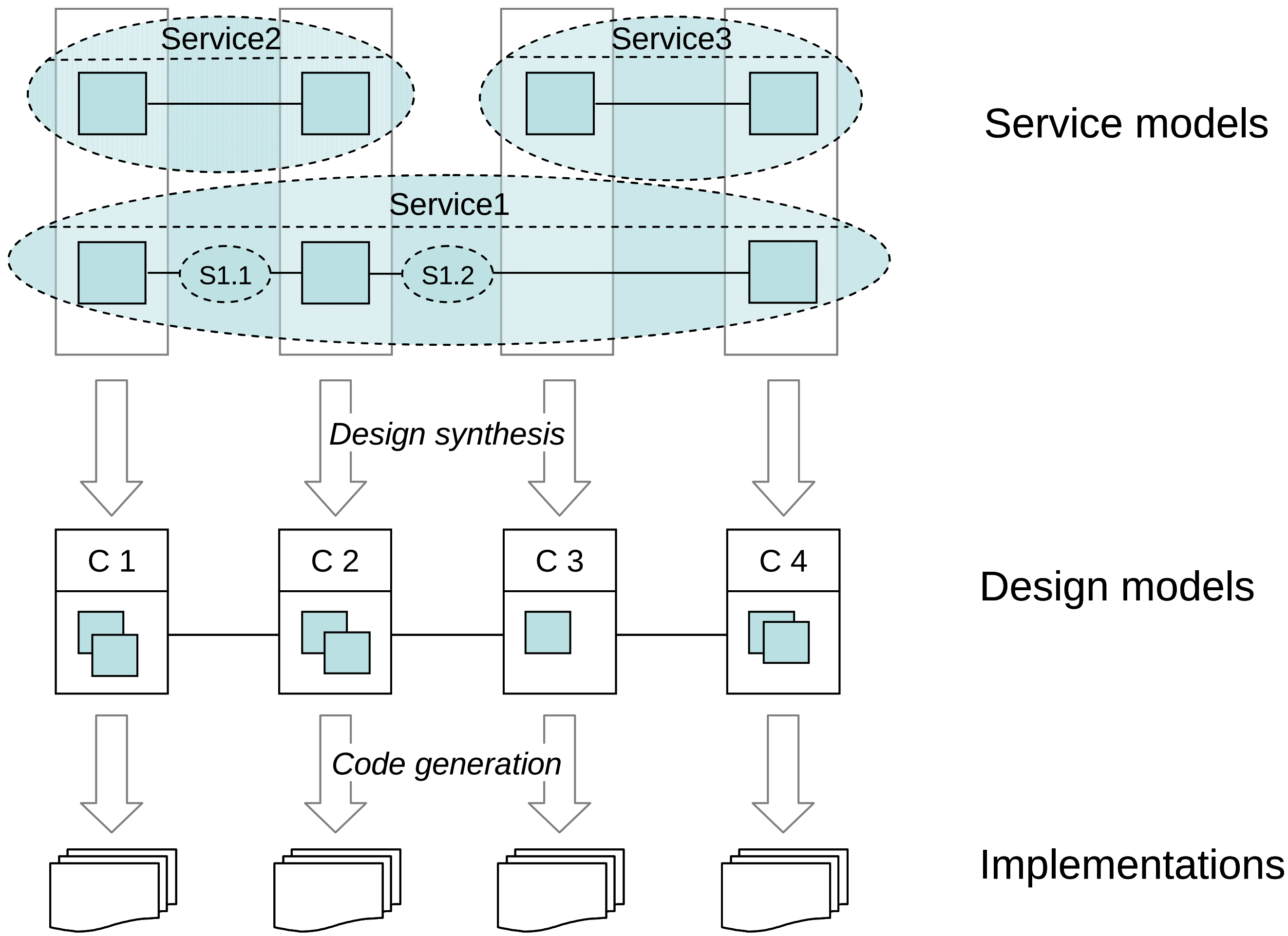
- Single FIFO input queue - No power of deciding the order of message consumption (e.g. SDL)
- Multiple input queues – decide which input queue to consider - possibly several (e.g. Estelle)
- Typed message pool – decide which type(s) of message to receive (e.g. SDL with Save, IBM's BPEL execution environment, stuck-free formalism)
- General message pool – as above, but decision may depend on parameter values of messages

Outline of talk

- History of modeling languages, concepts and tools
- Similarity of notations
- Distributed system design – an example
- Systematic design of distributed systems
- Substitution principle (inheritance) for state machines
- Concluding remarks

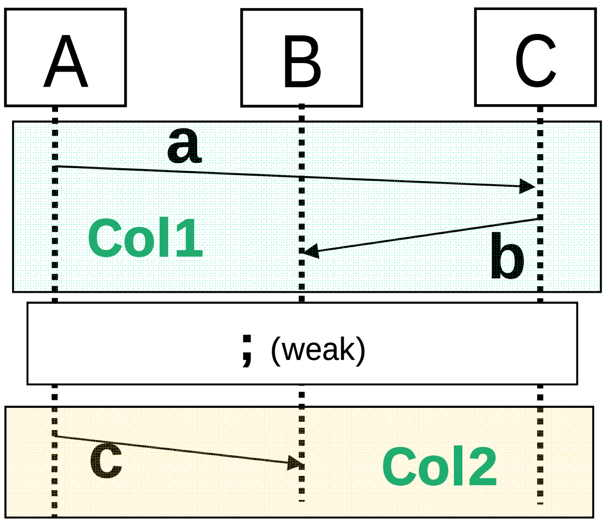
Distributed System Design from Global Requirements

Collaboration with Rolv Braek, Trondheim (2006) [Castejòn 2011]



Design method: by projection

A general method: Deriving a distributed system design from the global behavior specification by projecting the global behavior on each component *(we assume message pools)*



Hierarchical specification:
weak *sequence*
of two collaborations, Col1 and Col2

Projection on A:

$Proj_A(Col1 \;;_W Col2) =$
$Proj_A(Col1) \;;\; Proj_A(Col2)$

Resulting behaviors

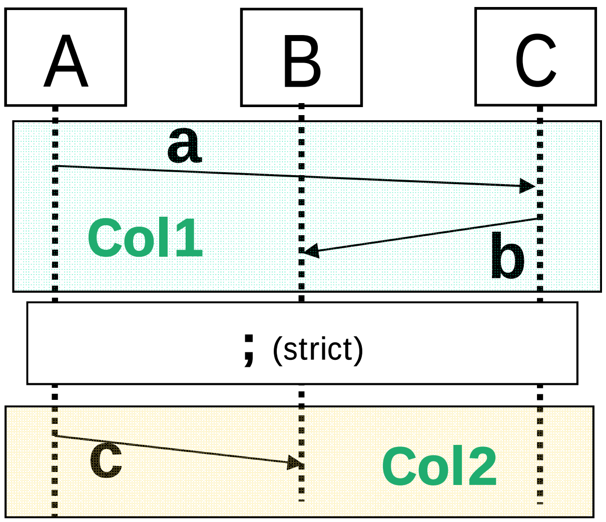
for A: a!; c!

for B: b?; c?

for C: a?; b!

The case of **strict** sequence

This case was discussed in Gotzhein & Bochmann in 1986



Hierarchical specification:
***strict** sequence*
of two collaborations, Col1 and Col2

Projection on A:

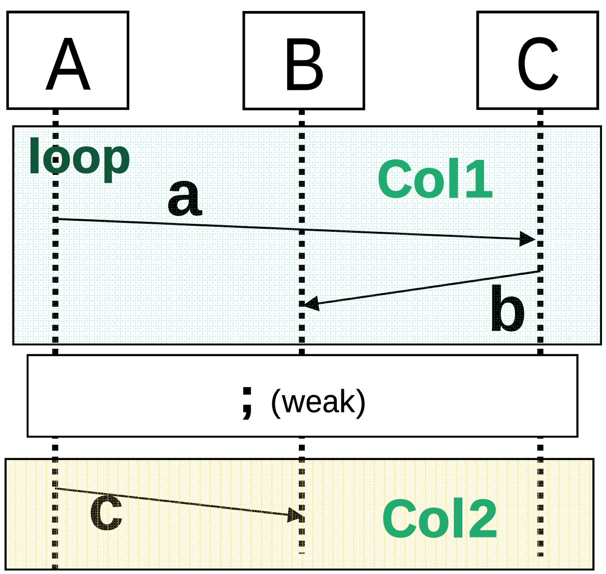
$\mathrm{Proj}_A(\mathrm{Col1}\ ;_s\ \mathrm{Col2}) =$
$\mathrm{Proj}_A(\mathrm{Col1})$;
exchange of required coordination messages ;
$\mathrm{Proj}_A(\mathrm{Col2})$

The required coordination messages depend on the terminating actions of Col1 and the initiating actions of Col2.
In this example we need a message from B to A.

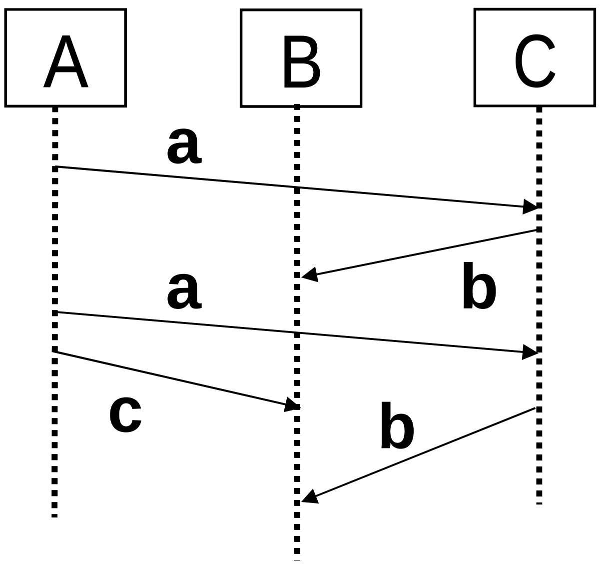
Resulting behaviors:
for A: a!; coord?; c!
for B: b?; coord!; c?
for C: a?; b!

Design method by projection: further complications

- Choice: No problem for local choice
  - However, need for coordination message ("choice indication") for components not involved in one of the alternatives (Bochmann 2008)
- Weak loop: *Problem example : Process B does not know how many b messages to expect.*

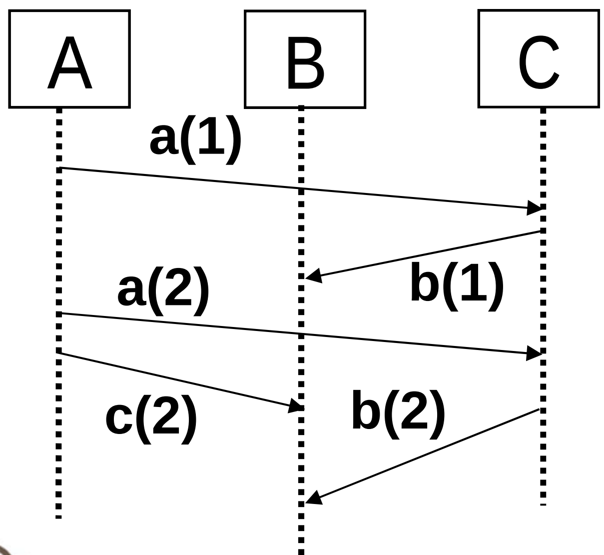


Example scenario:



Design method by projection: further complications

- Choice: No problem for local choice
  - However, need for coordination message ("choice indication") for components not involved in one of the alternatives (Bochmann 2008)
- Weak loop: **Need for message parameters representing the number of repetitions**



Example: messages b and c include repetition parameter, allowing component B to decide whether message c can be consumed (only if its repetition parameter is equal to the value in the last b message)

This means a "**general** message pool" is required (wait for messages with particular parameter values)

Outline of talk

- History of modeling languages, concepts and tools
- Similarity of notations
- Distributed system design – an example
- Systematic design of distributed systems
- Substitution principle (inheritance) for state machines
- Concluding remarks

History: Object orientation - inheritance

2005 UML-2

1992 SDL-92

1976 SDL

1955 FSM

1995 Substitution principle with assumptions (Abadi & Lamport)

1993 Conformance relations for behavior (Petrenko, Bochmann, Dssouli)

1991 Behavior specifications with assumptions (Misra & Chandy)

1987 Substitution principle based on behavior (Liskov)

Structural subtyping based on interfaces (Emerald language, Black)

1967 Object orientation, inheritance (Simula)

Substitution principle (Liskov 1987)

Principle: Where the system design asks for an instance of a class A, the implementation of a subclass B may be used.

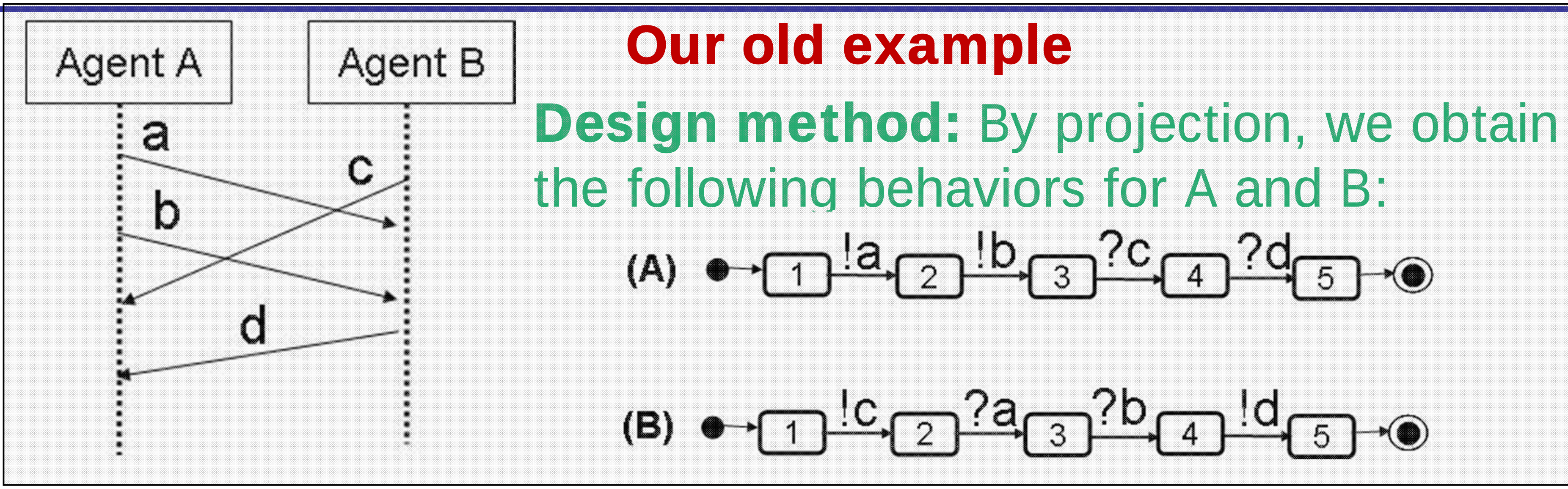
- Assuming that component properties are defined in some logic language:
  - P_I are the properties of the implementation I.
  - P_A are the properties of all instances of class A.
  - P_B are the properties of all instances of class B.
- Then: $P_I \Rightarrow P_B$ means that I is an instance of class B
- If we define "B is a subclass of A" iff $P_B \Rightarrow P_A$ then the substitution principle holds:
  - $P_I \Rightarrow P_B$ and $P_B \Rightarrow P_A$ implies $P_I \Rightarrow P_A$

Assumptions and Guarantees

Principle: A specification of a system component includes assumptions about its environment and guarantees that the component should satisfy (see for instance Abadi & Lamport 1995). Example: Method pre- and post-conditions.

- The specification of class A has the form:
  - $P_A = Ass_A \Rightarrow Gua_A$ *(if the assumptions Ass are satisfied then the behavior of the component satisfies the guarantees Gua)*
- Proposition: $Gua_B \Rightarrow Gua_A$ and $Ass_A \Rightarrow Ass_B$ implies $P_B \Rightarrow P_A$ *(B is a subclass of A)*
  - E.g. method pre-conditions for B are weaker than for A and post-conditions for B are stronger than for A
- This principles was nicely applied to interface signatures in the Emerald language (1987)

How does this apply to state machines ?



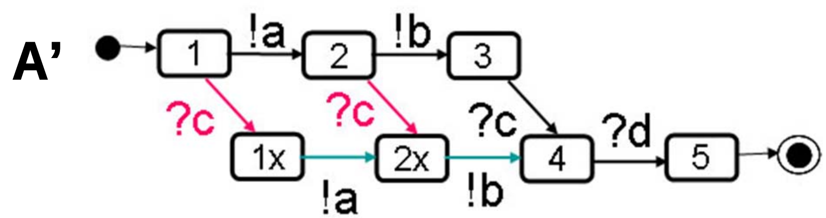
Our old example

Design method: By projection, we obtain the following behaviors for A and B:

- What is the behavior of machine A if in state 2 the message c is received ? – Three interpretations:
  1. Undefined behavior (Assumption: this will never happen)
  2. Machine goes into an error state (No assumption)
  3. Message will be dropped (SDL) (No assumption)

Subclass relationship for state machines

A' defines the behavior for the reception of c in state 2:



- Does the behavior of **A'** represent a subclass of **A** ?

Answer: This depends on the interpretation:

1. Undefined behavior: **YES**

   Assumptions of A' are weaker, and guarantees are stronger (A' has an extended behavior)

2. Machine goes into an error state: **NO**

   Guarantees are contradicting (behavior for reception of c in state 2 has been modified)

3. Message will be dropped (SDL): **NO** (similarly)

Conclusions on state machine inheritance

If the substitution principle should hold, the modeling language should include the following features:

- Inheritance of messages and their parameters: follow the approach of Emerald
- Inheritance for state machine behavior: a non-specified reception means “undefined behavior” (assumption that such a message would not be received in this state)

Conclusions

- System modeling using state machines goes back to the 1960ies
- Similarity of notations (UML state machines, activity diagrams and Use Case Maps) : do we really need so many different notations ???
- Non-local choice requires some protocol for resolving competing initiatives
  - Gouda's approach with different priorities; circulating token; or application-dependent solutions
- Rendezvous communication is an important abstract modeling paradigm
- Modeling **message consumption** with a message pool appears to be preferable to processing input messages in FIFO order
  - Avoidance of race conditions and non-specified receptions; reduction of implied scenarios
- With message consumption, a method for deriving distributed designs from a global behavior specification, based on projection, can deal with strict and weak sequencing.
- To validate the substitution principle of inheritance, an unspecified reception (partial definition) should mean that the behavior is undefined in such a case (formally an assumption about the behavior of the environment that such a case will never occur).

References

- M. Abadi and L. Lamport, *Conjoining specifications*, ACM Transactions on Programming Languages & Systems, vol.17, no.3, May 1995, pp. 507-34.
- R. Alur, K. Etessami and M. Yannakakis, "Inference of Message Sequence Charts", Proc. 22nd Intl. Conf. on Software Engineering (ICSE'00), 2000.
- T. P. Blumer and R. Tenney, *A formal specification technique and implementation method for protocols*, Computer Networks 6,3 (July 1982), pp. 201-217.
- A. Black, N. Hutchinson, E. Jul, H. Levy and L. Carter, *Distribution and Abstract Types in Emerald*, IEEE Trans. on Software Engineering, Vol. SE-13, no. 1, January 1987, pp.65-76.
- A. Black, N. Hutchinson, E. Jul, H. Levy, The development of the Emerald programming language. Proceedings of the third ACM SIGPLAN conference on History of programming languages, 2007, Pages 11-1-11-51.
- G. v. Bochmann, *Communication protocols and error recovery procedures*, Proceedings ACM Symposium on Interprocess Communication, SIGOPS Review, 9, No.3, 45-50 (1975).
- G. v. Bochmann, *Finite State Description of Communication Protocols*, Computer Networks, Vol. 2 (1978), pp. 361-372.
- G. v. Bochmann, G. Gerber and J.-M. Serre, *Semiautomatic implementation of communication protocols*, IEEE Tr. on SE, Vol. SE-13, No. 9, September 1987, pp. 989-1000.
- G. v. Bochmann, *Deriving component designs from global requirements*, Proc. Intern. Workshop on Model Based Architecting and Construction of Embedded Systems (ACES), Toulouse, Sept. 2008.
- G. v. Bochmann, D. Rayner and C. H. West, *Some notes on the history of protocol engineering*, Computer Networks journal, 54 (2010), pp 3197–3209.
- H. N. Castejòn, G. v. Bochmann and R. Braek, *On the realizability of collaborative services*, Journal of *Software and Systems Modeling*, Vol. 10 (12 October 2011), pp. 1-21.
- C. Fournet, T. Hoare, S. K. Rajamani, and J. Rehof, "Stuck-free Conformance", Proc. 16th Intl. Conf. on Computer Aided Verification (CAV'04), LNCS, vol. 3114, Springer, 2004
- M. G. Gouda and Y.-T. Yu, *Synthesis of communicating Finite State Machines with guaranteed progress*, IEEE Trans on Communications, vol. Com-32, No. 7, July 1984, pp. 779-788.
- D. Harel, *State charts: A visual formalism for complex systems*, Science of Computer Programming 8, 19987, pp. 231-274.
- G. J. Holzmann, *Limits and possibilities of protocol analysis*, Proc. IFIP Symp. on Protocol Specification, Testing and Verification: VII, North Holland, 1987, pp. 339-346.
- F. Khendek, X.J. Zhang, From MSC to SDL: Overview and an Application to the Autonomous Shuttle Transport System, LNCS Vol. 3466, 2005, pp 228-254.
- L. Lamport, *Time, clocks and the ordering of events in a distributed system*, Comm. ACM 21, 7 (July 1978), pp. 558-565.
- J. Misra and K. M. Chandy, *Proofs of networks of processes*, IEEE Tr. on SE, Vol. SE-7 (July 1991), pp. 417-426.
- A. J. Mooij, N. Goga and J. Romijn, "Non-local choice and beyond: Intricacies of MSC choice nodes", Proc. Intl. Conf. on Fundamental Approaches to Software Engineering (FASE'05), LNCS, 3442, Springer, 2005
- L. Pouzin, *Presentation and major design aspects of the CYCLADES computer network*, in Proc. 3rd ACM-IEEE Commun. Symp. Tampa, FL, Nov. 1973, pp. 80-87.
- A. Petrenko, G. v. Bochmann and R. Dssouli, *Conformance relations and test derivation*, (invited paper), Proc. Int. Workshop on Protocol Test Systems (IFIP), O. Rafiq (ed.), North Holland Publ. 1993, pp.157-178.
- C. H. West, *An automated technique of communications protocol validation*, IEEE Trans. COM-26 (1978), pp. 1271-1275.

Thanks !

Any comments

or questions ??

For copy of slides, see

http://www.eecs.uottawa.ca/~bochmann/talks/StateMachineConcepts.pdf