Specifying and Checking Java using CSP

Michael Möller

University of Oldenburg, Germany
Content

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

- Project ForMooS: **Formal Methods in object oriented Software Engineering**
- base: CSP-OZ [C. Fischer 2000]
  - *Communicating Sequential Processes* [Hoare ’85] +
  - *Object-Z* [Smith 2000]
- goal: UML (subset) → CSP-OZ → Java
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
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
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
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
**jassda Overview**

- Jass: **Java with assertions**
- jassda: **Jass Debug Architecture**
- Debug on Byte-code level
- modular structure (framework for debugging)
The diagram illustrates the architecture of the jassda system, which includes several core components and modules. The core components are:

- **GUI**
- **Broker**
- **Other core components**

The modules are:

- **Logger**
- **Trace-Checker**
- **...**

These components and modules are interconnected, forming the structure of the jassda system. The diagram also shows the integration with the JVM for debuggee 1, debuggee 2, and other JVM debuggees.
Serialised Event Stream

Using JDI means getting a **serialised** stream of events.

serialised stream = trace

CSP trace semantics

check: (trace of current run) ∈ 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 \texttt{begin}
- Normal method termination \texttt{end}
- Exceptional method termination \texttt{exception}
Event Properties

- Virtual Machine (Debugee)
- JDI
Event Properties

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

- 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)
Event Sets

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) ∈ eventset
Event Sets

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

- Prefix operator: “eventset → Process” (current event) ∈ eventset

- Definition of event sets

```java
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 → Process”
  (current event) ∈ eventset
- definition of event sets
  
  \[
  \text{eventset myset} = \{ \, \text{handler} = "..." \\
    \text{debuggee} = "..." \text{ thread} = "..." \ldots \, \}
  \]
  (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 $CSP_M$:

$\text{main} = \text{in?}x \rightarrow \text{out!}x \rightarrow \text{main}$
Binding Variables (2)

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

- Communication in $CSP_M$:
  $$\text{main }= \text{in?}x \rightarrow \text{out!}x \rightarrow \text{main}$$

- Variable binding in $CSP_{jassda}$:
Binding event set variables similar to communication via channels in $CSP_M$:

- Communication in $CSP_M$:
  
  \[
  \text{main} = \text{in}?x \rightarrow \text{out}!x \rightarrow \text{main}
  \]

- Variable binding in $CSP_{jassda}$:
  
  \[
  \text{main()} \{ \text{in}?x: [\text{arg0}] \rightarrow \text{out}!x \rightarrow \text{main()} \} 
  \]
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

\[ \text{traces}(a \rightarrow P \sqcap b \rightarrow Q) = \text{traces}(a \rightarrow P \sqcap b \rightarrow Q) \]
Determinism (2)

Nondeterministic Choice ($CSP_M$)

$$Spec = a \rightarrow P \square a \rightarrow Q$$
Determinism (2)

- Nondeterministic Choice ($CSP_M$)

\[ Spec = a \rightarrow P \n a \rightarrow Q \]

- nondeterministic: \[ Spec \xrightarrow{a} P \] or \[ Spec \xrightarrow{a} Q \]
Determinism (2)

- Nondeterministic Choice ($CSP_M$)
  \[ Spec = a \rightarrow P \square a \rightarrow Q \]

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

- Delayed Choice ($CSP_{jassda}$)
  \[ Spec = a \rightarrow P \square b \rightarrow Q \]
Determinism (2)

- Nondeterministic Choice ($CSP_M$)
  
  \[ Spec = a \rightarrow P \bigcirc a \rightarrow Q \]

- Nondeterministic: $Spec \overset{a}{\rightarrow} P$ or $Spec \overset{a}{\rightarrow} Q$

- Delayed Choice ($CSP_{jassda}$)
  
  \[ Spec = a \rightarrow P \bigcirc b \rightarrow Q \]

- Deterministic:
Determinism (2)

- Nondeterministic Choice ($CSP_M$)
  $$Spec = a \rightarrow P \lor a \rightarrow Q$$

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

- Delayed Choice ($CSP_{jassda}$)
  $$Spec = a \rightarrow P \land b \rightarrow Q$$

- deterministic:
  - $Spec \overset{\alpha}{\rightarrow} P$ if $\alpha \in (a \setminus b)$
Determinism (2)

- Nondeterministic Choice ($CSP_M$)
  \[ \text{Spec} = a \rightarrow P \sqcup a \rightarrow Q \]

- nondeterministic: \[ \text{Spec} \xrightarrow{a} P \text{ or } \text{Spec} \xrightarrow{a} Q \]

- Delayed Choice ($CSP_{jassda}$)
  \[ \text{Spec} = a \rightarrow P \sqcup b \rightarrow Q \]

- deterministic:
  \[ \text{Spec} \xrightarrow{\alpha} P \quad \text{if} \quad \alpha \in (a \setminus b) \]
  \[ \text{Spec} \xrightarrow{\alpha} Q \quad \text{if} \quad \alpha \in (b \setminus a) \]
Determinism (2)

- Nondeterministic Choice ($CSP_M$)

$$Spec = a \rightarrow P \square a \rightarrow Q$$

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

- Delayed Choice ($CSP_{jassda}$)

$$Spec = a \rightarrow P \square b \rightarrow Q$$

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

\[ CSP_{jassda} \]:

- **CSP** : 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
Conclusion

\[ \text{CSP}_{\text{jassda}} \]:

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

\[ \textit{CSP}_{\text{jassda}}: \]
\begin{itemize}
  \item specifying dynamic behaviour of Java programs
  \item addition to Design by Contract (not a replacement)
  \item extendable through handler classes
\end{itemize}

\begin{itemize}
  \item jassda tool:
\end{itemize}
**Conclusion**

\[\begin{align*}
\text{CSP}_{\text{jassda}}: \\
\quad \text{specifying dynamic behaviour of Java programs} \\
\quad \text{addition to Design by Contract (not a replacement)} \\
\quad \text{extendable through handler classes} \\
\text{jassda tool:} \\
\quad \text{operates on Byte-code level}
\end{align*}\]
Conclusion

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

\text{jassda} tool:
- operates on Byte-code level
- third party products checkable (without source-code)
Conclusion

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

\text{jassda tool:}
- operates on Byte-code level
- third party products checkable (without source-code)
- Java applications, applets and servlets checkable
Conclusion

\[ \text{CSP}_{jassda} : \]
- specifying dynamic behaviour of Java programs
- addition to Design by Contract (not a replacement)
- extendable through handler classes

\[ jassda \text{ tool:} \]
- 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 $\rightarrow$ CSP$_{\text{jassda}}$
(tool supported)
Future Work

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

翻译：CSP-OZ $\rightarrow CSP_{jassda}$

- 改进 $CSP_{jassda}$
- 改进 jassda
Future Work

Translation: CSP-OZ $\rightarrow CSP_{jassda}$ (tool supported)

- Improve $CSP_{jassda}$
- Improve jassda
  - Performance

Many thanks to Mark Brörken.
Future Work

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

→ translation: CSP-OZ $\rightarrow CSP_{jassda}$ (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 $\rightarrow CSP_{jassda}$ (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