Mohammad Mahdi Jaghoori
Centre for Mathematics and Computer Science
The Netherlands

Harnessing Tubs for Traveling Soundly in Seas
(TTSS’09)
Credo Methodology

- **Credo**
  - Highly reconfigurable systems
  - Top-down design
  - Compositional analysis

- **Methodology**
  - How to use the techniques
  - When to use the tools

- **A peer-to-peer system**
  - Nodes sharing files
  - Broker dispatching requests
Modeling Levels

Reo
- Interaction Analysis
- Deadlock Analysis
- Dataflow modeling

Conformance checking

Creol
- Functional Analysis
- Schedulability Analysis
- Object-oriented modeling

Conformance checking

C, Java, ...

Programming
Credo Tools & Analysis

Methodology

Introduction
A P2P System
Data-Flow level
OO Level

Conclusion

Testing Tool Suite
conformance checking

ECT
interaction analysis
deadlock analysis

network

component structural interface

component behavioral interface

object

UPPAAL
schedulability analysis

Creol Platform
functional analysis

interaction analysis
deadlock analysis

Testing Tool Suite
conformance checking
Modeling & Analyzing A Peer-to-Peer System in Credo
Nodes in the Peer-to-Peer System

- N1
- N2
- N3

- sReq
- cReq
- sAns
- cAns

Introduction
A P2P System
Data-Flow level
OO Level
Conclusion

Mahdi Jaghoori
Nodes in the Peer-to-Peer System

facade ClientSide begin
  port cReq : output
  port cAns : import
  sync_event openCS<req : output, ans : import>
    (in k : Data; out f : Bool)
  sync_event closeCS<req : output, ans : import>
end

openCS

closeCS

cReq

cAns
Nodes in the Peer-to-Peer System

\begin{center}
\begin{tikzpicture}
\node[draw, rounded rectangle] (N1) at (0,0) {\textbf{N1}};
\node[draw, rounded rectangle] (N2) at (2,0) {\textbf{N2}};
\node[draw, rounded rectangle] (N3) at (4,0) {\textbf{N3}};

\draw[->] (N1) -- (N2);
\draw[->] (N1) -- (N3);
\draw[->] (N2) -- (N3);
\end{tikzpicture}
\end{center}

\begin{verbatim}
facade ServerSide begin
  port sReq : import
  port sAns : output
  sync_event openSS<req:import, ans:output>()
  sync_event closeSS<req:import, ans:output>()
  register <>(in keyList : List [Data])
end
\end{verbatim}
Nodes in the Peer-to-Peer System

facade Peer inherits ClientSide, ServerSide begin
  update <> (in keyList : List[Data])
end
Nodes in the Peer-to-Peer System

- **Client**
  - openCS
  - closeCS
  - cAns
  - cReq

- **Peer**
  - register
  - cAns
  - update

- **Server**
  - openSS
  - closeSS
  - sReq
  - sAns
Modeling the Network

Connecting *Client*$_i$ and *Server*$_j$
Modeling the Network

Connecting \textit{Client}_i and \textit{Server}_j

\begin{tikzpicture}
  \node (s) at (0,0) [draw, circle] {$s$};
  \node (i) at (2,0) [draw, circle] {$i,j$};
  \node (e) at (0,-2) [draw, circle] {$e$};
  \node (f) at (2,-2) [draw, circle] {$f$};

  \draw[->] (s) -- (i); \node [midway, above] at (1,0) {$c\text{Req}_i, s\text{Req}_j$};
  \draw[->] (i) -- (s); \node [midway, below] at (1,0) {$c\text{Ans}_i, s\text{Ans}_j$};
  \draw[->] (e) -- (f); \node [midway, above] at (1,-2) {$\text{in}$};
  \draw[->] (f) -- (e); \node [midway, below] at (1,-2) {$\text{out}$};

  \draw[->] (s) -- (e); \node [midway, left] at (0,-1) {$\text{openCS}_i, \text{openSS}_j$};
  \draw[->] (f) -- (i); \node [midway, right] at (2,-1) {$\text{closeCS}_i, \text{closeSS}_j$};
\end{tikzpicture}
Modeling the Network

Connecting *Client*$_i$ and *Server*$_j$

Diagram showing the connection between a client and a server, with various states and transitions labeled with symbols such as `openCS`, `openSS`, `closeCS`, `closeSS`, `cReq`, `sReq`, `cAns`, `sAns`, `in`, and `out`. The diagram illustrates the flow of communication and state transitions in a P2P system.
Reo Channels in Modeling the Network

Methodology

A P2P System

Data-Flow level

OO Level

Conclusion

Mahdi Jaghoori
Creol Modeling Language

A language for Concurrent Objects:

- Dedicated **processors** (for modeling distributed systems)
- Asynchronous message passing
- A process is created for handling each incoming message
  - There is a method corresponding to each message
- Voluntary processor release points
  - can be conditional
- automatic reply signal upon termination of methods
  - The caller can wait for the reply
Creol Modeling Language

A language for Concurrent Objects:

- Dedicated processors (for modeling distributed systems)
- **Asynchronous** message passing
  - A process is created for handling each incoming message
    - There is a method corresponding to each message
  - Voluntary processor release points
    - can be conditional
  - automatic reply signal upon termination of methods
    - The caller can wait for the reply
Creol Modeling Language

A language for Concurrent Objects:

- Dedicated processors (for modeling distributed systems)
- Asynchronous message passing
- A process is created for handling each incoming message
  - There is a method corresponding to each message
  - Voluntary processor release points
    - can be conditional
  - automatic reply signal upon termination of methods
    - The caller can wait for the reply
Creol Modeling Language

A language for Concurrent Objects:
- Dedicated processors (for modeling distributed systems)
- Asynchronous message passing
- A process is created for handling each incoming message
  - There is a method corresponding to each message
- Voluntary processor release points
  - can be conditional
- automatic reply signal upon termination of methods
  - The caller can wait for the reply
Creol Modeling Language

A language for Concurrent Objects:

- Dedicated processors (for modeling distributed systems)
- Asynchronous message passing
- A process is created for handling each incoming message
  - There is a method corresponding to each message
- Voluntary processor release points
  - Can be conditional
- Automatic reply signal upon termination of methods
  - The caller can wait for the reply
Creol Modeling Language

A language for Concurrent Objects:

- Dedicated processors (for modeling distributed systems)
- Asynchronous message passing
- A process is created for handling each incoming message
  - There is a method corresponding to each message
- Voluntary processor release points
  - can be conditional
- automatic reply signal upon termination of methods
  - The caller can wait for the reply
Creol Modeling Language

A language for Concurrent Objects:

- Dedicated processors (for modeling distributed systems)
- Asynchronous message passing
- A process is created for handling each incoming message
  - There is a method corresponding to each message
- Voluntary processor release points
  - can be conditional
- automatic reply signal upon termination of methods
  - The caller can wait for the reply
Creol Modeling Language

A language for Concurrent Objects:

- Dedicated processors (for modeling distributed systems)
- Asynchronous message passing
- A process is created for handling each incoming message
  - There is a method corresponding to each message
- Voluntary processor release points
  - can be conditional
- automatic reply signal upon termination of methods
  - The caller can wait for the reply
Eclipse Environment for Creol

Methodology

Introduction

A P2P System

Data-Flow level

Data-Flow level

OO Level

Conclusion

Mahdi Jaghoori
Object Interfaces

```plaintext
interface Client(s:StoreClientPerspective, r:outport, a:inport) begin end

interface StoreClientPerspective begin
  with Client op add(in key:Data, info:Data)
end

interface Server(s:StoreClientPerspective, r:inport, a:outport) begin end

interface StoreServerPerspective begin
  with Server op find(in key:Data; out info:Data)
end

interface Store
  inherits StoreClientPerspective, StoreServerPerspective begin end
```
Object Interfaces

interface Client (s: StoreClientPerspective, r: outport, a: inport)
begin end

interface StoreClientPerspective begin
  with Client op add (in key: Data, info: Data)
end

interface Server (s: StoreClientPerspective, r: inport, a: outport)
begin end

interface StoreServerPerspective begin
  with Server op find (in key: Data; out info: Data)
end

interface Store
  inherits StoreClientPerspective, StoreServerPerspective
begin end
Object Interfaces

```plaintext
interface Client(s: StoreClientPerspective, r: outport, a: inport)
begin end

interface StoreClientPerspective begin
  with Client op add(in key: Data, info: Data)
end

interface Server(s: StoreClientPerspective, r: inport, a: outport)
begin end

interface StoreServerPerspective begin
  with Server op find(in key: Data; out info: Data)
end

interface Store
  inherits StoreClientPerspective, StoreServerPerspective
begin end
```
Server Class Implementation

class ServerImp ( store : StoreServerPerspective ,
  req : inport , ans : outport )

  inside Peer
  implements Server

begin
  op run ==
    var key , result : Data ;
    raise_event openSS < req , ans > ();
    req . take ( ; key );
    store . find ( key ; result );
    ans . write ( result );
    raise_event closeSS < req , ans > ();
    ! run ()

end
Executing Creol in Eclipse
Conformance Testing - Creol Code

What to test?
- one object in isolation

How to test?
- give a specification for the environment and the expected behavior
- the specification and the object under test are executed in Maude
- the specification drives the execution

What is the result?
- ok
- or a Maude execution trace that reveals unspecified behavior of the object under test
Creol Testing

The (simplified) specification of a server:

\[ \phi_S = \langle \text{event register(keyList)} \rangle ? \ . \ 
\text{rec } X . \ 
\langle \text{event openSS()} \rangle ? \ . \ 
\langle \text{port s.sReq(key)} \rangle ! \ . \ 
\langle \text{port s.sAns(data[key])} \rangle ? \ . \ 
\langle \text{event closeSS()} \rangle ? \ . X \]

- use variables to relate caller and callee
- use variables to relate input and output
- use “don’t-care” variables for underspecified behavior
Schedulability Analysis in UPPAAL

- **Schedulability**: All tasks finish in time
  - Modularly: Each object individually w.r.t its behavioral interface
  - Behavioral interface
    - Message arrival
    - Deadlines: Schedulability requirement
  - Scheduler: Order of executing messages

**Bounded Queue Theorem**
A schedulable object needs a queue length of at most \( \lceil d_{\text{max}} / b_{\text{min}} \rceil \)
Schedulability Analysis in UPPAAL

- Schedulability: All tasks finish in time
- Modularly: Each object individually w.r.t its behavioral interface

- Behavioral interface
  - Message arrival
  - Deadlines: Schedulability requirement
- Scheduler: Order of executing messages

Bounded Queue Theorem

A schedulable object needs a queue length of at most \( \left\lceil \frac{d_{\text{max}}}{b_{\text{min}}} \right\rceil \)
Schedulability Analysis in UPPAAL

- Schedulability: All tasks finish in time
- Modularly: Each object individually w.r.t its behavioral interface

**Behavioral interface**
- Message arrival
- Deadlines: Schedulability requirement
- Scheduler: Order of executing messages

**Bounded Queue Theorem**
A schedulable object needs a queue length of at most 
\[ \left\lceil \frac{d_{\text{max}}}{b_{\text{min}}} \right\rceil \]
Schedulability Analysis in UPPAAL

- Schedulability: All tasks finish in time
- Modularly: Each object individually w.r.t its behavioral interface

Behavioral interface
- Message arrival
- Deadlines: Schedulability requirement
- Scheduler: Order of executing messages

Bounded Queue Theorem
A schedulable object needs a queue length of at most \( \lceil \frac{d_{\text{max}}}{b_{\text{min}}} \rceil \)
Schedulability Analysis in UPPAAL

- Schedulability: All tasks finish in time
- Modularly: Each object individually w.r.t its behavioral interface
- Behavioral interface
  - Message arrival
  - Deadlines: Schedulability requirement
- Scheduler: Order of executing messages

Bounded Queue Theorem

A schedulable object needs a queue length of at most \( \lceil \frac{d_{max}}{b_{min}} \rceil \)
Schedulability Analysis in UPPAAL

- Schedulability: All tasks finish in time
- Modularly: Each object individually w.r.t its behavioral interface
  - Behavioral interface
    - Message arrival
    - Deadlines: Schedulability requirement
- Scheduler: Order of executing messages

Bounded Queue Theorem

A schedulable object needs a queue length of at most \( \lceil \frac{d_{\text{max}}}{b_{\text{min}}} \rceil \)
Schedulability Analysis in UPPAAL

- Schedulability: All tasks finish in time
- Modularly: Each object individually w.r.t its behavioral interface

Behavioral interface
- Message arrival
- Deadlines: Schedulability requirement
- Scheduler: Order of executing messages

**Bounded Queue Theorem**

A schedulable object needs a queue length of at most 
\[ \lceil d_{max} / b_{min} \rceil \]
Conclusions and Future Work

- **Data-flow level**
  - Modeling interaction
  - Deadlock analysis

- **Object-oriented level**
  - Modeling functionality and behavior
  - Functional analysis: simulation, assertion checking
  - Nonfunctional analysis: schedulability

- Testing conformance between the levels, based on behavioral interfaces
Conclusions and Future Work

- Data-flow level
  - Modeling interaction
  - Deadlock analysis
- Object-oriented level
  - Modeling functionality and behavior
  - Functional analysis: simulation, assertion checking
  - Nonfunctional analysis: schedulability
- Testing conformance between the levels, based on behavioral interfaces
Conclusions and Future Work

- **Data-flow level**
  - Modeling interaction
  - Deadlock analysis

- **Object-oriented level**
  - Modeling functionality and behavior
  - Functional analysis: simulation, assertion checking
  - Nonfunctional analysis: schedulability

- Testing conformance between the levels, based on behavioral interfaces
Conclusions and Future Work

- Data-flow level
  - Modeling interaction
  - Deadlock analysis

- Object-oriented level
  - Modeling functionality and behavior
  - Functional analysis: simulation, assertion checking
  - Nonfunctional analysis: schedulability

- Testing conformance between the levels, based on behavioral interfaces
Conclusions and Future Work

- **Data-flow level**
  - Modeling interaction
  - Deadlock analysis

- **Object-oriented level**
  - Modeling functionality and behavior
  - Functional analysis: simulation, assertion checking
  - Nonfunctional analysis: schedulability

- Testing conformance between the levels, based on behavioral interfaces
Conclusions and Future Work

- Data-flow level
  - Modeling interaction
  - Deadlock analysis

- Object-oriented level
  - Modeling functionality and behavior
  - Functional analysis: simulation, assertion checking
  - Nonfunctional analysis: schedulability

- Testing conformance between the levels, based on behavioral interfaces
Conclusions and Future Work

- Data-flow level
  - Modeling interaction
  - Deadlock analysis

- Object-oriented level
  - Modeling functionality and behavior
  - Functional analysis: simulation, assertion checking
  - Nonfunctional analysis: schedulability

- Testing conformance between the levels, based on behavioral interfaces
Conclusions and Future Work

- Data-flow level
  - Modeling interaction
  - Deadlock analysis

- Object-oriented level
  - Modeling functionality and behavior
  - Functional analysis: simulation, assertion checking
  - Nonfunctional analysis: schedulability

- Testing conformance between the levels, based on behavioral interfaces
Conclusions and Future Work

- **Data-flow level**
  - Modeling interaction
  - Deadlock analysis

- **Object-oriented level**
  - Modeling functionality and behavior
  - Functional analysis: simulation, assertion checking
  - Nonfunctional analysis: schedulability

- Testing conformance between the levels, based on behavioral interfaces

- Covering the full development life-cycle
  - Software architecture modeling
  - Deployment concerns, e.g., schedulers
Conclusions and Future Work

- **Data-flow level**
  - Modeling interaction
  - Deadlock analysis

- **Object-oriented level**
  - Modeling functionality and behavior
  - Functional analysis: simulation, assertion checking
  - Nonfunctional analysis: schedulability

- Testing conformance between the levels, based on behavioral interfaces

- Covering the full development life-cycle
  - Software architecture modeling
  - Deployment concerns, e.g., schedulers
Conclusions and Future Work

- **Data-flow level**
  - Modeling interaction
  - Deadlock analysis

- **Object-oriented level**
  - Modeling functionality and behavior
  - Functional analysis: simulation, assertion checking
  - Nonfunctional analysis: schedulability

- Testing conformance between the levels, based on behavioral interfaces

- Covering the full development life-cycle
  - Software architecture modeling
  - Deployment concerns, e.g., schedulers
Conclusions and Future Work

- Data-flow level
  - Modeling interaction
  - Deadlock analysis

- Object-oriented level
  - Modeling functionality and behavior
  - Functional analysis: simulation, assertion checking
  - Nonfunctional analysis: schedulability

- Testing conformance between the levels, based on behavioral interfaces

- Covering the full development life-cycle
  - Software architecture modeling
  - Deployment concerns, e.g., schedulers
Questions

問題
پرسش؟
Fragen?
Vragen?
Questions?
Comments?
Suggestions?

email: jaghouri@cwi.nl