DSLs mit Parsern komponieren

H. Krasemann, J. Brauer, C. Crasemann

2011

DSLs mit Parsern komponieren

- über mich
- DSLs - warum, was & wie
- Trecker, Panzer & SUV
- Beispiel AuDSL, MicrowaveOven
- Beispiel SStDSL
- Demo
- Fazit
- zum Lesen

über mich - meine Maßstäbe

- Praxis seit 1981
- Objektorientierung und Smalltalk seit 1984
- kommerzielles Grafik-Tool: ObjectDrawing
- OOAD-Tool ADvance für VisualWorks
- Metaprogrammierung für Tools in Smalltalk
- große Industrieprojekte in Smalltalk, C++, Java

- frühe Bindung, statische Typisierung
- viele „Technologien“ (Frameworks), viel technischer Code, viele Lernkurven,
- Abstraktionslücke

- späte Bindung, dynamische Sprachen
- Hierarchie von Sprachen
- orthogonale Sprachelemente
- Domain Specific Languages

Warum DSLs ?

- Lisp
  - Programmieren auf dem Syntax-Baum
  - Makros → DSLs

Paul Graham
„On Lisp“

- Objektorientierung
  - Zustand in der Neumann Maschine und in der Welt
  - Kapselung von Zustand
  - neues polymorphes Vokabular bei fixer Syntax & Semantik

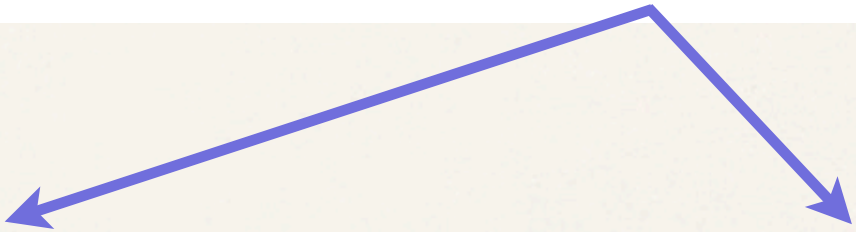
--> poor man‘s DSL in St

- DSL: OO + integrierte Spracherweiterung

DSLs - Kategorien

- neue Syntax + neue Semantik --> DSL
  - alte Syntax + neue Semantik --> Sprachredefinition
  - neue Syntax + alte Semantik --> zB Teil von Scala
- gemeinsamer Adressraum --> interne DSL

Begriffsverwirrung



compositional

getrennt kompiliert

Konfigurations-DSL

braucht „nur“ Compiler & API

computational

integriert kompiliert

spracherweiternde DSL

„Language Workbench“

oder „LanguageBox“

Helvetia Renggli

Konfigurations-DSL

- intern
- umrissener Gegenstand
- definierter Ausführungszeitpunkt
- = lokal im Code
- ohne Language Workbench oder -Box:
  der DSL-Compiler kann separat aufgerufen werden

Wie parsen ?

Host Syntax: Poor Man‘s DSL

geht besonders gut in Smalltalk, holpert aber

Parser Generator: z.B. ANTLR

das ganz große Geschütz. Lexer, Parser, und Code-Generator

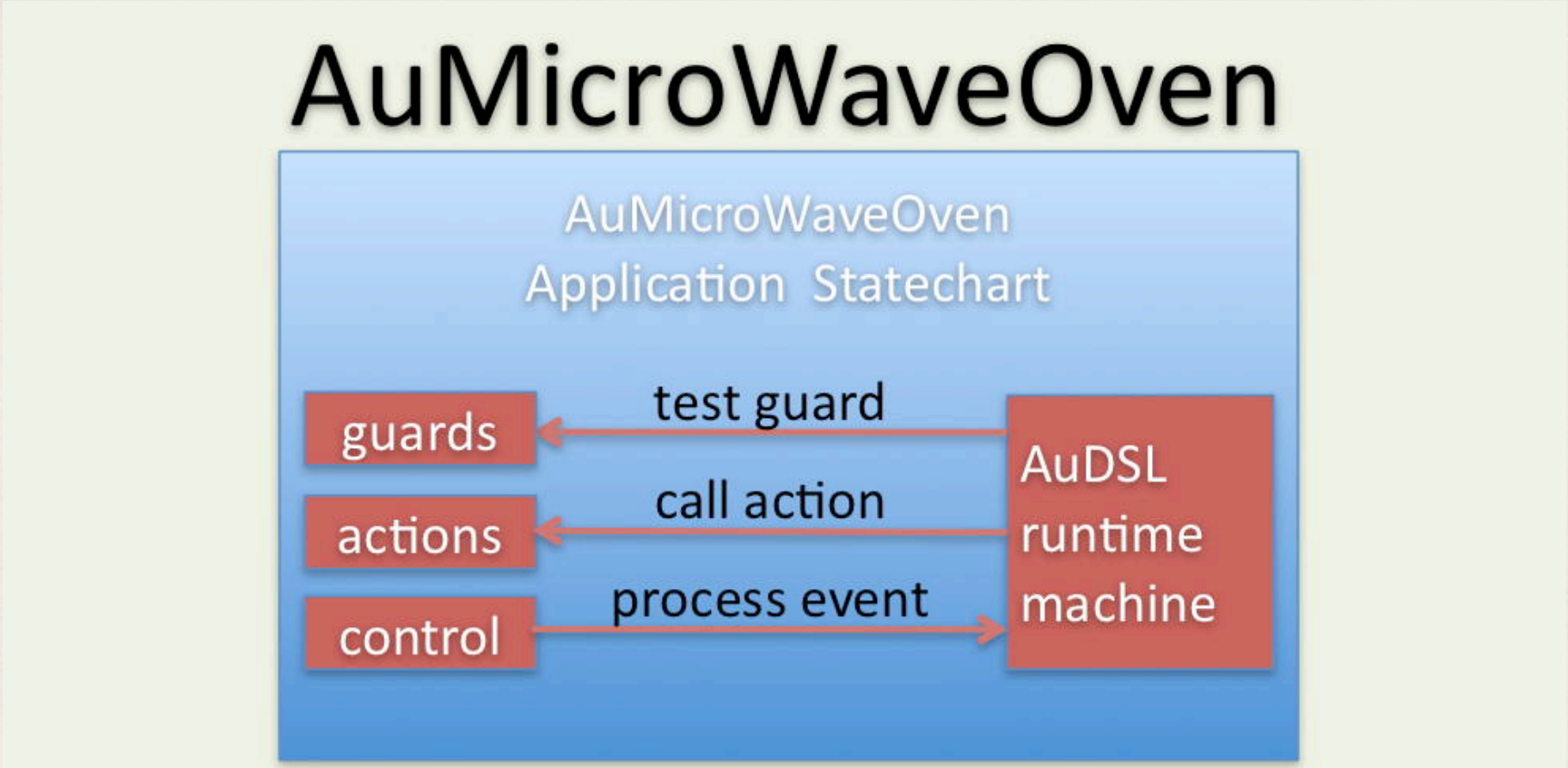
Parser Kombinatoren: PetitParser, Scala

sind elegant und mächtig: sogar nicht kontextfreie Grammatiken.

Language Box: Helvetia und der PetitParser

- Helvetia beruht auf 3 Technologien
  - Smalltalk
  - Petit Parser (Parser Kombination)
  - Language Box
- PetitParser (Parser Kombination)
  - Parsing Expression Grammars
  - Scannerlose Parser nutzen den vollen lexikalischen Kontext
  - Parser-Kombination aus einzelnen Parser-Bausteinen (syntaxorientiert)
  - Packrat-Parser sorgen für Effizienz
- Language Box
  - Erweiterungspunkte der Host-Grammatik für neue Grammatiken
  - Transformationen
    - zur Code-Repräsentation der Wirtssprache,
    - zur Code-Produktion,
    - Hervorhebungen und Aktionen (für den Editor)

Scala enthält ein Combinator Parsing Framework, damit lässt sich ebenfalls eine AST-separierte DSL bauen



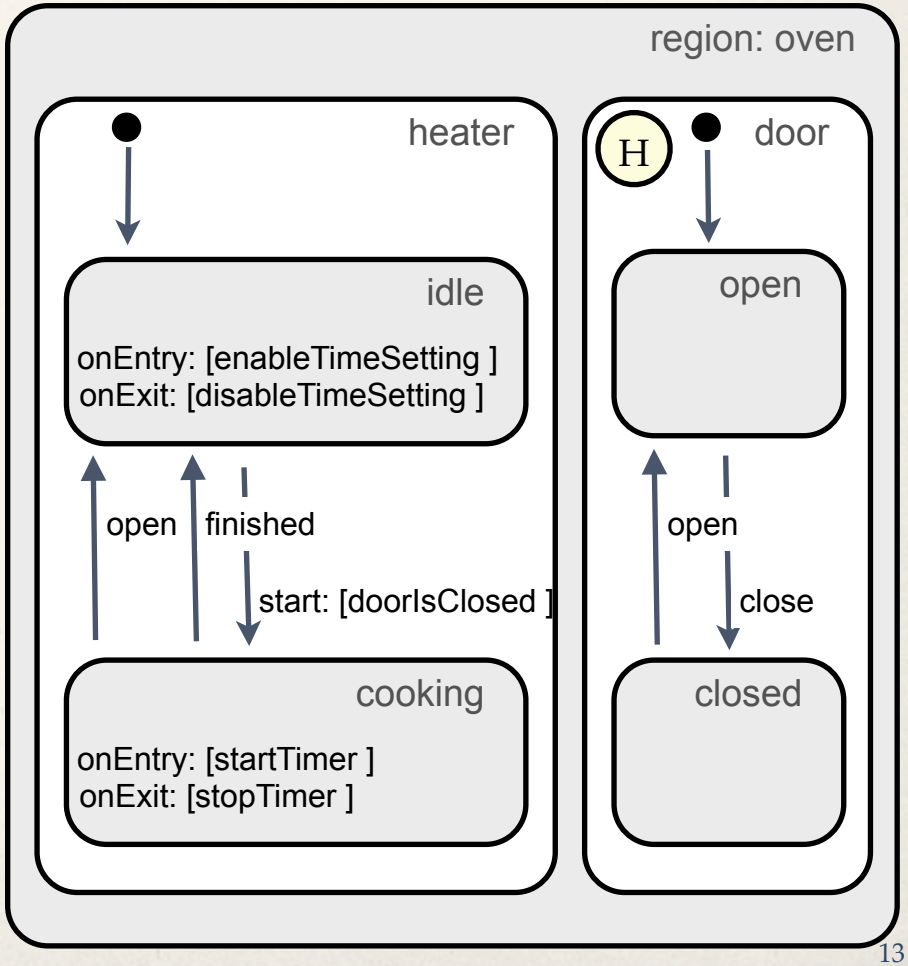
AuDSL - Beispiel MicrowaveOven

Alle Zustände und die Wechsel-Logik sind deklarativ im Statechart (Harel-Statechart)

Eine Compositional / Konfigurations-DSL

AuDSL - Beispiel

```
(r: oven
    (heater
        (idle
            onEntry: [enableTimeSetting ]
            onExit: [disableTimeSetting ]
            start [doorIsClosed ] -> cooking)
        (cooking
            onEntry: [startTimer ]
            onExit: [stopTimer]
            open -> idle
            finished -> idle))
    (door
        history
        (open
            close -> closed)
        (closed
            open -> open)))
```



AuDSL - BNF : Smalltalk

```
state       = '(' (regiontype | xortype) {onentry} {onexit} {transition+} {state+} ')'
regiontype  = 'r:' identifier
xortype     = identifier {history}
history     = 'history'
onentry     = 'onEntry:' '[' identifier+ ']'
onexit      = 'onExit:' '[' identifier+ ']'
transition  = identifier { '[' identifier+ ']' } '->' identifier { '[' identifier+ ']' }
identifier  = letter word*
```

states sind rekursiv !

event [guards] -> target [actions]

```
state       := PPUnresolvedParser new.
identifier  := (#letter asParser , #word asParser star) token trim.
transition  := identifier, ($[ asParser, identifier plus, $] asParser) trim optional,
                    '->' asParser, identifier, ($[ asParser, identifier plus, $] asParser) trim optional.
onexit      := 'onExit:' asParser trim, $[ asParser, identifier plus, $] asParser.
onentry     := 'onEntry:' asParser trim, $[ asParser, identifier plus, $] asParser.
history     := 'history' asParser trim.
xortype     := identifier, history optional.
regiontype  := 'r:' asParser, identifier.
state  def:   $( asParser, (regiontype/xortype), onentry optional, onexit optional,
                                transition plus optional, state plus optional, $) asParser trim.
auDSL       := state trim end.
```

trim entfernt Whitespace

rekursiv durch Redefinition !

AuDSL - API und Codegrößen

Schnittstelle

1 Kl.; 26 LOC; 1 Var.; 7 Meth.

Au3Model

AuDSL-Parser

1 Skript mit 56 LOC

oder:

Au3Grammar, Au3Compiler

2 Kl.; 62 LOC; 9 Var.; 19 Meth.

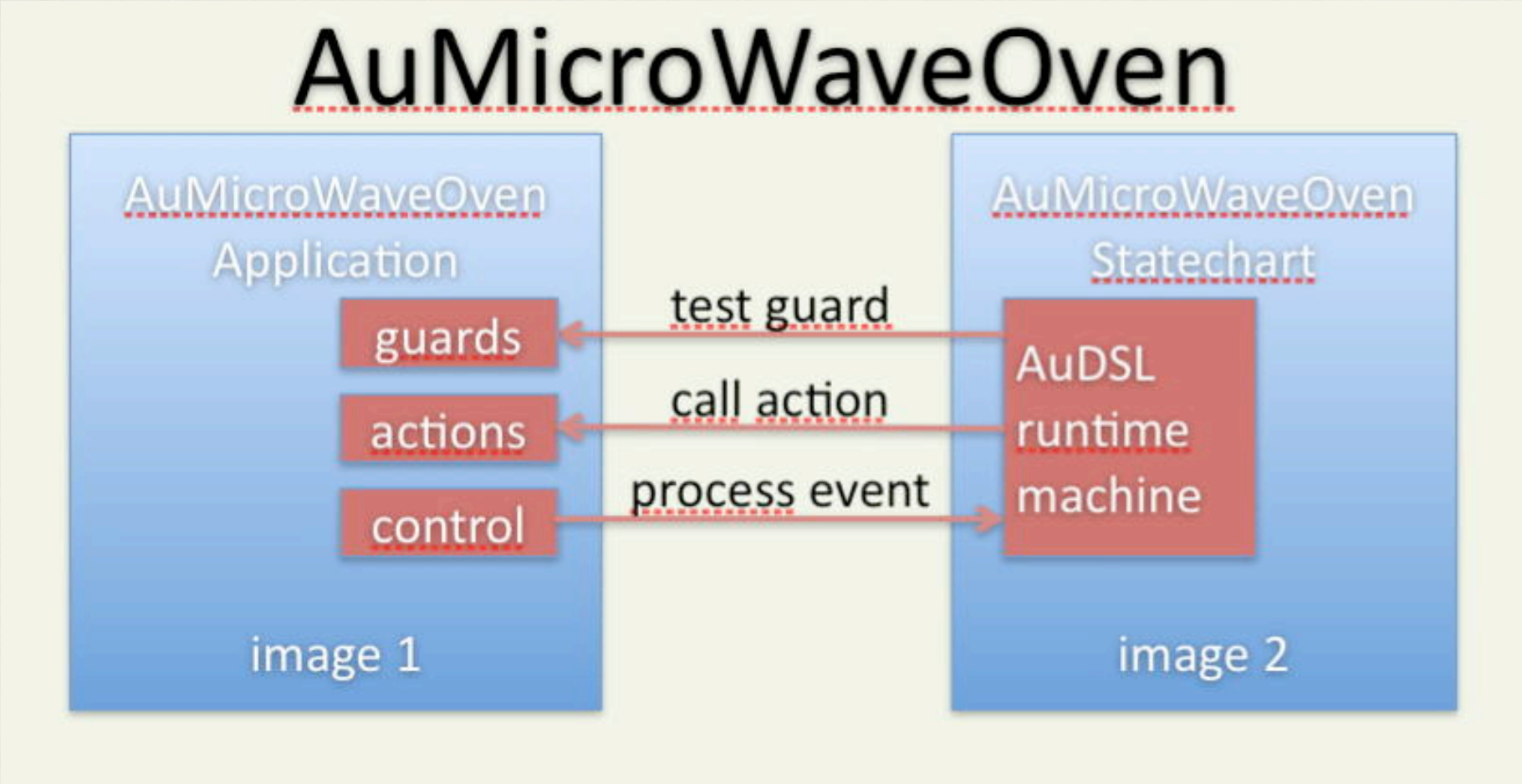
- API der Anwendung

Spec (= ein String)
createStates, super initialize
process: #event
actions aus der DSL
predicates aus der DSL

Semantisches Modell

4 Kl.; 156 LOC; 18 Var.; 62 Meth.

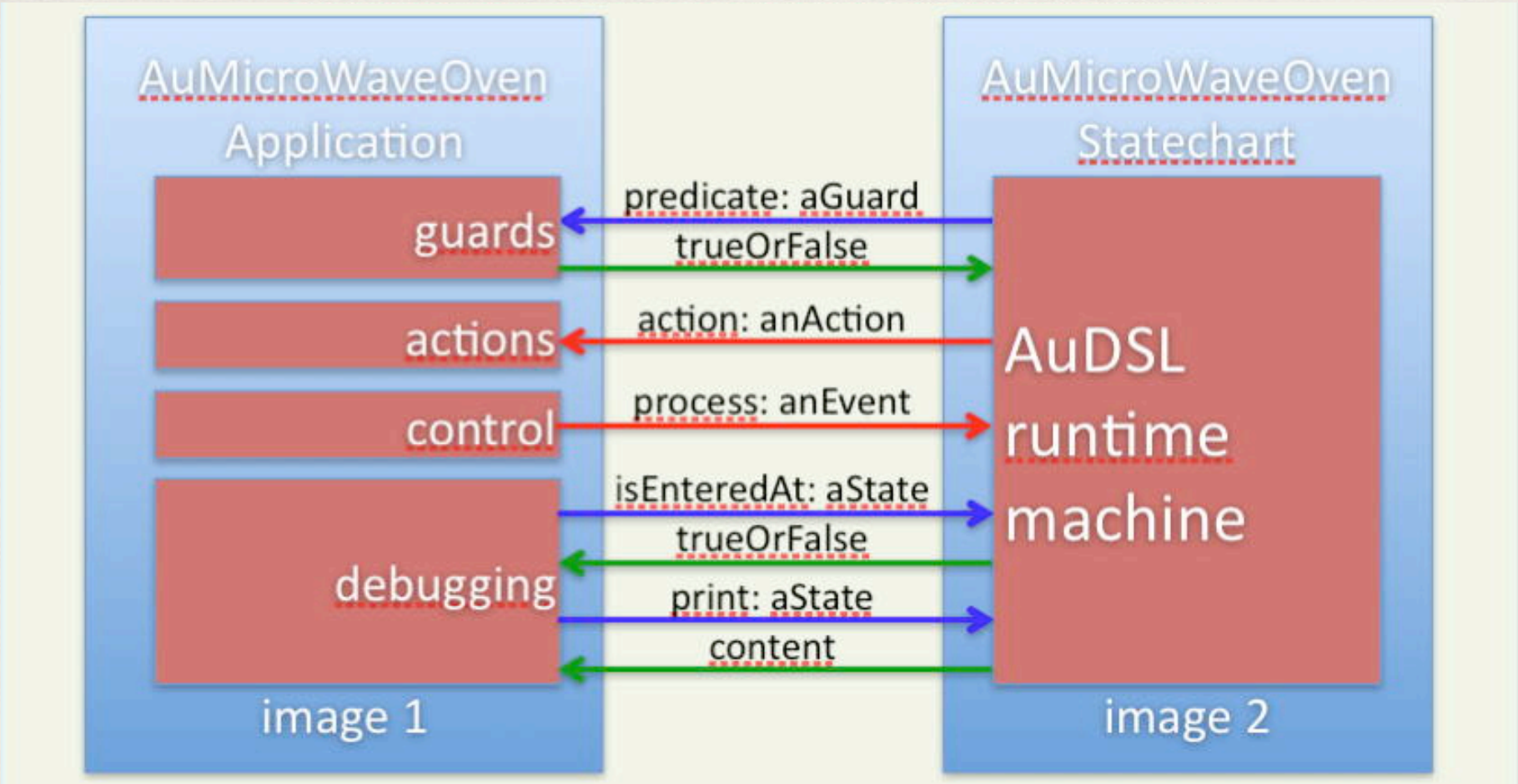
Au3State
Au3XORState, Au3Region
Au3Transition



SStDSL - Beispiel für MicrowaveOven

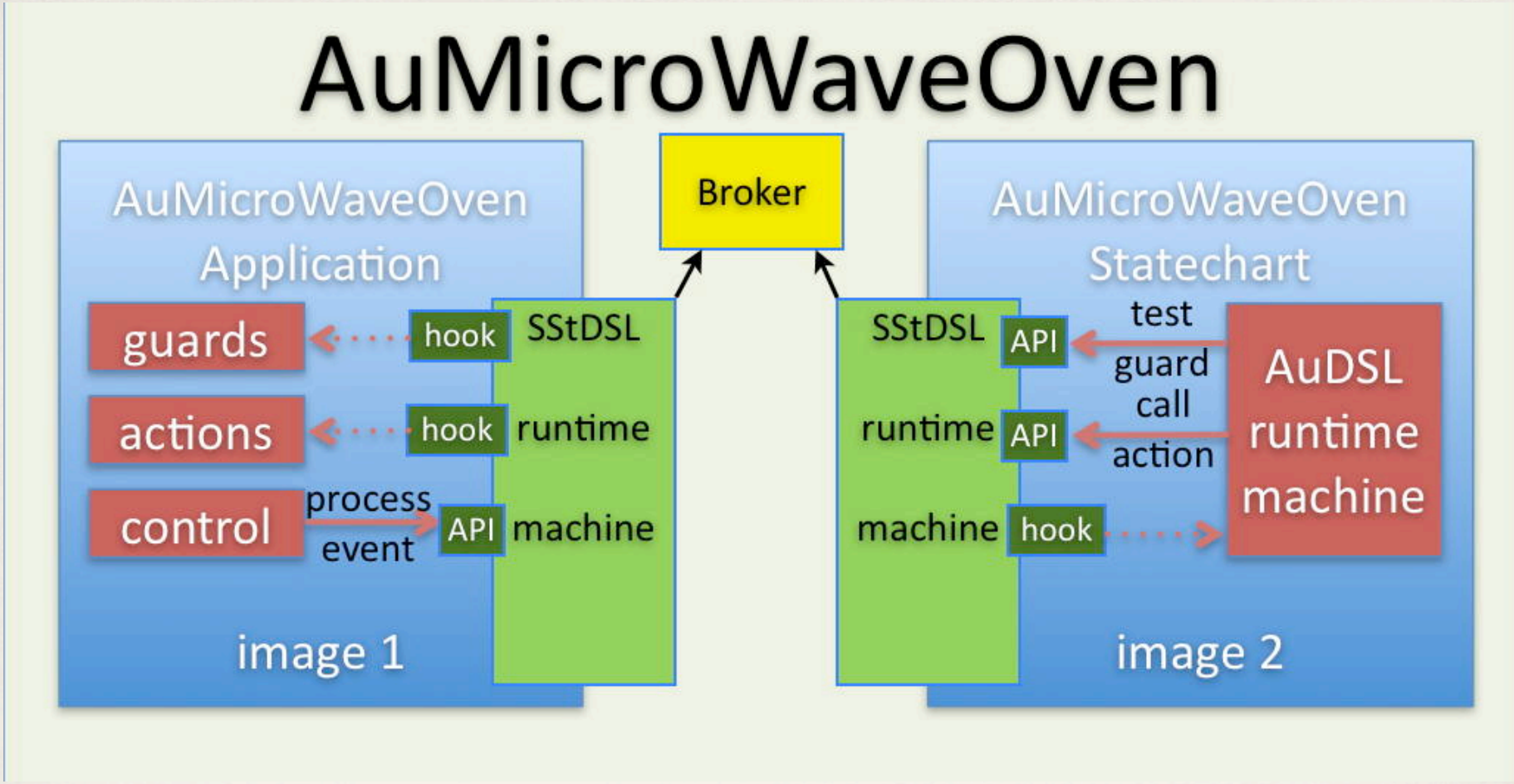
AuMicroWaveOven zerlegen in Statechart und zustandslose Anwendung

Eine Compositional / Konfigurations-DSL



Nachrichten: operativ und für Debugging

funktioniert mit 2 Images oder mit 2 Rechnern



SStDSL - Beispiel für MicrowaveOven

je Nachricht ein API zum Senden und ein Hook zum Empfangen

SStDSL - Beispiel

```
INTERFACE MicrowaveStatechart          /* bei der MicrowaveApp */
  TECHNOLOGY FTP
  URL www.myhomepage.com
  VERSION 1.0
  SEND (
    COMMAND process VERSION 1.0
    REQUEST print VERSION 0.9
      RESPONSE content VERSIONS [1.0 0.9] TIMEOUT 500
    REQUEST isEnteredAt VERSION 0.9
       RESPONSE trueOrFalse VERSIONS [0.9 1.0 1.1] TIMEOUT 600 )
  RECEIVE toApp (                      /* toApp is the receiver queue */
    COMMAND action VERSIONS [0.9 1.0, 1.1]
    REQUEST predicate VERSIONS [0.9 1.0]
       RESPONSE trueOrFalse VERSION 0.9 )
  LOGLEVEL                             /* vordefinierter Loglevel */
  TIMING  (print, isEnteredAt)
  MONITORING                                    /* Monitorqueue */

INTERFACE MicrowaveApp            /* beim MicrowaveStatechart */
  ......
  SEND (
    COMMAND action VERSION 1.0
    REQUEST predicate VERSION 1.0
      RESPONSE trueOrFalse VERSION [0.9 1.0] TIMEOUT 400 )
  ......
```

```
MESSAGES (
  COMMAND process
    VERSION 1.0
    DATA event
  COMMAND do
    VERSION 1.0
    DATA actionName
  REQUEST test
    VERSION 1.0
    DATA predicateName
  RESPONSE trueOrFalse
    VERSION 1.0
    DATA trueOrFalse
  REQUEST isEnteredAt
    VERSION 1.0
    DATA stateName
  REQUEST print
    VERSION 1.0
    DATA scName
  RESPONSE content
    VERSION 1.0
    DATA contentString )
```

SStDSL - BNF

- sstDSL = interface | messages

```
/* ========================= Regeln für das INTERFACE */
interface                  =  'INTERFACE' identifier,
                              'TECHNOLOGY' technology
                              'URL' url,
                              'VERSION' version
                              'SEND' send*,
                              'RECEIVE' queue2app receive*
                              'LOGLEVEL'  ... /* hier noch ausgelassen */
                              'TIMING'  ...   /* hier noch ausgelassen */
                             'MONITORING' .../* hier noch ausgelassen */
technology                 =  ftp | jms | amqp | sqs
ftp                        =  'FTP' /* und entsprechend für die anderen Technologien */
url                        =  (letter | '.' | '/')+
send                       =  sendCommand | sendNotification | sendRequestResponse
sendCommand                = 'COMMAND' identifier 'VERSION' version
sendNotification           = 'NOTIFICATION' identifier 'VERSION' version
sendRequestResponse        = sendRequest receiveResponse timeout
sendRequest                = 'REQUEST' identifier 'VERSION' version
receiveResponse            = 'RESPONSE' identifier 'VERSIONS' versions
timeout                    = 'TIMEOUT' integer
queue2app                  = identifier
receive                    = receiveCommand | receiveNotification | receiveRequestResponse
receiveCommand             = 'COMMAND' identifier 'VERSIONS' versions
receiveNotification        = 'NOTIFICATION' identifier 'VERSIONS' versions
receiveRequestResponse     = receiveRequest sendResponse
receiveRequest             = 'REQUEST' identifier 'VERSIONS' versions
sendResponse               = 'RESPONSE' identifier 'VERSION' version
/* ========================= Regeln für die MESSAGES */
messages                   = 'MESSAGES' message+
message                    = command | notification | request | response
command                    = 'COMMAND' identifier 'VERSION' version 'DATA' identifier
notification               = 'NOTIFICATION' identifier 'VERSION' version 'DATA' identifier
request                    = 'REQUEST' identifier 'VERSION' version 'DATA' identifier
response                   = 'RESPONSE' identifier 'VERSION' version 'DATA' identifier
/* ========================= gemeinsam genutzte Regeln */
versions                   = '[' version+ ']'
version                    = (letter | digit | '.')+
integer                    = ziffer+
identifier                 = letter word*
```

SStDSL - API

INTERFACE MicrovaweApp , ähnlich für MicrovaweStatechart

MicrowaveApp methodsFor: 'SStDSL sends'

process: anEvent ask: aReceiverQueue
```
"(command) generated by SStDSL
call this method from your code - do not change"
self send: (self newMsg: #process content: anEvent) to: aReceiverQueue
```

MicrowaveApp methodsFor: 'SStDSL requests'

isEnteredAt: aStateName ask: aReceiverQueue
```
"(request) generated by SStDSL - answers the response content of message #trueOrFalse: aTrueOrFalse
call this method from your code - do not change"
^self request: (self newMsg: #isEnteredAt content: aStateName) to: aReceiverQueue
```

print: aScName ask: aReceiverQueue
```
"(request) generated by SStDSL - answers the response content of message #content: aContentString
call this method from your code - do not change"
^self request: (self newMsg: #print content: aScName) to: aReceiverQueue
```

MicrowaveApp methodsFor: 'SStDSL receive hooks'

test: aPredicateName
```
"(request) hook method generated by SStDSL"
"please define your code - answer aTrueOrFalse "
^'exampleString'
```

do: anActionName
```
"(command) hook method generated by SStDSL"
"please define your code"
```

Demo

Fazit: Anwendungscodegrößen mit DSLs

- 67 LOC — AuMicrovaweOven-Beispiel
- 17 + 18 LOC — beide *INTERFACE* Spezifikationen
- 22 LOC — *MESSAGES* Spezifikation
- 13 LOC — Statechart Glue-Code
- 19 LOC — Statechart

Fazit: Codequalität

- Spezifikationen sind deklarativ = lesbar und verstehbar
- angemessene Syntax bei jeder Problemstellung
- keine Belastung mit Schnittstellen-Semantik & Technik
- keine Lernkurven (die hat der Lieferant der DSL)

- was für ein Unterschied zu heutigem Statechart / Schnittstellen-Code !

auf den Schultern der folgenden Riesen

- **Generell**
  J. Brauer, C. Crasemann, H. Krasemann, Auf dem Weg zu idealen Programmierwerkzeugen – Bestandsaufnahme und Ausblick, in: Informatik-Spektrum, 31(6), Dezember 2008
  M. Fowler, Domain Specific Languages, Addison-Wesley 2010 und Referenzen dort
  Krasemann, H., J. Brauer und C. Crasemann: DSL MIT PARSER- KOMBINATOREN: Mit wenig Code zu einer Harel- Statechart-DSL. OBJEKTspektrum, (04), Juli/August 2011
- **Parser-Komposition**
  G. Hutton, Higher-order functions for parsing, in: Journal of Functional Programming, 2(3), 1992
  B. Ford, Parsing expression grammars: a recognition-based syntactic foundation, in: POPL '04: Proc. of the 31st ACM SIGPLAN-SIGACT, New York 2004
- **Scala und Scala Parser Combinators**
  M. Odersky, L. Spoon, B. Venners, Programming in Scala, Artima 2008
  F. Piessens, A. Moors, M. Odersky, Celestijnenlaan 200A-B-3001 Heverlee (Belgium) Parser Combinators in Scala Adriaan Moors Frank Piessens, Techn. Ber., 2008
  S. Freund, Combinator Parsing in Scala, Vortrag im Arbeitskreis Objekttechnologie Norddeutschland der HAW Hamburg, Juni 2010, siehe: http://users.informatik.haw-hamburg.de/~sarstedt/AKOT
- **PetitParser**
  L. Renggli, S. Ducasse, T. Gîrba, O. Nierstrasz, Practical Dynamic Grammars for Dynamic Languages. in: 4th Workshop on Dynamic Languages and Applications (DYLA 2010), ACM 2010

und ...

- **Helvetia**
  L. Renggli, M. Denker, O. Nierstrasz, Language Boxes: Bending the Host Language with Modular Language Changes, in: Proc. of 2nd Int. Conf., SLE 2009, Springer 2009
  L. Renggli, T. Gîrba, O. Nierstrasz, Embedding Languages Without Breaking Tools, in: ECOOP 2010: Proc. of the 24th European Conference on Object-Oriented Programming, Springer 2010
  Renggli, L.: Dynamic Language Embedding. Doktorarbeit, Philosophisch-Naturwissenschaftliche Fakultät der Universität Bern, 2010
- **Statecharts**
  D. Harel, Statecharts: A visual formalism for complex systems, in: Sci. Comput. Program., 8(3), 1987
- **Schnittstellen**
  Hohpe, G. und B. Woolf: *Enterprise Integration Patterns*. A Martin Fowler Signature Book. Pearson Education, 2004
  Meyer, B.: *Object-oriented Software Construction*. C.A.R. Hoare Series. Prentice Hall, 1988, Seite 133
  Advanced Message Queuing Protocol - www.amqp.org
  RabbitMQ - AMQP Messenger www.rabbitmq.com
  Amazon: *Amazon Simple Queue Service (Amazon SQS) aws.amazon.com/de/sqs*, 2011
  Sun Microsystems: *Java Message Service 1.1*, April 2002
- **Code**
  SqueakSource, DSL for Harel Statecharts (Helvetia edition), siehe: www.squeaksource.com/AuDSL.html
  SqueakSource, Schnittstellen-DSL, siehe: http://www.squeaksource.com/SStDSL.html