Components with N-Party Rendezvous and Symbolic Transition Systems

Fabrício de Alexandria Fernandes[1]
Robin Passama[2]
Jean-Claude Royer[1]

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

13/05/2008

Outline

Outline

Outline

Outline

Outline

Outline

General Motivations

- Component based software engineering: To get a formal and executable model
- Explicit protocols integrated into component interfaces to describe their behaviour in a formal way
- Explicit protocols are often dissociated from component code
- Generally only binary synchronisation are provided
- Computing more than flat behavioural models
- Fill the gap between high-level formal models and implementation of protocols
- Formal analysis methods to verify components and their interactions
- Tool support: an API with parsers, and some analysis tools

http://www.emn.fr/x-info/jroyer/WEBLIB/index.html

General Motivations

- Component based software engineering: To get a formal and executable model
- Explicit protocols integrated into component interfaces to describe their behaviour in a formal way
- Explicit protocols are often dissociated from component code
- Generally only binary synchronisation are provided
- Computing more than flat behavioural models
- Fill the gap between high-level formal models and implementation of protocols
- Formal analysis methods to verify components and their interactions
- Tool support: an API with parsers, and some analysis tools

http://www.emn.fr/x-info/jroyer/WEBLIB/index.html

General Motivations

- Component based software engineering: To get a formal and executable model
- Explicit protocols integrated into component interfaces to describe their behaviour in a formal way
- Explicit protocols are often dissociated from component code
- Generally only binary synchronisation are provided
- Computing more than flat behavioural models
- Fill the gap between high-level formal models and implementation of protocols
- Formal analysis methods to verify components and their interactions
- Tool support: an API with parsers, and some analysis tools

`http://www.emn.fr/x-info/jroyer/WEBLIB/index.html`

General Motivations

- Component based software engineering: To get a formal and executable model
- Explicit protocols integrated into component interfaces to describe their behaviour in a formal way
- Explicit protocols are often dissociated from component code
- Generally only binary synchronisation are provided
- Computing more than flat behavioural models
- Fill the gap between high-level formal models and implementation of protocols
- Formal analysis methods to verify components and their interactions
- Tool support: an API with parsers, and some analysis tools

http://www.emn.fr/x-info/jroyer/WEBLIB/index.html

General Motivations

- Component based software engineering: To get a formal and executable model
- Explicit protocols integrated into component interfaces to describe their behaviour in a formal way
- Explicit protocols are often dissociated from component code
- Generally only binary synchronisation are provided
- Computing more than flat behavioural models
- Fill the gap between high-level formal models and implementation of protocols
- Formal analysis methods to verify components and their interactions
- Tool support: an API with parsers, and some analysis tools

http://www.emn.fr/x-info/jroyer/WEBLIB/index.html

General Motivations

- Component based software engineering: To get a formal and executable model
- Explicit protocols integrated into component interfaces to describe their behaviour in a formal way
- Explicit protocols are often dissociated from component code
- Generally only binary synchronisation are provided
- Computing more than flat behavioural models
- Fill the gap between high-level formal models and implementation of protocols
- Formal analysis methods to verify components and their interactions
- Tool support: an API with parsers, and some analysis tools

http://www.emn.fr/x-info/jroyer/WEBLIB/index.html

General Motivations

- Component based software engineering: To get a formal and executable model
- Explicit protocols integrated into component interfaces to describe their behaviour in a formal way
- Explicit protocols are often dissociated from component code
- Generally only binary synchronisation are provided
- Computing more than flat behavioural models
- Fill the gap between high-level formal models and implementation of protocols
- Formal analysis methods to verify components and their interactions
- Tool support: an API with parsers, and some analysis tools

`http://www.emn.fr/x-info/jroyer/WEBLIB/index.html`

General Motivations

- Component based software engineering: To get a formal and executable model
- Explicit protocols integrated into component interfaces to describe their behaviour in a formal way
- Explicit protocols are often dissociated from component code
- Generally only binary synchronisation are provided
- Computing more than flat behavioural models
- Fill the gap between high-level formal models and implementation of protocols
- Formal analysis methods to verify components and their interactions
- Tool support: an API with parsers, and some analysis tools

`http://www.emn.fr/x-info/jroyer/WEBLIB/index.html`

The Lamport Algorithm

- One server and several processes
- The server has to manage mutual exclusion of the processes
- Here two processes to simplify
- Several variants which use integer (bounded or not) to control critical section
- Ensures mutual exclusion, deadlock freeness, but not fairness

The Lamport Algorithm

- One server and several processes
- The server has to manage mutual exclusion of the processes
- Here two processes to simplify
- Several variants which use integer (bounded or not) to control critical section
- Ensures mutual exclusion, deadlock freeness, but not fairness

The Lamport Algorithm

- One server and several processes
- The server has to manage mutual exclusion of the processes
- Here two processes to simplify
- Several variants which use integer (bounded or not) to control critical section
- Ensures mutual exclusion, deadlock freeness, but not fairness

The Lamport Algorithm

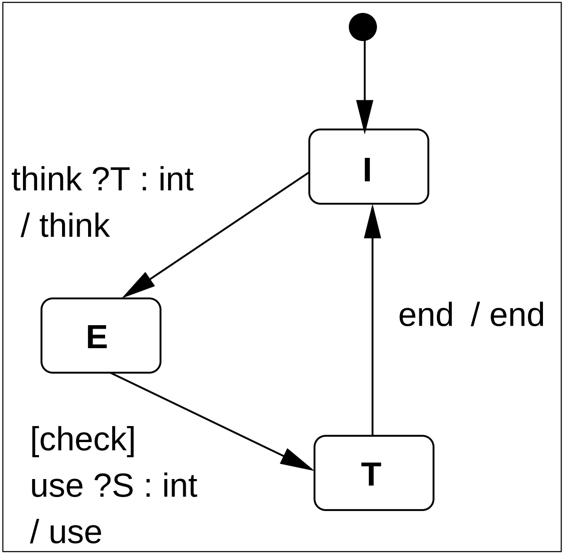
- One server and several processes
- The server has to manage mutual exclusion of the processes
- Here two processes to simplify
- Several variants which use integer (bounded or not) to control critical section
- Ensures mutual exclusion, deadlock freeness, but not fairness

The Lamport Algorithm

- One server and several processes
- The server has to manage mutual exclusion of the processes
- Here two processes to simplify
- Several variants which use integer (bounded or not) to control critical section
- Ensures mutual exclusion, deadlock freeness, but not fairness

Outline

The Process Dynamic Part



STS State Machine

- Symbolic Transition System = Dynamic Part + Data Part
- Dynamic Part: Guarded Input/Output finite state machine with actions
- Data Part: Might be provided as an ADT specification
- The ADT to Java translation has been tested but not integrated
- Data Part = a Java interface (optional) and a Java class

STS State Machine

- Symbolic Transition System = Dynamic Part + Data Part
- Dynamic Part: Guarded Input/Output finite state machine with actions
- Data Part: Might be provided as an ADT specification
- The ADT to Java translation has been tested but not integrated
- Data Part = a Java interface (optional) and a Java class

STS State Machine

- Symbolic Transition System = Dynamic Part + Data Part
- Dynamic Part: Guarded Input/Output finite state machine with actions
- Data Part: Might be provided as an ADT specification
- The ADT to Java translation has been tested but not integrated
- Data Part = a Java interface (optional) and a Java class

STS State Machine

- Symbolic Transition System = Dynamic Part + Data Part
- Dynamic Part: Guarded Input/Output finite state machine with actions
- Data Part: Might be provided as an ADT specification
- The ADT to Java translation has been tested but not integrated
- Data Part = a Java interface (optional) and a Java class

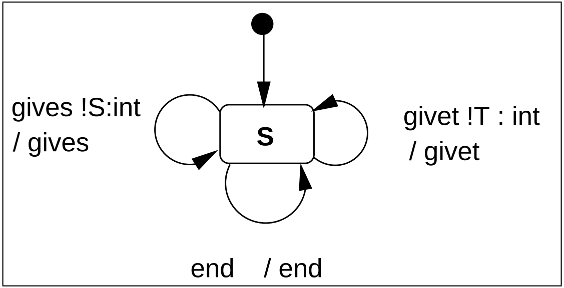
STS State Machine

- Symbolic Transition System = Dynamic Part + Data Part
- Dynamic Part: Guarded Input/Output finite state machine with actions
- Data Part: Might be provided as an ADT specification
- The ADT to Java translation has been tested but not integrated
- Data Part = a Java interface (optional) and a Java class

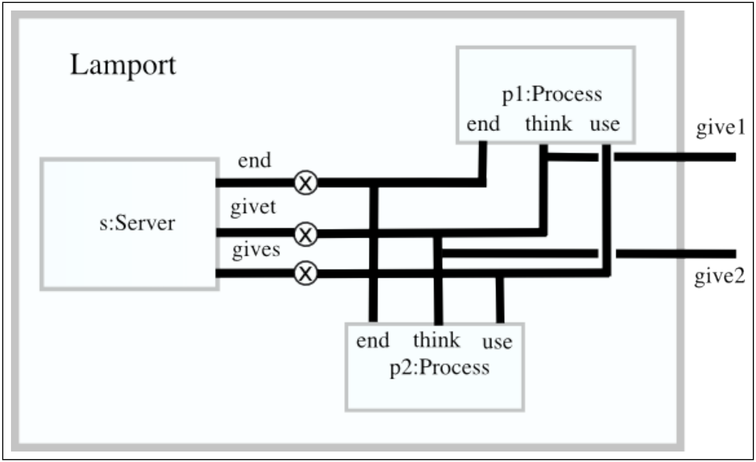
The Process Java Class

```
public class Process extends Data {

  //The local ticket.
  public int A;

  // Default constructor.
  public Process() { this.A = 0; }

  // Get a ticket.
  public void think(int t) { this.A = t; }

  // the check guard
  public boolean check(int s)
                     { return this.A == s; }

  // Enter critical section.
  public void use(int s) { }

  // End section.
  public void end() {}
}
```

The server component



An Architecture with Two processes



Lamport Architecture

```
ROOT examples  lamport
PACKAGE examples.lamport

NEED   Server::
       Process::

LOCALS s:Server p1 p2 : Process

COMMUNICATIONS XOR s.givet p1.think p2.think
COMMUNICATIONS XOR s.gives p1.use p2.use
COMMUNICATIONS XOR s.end p1.end p2.end

BINDINGS s.givet p1.think -> give1
BINDINGS s.givet p2.think -> give2
```

N-Party Rendezvous

- Previous first communication line is equivalent to:
  `COMMUNICATIONS AND s.givet p1.think`
  `COMMUNICATIONS AND s.givet p2.think`
- It expresses two synchronisations (here binary rendezvous)
- N-party rendezvous (`AND`) can involve any number of participants
- All of them execute “synchronously” their actions
- It allows one way but multiple value exchanges
- Communication requires compatibility offer

N-Party Rendezvous

- Previous first communication line is equivalent to:
  `COMMUNICATIONS AND s.givet p1.think`
  `COMMUNICATIONS AND s.givet p2.think`
- It expresses two synchronisations (here binary rendezvous)
- N-party rendezvous (`AND`) can involve any number of participants
- All of them execute “synchronously” their actions
- It allows one way but multiple value exchanges
- Communication requires compatibility offer

N-Party Rendezvous

- Previous first communication line is equivalent to:
  `COMMUNICATIONS AND s.givet p1.think`
  `COMMUNICATIONS AND s.givet p2.think`
- It expresses two synchronisations (here binary rendezvous)
- N-party rendezvous (`AND`) can involve any number of participants
- All of them execute “synchronously” their actions
- It allows one way but multiple value exchanges
- Communication requires compatibility offer

N-Party Rendezvous

- Previous first communication line is equivalent to:
  `COMMUNICATIONS AND s.givet p1.think`
  `COMMUNICATIONS AND s.givet p2.think`
- It expresses two synchronisations (here binary rendezvous)
- N-party rendezvous (`AND`) can involve any number of participants
- All of them execute “synchronously” their actions
- It allows one way but multiple value exchanges
- Communication requires compatibility offer

N-Party Rendezvous

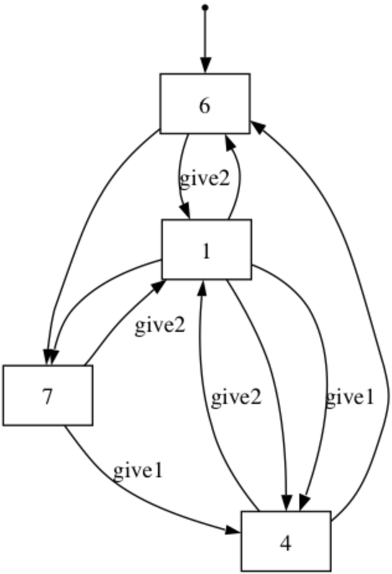
- Previous first communication line is equivalent to:
  `COMMUNICATIONS AND s.givet p1.think`
  `COMMUNICATIONS AND s.givet p2.think`
- It expresses two synchronisations (here binary rendezvous)
- N-party rendezvous (`AND`) can involve any number of participants
- All of them execute “synchronously” their actions
- It allows one way but multiple value exchanges
- Communication requires compatibility offer

N-Party Rendezvous

- Previous first communication line is equivalent to:
  `COMMUNICATIONS AND s.givet p1.think`
  `COMMUNICATIONS AND s.givet p2.think`
- It expresses two synchronisations (here binary rendezvous)
- N-party rendezvous (`AND`) can involve any number of participants
- All of them execute “synchronously” their actions
- It allows one way but multiple value exchanges
- Communication requires compatibility offer

Outline

A Classic View



Principle Summary

- Computing a flat behaviour is not sufficient for architectures
- Based on the synchronous product of LTS
- But need to consider guards, communications and actions
- Communications are synchronisation vectors
- Guards, events and actions are structured
- But it cannot takes into account a complex hierarchy with more than one level

Principle Summary

- Computing a flat behaviour is not sufficient for architectures
- Based on the synchronous product of LTS
- But need to consider guards, communications and actions
- Communications are synchronisation vectors
- Guards, events and actions are structured
- But it cannot takes into account a complex hierarchy with more than one level

Principle Summary

- Computing a flat behaviour is not sufficient for architectures
- Based on the synchronous product of LTS
- But need to consider guards, communications and actions
- Communications are synchronisation vectors
- Guards, events and actions are structured
- But it cannot takes into account a complex hierarchy with more than one level

Principle Summary

- Computing a flat behaviour is not sufficient for architectures
- Based on the synchronous product of LTS
- But need to consider guards, communications and actions
- Communications are synchronisation vectors
- Guards, events and actions are structured
- But it cannot takes into account a complex hierarchy with more than one level

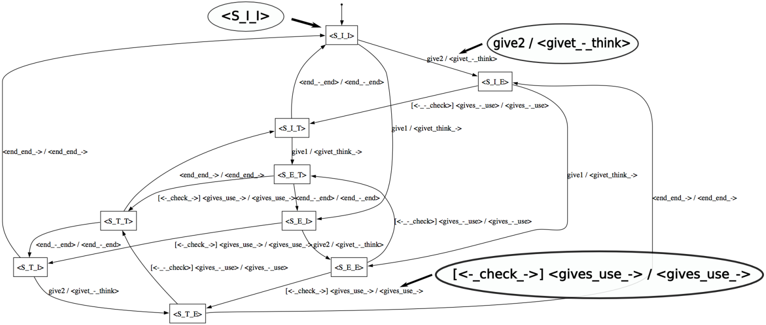
Principle Summary

- Computing a flat behaviour is not sufficient for architectures
- Based on the synchronous product of LTS
- But need to consider guards, communications and actions
- Communications are synchronisation vectors
- Guards, events and actions are structured
- But it cannot takes into account a complex hierarchy with more than one level

Principle Summary

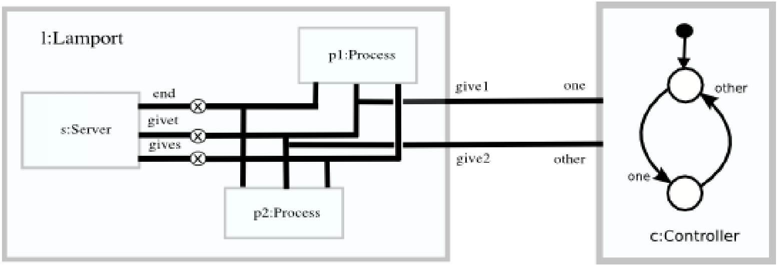
- Computing a flat behaviour is not sufficient for architectures
- Based on the synchronous product of LTS
- But need to consider guards, communications and actions
- Communications are synchronisation vectors
- Guards, events and actions are structured
- But it cannot takes into account a complex hierarchy with more than one level

Synchronous Product



Outline

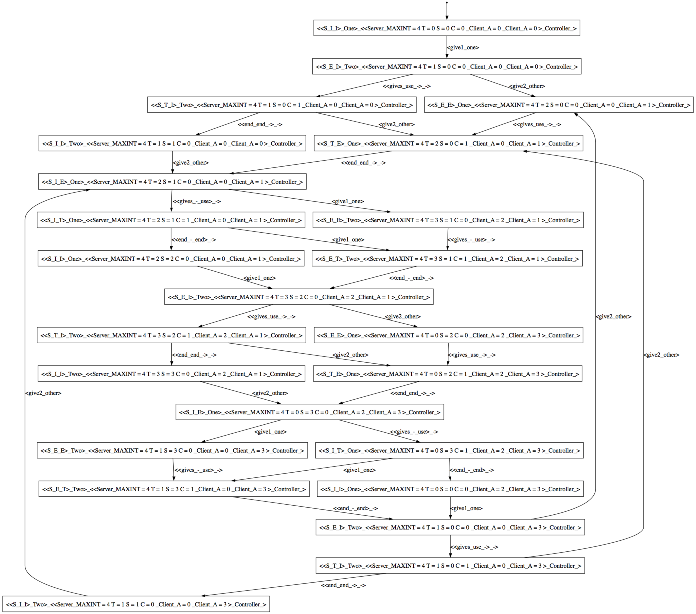
The Fairness Controller



Fairness Controller Architecture

```
NEED
            # the controller .sts
            Controller:        :
            # the inner Lamport.comp
            Inner :      Lamport   :

LOCALS      l:Inner c:Controller

COMMUNICATIONS AND c.one l.give1
COMMUNICATIONS AND c.other l.give2

# no bindings
```

The Resulting Behaviour



Our Needs

- To synchronise complex state machines with guard, communication and actions
- To allow N-party rendezvous
- To keep the structure of the composite in the result
- To hide, export or rename events
- To allow various bindings connections:
    - What to export outside ?
    - What kind of connection are allowed ?
    - What are the communication rules ?
    - ...

Our Needs

- To synchronise complex state machines with guard, communication and actions
- To allow N-party rendezvous
- To keep the structure of the composite in the result
- To hide, export or rename events
- To allow various bindings connections:
    - What to export outside ?
    - What kind of connection are allowed ?
    - What are the communication rules ?
    - ...

Our Needs

- To synchronise complex state machines with guard, communication and actions
- To allow N-party rendezvous
- To keep the structure of the composite in the result
- To hide, export or rename events
- To allow various bindings connections:
  - What to export outside ?
  - What kind of connection are allowed ?
  - What are the communication rules ?
  - ...

Our Needs

- To synchronise complex state machines with guard, communication and actions
- To allow N-party rendezvous
- To keep the structure of the composite in the result
- To hide, export or rename events
- To allow various bindings connections:
  - What to export outside ?
  - What kind of connection are allowed ?
  - What are the communication rules ?
  - ...

Our Needs

- To synchronise complex state machines with guard, communication and actions
- To allow N-party rendezvous
- To keep the structure of the composite in the result
- To hide, export or rename events
- To allow various bindings connections:
    - What to export outside ?
    - What kind of connection are allowed ?
    - What are the communication rules ?
    - ...

Our Needs

- To synchronise complex state machines with guard, communication and actions
- To allow N-party rendezvous
- To keep the structure of the composite in the result
- To hide, export or rename events
- To allow various bindings connections:
    - What to export outside ?
    - What kind of connection are allowed ?
    - What are the communication rules ?
    - ...

Our Needs

- To synchronise complex state machines with guard, communication and actions
- To allow N-party rendezvous
- To keep the structure of the composite in the result
- To hide, export or rename events
- To allow various bindings connections:
  - What to export outside ?
  - What kind of connection are allowed ?
  - What are the communication rules ?
  - ...

Our Needs

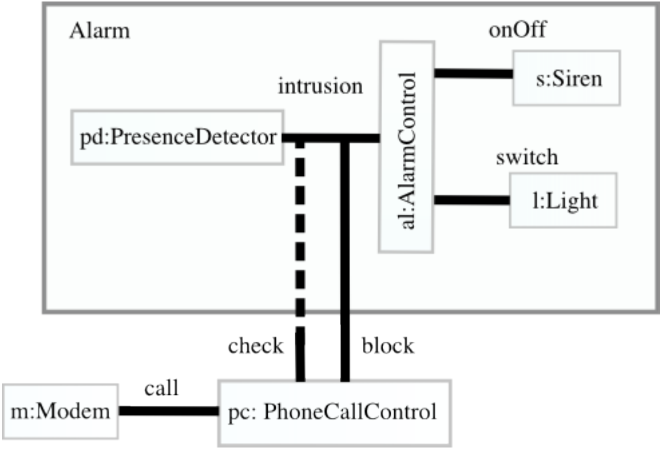
- To synchronise complex state machines with guard, communication and actions
- To allow N-party rendezvous
- To keep the structure of the composite in the result
- To hide, export or rename events
- To allow various bindings connections:
  - What to export outside ?
  - What kind of connection are allowed ?
  - What are the communication rules ?
  - ...

Our Needs

- To synchronise complex state machines with guard, communication and actions
- To allow N-party rendezvous
- To keep the structure of the composite in the result
- To hide, export or rename events
- To allow various bindings connections:
  - What to export outside ?
  - What kind of connection are allowed ?
  - What are the communication rules ?
  - ...

Outline

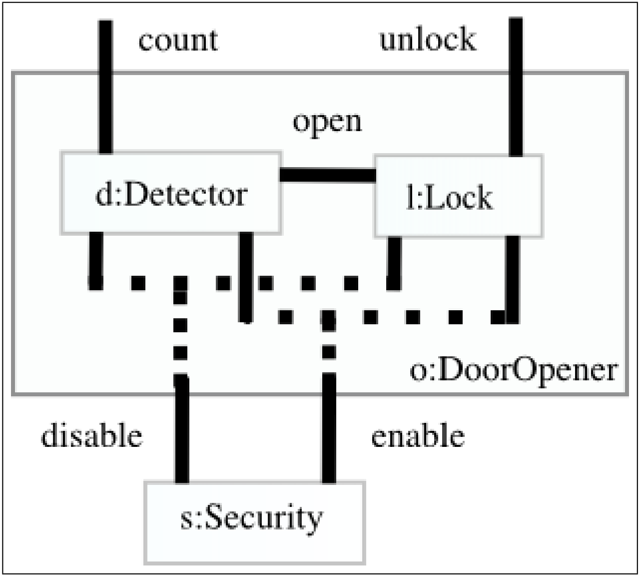
Event Interactions in Smart Home



Smart Home Architecture

```
# Internal Alarm assembly
NEED   Light:   :
       Siren:   :
       AlarmControl:   :
       PresenceDetector:  :

LOCALS   pd:PresenceDetector  l:Light
         s:Siren  a:AlarmControl

COMMUNICATIONS AND pd.intrusion  a.intrusion
COMMUNICATIONS AND a.onOff  s.onOff
COMMUNICATIONS AND a.switch  l.switch

# duplicated port
BINDINGS  pd.intrusion   -> check
# branch port
BINDINGS  pd.intrusion  a.intrusion -> block
```

Merged Port Example



DoorOpener

```
# DoorOpener definition
ROOT test fr emn stslib SH doorOpener
PACKAGE fr.emn.stslib.SH.doorOpener

NEED  Detector:  :
      Lock: :

LOCALS d: Detector l:Lock

COMMUNICATIONS AND d.open l.open

# two simple ports
BINDINGS l.unlock -> unlock
BINDINGS d.count -> count
# two merged ports
BINDINGS d.disable l.disable -> disable
BINDINGS d.enable l.enable -> enable
```

DoorOpener and Security

```
# DoorOpener + Security definition
ROOT test fr emn stslib SH doorOpener
PACKAGE fr.emn.stslib.SH.doorOpener

NEED   DoorOpener:      :
       Security:  :

LOCALS o: DoorOpener s:Security

COMMUNICATIONS AND o.enable s.off
COMMUNICATIONS AND o.disable s.on
```

Outline

Interactions Descriptions

- COMMUNICATIONS: responsible to define the internal rendezvous
- BINDINGS: defines and exports events to outside
- Explicit optional renaming:
  `BINDINGS s.gives p1.use -> entering1`
- Export an event:
  - **Simple port**: non connected port exported outside
  - **Duplicated port**: connected port exported outside
  - **Branch port**: connection point on an existing synchronisation
  - **Merged port**: new synchronisation between internal ports and export it outside
- Communication offer: ! or * if any

Interactions Descriptions

- COMMUNICATIONS: responsible to define the internal rendezvous
- BINDINGS: defines and exports events to outside
- Explicit optional renaming:
  `BINDINGS s.gives p1.use -> entering1`
- Export an event:
  - **Simple port**: non connected port exported outside
  - **Duplicated port**: connected port exported outside
  - **Branch port**: connection point on an existing synchronisation
  - **Merged port**: new synchronisation between internal ports and export it outside
- Communication offer: ! or * if any

Interactions Descriptions

- COMMUNICATIONS: responsible to define the internal rendezvous
- BINDINGS: defines and exports events to outside
- Explicit optional renaming:
  `BINDINGS s.gives p1.use -> entering1`
- Export an event:
  - **Simple port**: non connected port exported outside
  - **Duplicated port**: connected port exported outside
  - **Branch port**: connection point on an existing synchronisation
  - **Merged port**: new synchronisation between internal ports and export it outside
- Communication offer: ! or * if any

Interactions Descriptions

- COMMUNICATIONS: responsible to define the internal rendezvous
- BINDINGS: defines and exports events to outside
- Explicit optional renaming:
  `BINDINGS s.gives p1.use -> entering1`
- Export an event:
  - **Simple port**: non connected port exported outside
  - **Duplicated port**: connected port exported outside
  - **Branch port**: connection point on an existing synchronisation
  - **Merged port**: new synchronisation between internal ports and export it outside
- Communication offer: ! or * if any

Interactions Descriptions

- COMMUNICATIONS: responsible to define the internal rendezvous
- BINDINGS: defines and exports events to outside
- Explicit optional renaming:
  `BINDINGS s.gives p1.use -> entering1`
- Export an event:
  - **Simple port**: non connected port exported outside
  - **Duplicated port**: connected port exported outside
  - **Branch port**: connection point on an existing synchronisation
  - **Merged port**: new synchronisation between internal ports and export it outside
- Communication offer: ! or * if any

Interactions Descriptions

- COMMUNICATIONS: responsible to define the internal rendezvous
- BINDINGS: defines and exports events to outside
- Explicit optional renaming:
  `BINDINGS s.gives p1.use -> entering1`
- Export an event:
  - **Simple port**: non connected port exported outside
  - **Duplicated port**: connected port exported outside
  - **Branch port**: connection point on an existing synchronisation
  - **Merged port**: new synchronisation between internal ports and export it outside
- Communication offer: ! or * if any

Interactions Descriptions

- COMMUNICATIONS: responsible to define the internal rendezvous
- BINDINGS: defines and exports events to outside
- Explicit optional renaming:
  `BINDINGS s.gives p1.use -> entering1`
- Export an event:
  - **Simple port**: non connected port exported outside
  - **Duplicated port**: connected port exported outside
  - **Branch port**: connection point on an existing synchronisation
  - **Merged port**: new synchronisation between internal ports and export it outside
- Communication offer: ! or * if any

Interactions Descriptions

- COMMUNICATIONS: responsible to define the internal rendezvous
- BINDINGS: defines and exports events to outside
- Explicit optional renaming:
  `BINDINGS s.gives p1.use -> entering1`
- Export an event:
  - **Simple port**: non connected port exported outside
  - **Duplicated port**: connected port exported outside
  - **Branch port**: connection point on an existing synchronisation
  - **Merged port**: new synchronisation between internal ports and export it outside
- Communication offer: ! or * if any

Interactions Descriptions

- COMMUNICATIONS: responsible to define the internal rendezvous
- BINDINGS: defines and exports events to outside
- Explicit optional renaming:
  `BINDINGS s.gives p1.use -> entering1`
- Export an event:
  - **Simple port**: non connected port exported outside
  - **Duplicated port**: connected port exported outside
  - **Branch port**: connection point on an existing synchronisation
  - **Merged port**: new synchronisation between internal ports and export it outside
- Communication offer: ! or * if any

Structured Product

- More sophisticated than usual synchronous product of automata
- Complex transitions with guard, communication offers and actions
- Keep inside states, transitions and other elements the structure of the system
- N-party rendezvous and complex bindings, and renaming
- Global semantics for a composite: synchronous product at each level + some additional computation to manage bindings and renaming

Structured Product

- More sophisticated than usual synchronous product of automata
- Complex transitions with guard, communication offers and actions
- Keep inside states, transitions and other elements the structure of the system
- N-party rendezvous and complex bindings, and renaming
- Global semantics for a composite: synchronous product at each level + some additional computation to manage bindings and renaming

Structured Product

- More sophisticated than usual synchronous product of automata
- Complex transitions with guard, communication offers and actions
- Keep inside states, transitions and other elements the structure of the system
- N-party rendezvous and complex bindings, and renaming
- Global semantics for a composite: synchronous product at each level + some additional computation to manage bindings and renaming

Structured Product

- More sophisticated than usual synchronous product of automata
- Complex transitions with guard, communication offers and actions
- Keep inside states, transitions and other elements the structure of the system
- N-party rendezvous and complex bindings, and renaming
- Global semantics for a composite: synchronous product at each level + some additional computation to manage bindings and renaming

Structured Product

- More sophisticated than usual synchronous product of automata
- Complex transitions with guard, communication offers and actions
- Keep inside states, transitions and other elements the structure of the system
- N-party rendezvous and complex bindings, and renaming
- Global semantics for a composite: synchronous product at each level + some additional computation to manage bindings and renaming

Structured Product Algorithm

- Algorithm to compute the global semantics:
    - if it is a simple STS then return it
    - if it is a composite then
        - recursively compute the global semantics of each subcomponent,
        - update the synchronisation list to cope with duplicated and merged,
          If connected outside left bindings are added as new synchronisations
        - compute the synchronous product,
        - and rename the events according to bindings
- Need a notion of $Nary - STS$

Structured Product Algorithm

- Algorithm to compute the global semantics:
    - if it is a simple STS then return it
    - if it is a composite then
        - recursively compute the global semantics of each subcomponent,
        - update the synchronisation list to cope with duplicated and merged,
          If connected outside left bindings are added as new synchronisations
        - compute the synchronous product,
        - and rename the events according to bindings
- Need a notion of $Nary - STS$

Structured Product Algorithm

- Algorithm to compute the global semantics:
  - if it is a simple STS then return it
  - if it is a composite then
    - recursively compute the global semantics of each subcomponent,
    - update the synchronisation list to cope with duplicated and merged,
      If connected outside left bindings are added as new synchronisations
    - compute the synchronous product,
    - and rename the events according to bindings
- Need a notion of $Nary - STS$

Structured Product Algorithm

- Algorithm to compute the global semantics:
  - if it is a simple STS then return it
  - if it is a composite then
    - recursively compute the global semantics of each subcomponent,
    - update the synchronisation list to cope with duplicated and merged,
      If connected outside left bindings are added as new synchronisations
    - compute the synchronous product,
    - and rename the events according to bindings
- Need a notion of $Nary - STS$

Structured Product Algorithm

- Algorithm to compute the global semantics:
  - if it is a simple STS then return it
  - if it is a composite then
    - recursively compute the global semantics of each subcomponent,
    - update the synchronisation list to cope with duplicated and merged,
      If connected outside left bindings are added as new synchronisations
    - compute the synchronous product,
    - and rename the events according to bindings
- Need a notion of $Nary - STS$

Structured Product Algorithm

- Algorithm to compute the global semantics:
  - if it is a simple STS then return it
  - if it is a composite then
    - recursively compute the global semantics of each subcomponent,
    - update the synchronisation list to cope with duplicated and merged,
      If connected outside left bindings are added as new synchronisations
    - compute the synchronous product,
    - and rename the events according to bindings
- Need a notion of $Nary - STS$

Structured Product Algorithm

- Algorithm to compute the global semantics:
  - if it is a simple STS then return it
  - if it is a composite then
    - recursively compute the global semantics of each subcomponent,
    - update the synchronisation list to cope with duplicated and merged,
      If connected outside left bindings are added as new synchronisations
    - compute the synchronous product,
    - and rename the events according to bindings
- Need a notion of $Nary - STS$

Structured Product Algorithm

- Algorithm to compute the global semantics:
  - if it is a simple STS then return it
  - if it is a composite then
    - recursively compute the global semantics of each subcomponent,
    - update the synchronisation list to cope with duplicated and merged,
      If connected outside left bindings are added as new synchronisations
    - compute the synchronous product,
    - and rename the events according to bindings
- Need a notion of $Nary - STS$

Component Compatibility

- A component is compatible with an assembly iff the composite is deadlock free
- Build a composite with the components, compute the structured product, and check for deadlocks
- Undecidable: It covers more than simple behavioural compatibility since STS are a general model of computation
- Decidable in case of bounded system or I/O STS components
- Problem: there is neither sufficient nor necessary criterion thus the property is difficult to check on real examples with guards

Component Compatibility

- A component is compatible with an assembly iff the composite is deadlock free
- Build a composite with the components, compute the structured product, and check for deadlocks
- Undecidable: It covers more than simple behavioural compatibility since STS are a general model of computation
- Decidable in case of bounded system or I/O STS components
- Problem: there is neither sufficient nor necessary criterion thus the property is difficult to check on real examples with guards

Component Compatibility

- A component is compatible with an assembly iff the composite is deadlock free
- Build a composite with the components, compute the structured product, and check for deadlocks
- Undecidable: It covers more than simple behavioural compatibility since STS are a general model of computation
- Decidable in case of bounded system or I/O STS components
- Problem: there is neither sufficient nor necessary criterion thus the property is difficult to check on real examples with guards

Component Compatibility

- A component is compatible with an assembly iff the composite is deadlock free
- Build a composite with the components, compute the structured product, and check for deadlocks
- Undecidable: It covers more than simple behavioural compatibility since STS are a general model of computation
- Decidable in case of bounded system or I/O STS components
- Problem: there is neither sufficient nor necessary criterion thus the property is difficult to check on real examples with guards

Component Compatibility

- A component is compatible with an assembly iff the composite is deadlock free
- Build a composite with the components, compute the structured product, and check for deadlocks
- Undecidable: It covers more than simple behavioural compatibility since STS are a general model of computation
- Decidable in case of bounded system or I/O STS components
- Problem: there is neither sufficient nor necessary criterion thus the property is difficult to check on real examples with guards

Event Strictness

- With the controller example (exports entering1 and s.end p1.end -> entering2): we cannot observe synchronisations with the first process but it is still compatible !
- *Event strict* architecture: at each level, each synchronisation declared in the `COMMUNICATIONS` clauses occurs at least once in the behaviour
- Checking undecidable but decidable with bounded system or I/O STS components
- Static checking: if the structured product is not event strict then the system is not event strict
- First check event strictness then compatibility

Event Strictness

- With the controller example (exports entering1 and s.end p1.end -> entering2): we cannot observe synchronisations with the first process but it is still compatible !
- *Event strict* architecture: at each level, each synchronisation declared in the `COMMUNICATIONS` clauses occurs at least once in the behaviour
- Checking undecidable but decidable with bounded system or I/O STS components
- Static checking: if the structured product is not event strict then the system is not event strict
- First check event strictness then compatibility

Event Strictness

- With the controller example (exports entering1 and s.end p1.end -> entering2): we cannot observe synchronisations with the first process but it is still compatible !
- *Event strict* architecture: at each level, each synchronisation declared in the `COMMUNICATIONS` clauses occurs at least once in the behaviour
- Checking undecidable but decidable with bounded system or I/O STS components
- Static checking: if the structured product is not event strict then the system is not event strict
- First check event strictness then compatibility

Event Strictness

- With the controller example (exports entering1 and s.end p1.end -> entering2): we cannot observe synchronisations with the first process but it is still compatible !
- *Event strict* architecture: at each level, each synchronisation declared in the `COMMUNICATIONS` clauses occurs at least once in the behaviour
- Checking undecidable but decidable with bounded system or I/O STS components
- Static checking: if the structured product is not event strict then the system is not event strict
- First check event strictness then compatibility

Event Strictness

- With the controller example (exports entering1 and s.end p1.end -> entering2): we cannot observe synchronisations with the first process but it is still compatible !
- *Event strict* architecture: at each level, each synchronisation declared in the `COMMUNICATIONS` clauses occurs at least once in the behaviour
- Checking undecidable but decidable with bounded system or I/O STS components
- Static checking: if the structured product is not event strict then the system is not event strict
- First check event strictness then compatibility

Conclusions

- A hierarchical component model based on STS, a N-party rendezvous and sophisticated protocols with guards
- Use of N-party rendezvous: required to control component behaviour keeping a black box approach
- Tools to compute the global protocol associated to assemblies and also to analyze and check syntactic and behavioural properties
- Necessary support to analyze the communications and the compatibility of components
- Turing complete notion as STS and N-party rendezvous: control any kind of components in a truly compositional way

Conclusions

- A hierarchical component model based on STS, a N-party rendezvous and sophisticated protocols with guards
- Use of N-party rendezvous: required to control component behaviour keeping a black box approach
- Tools to compute the global protocol associated to assemblies and also to analyze and check syntactic and behavioural properties
- Necessary support to analyze the communications and the compatibility of components
- Turing complete notion as STS and N-party rendezvous: control any kind of components in a truly compositional way

Conclusions

- A hierarchical component model based on STS, a N-party rendezvous and sophisticated protocols with guards
- Use of N-party rendezvous: required to control component behaviour keeping a black box approach
- Tools to compute the global protocol associated to assemblies and also to analyze and check syntactic and behavioural properties
- Necessary support to analyze the communications and the compatibility of components
- Turing complete notion as STS and N-party rendezvous: control any kind of components in a truly compositional way

Conclusions

- A hierarchical component model based on STS, a N-party rendezvous and sophisticated protocols with guards
- Use of N-party rendezvous: required to control component behaviour keeping a black box approach
- Tools to compute the global protocol associated to assemblies and also to analyze and check syntactic and behavioural properties
- Necessary support to analyze the communications and the compatibility of components
- Turing complete notion as STS and N-party rendezvous: control any kind of components in a truly compositional way

Conclusions

- A hierarchical component model based on STS, a N-party rendezvous and sophisticated protocols with guards
- Use of N-party rendezvous: required to control component behaviour keeping a black box approach
- Tools to compute the global protocol associated to assemblies and also to analyze and check syntactic and behavioural properties
- Necessary support to analyze the communications and the compatibility of components
- Turing complete notion as STS and N-party rendezvous: control any kind of components in a truly compositional way

Future work

- Implementation of GUI to allow edition and visualization of STS and composite
- More complete set of verifications, specific attention will be on abstraction methods and bisimulation
- Enrich our set of interactions and to assist the user in the choice and the use of these interactions

Future work

- Implementation of GUI to allow edition and visualization of STS and composite
- More complete set of verifications, specific attention will be on abstraction methods and bisimulation
- Enrich our set of interactions and to assist the user in the choice and the use of these interactions

Future work

- Implementation of GUI to allow edition and visualization of STS and composite
- More complete set of verifications, specific attention will be on abstraction methods and bisimulation
- Enrich our set of interactions and to assist the user in the choice and the use of these interactions