Generic Interacting State Machines and their instantiation with dynamic features

David von Oheimb and Volkmar Lotz
Siemens, Corporate Technology, D-81730 Munich
{David.von.Oheimb|Volkmar.Lotz}@siemens.com

Introduction

Aims:

Formal security modeling and verification
for requirements analysis and evaluation/certification

Applications:

- Abstractions of practical IT systems, e.g.
  smartcards, protocols, databases, mobile agents, ...
- Systems involve both state and interaction,
  with dynamic structure and behavior

Tools:

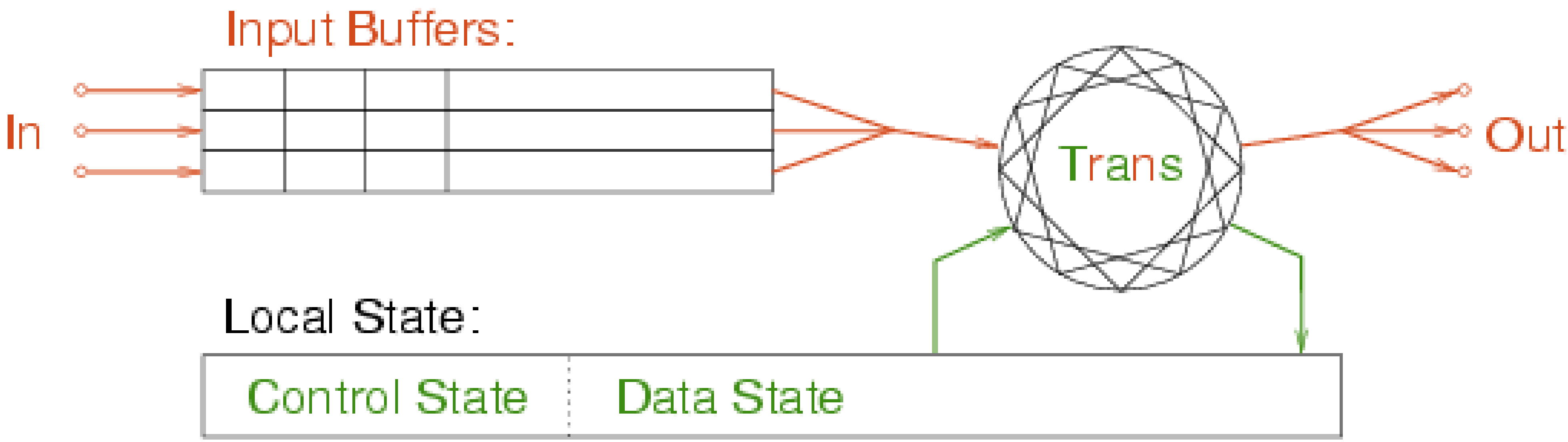
Some suitable specification language and proof assistant

Overview

- Introduction
- ISMs
  - Concepts, Semantics, Tools
- Generic ISMs
  - Concepts, Semantics, Instantiations
- Example: Client/Server
  - representation in AutoFocus, Isabelle
- Conclusion

Interacting State Machines (ISMs)

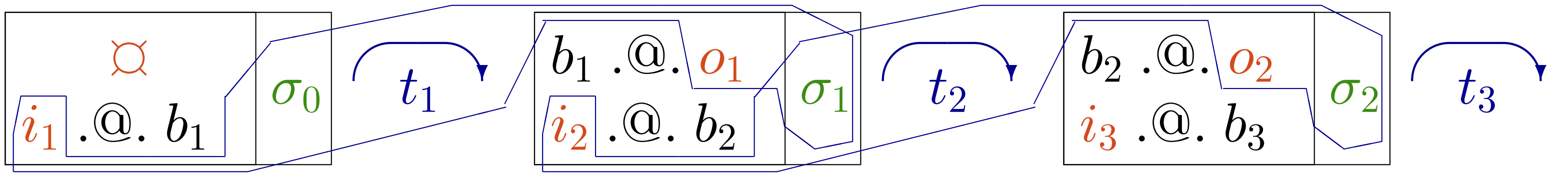
Automata with (nondeterministic) state transitions +
buffered i/o simultaneously on multiple connections



Transitions in executable and/or property-oriented style

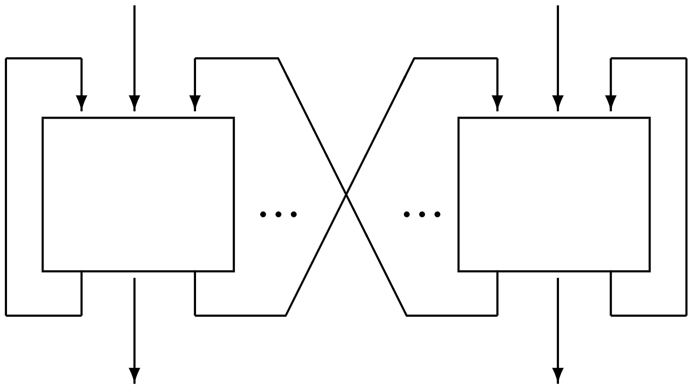
Trace Semantics

Runs by repeated application of transitions, with input from environment



Finite executions only ($\leadsto$ no general liveness properties)

Parallel composition by interleaving transitions on product state



With internal communication, feedback, and multicast

Tool Support

AutoFocus: graphical specification (and simulation)

- Syntactic perspective
- Graphical documentation
- Type and consistency checks

Isabelle/HOL: powerful interactive theorem prover

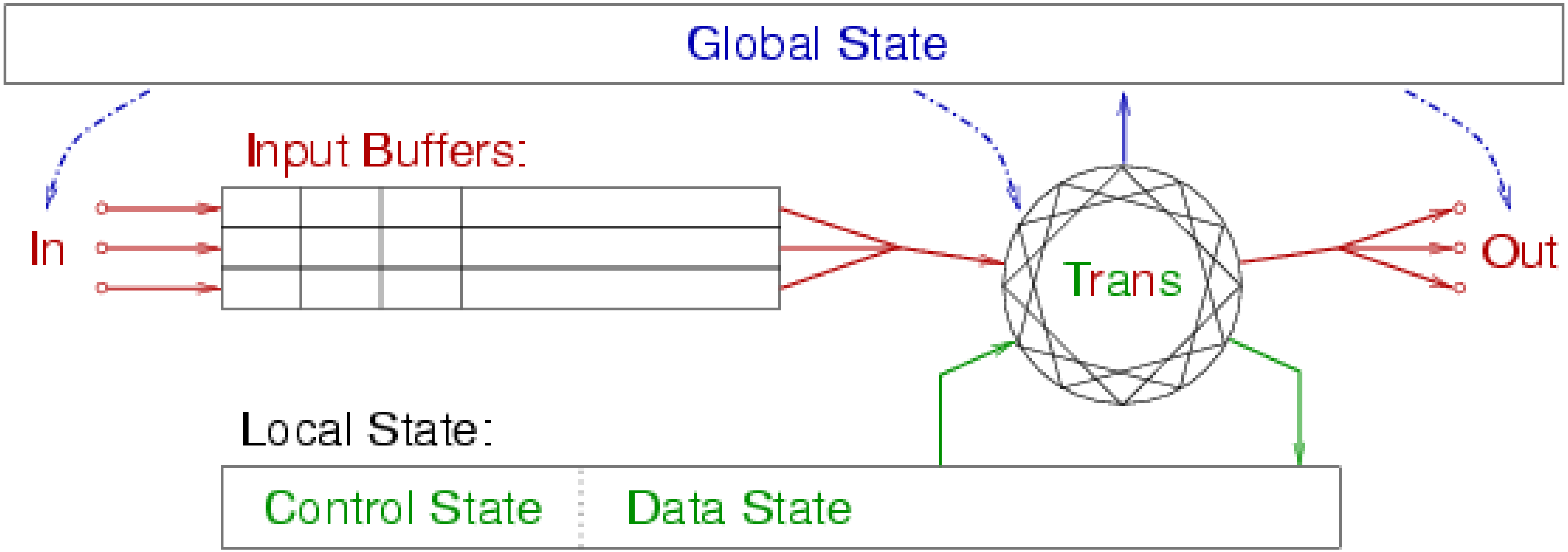
- Semantic perspective
- Textual documentation
- Validation and correctness proofs

AutoFocus drawing $\longrightarrow$ *Quest* file $\xrightarrow{\text{Conv}_1}$ *Isabelle theory* file

Within Isabelle: ***ism*** *sections* $\xrightarrow{\text{Conv}_2}$ Standard *HOL definitions*

Generic ISM Concepts

Idea: make ISM system depend on global state



Global state influenced by user commands,
but not directly visible in user transitions

Semantics

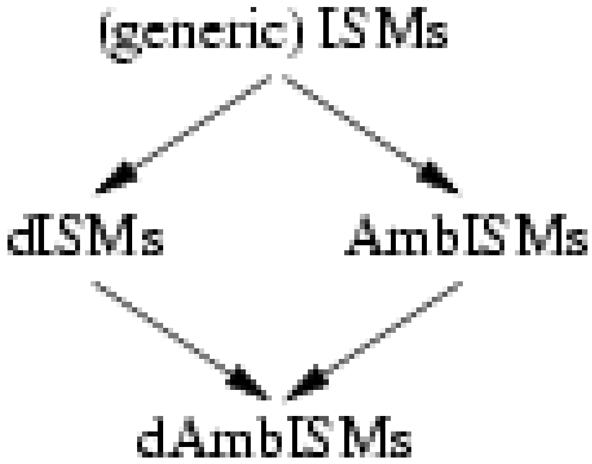
Composite runs of generic ISM family $As(\gamma) = (As(\gamma)_j)_{j \in I(\gamma)}$:

$$\frac{}{\langle(\maltese, (\gamma_0, S_0(As(\gamma_0))))\rangle \in CRuns(As, \gamma_0, gtrans)}$$

$$\frac{\begin{array}{c} j \in I(\gamma) \\ cs \frown (i \;.@.\; b, (\gamma, S[j := \sigma])) \in CRuns(As, \gamma_0, gtrans) \\ ((i, \sigma), c, (o, \sigma')) \in Trans(As(\gamma)_j) \\ mdom(i) \subseteq In(As(\gamma)_j) \cap AllOut(As(\gamma)) \\ mdom(o) \subseteq Out(As(\gamma)_j) \cap AllIn(As(\gamma)) \\ (\gamma, c, \gamma') \in gtrans(j) \end{array}}{\begin{array}{l} cs \frown (i \;.@.\; b, (\gamma\;, S[j := \sigma\;])) \\ \quad \frown (b \;.@.\; o, (\gamma', S[j := \sigma'])) \in CRuns(As, \gamma_0, gtrans) \end{array}}$$

Instantiations

ISM hierarchy:

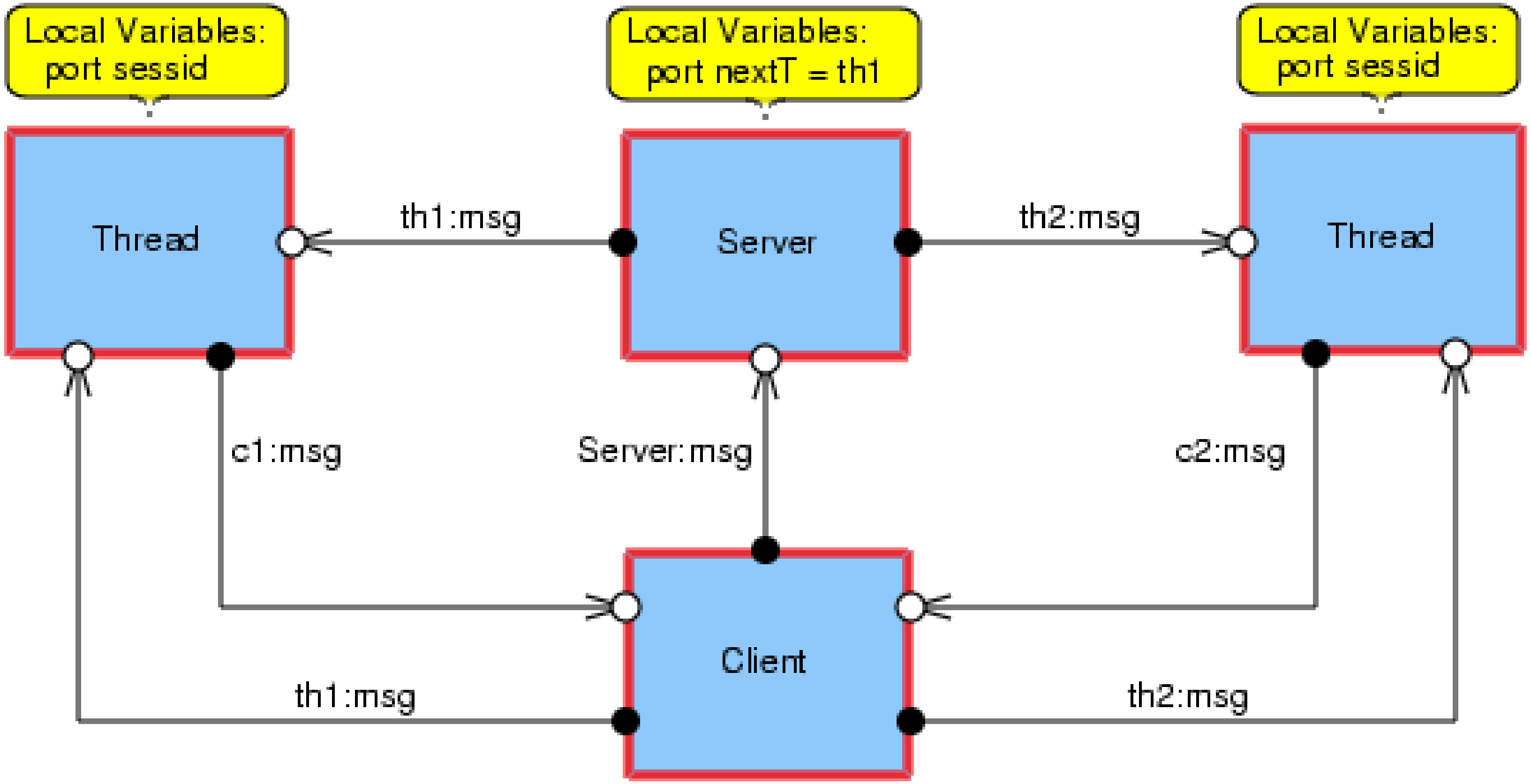


dISMs commands: *Run i*, *Stop i*, *New p*, *Convey p i*, *Enable p*, *Disable p*

AmbISMs: accompanying paper by Thomas Kuhn and David von Oheimb
Interacting State Machines for Mobility. FM 2003.

dAmbISMs: combination of dISMs and AmbISMs + *locality* constraint

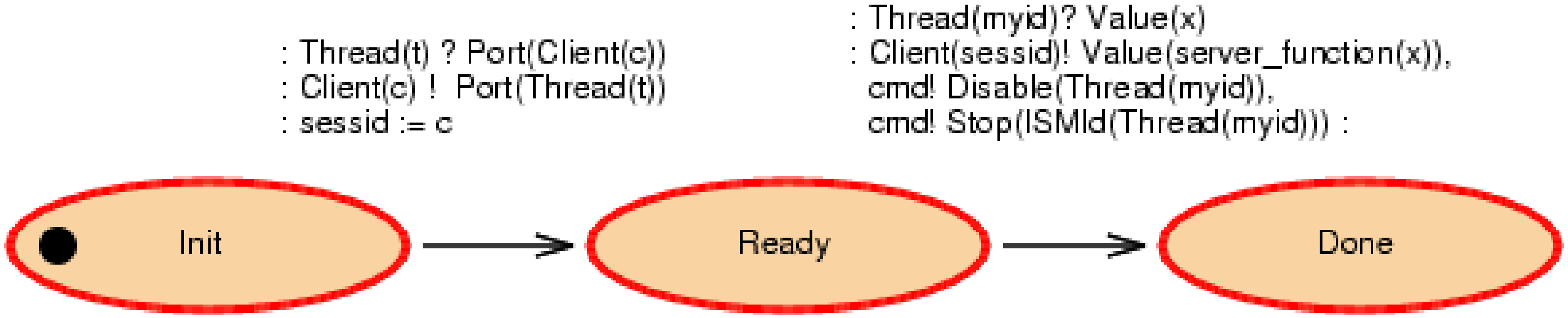
Client/Server example: System Structure Diagram



The server *Run*s a new working thread for each client request, *Convey*ing a *New*ly created port to the new thread.

State Transition Diagram: `Thread`

Transition format: [`guard`] : [`inputs`] : [`outputs`] : [`assignments`]



When finished, the thread *`Disable`*s its port and *`Stop`*s.

Isabelle Representation

```
ism Thread (myid::tid) =
 ports port
   inputs  "{}"
   outputs "{Client c
             |c. True}"
 messages msg
 commands cs_cmds
   default "[]"
 states state
  control Thread_control
    init "Init"
  data    Thread_data
```

```
transitions

init: Init → Ready
 in  "Thread myid" "[Port (Client c)]"
 out "Client c" "[Port (Thread myid)]"
 post sessid := "c"

work: Ready → Done
 in  "Thread myid" "[Value x]"
 out "Client (sessid s)"
          "[Value (server_function x)]"
 cmd "[Disable (Thread myid),
        Stop (ISMId (Thread myid))]"
```

Parameterized ISM, asynchronous multiple I/O, explicit state,
dISM commands, transition rules $\rightsquigarrow$ *rule induction principle*

Related Work

I/O Automata (IOAs) modeling asynchronous distributed computation

- $\oplus$ well-developed meta theory, good tool support
- $\ominus$ low-level interaction scheme (singleton & unbuffered input/output)

AutoFocus Automata modeling embedded systems

- $\oplus$ graphical design, simulation, model checking, code generation
- $\ominus$ clock-synchronous semantics

π-Calculus modeling communicating processes

- $\oplus$ very concise and flexible modeling of communication
- $\ominus$ state and local computation cumbersome to encode

StateMate modeling complex state-based systems

- $\oplus$ rich structural notions, modeling support
- $\ominus$ basic communication notion, poor proof support

Conclusion

- ISMs can be seen . . .
  - as high-level *Input-/Output Automata*
  - as asynchronous *AutoFocus Automata*
  - as stateful communicating *processes*
- Generic ISMs serve as toolkit for enhancing expressiveness
- Current instantiations support dynamic (e.g., ambient) systems
- Rather intuitive use, textual and graphical representation

$\Rightarrow$ Generic ISMs provide good support for practical formal system analysis

Future work: more applications, extended meta theory