Yakindu Tools & Domain-Specific Statecharts

Axel Terfloth
itemis AG
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 SCT
Statechart Tools

heating

interface hmi:
in event toggleOnOff
in event setup
in event increaseTemp
in event decreaseTemp
interface controller:
var tempSetPoint : integer
in event tempDrop : integer
in event tempChanged : integer
interface system:
in event day
in event night
internal:
var desiredTemp : integer
var actualTemp : integer
var nightReduction : integer = 3
var tempChanged / actualTemp = temp...

SCT
Damos
CReMa
CoMo
Eclipse Platform
YAKINDU Damos
Dataflow-oriented Modeling
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 different domains?
• How can tools support this adoption?
Example: Domain Concepts - HMI

```java
app cc {
    scene Menu {
        Button : info
        Button : media
        Button : navigation
    }
    scene Info {
        InfoArea : top
        InfoArea : middle
        InfoArea : bottom
        InfoPane : welcome
        InfoPane : clock
        InfoPane : averageSpeed
        InfoPane : tripDistance
        InfoPane : temp
        InfoPane : pressure
    }
    scene Media { ... }
    scene Navigation { ... }
}
```
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

```plaintext
app cc {
    scene Menu {
        Button : info
        Button : media
        Button : navigation
    }
    scene Info {
        InfoArea : top
        InfoArea : middle
        InfoArea : bottom
        InfoPane : welcome
        InfoPane : clock
        InfoPane : averageSpeed
        InfoPane : tripDistance
        InfoPane : temp
        InfoPane : pressure
    }
    scene Media { ... }
    scene Navigation { ... }
}
```
Yakindu SCT - Extensibility

- 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 (SEexec)
• Adoption of existing or integration of custom code generators
• Integration of custom type system, augmentation by application types
• Integration of additional validation constraints
YAKINDU SCT Approach

Domain-Specific

HMI Meta Model

Generic

Structural Concepts (SGraph)

HMI Declarations

grammar com.yakindu.hmi.sctmodel.HMIText with org.yakindu.sct.model.stext.SText

/* ---- root rules ----
These root rules are not relevant for the grammar integration into a single grammar.
*/
Root:
  (roots+=DefRoot)*;

DefRoot:
  StatechartRoot | StateRoot | TransitionRoot;

Scope returns sct::Scope:
(SimpleScope | StatechartScope);
// a SimpleScope is used for states and regions
SimpleScope returns sct::Scope:
(SimpleScope | InternalScope);
// defines the possible scopes for statecharts
StatechartScope returns sct::Scope:
(InterfaceScope | InternalScope);

Declarations & Expressions (SText)

grammar org.yakindu.sct.model.stext.SText with org.eclipse.xtext.common.Terminals

/* ---- root rules ----
These root rules are not relevant for the grammar integration into a single grammar.
*/
Root:
  (roots+=DefRoot)*;

DefRoot:
  StatechartRoot | StateRoot | TransitionRoot;

Scope returns sct::Scope:
(SimpleScope | StatechartScope);
// a SimpleScope is used for states and regions
SimpleScope returns sct::Scope:
(SimpleScope | InternalScope);
// defines the possible scopes for statecharts
StatechartScope returns sct::Scope:
(InterfaceScope | InternalScope);

references
extends
YAKINDU SCT Approach

Domain-Specific

HMI Meta Model

Generic

Structural Concepts (SGraph)

Domain-Specific Statechart

Declarations & Expressions (SText)

HMI Declarations

This image illustrates the YAKINDU SCT approach, which involves domain-specific specialization. The diagram shows how domain-specific concepts, such as HMI meta models and structural concepts, are integrated into a generic framework. The approach emphasizes the use of domain-specific declarations within a larger statechart context, facilitating a seamless integration of specific requirements into a general model. The structural concepts (SGraph) provide a foundation for defining and extending both generic and domain-specific elements, ensuring flexibility and reusability in software development.
Yakindu SCT

• Open Source / EPL
• Hosted at EclipseLabs
• SCT Eclipse-Proposals planned for 2012
  • Damos already sumitted
  • 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?