N-Party Rendezvous versus Sending Messages

Jean-Claude Royer[1]

[1]École des Mines de Nantes
Department of Computer Science – OBASCO Group
INRIA Research Centre Rennes - Bretagne Atlantique – LINA

20/09/2008

Outline

Outline

Outline

General Motivations

- Try to better understand the use of N-party rendezvous
- To compare its applicability with binary messages
- Not directly related to extraction of components from code
- However could be used to compare the models we have
- Kmelia uses a mutliparty, STSLib has N-party rendezvous, ...

General Motivations

- Try to better understand the use of N-party rendezvous
- To compare its applicability with binary messages
- Not directly related to extraction of components from code
- However could be used to compare the models we have
- Kmelia uses a mutliparty, STSLib has N-party rendezvous, ...

General Motivations

- Try to better understand the use of N-party rendezvous
- To compare its applicability with binary messages
- Not directly related to extraction of components from code
- However could be used to compare the models we have
- Kmelia uses a mutliparty, STSLib has N-party rendezvous, ...

General Motivations

- Try to better understand the use of N-party rendezvous
- To compare its applicability with binary messages
- Not directly related to extraction of components from code
- However could be used to compare the models we have
- Kmelia uses a mutliparty, STSLib has N-party rendezvous, ...

General Motivations

- Try to better understand the use of N-party rendezvous
- To compare its applicability with binary messages
- Not directly related to extraction of components from code
- However could be used to compare the models we have
- Kmelia uses a mutliparty, STSLib has N-party rendezvous, ...

Binary Message versus Rendezvous

- Binary message is the main way for components to communicate
- It is a sychronisation which triggers an action on the receiver side
- Rendezvous principle as in Ada is the same
- We rather want to analyse general rendezvous as in LOTOS or BIP
- 2-party rendezvous can synchronously execute an action on each side, one for the emitter and one for the receiver
- It can be viewed as an atomic and related set of binary messages

Binary Message versus Rendezvous

- Binary message is the main way for components to communicate
- It is a sychronisation which triggers an action on the receiver side
- Rendezvous principle as in Ada is the same
- We rather want to analyse general rendezvous as in LOTOS or BIP
- 2-party rendezvous can synchronously execute an action on each side, one for the emitter and one for the receiver
- It can be viewed as an atomic and related set of binary messages

Binary Message versus Rendezvous

- Binary message is the main way for components to communicate
- It is a sychronisation which triggers an action on the receiver side
- Rendezvous principle as in Ada is the same
- We rather want to analyse general rendezvous as in LOTOS or BIP
- 2-party rendezvous can synchronously execute an action on each side, one for the emitter and one for the receiver
- It can be viewed as an atomic and related set of binary messages

Binary Message versus Rendezvous

- Binary message is the main way for components to communicate
- It is a sychronisation which triggers an action on the receiver side
- Rendezvous principle as in Ada is the same
- We rather want to analyse general rendezvous as in LOTOS or BIP
- 2-party rendezvous can synchronously execute an action on each side, one for the emitter and one for the receiver
- It can be viewed as an atomic and related set of binary messages

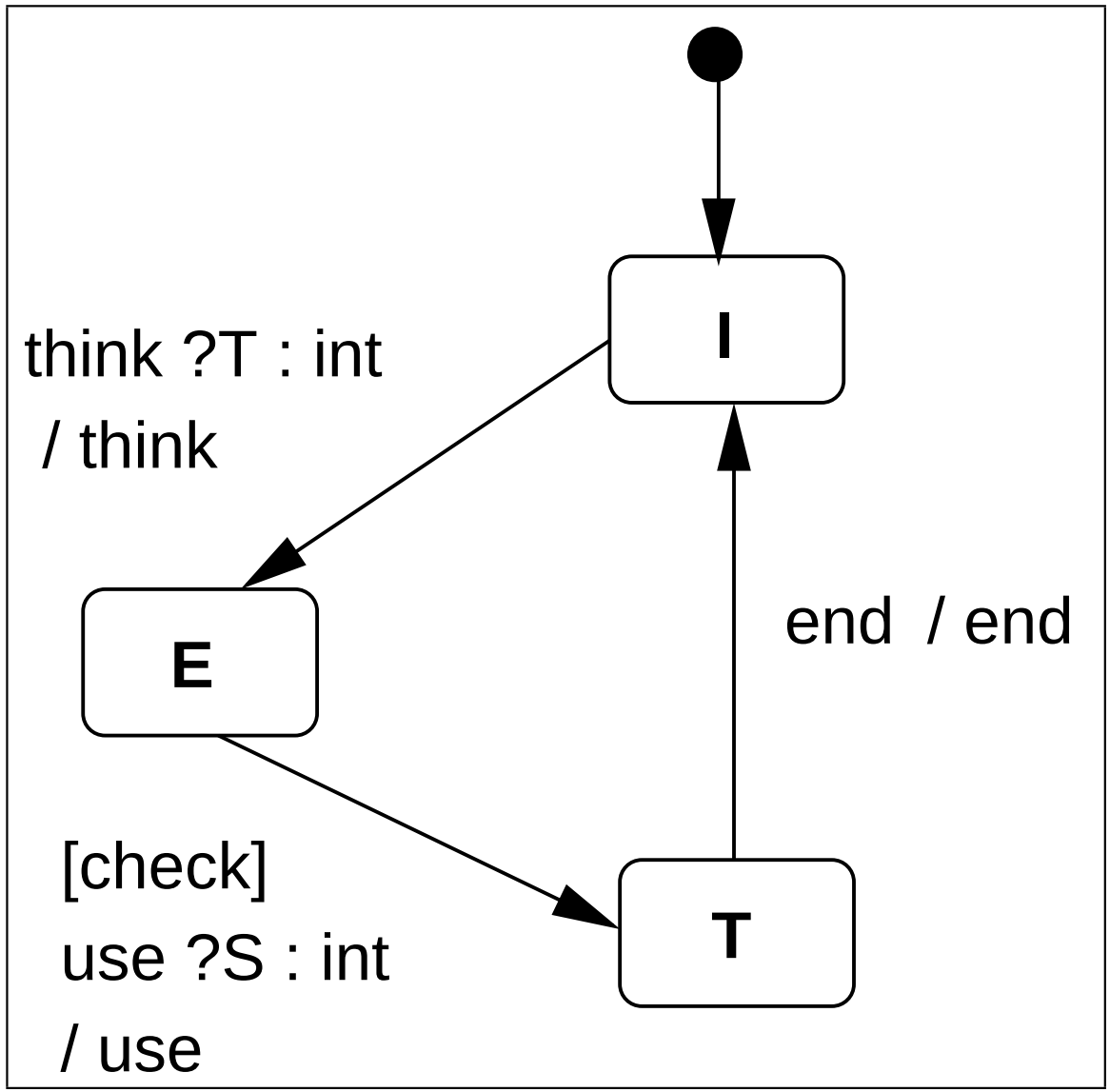
Binary Message versus Rendezvous

- Binary message is the main way for components to communicate
- It is a sychronisation which triggers an action on the receiver side
- Rendezvous principle as in Ada is the same
- We rather want to analyse general rendezvous as in LOTOS or BIP
- 2-party rendezvous can synchronously execute an action on each side, one for the emitter and one for the receiver
- It can be viewed as an atomic and related set of binary messages

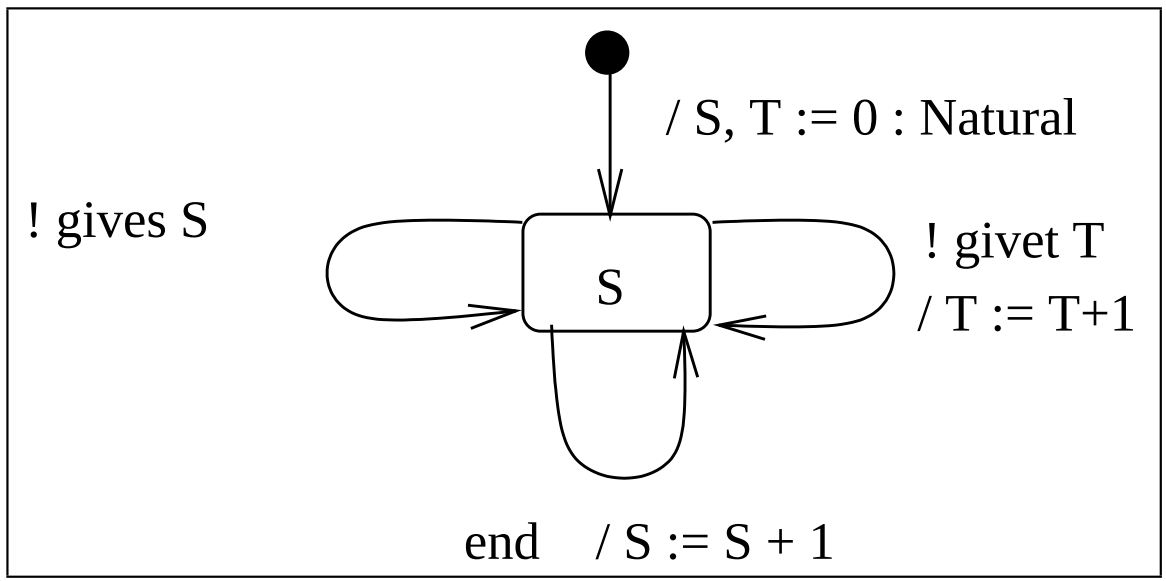
Binary Message versus Rendezvous

- Binary message is the main way for components to communicate
- It is a sychronisation which triggers an action on the receiver side
- Rendezvous principle as in Ada is the same
- We rather want to analyse general rendezvous as in LOTOS or BIP
- 2-party rendezvous can synchronously execute an action on each side, one for the emitter and one for the receiver
- It can be viewed as an atomic and related set of binary messages

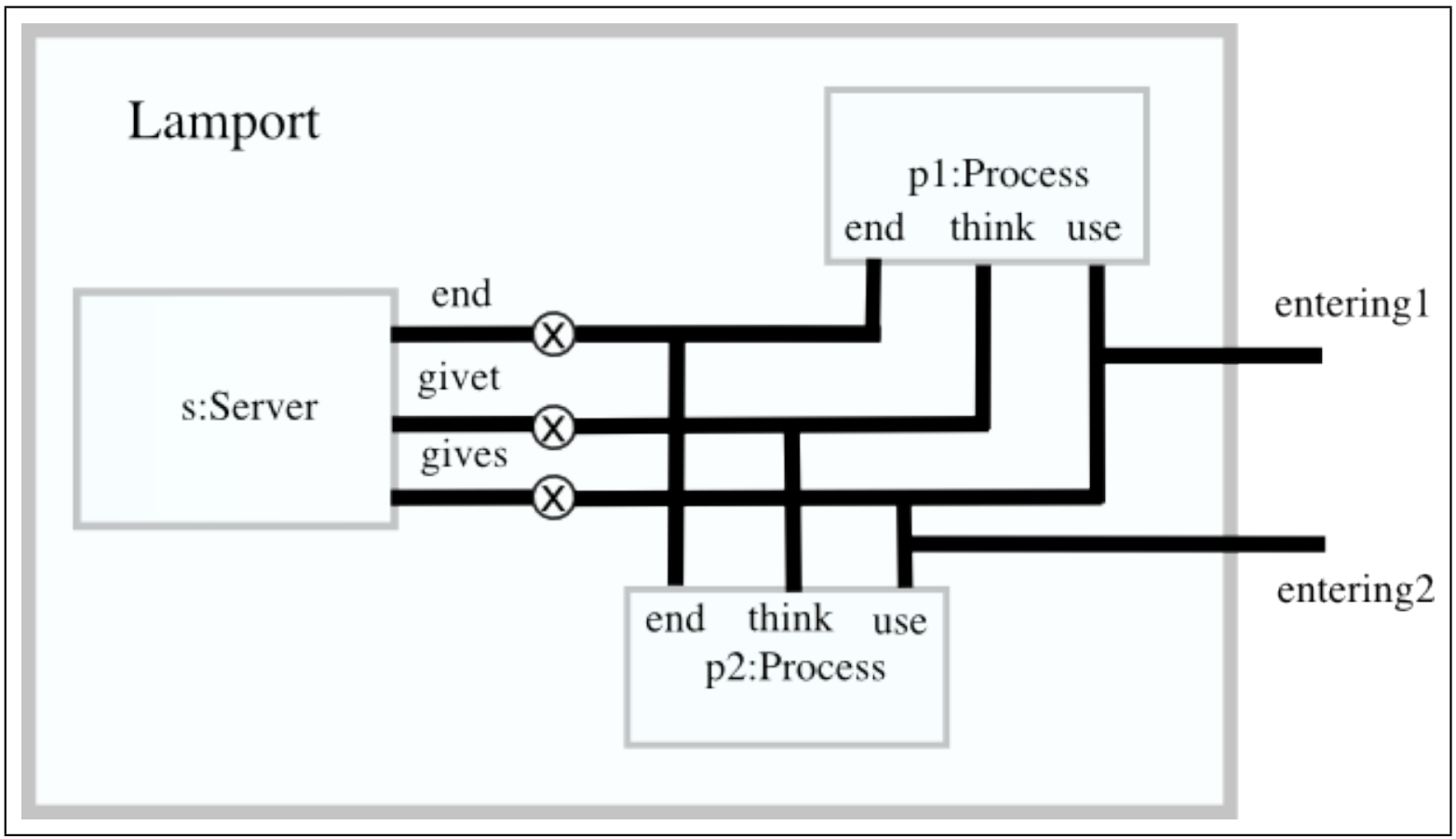
The Process Dynamic Part



The server component

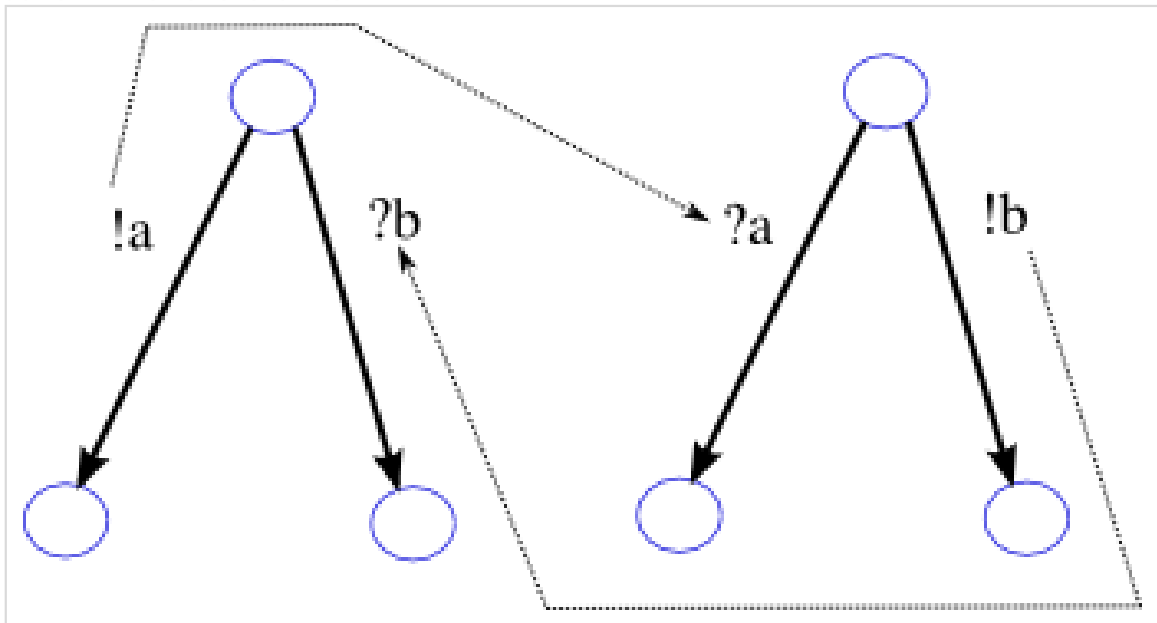


An Architecture with Two processes



Mixed State problem

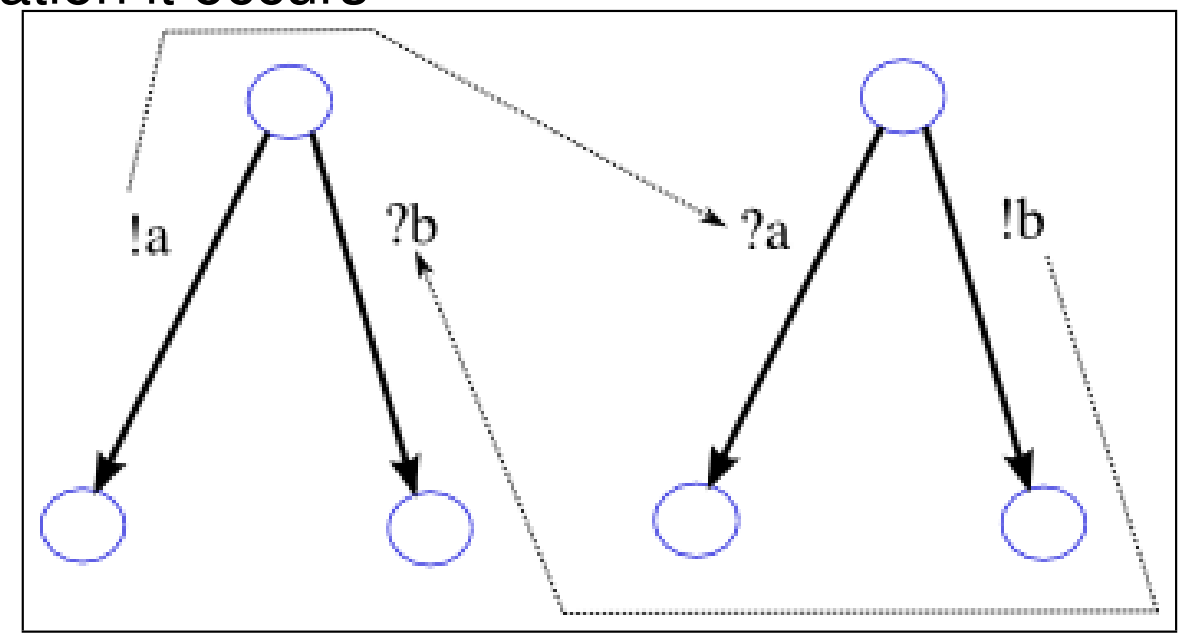
- LTS or other formalisms use a synchronisation rule which avoids this problem
- With simple messages and without a global coordination it occurs



- An STS message is a message without mixed state problem

Mixed State problem

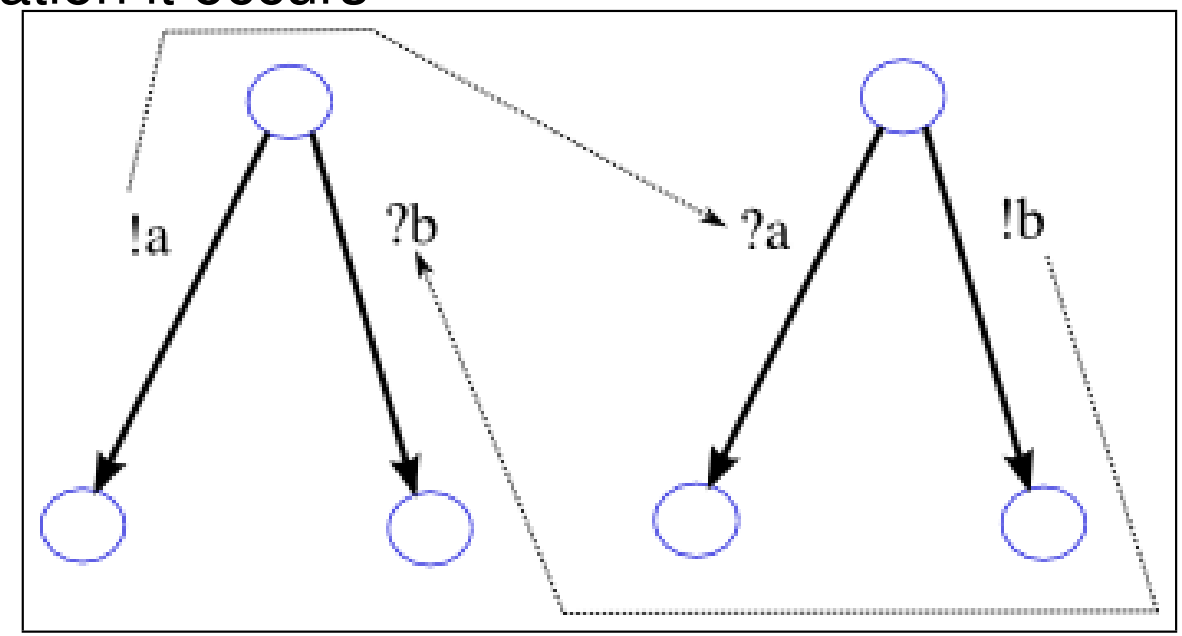
- LTS or other formalisms use a synchronisation rule which avoids this problem
- With simple messages and without a global coordination it occurs



- An STS message is a message without mixed state problem

Mixed State problem

- LTS or other formalisms use a synchronisation rule which avoids this problem
- With simple messages and without a global coordination it occurs



- An STS message is a message without mixed state problem

Some Transformations

- 2-party rendezvous can be transformed into set of messages
- Double action: `server.givet / T++ - process.think / A=T`
- Guard with receipt: `server.gives !S - process.[A=S] use ? S`
- Combination of both cases

Some Transformations

- 2-party rendezvous can be transformed into set of messages
- Double action: `server.givet / T++ - process.think / A=T`
- Guard with receipt: `server.gives !S - process.[A=S] use ? S`
- Combination of both cases

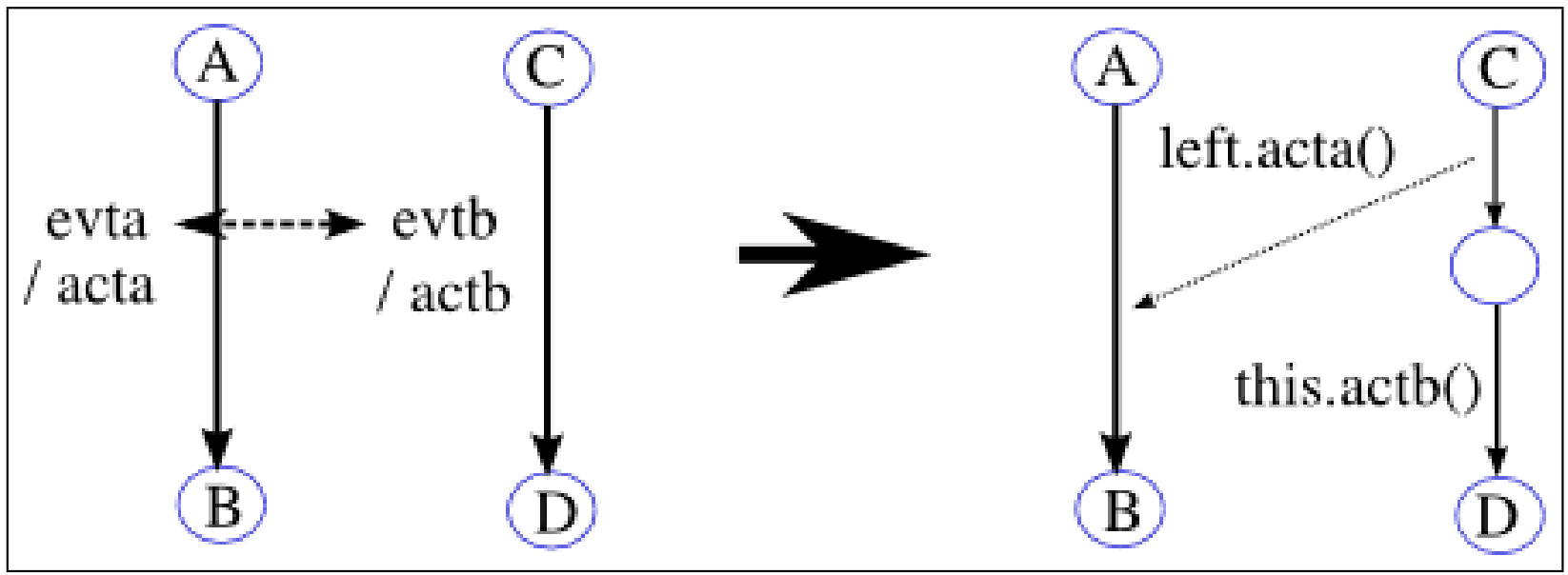
Some Transformations

- 2-party rendezvous can be transformed into set of messages
- Double action: `server.givet / T++ - process.think / A=T`
- Guard with receipt: `server.gives !S - process.[A=S] use ? S`
- Combination of both cases

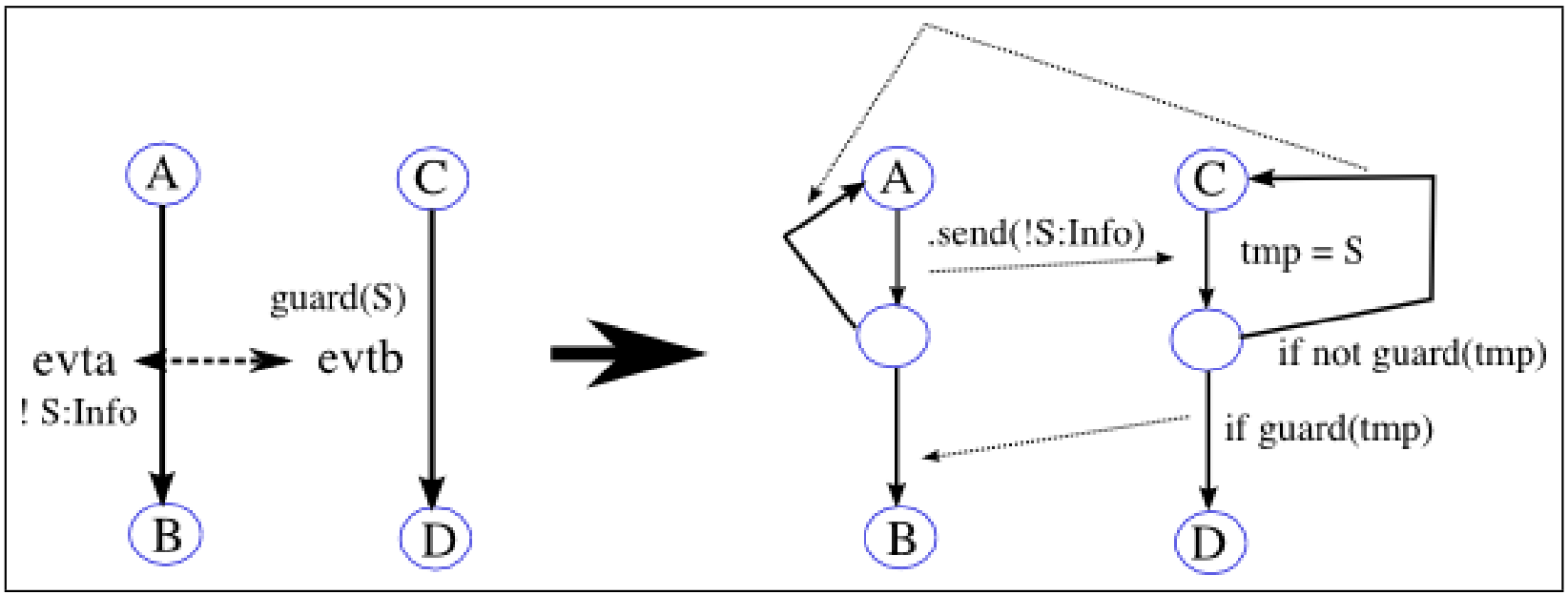
Some Transformations

- 2-party rendezvous can be transformed into set of messages
- Double action: `server.givet / T++ - process.think / A=T`
- Guard with receipt: `server.gives !S - process.[A=S] use ? S`
- Combination of both cases

Double Action Transformation



Guard with Receipt Transformation



Main principles but

- Additional constraints come from ! and ?
- Guards with receipt imply more events, more actions and more synchronisations
- There are also some possible variations with
  - message direction
  - ordering of actions
- Transformations increase the complexity (number of states and transitions)
- Double adds 1+1 and Guard with receipt 2+4

Main principles but

- Additional constraints come from ! and ?
- Guards with receipt imply more events, more actions and more synchronisations
- There are also some possible variations with
  - message direction
  - ordering of actions
- Transformations increase the complexity (number of states and transitions)
- Double adds 1+1 and Guard with receipt 2+4

Main principles but

- Additional constraints come from ! and ?
- Guards with receipt imply more events, more actions and more synchronisations
- There are also some possible variations with
  - message direction
  - ordering of actions
- Transformations increase the complexity (number of states and transitions)
- Double adds 1+1 and Guard with receipt 2+4

Main principles but

- Additional constraints come from `!` and `?`
- Guards with receipt imply more events, more actions and more synchronisations
- There are also some possible variations with
  - message direction
  - ordering of actions
- Transformations increase the complexity (number of states and transitions)
- Double adds 1+1 and Guard with receipt 2+4

Main principles but

- Additional constraints come from `!` and `?`
- Guards with receipt imply more events, more actions and more synchronisations
- There are also some possible variations with
  - message direction
  - ordering of actions
- Transformations increase the complexity (number of states and transitions)
- Double adds 1+1 and Guard with receipt 2+4

Main principles but

- Additional constraints come from `!` and `?`
- Guards with receipt imply more events, more actions and more synchronisations
- There are also some possible variations with
  - message direction
  - ordering of actions
- Transformations increase the complexity (number of states and transitions)
- Double adds 1+1 and Guard with receipt 2+4

Main principles but

- Additional constraints come from `!` and `?`
- Guards with receipt imply more events, more actions and more synchronisations
- There are also some possible variations with
  - message direction
  - ordering of actions
- Transformations increase the complexity (number of states and transitions)
- Double adds 1+1 and Guard with receipt 2+4

Lamport Example

- Design as a 2-party system, it works with any number of processes
- Change it into a binary message system
- Change one double action the `givet - think` interaction
- Change the `check` guard with receipt
- With four processes: product *3 2 and cfg *2.7 1.8
- An ongoing experiment with a smart home system

Lamport Example

- Design as a 2-party system, it works with any number of processes
- Change it into a binary message system
- Change one double action the `givet - think` interaction
- Change the `check` guard with receipt
- With four processes: product *3 2 and cfg *2.7 1.8
- An ongoing experiment with a smart home system

Lamport Example

- Design as a 2-party system, it works with any number of processes
- Change it into a binary message system
- Change one double action the `givet - think` interaction
- Change the `check` guard with receipt
- With four processes: product *3 2 and cfg *2.7 1.8
- An ongoing experiment with a smart home system

Lamport Example

- Design as a 2-party system, it works with any number of processes
- Change it into a binary message system
- Change one double action the `givet - think` interaction
- Change the `check` guard with receipt
- With four processes: product *3 2 and cfg *2.7 1.8
- An ongoing experiment with a smart home system

Lamport Example

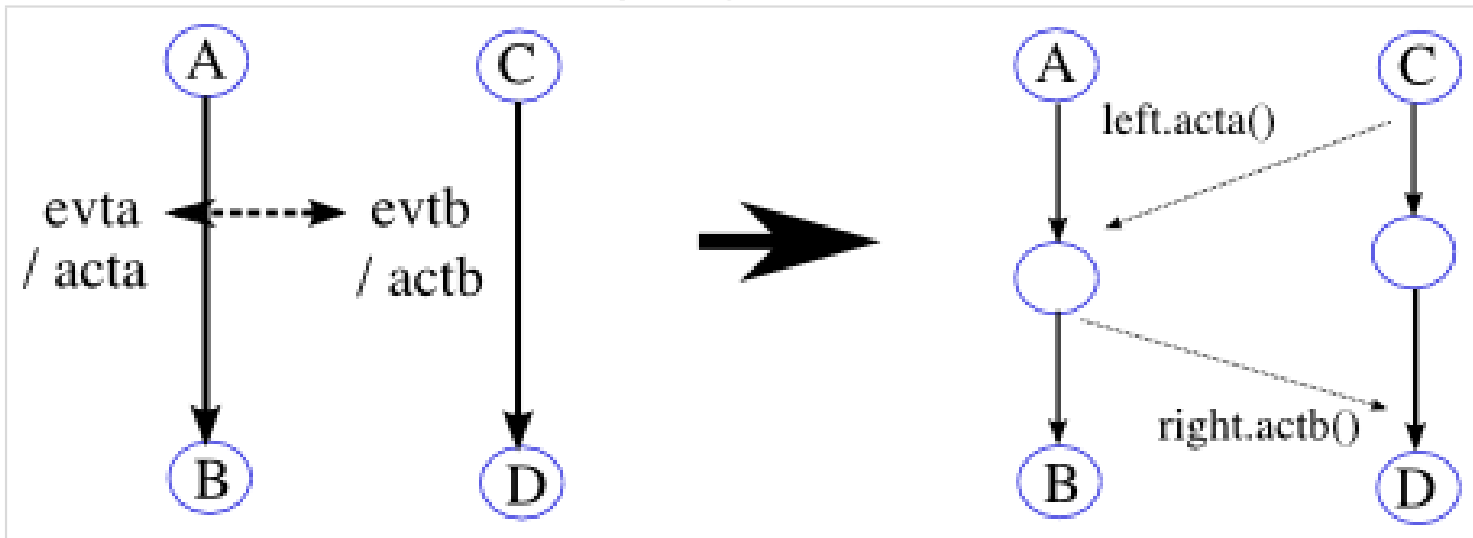
- Design as a 2-party system, it works with any number of processes
- Change it into a binary message system
- Change one double action the `givet - think` interaction
- Change the `check` guard with receipt
- With four processes: product *3 2 and cfg *2.7 1.8
- An ongoing experiment with a smart home system

Lamport Example

- Design as a 2-party system, it works with any number of processes
- Change it into a binary message system
- Change one double action the `givet - think` interaction
- Change the `check` guard with receipt
- With four processes: product *3 2 and cfg *2.7 1.8
- An ongoing experiment with a smart home system

Decoupling is not correct

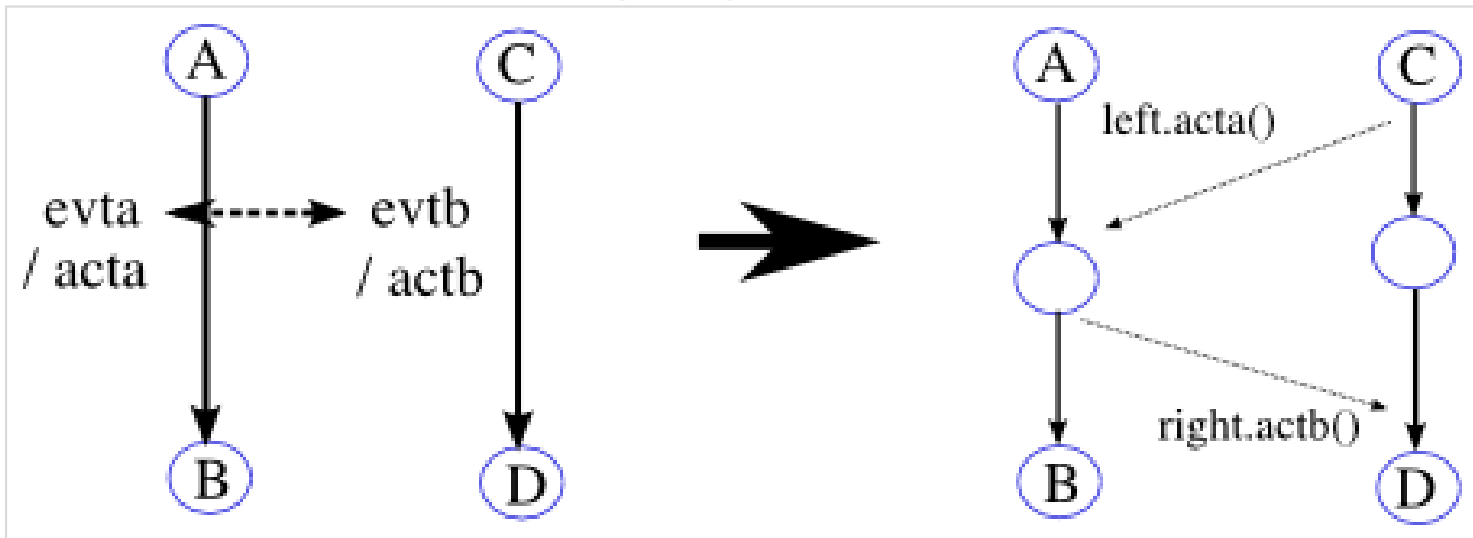
- Decoupling (case double action and others) is not correct: no end of synchronisation
- `A.a ; B.c` and actions on the components can be in reverse ordering
- One solution: a kind of 2-party rendezvous



- Cannot be generalised to N without adding more glue code

Decoupling is not correct

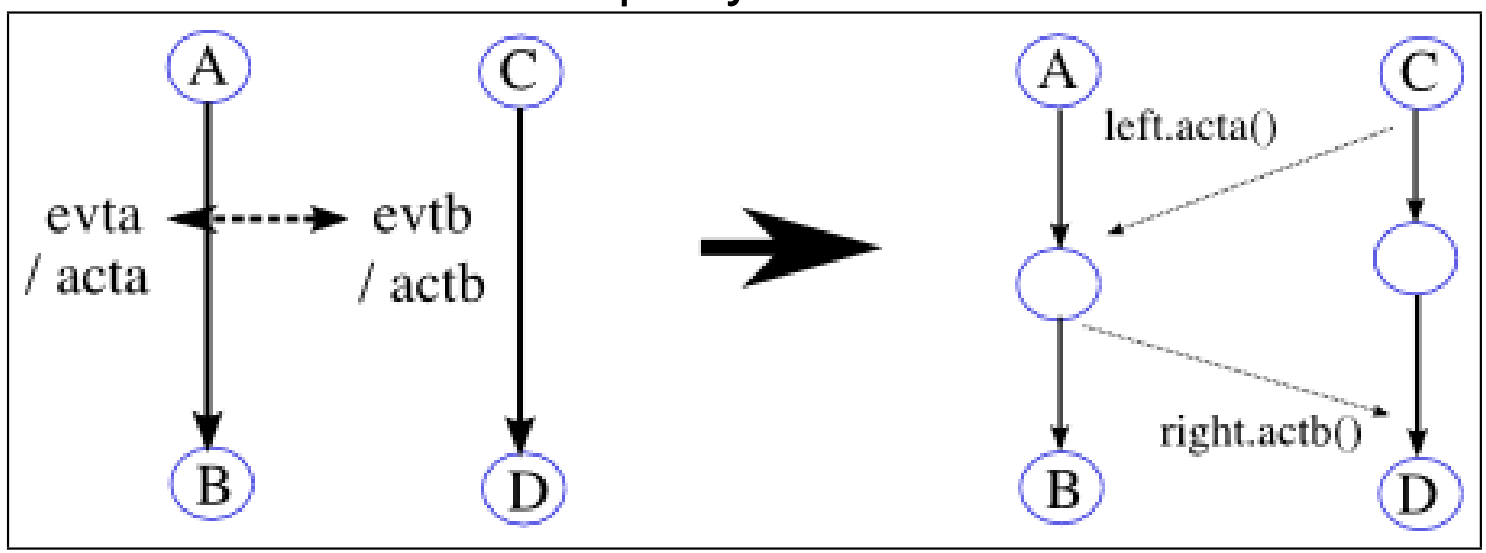
- Decoupling (case double action and others) is not correct: no end of synchronisation
- `A.a ; B.c` and actions on the components can be in reverse ordering
- One solution: a kind of 2-party rendezvous



- Cannot be generalised to N without adding more glue code

Decoupling is not correct

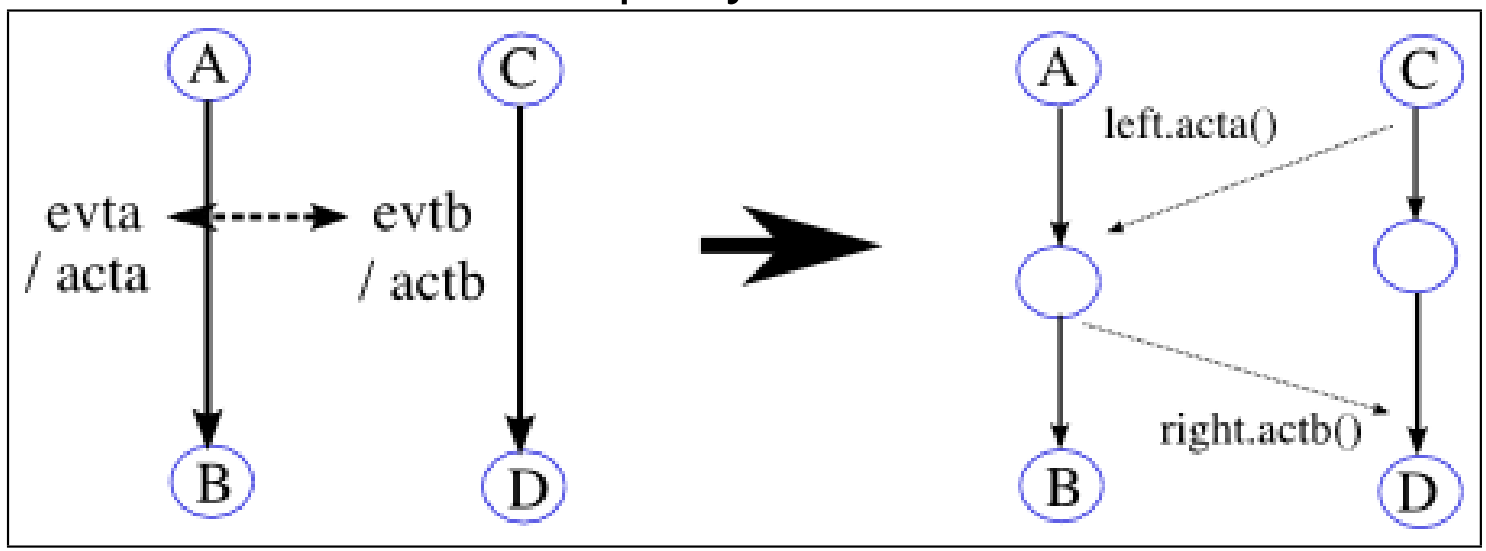
- Decoupling (case double action and others) is not correct: no end of synchronisation
- `A.a ; B.c` and actions on the components can be in reverse ordering
- One solution: a kind of 2-party rendezvous



- Cannot be generalised to N without adding more glue code

Decoupling is not correct

- Decoupling (case double action and others) is not correct: no end of synchronisation
- `A.a ; B.c` and actions on the components can be in reverse ordering
- One solution: a kind of 2-party rendezvous



- Cannot be generalised to N without adding more glue code

Message with return

- Not existing with N-party, such a generalisation does not seem natural and even useful
- Even with message it does not seem a flexible way to interact with components
- We have the feeling that most of the developers use only one way call (or generalisation of it)

Message with return

- Not existing with N-party, such a generalisation does not seem natural and even useful
- Even with message it does not seem a flexible way to interact with components
- We have the feeling that most of the developers use only one way call (or generalisation of it)

Message with return

- Not existing with N-party, such a generalisation does not seem natural and even useful
- Even with message it does not seem a flexible way to interact with components
- We have the feeling that most of the developers use only one way call (or generalisation of it)

N-party rendezvous

- N-party rendezvous is a generalisation to *N* participants
- It allows one way or multiway communication
- It needs two synchronisation barriers, one for entering the synchronisation area and one for leaving the area.
- With distributed systems it is not realistic
- However in local network it is possible to use it
- Automation systems, controller synthesis, modular robotics, ...
- Logical or even real-time rendezvous is possible (BIP)

N-party rendezvous

- N-party rendezvous is a generalisation to *N* participants
- It allows one way or multiway communication
- It needs two synchronisation barriers, one for entering the synchronisation area and one for leaving the area.
- With distributed systems it is not realistic
- However in local network it is possible to use it
- Automation systems, controller synthesis, modular robotics, ...
- Logical or even real-time rendezvous is possible (BIP)

N-party rendezvous

- N-party rendezvous is a generalisation to *N* participants
- It allows one way or multiway communication
- It needs two synchronisation barriers, one for entering the synchronisation area and one for leaving the area.
- With distributed systems it is not realistic
- However in local network it is possible to use it
- Automation systems, controller synthesis, modular robotics, ...
- Logical or even real-time rendezvous is possible (BIP)

N-party rendezvous

- N-party rendezvous is a generalisation to *N* participants
- It allows one way or multiway communication
- It needs two synchronisation barriers, one for entering the synchronisation area and one for leaving the area.
- With distributed systems it is not realistic
- However in local network it is possible to use it
- Automation systems, controller synthesis, modular robotics, ...
- Logical or even real-time rendezvous is possible (BIP)

N-party rendezvous

- N-party rendezvous is a generalisation to *N* participants
- It allows one way or multiway communication
- It needs two synchronisation barriers, one for entering the synchronisation area and one for leaving the area.
- With distributed systems it is not realistic
- However in local network it is possible to use it
- Automation systems, controller synthesis, modular robotics, ...
- Logical or even real-time rendezvous is possible (BIP)

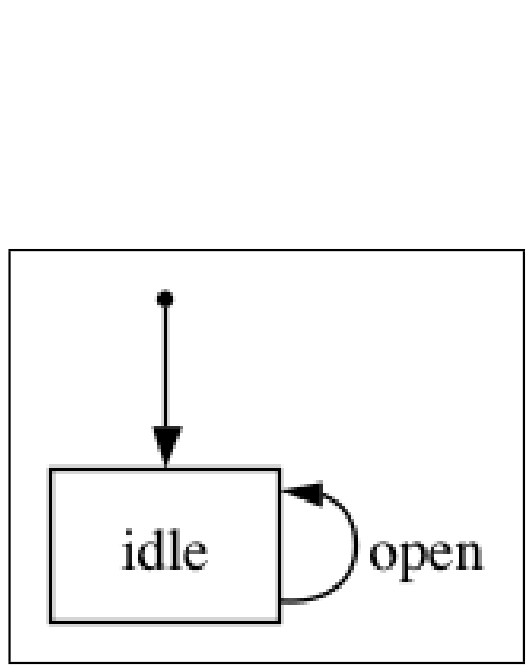
N-party rendezvous

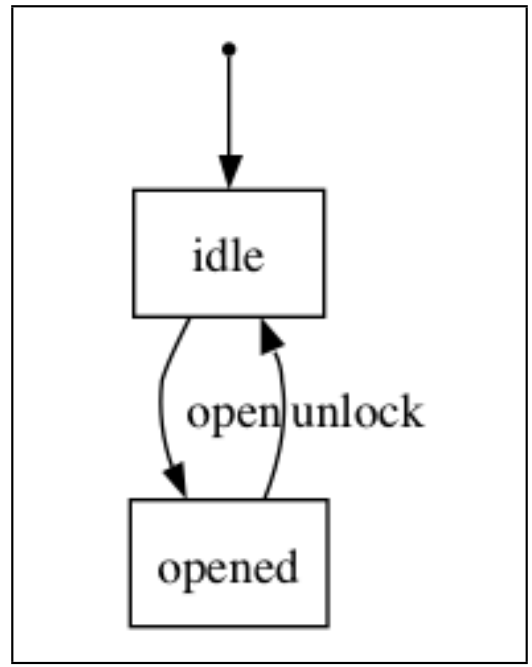
- N-party rendezvous is a generalisation to *N* participants
- It allows one way or multiway communication
- It needs two synchronisation barriers, one for entering the synchronisation area and one for leaving the area.
- With distributed systems it is not realistic
- However in local network it is possible to use it
- Automation systems, controller synthesis, modular robotics, ...
- Logical or even real-time rendezvous is possible (BIP)

N-party rendezvous

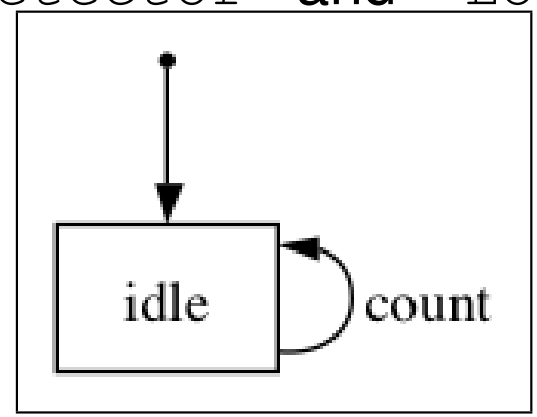
- N-party rendezvous is a generalisation to *N* participants
- It allows one way or multiway communication
- It needs two synchronisation barriers, one for entering the synchronisation area and one for leaving the area.
- With distributed systems it is not realistic
- However in local network it is possible to use it
- Automation systems, controller synthesis, modular robotics, ...
- Logical or even real-time rendezvous is possible (BIP)

The Primitive Components



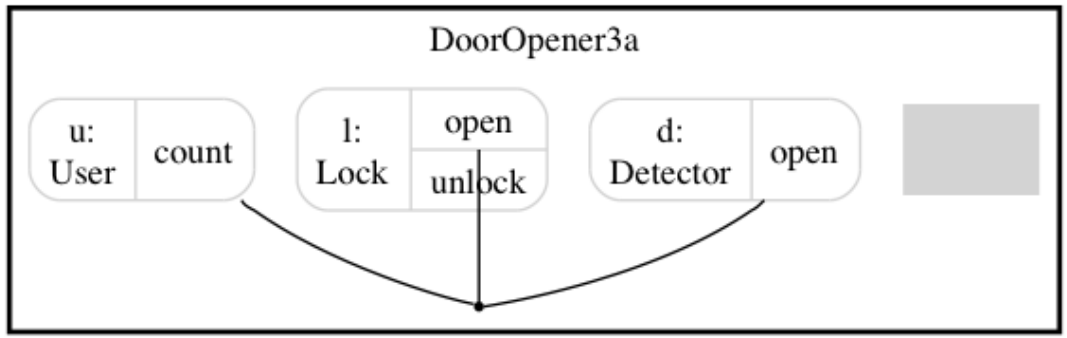


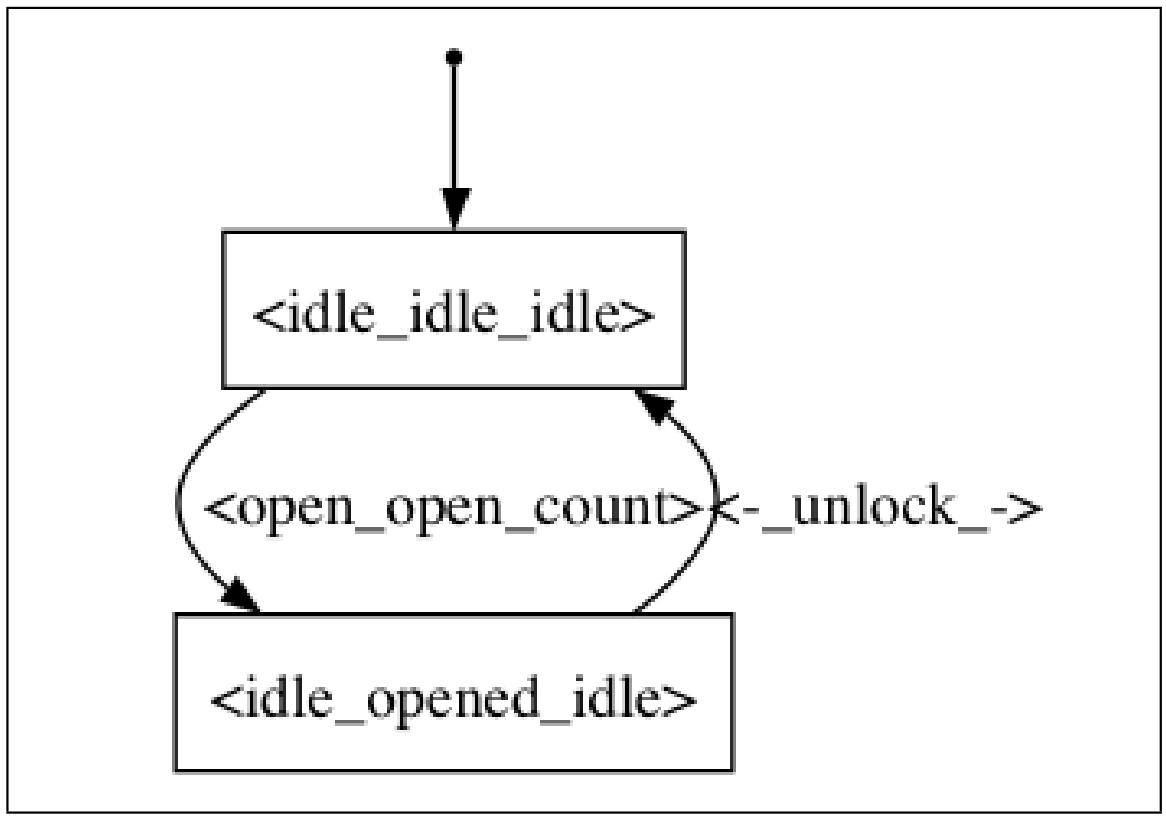
`Detector` and `Lock`



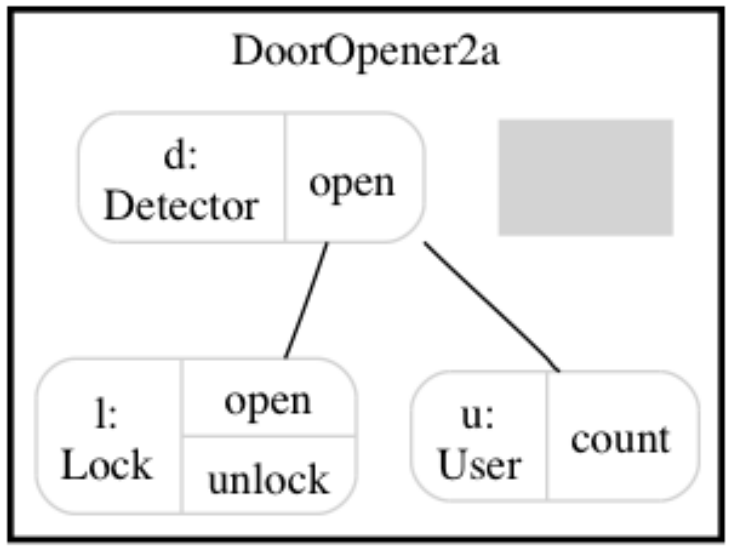
`User`

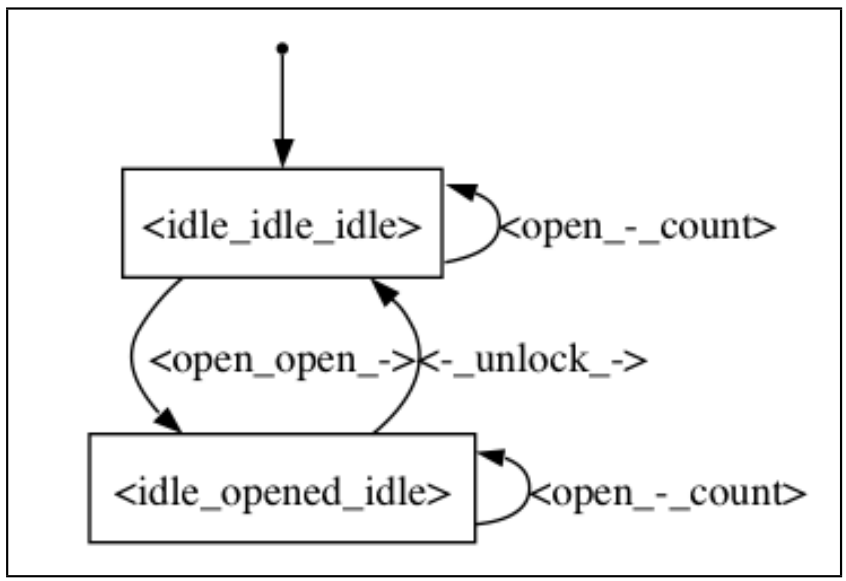
DoorOpener with 3-party





DoorOpener with 2-party





DoorOpener with 2-party

- There is no way to compound the components to get the same behaviour
- Not a proof but may be not too difficult to see
- Even we can get non compatible systems
- Without adding new behaviours we cannot realise the same behaviour
- Thus it needs some adaptors realising the synchronisation of several participants

DoorOpener with 2-party

- There is no way to compound the components to get the same behaviour
- Not a proof but may be not too difficult to see
- Even we can get non compatible systems
- Without adding new behaviours we cannot realise the same behaviour
- Thus it needs some adaptors realising the synchronisation of several participants

DoorOpener with 2-party

- There is no way to compound the components to get the same behaviour
- Not a proof but may be not too difficult to see
- Even we can get non compatible systems
- Without adding new behaviours we cannot realise the same behaviour
- Thus it needs some adaptors realising the synchronisation of several participants

DoorOpener with 2-party

- There is no way to compound the components to get the same behaviour
- Not a proof but may be not too difficult to see
- Even we can get non compatible systems
- Without adding new behaviours we cannot realise the same behaviour
- Thus it needs some adaptors realising the synchronisation of several participants

DoorOpener with 2-party

- There is no way to compound the components to get the same behaviour
- Not a proof but may be not too difficult to see
- Even we can get non compatible systems
- Without adding new behaviours we cannot realise the same behaviour
- Thus it needs some adaptors realising the synchronisation of several participants

Summary

- There are strictly more opportunity to compound with N-party: there are more combinations and some of them cannot be realised with binary interactions
- Powerful, but costly, mechanism
- Local network
- More abstract thus simpler behaviour
- On the other hand: The use of messages
  - Increase complexity and add problems
  - Add adaptors or controllers thus more or less a N-party mechanism
  - Safe transformation ? possible

Summary

- There are strictly more opportunity to compound with N-party: there are more combinations and some of them cannot be realised with binary interactions
- Powerful, but costly, mechanism
- Local network
- More abstract thus simpler behaviour
- On the other hand: The use of messages
  - Increase complexity and add problems
  - Add adaptors or controllers thus more or less a N-party mechanism
  - Safe transformation ? possible

Summary

- There are strictly more opportunity to compound with N-party: there are more combinations and some of them cannot be realised with binary interactions
- Powerful, but costly, mechanism
- Local network
- More abstract thus simpler behaviour
- On the other hand: The use of messages
  - Increase complexity and add problems
  - Add adaptors or controllers thus more or less a N-party mechanism
  - Safe transformation ? possible

Summary

- There are strictly more opportunity to compound with N-party: there are more combinations and some of them cannot be realised with binary interactions
- Powerful, but costly, mechanism
- Local network
- More abstract thus simpler behaviour
- On the other hand: The use of messages
  - Increase complexity and add problems
  - Add adaptors or controllers thus more or less a N-party mechanism
  - Safe transformation ? possible

Summary

- There are strictly more opportunity to compound with N-party: there are more combinations and some of them cannot be realised with binary interactions
- Powerful, but costly, mechanism
- Local network
- More abstract thus simpler behaviour
- On the other hand: The use of messages
  - Increase complexity and add problems
  - Add adaptors or controllers thus more or less a N-party mechanism
  - Safe transformation ? possible

Summary

- There are strictly more opportunity to compound with N-party: there are more combinations and some of them cannot be realised with binary interactions
- Powerful, but costly, mechanism
- Local network
- More abstract thus simpler behaviour
- On the other hand: The use of messages
  - Increase complexity and add problems
  - Add adaptors or controllers thus more or less a N-party mechanism
  - Safe transformation ? possible

Summary

- There are strictly more opportunity to compound with N-party: there are more combinations and some of them cannot be realised with binary interactions
- Powerful, but costly, mechanism
- Local network
- More abstract thus simpler behaviour
- On the other hand: The use of messages
  - Increase complexity and add problems
  - Add adaptors or controllers thus more or less a N-party mechanism
  - Safe transformation ? possible

Summary

- There are strictly more opportunity to compound with N-party: there are more combinations and some of them cannot be realised with binary interactions
- Powerful, but costly, mechanism
- Local network
- More abstract thus simpler behaviour
- On the other hand: The use of messages
  - Increase complexity and add problems
  - Add adaptors or controllers thus more or less a N-party mechanism
  - Safe transformation ? possible