YAKINDU is a modular toolkit for model-based development (of embedded systems)
YAKINDU Modules

- independent and self-contained
- not bound to a specific methodology
- usable on their own
- open & extendable
- composable to (domain-specific) language workbenches

➡ Reuse of
  - modeling language
  - tools
YAKINDU Statechart

- **Formalism** similar to state machines as defined by David Harel, but:
  - **self-contained** with a well defined **interface**
  - with a **cycle-based** execution semantics

- allows processing concurrent events
- event-driven behavior can be defined on top
- time control is delegated to the environment
Yakindu SCT - Editing
Yakindu SCT - Editing

Integrated Graphical & Textual Modeling
Yakindu SCT - Simulation
Yakindu SCT - Simulation

Editor

SGraph

SText

Easy adoptable simulation

SExec

Simulator

Donnerstag, 29. März 12
Yakindu SCT - Code Generation

• Yakindu comprises code generators for Java, C, C++
• All generators can be „customized“ by a generator model

• Custom generators based on Xpand & Xtend2/Java can be easily integrated
Yakindu SCT - Code Generation

- Yakindu comprises code generators for Java, C, C++
- All generators can be "customized" by a generator model
- Custom generators based on Xpand & Xtend2/Java can be easily integrated

Flexible Code Generation

- Editor
- SGraph
- SText
- SExec
- SGen
- Simulator
- Code-Generators

Donnerstag, 29. März 12
DEMO

interface :
  in event e1
  in event e2
  var counter : integer = 0

main region

S1
  e1
  after 2s
  e2/counter = 0

S2
  entry / counter = counter + 1
Yakindu SCT - Extensibility

- Recap: different models are used around the Statechart formalism

- **SGraph** (EMF): specification of graphical structures
- **SText** (Xtext): textual specification of declarations & expressions
- **SExec** (EMF): sequentialized statechart execution
- **SGen** (Xtext): code generator parameterization
Built-In Extensibility

- Restriction of structural concepts (SGraph)
- Customization of declarations & expressions (SText)
- Adoption of the execution semantics (SExec)
- Adoption of existing or integration of custom code generators
- Integration of custom type system, augmentation by application types
- Integration of additional validation constraints
The Statechart Application Gap

State-based modeling is useful in many domains
The Statechart Application Gap

State-based modeling is useful in many domains

Typically, statecharts are independent of any domain
The Statechart Application Gap

State-based modeling is useful in many domains

Typically, statecharts are independent of any domain

• How can statecharts be adopted to a specific domain?
• How can tools support this adoption?
Example: Domain Concepts - HMI

```app
HeadUnit {

ACScene Ac {
  animation Intro
  animation Outro reverse Intro1

  Slider : tempSetPoint
  Label : temperature
}

scene Main {
  Button : climate
  Button : media
  Button : car
  Button : phone
}

scene Media { ... }
scene Car { ... }
layouted_scene Phone { ... }
}
```
Domain Specific Statecharts

• Improving expressiveness and semantic integration by adopting domain concepts:
  
  • Refer to domain concepts within declarations (events, variables) and expressions (feature-calls)
  
  • Concepts from HMI domain: widget (button, label, etc.), scene, popup, animation, button-click, intro, outro,...
Integration of HMI Concepts

```javascript
app HeadUnit {
    scene Main {
        Button : climate
        Button : media
        Button : car
        Button : phone
    }
    ACScene Ac {
        animation Intro
        animation Outro reverse Intro1
        Slider : tempSetPoint
        Label : temperature
    }
    scene Media {...}
    scene Car {...}
    layouted_scene Phone {...}
}
```
DEMO
Domain-Specific Statechart Approach
Domain-Specific Statechart Approach

Structural Concepts (SGraph)

Generic
Domain-Specific Statechart Approach

Structural Concepts (SGraph)

Generic

Declarations & Expressions (SText)

/* ---- root rules ----
These root rules are not relevant for the grammar integration into a single grammar. */

Root: (roots+={Root}) ;

DefRoot: StatechartRoot | StateRoot | TransitionRoot;

Scope returns sct::Scope
   SimpleScope | StatechartScope;
// a SimpleScope and some other possible Scope classes

SimpleScope returns sct::Scope
   {SimpleScope} (declarations=declarations);
// defines the possible scopes for statecharts

StatechartScope returns sct::Scope
   InterfaceScope | InternalScope;

InterfaceScope returns sct::Scope:
Domain-Specific Statechart Approach

Domain-Specific

HMI Meta Model

Structural Concepts (SGraph)

Generic

Declarations & Expressions (SText)

Donnerstag, 29. März 12
Domain-Specific Statechart Approach

Domain-Specific

HMI Meta Model

Structural Concepts (SGraph)

Generic

Donnerstag, 29. März 12
Domain-Specific Statechart Approach

Domain-Specific

HMI Meta Model

references

HMI Declarations

declares

extends

Structural Concepts (SGraph)

extends

Declarations & Expressions (SText)

Donnerstag, 29. März 12
Domain-Specific Statechart Approach

Domain-Specific

HMI Meta Model

Structural Concepts (SGraph)

Generic

HMI Declarations

Declarations & Expressions (SText)

Donnerstag, 29. März 12
Yakindu SCT

- Open Source / EPL
- Hosted at EclipseLabs
- Eclipse-Proposal planned for 2012
  - Interested parties welcome!

- Important Links:
  - Project Site: http://yakindu.org
  - Eclipse Labs Site: http://code.google.com/a/eclipselabs.org/p/yakindu/
  - Update Site: http://updates.yakindu.com/indigo/milestones/
Thank You! Questions?