CARL VON OSSIETZKY universität OLDENBURG

Semantics Group

Specifying and Checking Java using CSP

Michael Möller

University of Oldenburg, Germany

CARL VON OSSIETZKY universität OLDENBURG

Content

Semantics Group

- Why CSP?
- CSP-OZ to Java
- jassda
- Differences to CSP_M
- Conclusion and Future Work

CARL VON OSSIETZKY universität OLDENBURG

Why CSP?

Semantics Group

- Project ForMooS: **For**mal **M**ethods in **o**bject **o**riented **S**oftware Engineering
- base: CSP-OZ [C. Fischer 2000]
  - *Communicating Sequential Processes* [Hoare '85] +
  - *Object-Z* [Smith 2000]
- goal: UML (subset) → CSP-OZ → Java

CSP-OZ-Classes

- The three parts of a CSP-OZ-Class:
  1. Interface
  2. dynamical behaviour: CSP-Process
  3. State space and state transformation: Z-Part
- goal: map every part to the implementation

CSP-OZ-Classes

- The three parts of a CSP-OZ-Class:
  1. Interface
  2. dynamical behaviour: CSP-Process
  3. State space and state transformation: Z-Part
- goal: map every part to the implementation
  1. Java - Interface

CSP-OZ-Classes

- The three parts of a CSP-OZ-Class:
  1. Interface
  2. dynamical behaviour: CSP-Process
  3. State space and state transformation: Z-Part
- goal: map every part to the implementation
  1. Java - Interface
  3. Design-by-Contract, BISL

CSP-OZ-Classes

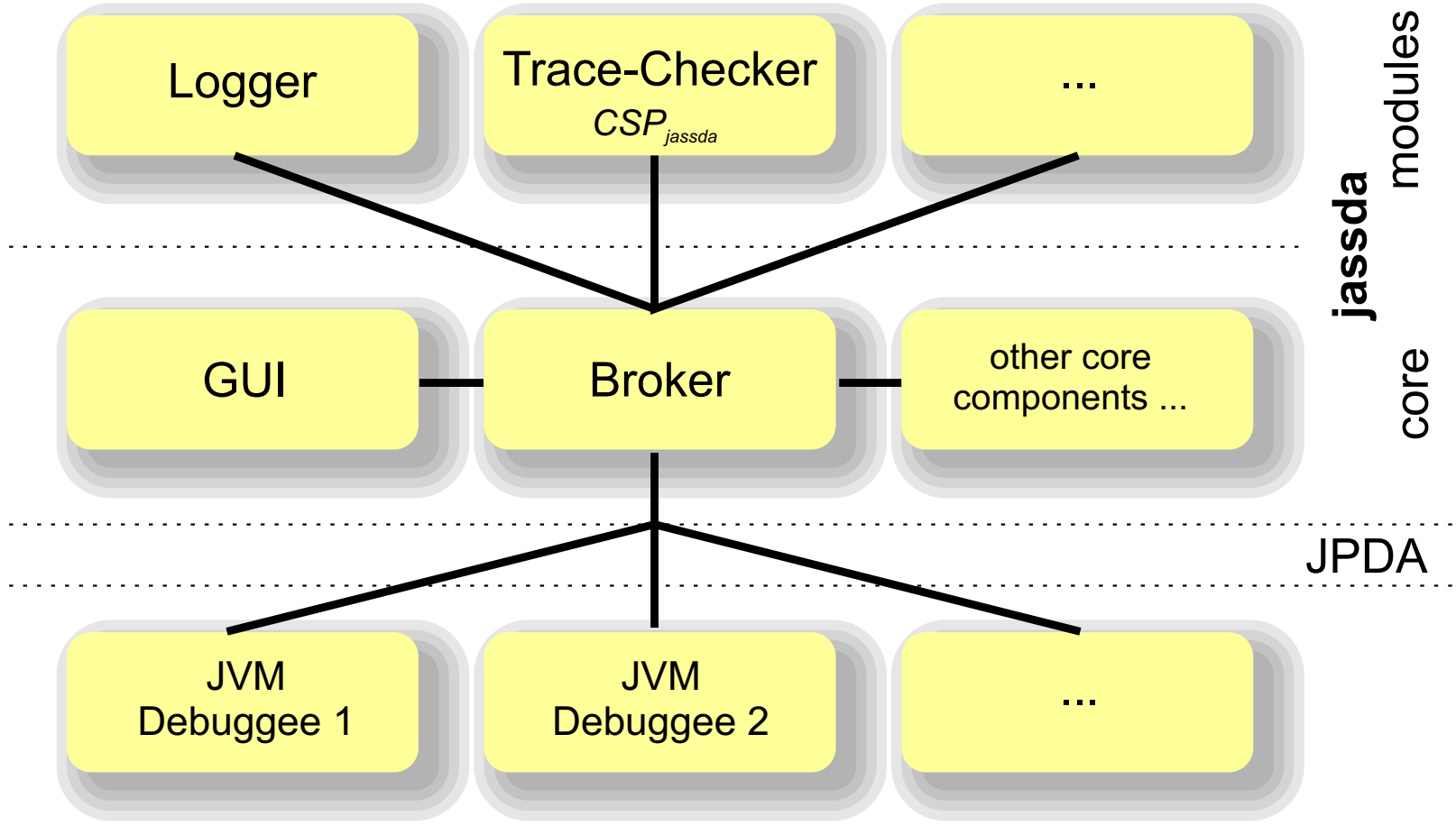
- The three parts of a CSP-OZ-Class:
  1. Interface
  2. dynamical behaviour: CSP-Process
  3. State space and state transformation: Z-Part
- goal: map every part to the implementation
  1. Java - Interface
  2. Trace Assertions
  3. Design-by-Contract, BISL

skip CSP-OZ

jassda Overview

- Jass: **J**ava with **ass**ertions
- jassda: **Jass D**ebug **A**rchitecture
- Debug on Byte-code level
- modular structure (framework for debugging)

jassda Architecture



Serialised Event Stream

- Using JDI means getting a **serialised** stream of events.
- serialised stream = trace
- CSP trace semantics
  - check: (trace of current run) $\in$ CSP trace semantics
  - “checking operational”:
    do not expand the trace semantics
    describe further trace(s) by processes (for every step)

Events

Possible *events*: Everything that can stop the VM via JDI. CSP_{jassda} basic events: (analogous to Design by Contract clauses

- Method entry point `begin`
- Normal method termination `end`
- Exceptional method termination `exception`

Event Properties

- Virtual Machine (Debugee)
- JDI

CARL VON OSSIETZKY universität OLDENBURG

Event Properties

Semantics Group

- Virtual Machine (Debugee)
- thread
- class
- instance
- method

CARL VON OSSIETZKY universität OLDENBURG

Event Properties

[[S]] Semantics Group

- Virtual Machine (Debugee)
- thread
- class
- instance
- method
- method arguments

Event Properties

- Virtual Machine (Debugee)
- thread
- class
- instance
- method
- method arguments
- result
  (with simple modification of the byte code)

Event Properties

- Virtual Machine (Debugee)
- thread
- class
- instance
- method
- method arguments
- result
  (with simple modification of the byte code)

Filtering of events through properties by Java classes (handler class)

CARL VON OSSIETZKY universität OLDENBURG

Event Sets

[[S]] Semantics Group

- Exactly one event is not very handy (how to define e.g. the thread?)

Event Sets

- Exactly one event is not very handy (how to define e.g. the thread?)
- Prefix operator: "$eventset \rightarrow Process$"
  (current event)$\in eventset$

Event Sets

- Exactly one event is not very handy (how to define e.g. the thread?)
- Prefix operator: "$eventset \rightarrow Process$" (current event)$\in eventset$
- definition of event sets

```
eventset myset = { handler="..."
debuggee="..." thread="..." ... }
```

(default property for handler and predefined event sets)

Event Sets

- Exactly one event is not very handy (how to define e.g. the thread?)
- Prefix operator: "$eventset \rightarrow Process$" (current event)$\in eventset$
- definition of event sets

```
eventset myset = { handler="..."
debuggee="..." thread="..." ... }
```

(default property for handler and predefined event sets)

- operations on sets: intersection, union

Binding Variables (2)

Binding event set variables similar to communication via channels in CSP_M:

- Communication in CSP_M:

Binding Variables (2)

Binding event set variables similar to communication via channels in CSP_M:

- Communication in CSP_M:

```
main = in?x -> out!x -> main
```

Binding Variables (2)

Binding event set variables similar to communication via channels in CSP_M:

- Communication in CSP_M:

```
main = in?x -> out!x -> main
```

- Variable binding in CSP_{jassda}:

Binding Variables (2)

Binding event set variables similar to communication via channels in CSP_M:

- Communication in CSP_M:

```
main = in?x -> out!x -> main
```

- Variable binding in CSP_{jassda}:

```
main() { in?x:[arg0] -> out!x -> main() }
```

Determinism (1)

- Internal Choice (CSP_M)

$$a \rightarrow P \sqcap b \rightarrow Q$$

Determinism (1)

- Internal Choice (CSP_M)

$$a \rightarrow P \sqcap b \rightarrow Q$$

- operational: nondeterministic choice

Determinism (1)

- Internal Choice (CSP_M)

$$a \rightarrow P \sqcap b \rightarrow Q$$

- operational: nondeterministic choice
- may reject a or b (without external influence)

Determinism (1)

- Internal Choice (CSP_M)

$$a \rightarrow P \sqcap b \rightarrow Q$$

- operational: nondeterministic choice
- may reject a or b (without external influence)
- for testing: program never satisfies *Spec*

Determinism (1)

- Internal Choice (CSP_M)

$$a \rightarrow P \sqcap b \rightarrow Q$$

- operational: nondeterministic choice
- may reject a or b (without external influence)
- for testing: program never satisfies $Spec$
- so: use trace semantics

$$traces(a \rightarrow P \sqcap b \rightarrow Q) = traces(a \rightarrow P \,\Box\, b \rightarrow Q)$$

Determinism (2)

- Nondeterministic Choice (CSP_M)

$$Spec = a \rightarrow P \,\Box\, a \rightarrow Q$$

Determinism (2)

- Nondeterministic Choice (CSP_M)

$$Spec = a \rightarrow P \,\Box\, a \rightarrow Q$$

- nondeterministic: $Spec \xrightarrow{a} P$ or $Spec \xrightarrow{a} Q$

Determinism (2)

- Nondeterministic Choice (CSP_M)

$$Spec = a \rightarrow P \,\Box\, a \rightarrow Q$$

- nondeterministic: $Spec \xrightarrow{a} P$ or $Spec \xrightarrow{a} Q$
- Delayed Choice (CSP_{jassda})

$$Spec = a \rightarrow P \,\Box\, b \rightarrow Q$$

Determinism (2)

- Nondeterministic Choice (CSP_M)

$$Spec = a \rightarrow P \,\Box\, a \rightarrow Q$$

- nondeterministic: $Spec \xrightarrow{a} P$ or $Spec \xrightarrow{a} Q$
- Delayed Choice (CSP_{jassda})

$$Spec = a \rightarrow P \,\Box\, b \rightarrow Q$$

- deterministic:

Determinism (2)

- Nondeterministic Choice (CSP_M)

$$Spec = a \rightarrow P \,\Box\, a \rightarrow Q$$

- nondeterministic: $Spec \xrightarrow{a} P$ or $Spec \xrightarrow{a} Q$
- Delayed Choice (CSP_{jassda})

$$Spec = a \rightarrow P \,\Box\, b \rightarrow Q$$

- deterministic:
  - $Spec \xrightarrow{\alpha} P$ if $\alpha \in (a \setminus b)$

Determinism (2)

- Nondeterministic Choice (CSP_M)

$$Spec = a \rightarrow P \,\Box\, a \rightarrow Q$$

- nondeterministic: $Spec \xrightarrow{a} P$ or $Spec \xrightarrow{a} Q$
- Delayed Choice (CSP_{jassda})

$$Spec = a \rightarrow P \,\Box\, b \rightarrow Q$$

- deterministic:
  - $Spec \xrightarrow{\alpha} P$ if $\alpha \in (a \setminus b)$
  - $Spec \xrightarrow{\alpha} Q$ if $\alpha \in (b \setminus a)$

Determinism (2)

- Nondeterministic Choice (CSP_M)

$$Spec = a \rightarrow P \,\Box\, a \rightarrow Q$$

- nondeterministic: $Spec \xrightarrow{a} P$ or $Spec \xrightarrow{a} Q$
- Delayed Choice (CSP_{jassda})

$$Spec = a \rightarrow P \,\Box\, b \rightarrow Q$$

- deterministic:
  - $Spec \xrightarrow{\alpha} P$ if $\alpha \in (a \setminus b)$
  - $Spec \xrightarrow{\alpha} Q$ if $\alpha \in (b \setminus a)$
  - $Spec \xrightarrow{\alpha} P \,\Box\, Q$ if $\alpha \in (a \cap b)$

Hello World

```
eventset helloWorld
 { handler="jass.debugger.jdi.eventset.GenericSet",
   class  ="HelloWorld" }
eventset start { eventtype="begin" method="start"}
eventset stop  { eventtype="begin" method="stop"}

main() {
  ||x:[instance] @ helloWorldProc(x)
}
helloWorldProc(x) {
  helloWorld.start.x ->
     helloWorld.stop.x -> helloWorldProc(x)
}
```

Conclusion

- CSP_{jassda}:

Conclusion

- CSP_{jassda}:
  - specifying dynamic behaviour of Java programs

Conclusion

- CSP_{jassda}:
  - specifying dynamic behaviour of Java programs
  - addition to Design by Contract (not a replacement)

Conclusion

- CSP_{jassda}:
  - specifying dynamic behaviour of Java programs
  - addition to Design by Contract (not a replacement)
  - extendable through handler classes

Conclusion

- CSP_{jassda}:
  - specifying dynamic behaviour of Java programs
  - addition to Design by Contract (not a replacement)
  - extendable through handler classes
- jassda tool:

Conclusion

- CSP_{jassda}:
  - specifying dynamic behaviour of Java programs
  - addition to Design by Contract (not a replacement)
  - extendable through handler classes
- jassda tool:
  - operates on Byte-code level

Conclusion

- CSP_{jassda}:
  - specifying dynamic behaviour of Java programs
  - addition to Design by Contract (not a replacement)
  - extendable through handler classes
- jassda tool:
  - operates on Byte-code level
  - third party products checkable (without source-code)

Conclusion

- CSP_{jassda}:
  - specifying dynamic behaviour of Java programs
  - addition to Design by Contract (not a replacement)
  - extendable through handler classes
- jassda tool:
  - operates on Byte-code level
  - third party products checkable (without source-code)
  - Java applications, applets and servlets checkable

Conclusion

- CSP_{jassda}:
  - specifying dynamic behaviour of Java programs
  - addition to Design by Contract (not a replacement)
  - extendable through handler classes
- jassda tool:
  - operates on Byte-code level
  - third party products checkable (without source-code)
  - Java applications, applets and servlets checkable
  - extendable through modular structure

Future Work

- translation: CSP-OZ $\longrightarrow$ CSP_{jassda} (tool supported)

Future Work

- translation: CSP-OZ $\longrightarrow$ CSP_{jassda} (tool supported)
- improve CSP_{jassda}

Future Work

- translation: CSP-OZ $\longrightarrow$ CSP_{jassda} (tool supported)
- improve CSP_{jassda}
- improve jassda

Future Work

- translation: CSP-OZ $\longrightarrow$ CSP_{jassda} (tool supported)
- improve CSP_{jassda}
- improve jassda
  - performance

Future Work

- translation: CSP-OZ $\longrightarrow$ CSP_{jassda} (tool supported)
- improve CSP_{jassda}
- improve jassda
  - performance
  - usability of GUI

Future Work

- translation: CSP-OZ $\longrightarrow$ CSP_{jassda} (tool supported)
- improve CSP_{jassda}
- improve jassda
  - performance
  - usability of GUI
- case studies: expressiveness of CSP_{jassda}, scalability and overhead of debug architecture

Future Work

- translation: CSP-OZ $\longrightarrow$ CSP_{jassda} (tool supported)
- improve CSP_{jassda}
- improve jassda
  - performance
  - usability of GUI
- case studies: expressiveness of CSP_{jassda}, scalability and overhead of debug architecture

Many thanks to Mark Brörkens