The STSLIB Project: Towards a Formal Model Component Based on STS

Fabrício de Alexandria Fernandes
Jean-Claude Royer

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

21-09-2007 / Nice / FACS 2007

Outline

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
- Problem: explicit protocols are often dissociated from component code, not ensured that component execution will respect protocols rules
- Fill the gap between high-level formal models and implementation of protocols
- Formal analysis methods to analyze component and their interactions
- Try to ensure consistency between analysis and execution phases
- Tool support: a library with parsers and analysis tools

Related Work

- Java/A is the most relevant since it provides a formal model and a Java compiler
  - It uses protocol to describe ports: I/O Automata formalism
  - There are a compiler and a model-checker
  - Formalization with I/O transition systems and states as algebras
  - The authors prove consistency results for assemblies
- CADP as a good representative of the verification tools
  - Rich tool box set
  - Efficient
  - Model-checking by bounding variables
  - A simulator (C code and Petri nets) not a real runtime support

Related Work

- Java/A is the most relevant since it provides a formal model and a Java compiler
  - It uses protocol to describe ports: I/O Automata formalism
  - There are a compiler and a model-checker
  - Formalization with I/O transition systems and states as algebras
  - The authors prove consistency results for assemblies
- CADP as a good representative of the verification tools
  - Rich tool box set
  - Efficient
  - Model-checking by bounding variables
  - A simulator (C code and Petri nets) not a real runtime support

reprendre le papier

Related Work Continued

- Use of explicit behavioural protocols
  - PROCOL: sequences of events, data types and guards, 1-1 communication
  - SOFA: sequences of events, synchronous communications 1-1 RPC calls
  - Cooperative Objects: Petri-Net, data types and guards, synchronous communications 1-1 RPC calls
- Finite State Processes (FSP) with Java constructions: process algebra based CSP, synchronization based on rendezvous mechanism
- JCSP: provides a CSP model for the Java thread model, Java library, shared channels to synchronize processes, safer alternative than threads

The Cash-Point Example

- We consider the FM'99 cash-point service benchmark (Vol 12 of Formal Aspect of Computing Science)
- Several tills can access a central bank with a database
- A comprehensive report is available at `www.emn.fr/x-info/jroyer/cashpoint.pdf`
- Compared to other approaches: more precise management of the card, structured, readable and homogeneous semantics

The Cash-Point Example
- We consider the FM'99 cash-point service benchmark (Vol 12 of Formal Aspect of Computing Science)
- Several tills can access a central bank with a database
- A comprehensive report is available at www.emn.fr/x-info/jroyer/cashpoint.pdf
- Compared to other approaches: more precise management of the card, structured, readable and homogeneous semantics

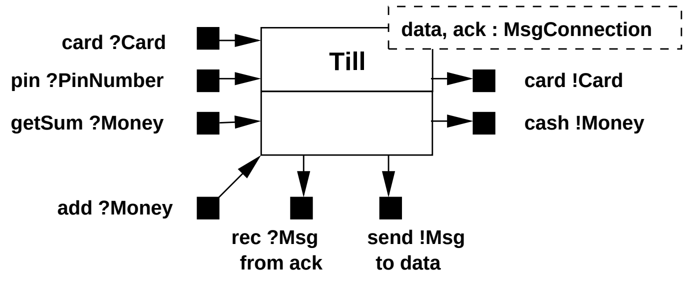
Si inutile virer ensuite

The Cash-Point Example

- We consider the system as a set of interacting components
- Indeed we describe component types (not component instance)
- Each component has an interface enriched by a dynamic behaviour notation
- This interface can be completed by an algebraic data type description

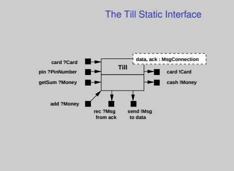
The STSLIB Project
Fabrício Fernandes, Jean-Claude Royer
Introduction
Illustrating Example
The Cash-Point Case Study
The STS Till Component
Architecture and Communications
The STSLIB Project
STSLIB Current State
Verification Based on Configuration Graph
Runtime Interpreter
Conclusions

The Till Static Interface



2007-08-31

The STSLIB Project
└ Illustrating Example
  └ The STS Till Component
    └ The Till Static Interface



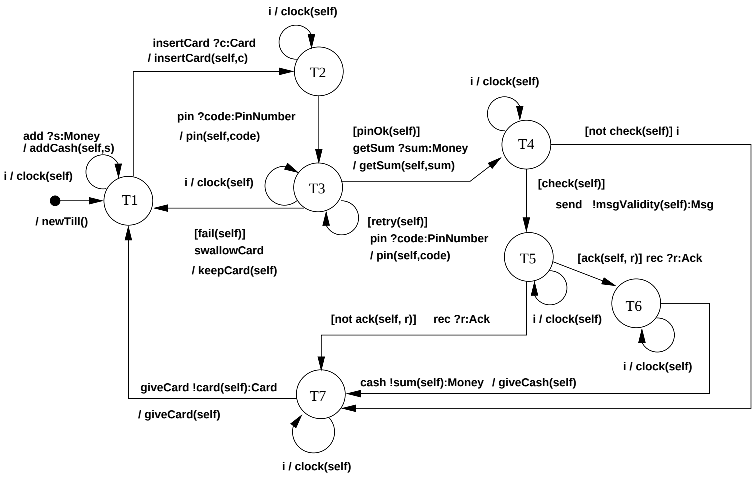
One may obviously give a static interface for this component. It may look like this with port and event name. But we consider that it is better if we have additionnaly the dynamic behaviour.

The STSLIB Project
Fabrício Fernandes, Jean-Claude Royer
Introduction
Illustrating Example
The Cash-Point Case Study
The STS Till Component
Architecture and Communications
The STSLIB Project
STSLIB Current State
Verification Based on Configuration Graph
Runtime Interpreter
Conclusions

The Dynamic Part

- An STS has two parts: the dynamic part and the data part
- This is a finite state transition machine
- Transition are labelled by
  - Guard: `[guard]` a condition to trigger the transition
  - Event name: the event
  - Communication offers:
    - `! value` emission, `value` is a function name called an emitter
    - `? var:Type` a receipt
  - Action: `/ action` the name of an operation related to the data part
- ...

The Till Dynamic Behaviour



The Till Data Type

- Describe the operation semantics (emitters, guards, actions) occurring in the dynamic part.
- We use an algebraic style with positive conditional axioms

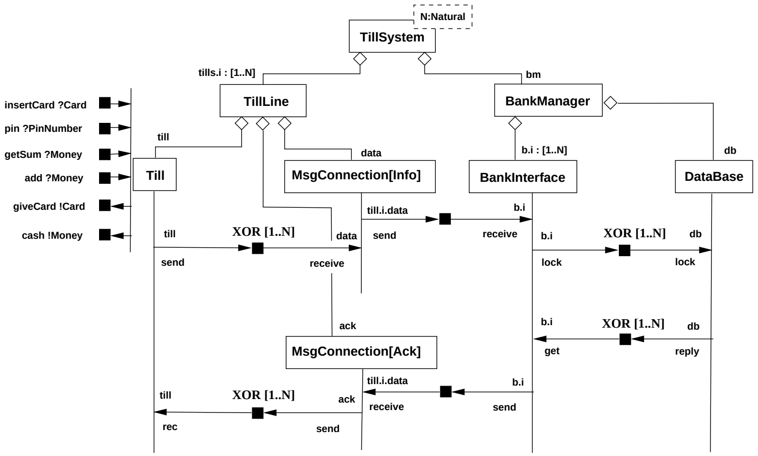
Some Axioms

```
Variables: self:Till; a,sum:Money; c:Card; code:PinNumber; ...

/* generator for till */
newTill : Money Card PinNumber Money Date Natural -> Till

/* card selector */
card : Till -> Card
card(newTill(a,c,code,sum,today,cpt)) = c

/* guard to check PIN code */
fail : Till -> Boolean
fail(self) = not equals(crypt(code(self)), code(card(self)))
              AND counter(self) >= 3

/* action to give cash and update card information */
giveCash : Till -> Till
giveCash(newTill(a,c,code,sum,today,cpt)) =
     newTill(a-sum,updateDailyLimit(card(self),sum,today),     code,sum,today
```


Communication Diagram

- We consider a variant of UML composition diagram to represent the system structure
- It is called a communication diagram since it expresses also the communication links
- Link between ports denote event synchronization with communications
- We use a range notation and some operators to define communications (but we do not detail this aspect here)
- We consider this kind of diagram better suited to our purpose than for instance component diagram since semantics are different on several points

The Cash-Point Architecture



From Notation to Tool Support

- This was rather the KADL graphic notation
- The KADL model is dedicated to analysis and verification purposes
- We want to get a concrete syntax and tool support
- Currently we simplify the communication glue which is very expressive in KADL
- We simplify some features:
  - no inheritance between STS,
  - ADT description are less general
  - +++ ?

Our Plans

- An environnement to define STS, components, architectures, ...
- Surely with GUIs (under development)
- One level devoted to verification
  - Theorem prover based: in the past we have experimented an approach using PVS
  - A new approach rather close to model-checking and based on the notion of configuration graph
- An operational level which is able to execute component description
  - Real code execution not a simulator
  - Automatic translation as possible

Defining an STS

2007-08-31

The STSLib Project
└ STSLib Current State
  └ Defining STS
    └ Defining an STS

refaire le 7 apres regeneration// premier .odp manque Till// le TillJava aussi// Data instance

Summary: Defining an STS

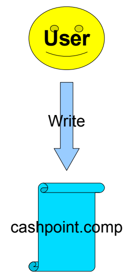
- One first implementation was done Python
- We are rewriting it in Java 1.5 under Eclipse
- The user write a `.sts` file
- A parser exists for the dynamic part
- An interface generator built an ADT skeleton `.adt`
- User has to fill the axioms
- A Java translator generate a Java class from the ADT (experimental)
- But the user may also write a Java class or reuse an existing one
- One STS is not sufficient enough for our example ...

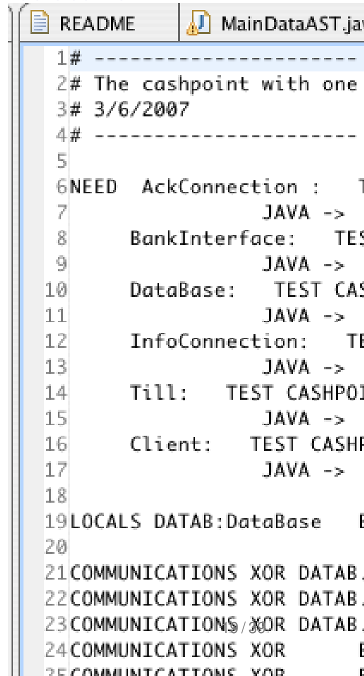
2007-08-31

The STSLib Project
└ STSLib Current State
  └ Defining STS
    └ Summary: Defining an STS

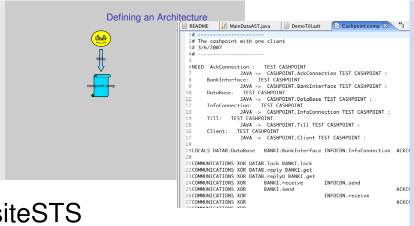
avoir qq elements pour dire ca va mieux avec Java...
Y'a quand meme contrat minimal sur la Java class
Revoir les parsers ca marche ?

Defining an Architecture






```
# ----------------------
# The cashpoint with one
# 3/6/2007
# ----------------------

NEED  AckConnection :   T
                JAVA ->
      BankInterface:   TES
                JAVA ->
      DataBase:   TEST CAS
                JAVA ->
      InfoConnection:   TE
                JAVA ->
      Till:   TEST CASHPOI
                JAVA ->
      Client:   TEST CASHP
                JAVA ->

LOCALS DATAB:DataBase   E

COMMUNICATIONS XOR DATAB.
COMMUNICATIONS XOR DATAB.
COMMUNICATIONS XOR DATAB.
COMMUNICATIONS XOR      E
COMMUNICATIONS XOR
```



Dans JAVA -> a quoi sert repertoire ?// verifier CompositeSTS instance

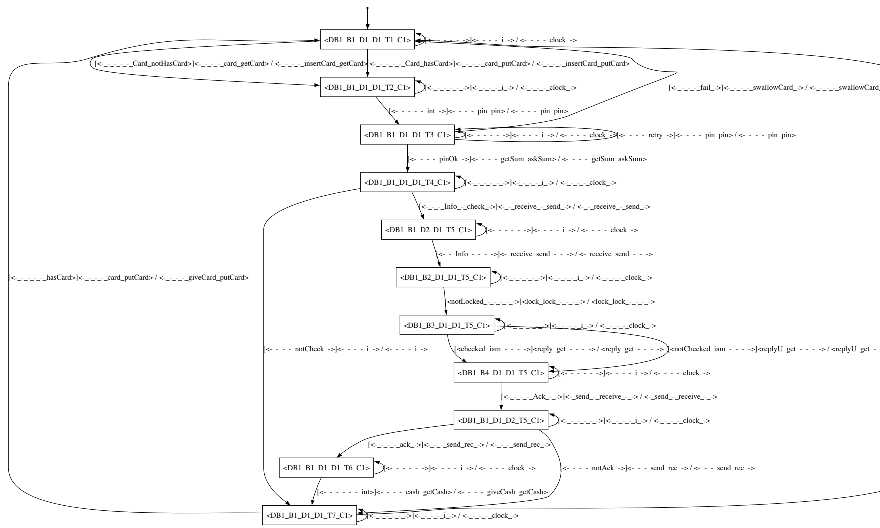
Summary: Defining an Architecture

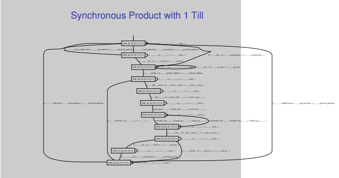
- A grammar allow to express component types, local instances, bindings and exports
- This is a bit more complex since the parser have to be recursive
- It also needs to call the STS loader
- When loading an architecture we have also to specify where are the Java classes
- The instantiation process is also a bit more complex
- Not completly stable since some choices about bindigns are pending

Formalizations

- Formal definition of STS, configuration graph and synchronous product are given in the paper
- Currently our verification process is:
  - Compute the synchronous product and check it
  - Compute the configuration graph and prove some properties
- In FMOODS 2006 we present a way to decompose a system to get an abstraction of the configuration graph allowing to prove some safety properties
- We also prove some properties of the synchronous product and decomposition principle
- What we want to discuss is our verification approach which is a bit original

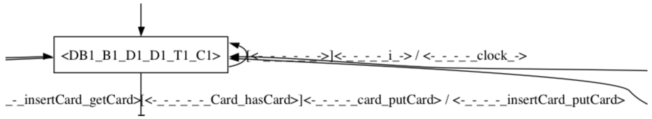
Synchronous Product with 1 Till



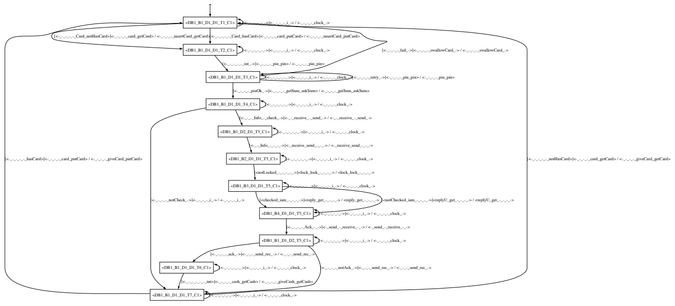


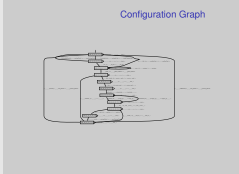
zoomer sur une transition ... Note that our state machines keeps inside states and transition labels the structure of the system. Obvioulsy if it is not useful an abstraction can easily remove it. TODO refaire la vue ici

Structuration



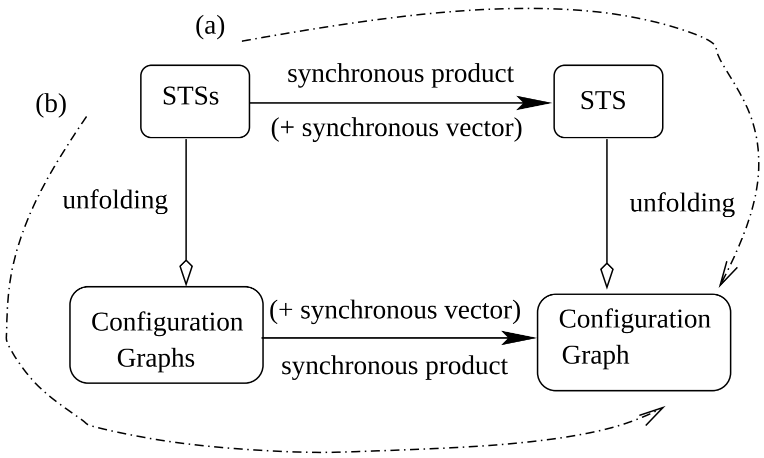
Configuration Graph

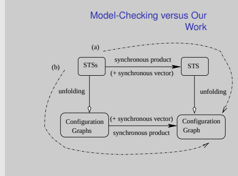




Soit bricole soit montrer celui en Python plutot !!!

Model-Checking versus Our Work



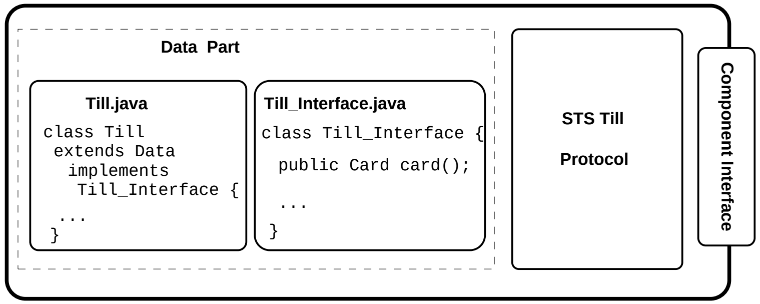


pas si clair la diff avec on the fly

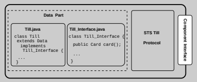
Some Results

- On the cashpoint we are able to built it up to 4 Tills
- With the currrent requirements we check that `swallowCard` leads to a livelock
- PIN counter = 3 after a `swallowCard`
- The database and the till amount are ≥ 0
- Eclusive access to any bank account
- To check that the card is owned by the proper client or by its connected till or lost
- We check various, mainly safety properties
- Not really efficient but we argue that the approach can be useful

STS Implementation



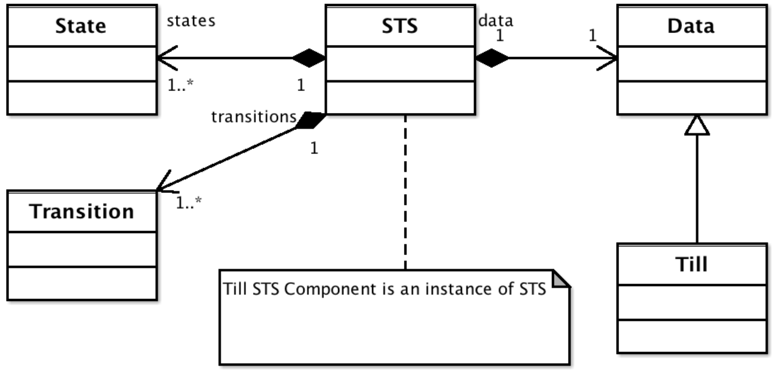
STS Implementation



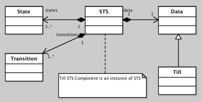
y a rules mais bof

Fabrício Fernandes, Jean-Claude Royer

Class Diagram



Class Diagram



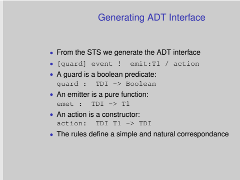
The STSLIB Project

Fabrício Fernandes, Jean-Claude Royer

Introduction
Illustrating Example
STSLIB Current State
Verification Based on Configuration Graph
Runtime Interpreter
Implementation Overview
Translating Data Types
Rendezvous Mechanism
Conclusions

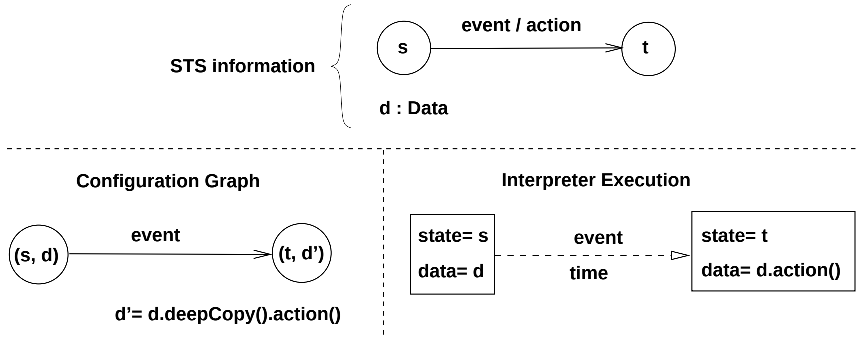
Generating ADT Interface

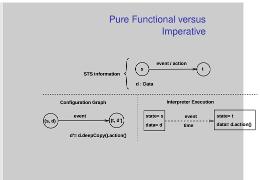
- From the STS we generate the ADT interface
- `[guard] event ! emit:T1 / action`
- A guard is a boolean predicate:
  `guard : TDI -> Boolean`
- An emitter is a pure function:
  `emet : TDI -> T1`
- An action is a constructor:
  `action: TDI T1 -> TDI`
- The rules define a simple and natural correspondance



transition la pas clair

Pure Functional versus Imperative





idem ca

ADT Hypotheses

- One generator with selectors
- Axiom with conditions
- Functional form for the left-conclusion term
- Terminating and confluent system ?
- Prefix grammar without overloading
- `exemple`

ADT Hypotheses

- One generator with selectors
- Axiom with conditions
- Functional form for the left-conclusion term
- Terminating and confluent system ?
- Prefix grammar without overloading
- `exemple`

parler des regles pour l'interface?+emitter+guard+action depuis le STS

Translation Mechanism

- A LL(2) grammar and parser have been defined using ANTLR
- An AST builder
- +++

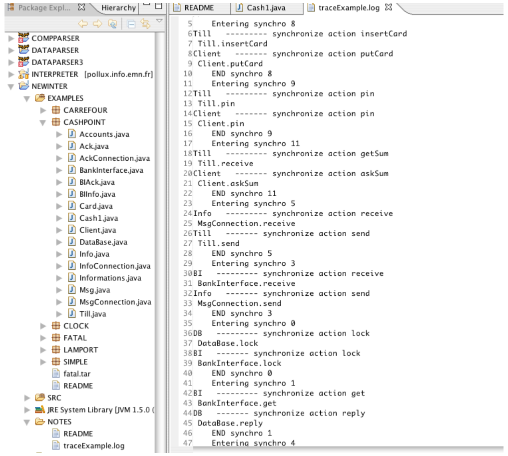
Rendezvous Mechanism

- Presented and published at CPA ...
- `STS` is an active class owning a Java thread
- The technical part is to define a synchronization area for several participant
- We use a two barriers implemented using the Java monitor facility
- Delicate things are:
  - The management of guards
  - To ...

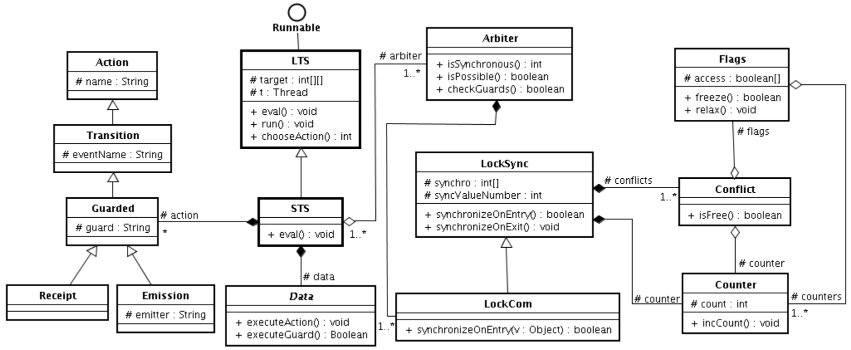
Rendezvous Mechanism
- Presented and published at CPA ...
- STS is an active class owning a Java thread
- The technical part is to define a synchronization area for several participant
- We use a two barriers implemented using the Java monitor facility
- Delicate things are:
  - The management of guards
  - To ...

to be short : montrer execution ?

An Execution Trace



Class Diagram for Runtime Support



Conclusions

- Provides an operational interpreter to program primitive components in Java with STS and a powerful way to compose them
- Protocols as Symbolic Transition Systems with full data types, guards and communications
- Relating verification and execution of component systems
- Tools for the parsing and generation of +++

Future Work

- Definition of a component programming language with STS, asynchronous and synchronous communications
- Current version: reflexivity used to glue protocols and data parts. Compiler version: direct call to the data parts methods
- Define a true compiler support
- Implement classic abstraction mechanisms and +++
- Prove the correctness of the solution for the rendezvous

Questions?

- Questions?

The STSLIB Project

Fabrício Fernandes, Jean-Claude Royer

Introduction

Illustrating Example

STSLIB Current State

Verification Based on Configuration Graph

Runtime Interpreter

Conclusions

The STSLIB Project: Towards a Formal Model Component Based on STS

Fabrício de Alexandria Fernandes
Jean-Claude Royer

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

21-09-2007 / Nice / FACS 2007