Yakindu HMI

Model-driven HMI Development

Axel Terfloth
itemis AG

HMI = Human Machine Interface

≈

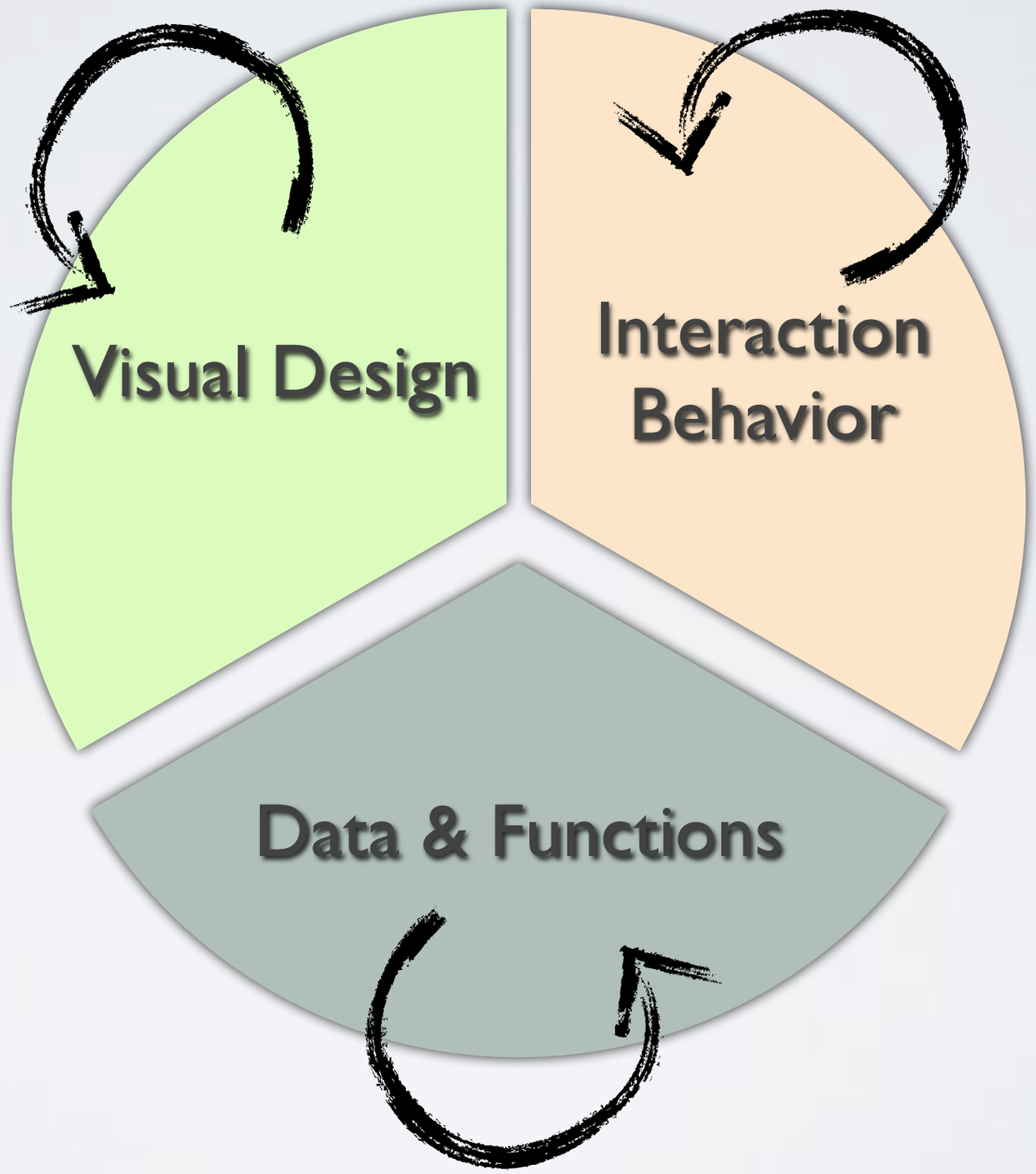
User Interface = UI

Anatomy of an HMI

UI-Technologies

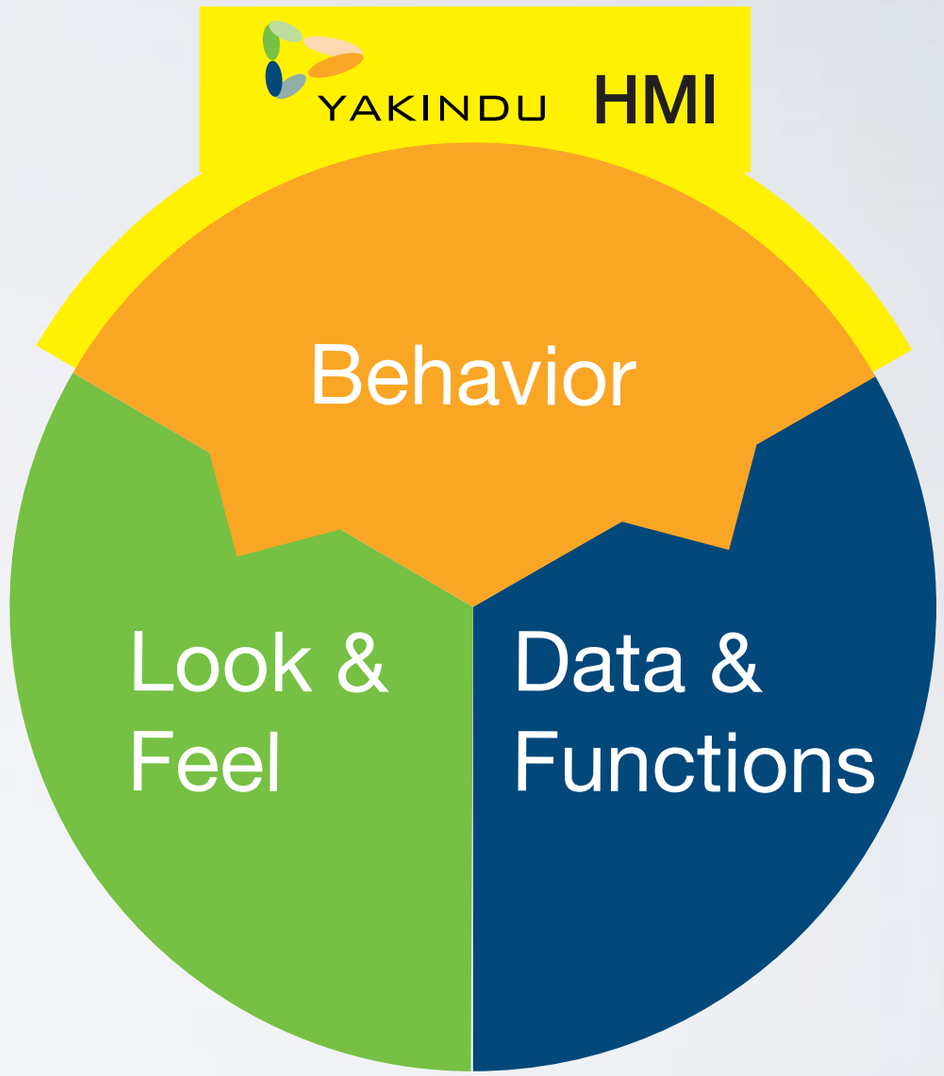
FUJITSU

CGI Studio



YAKINDU HMI

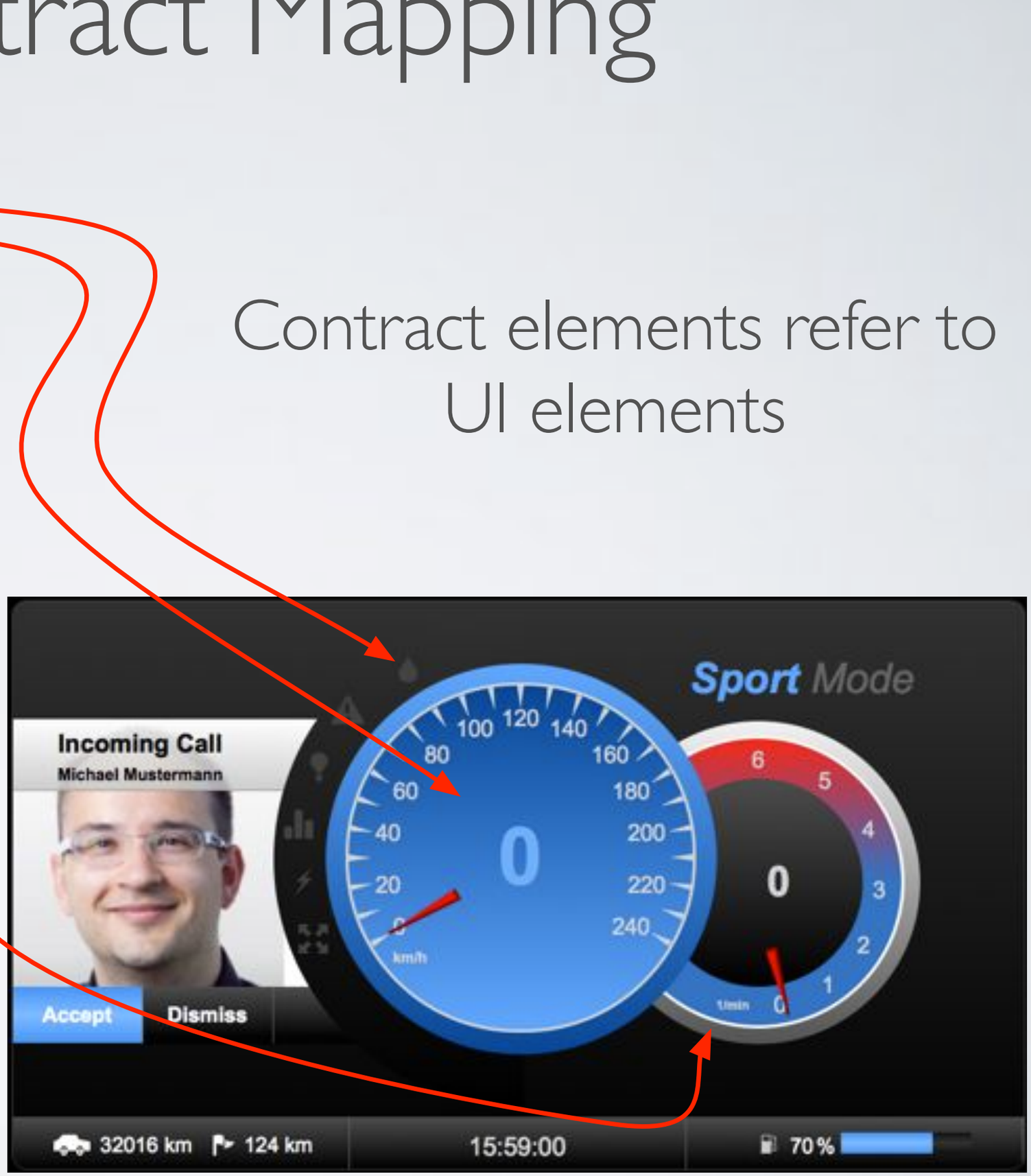
- focusses on behavior
  - makes use of state charts
- integrates with „visualization models“
- modular toolchain
- based on Eclipse technologies



Contract Mapping

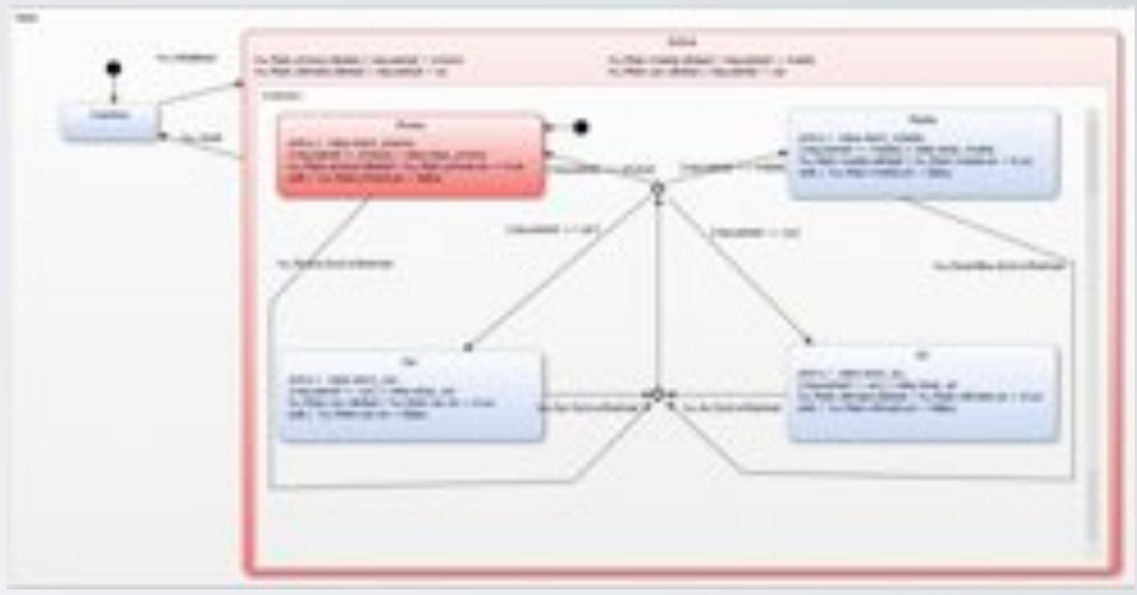
```
app YCluster  {
        scene main {
                ControlLight: oilPressure
                Speedometer: meter
                InfoArea: infoarea
                FuelIndicator: fuel
        }

        scene sport {
                animation intro
                animation outro
                Revmeter: revmeter
        }

        scene eco { ... }

        menu options {
                Item: eco
                Item: sport
                Item: tripInfo
                Item: radio
        }
}
```

Contract elements refer to UI elements



HMI Visualization

HMI Behavior



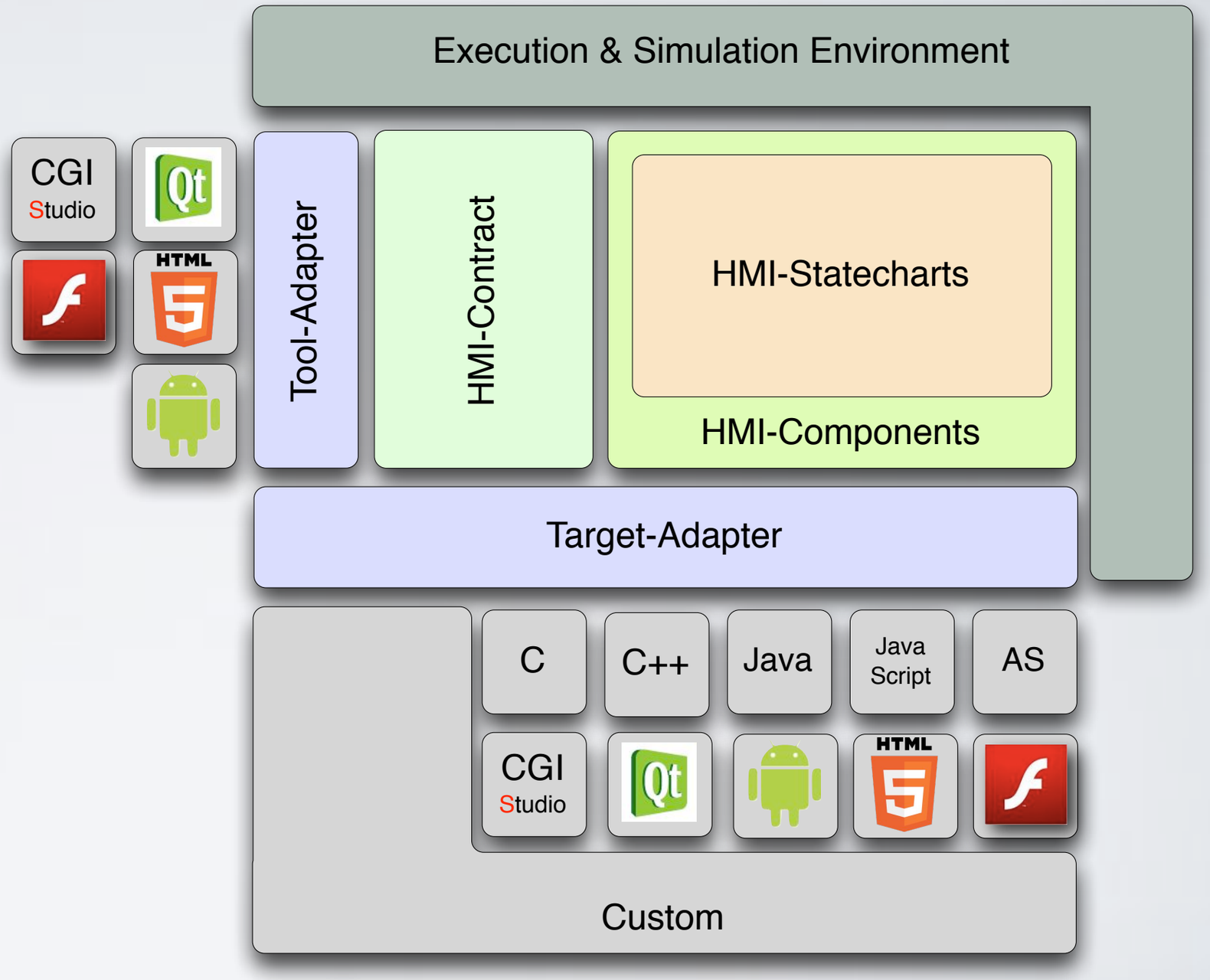
HMI Contract

```
app YCluster  {
       scene main {
              ControlLight: oilPressure
              Speedometer: meter
              InfoArea: infoarea
              FuelIndicator: fuel
       }

       scene sport {
              animation intro
              animation outro
              Revmeter: revmeter
       }
}
```

YAKINDU HMI Components

tool adapters
care about
contract
integration

target adapters
generate code

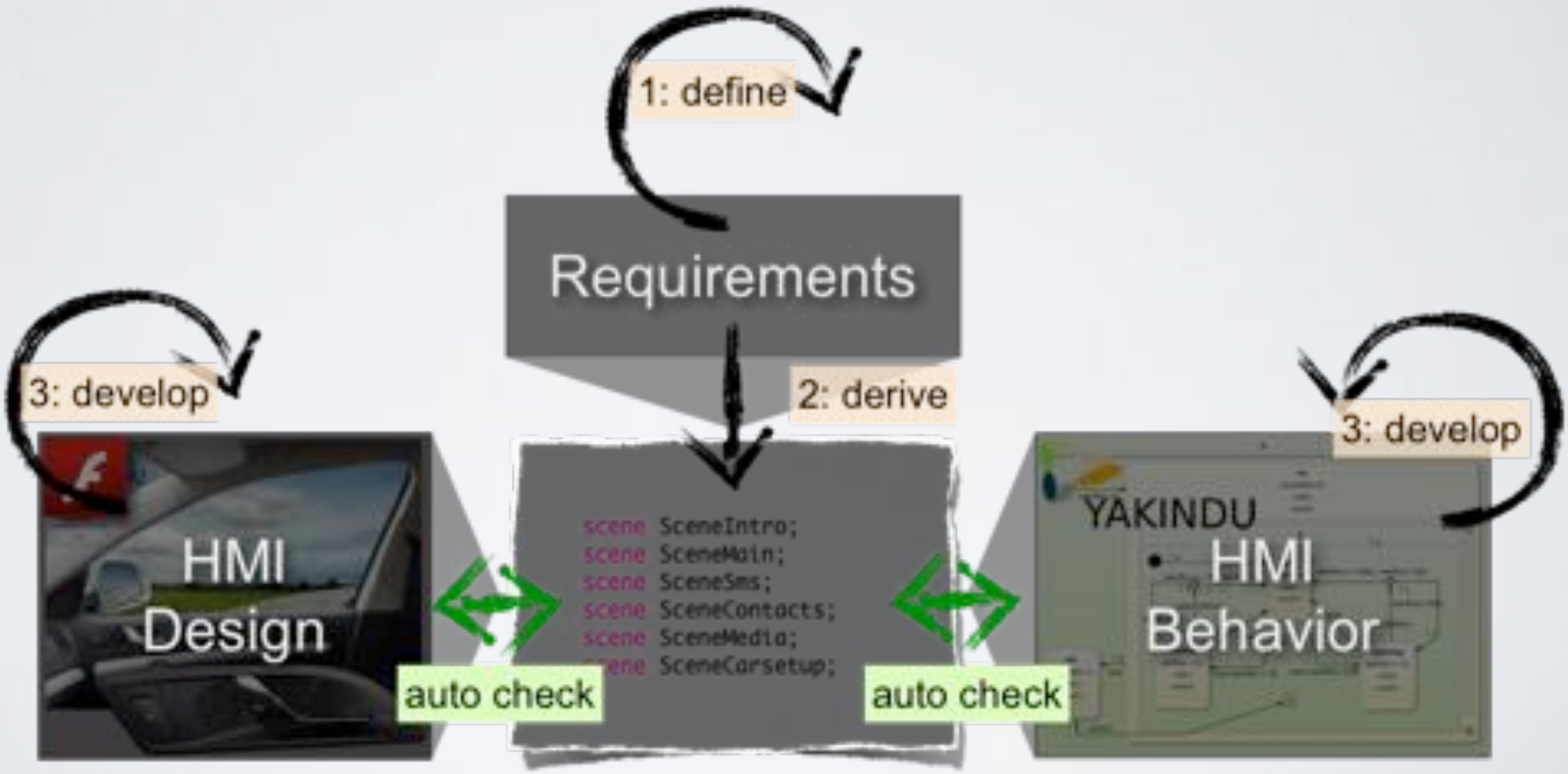


HMI Contract

decouples different system engineering activities

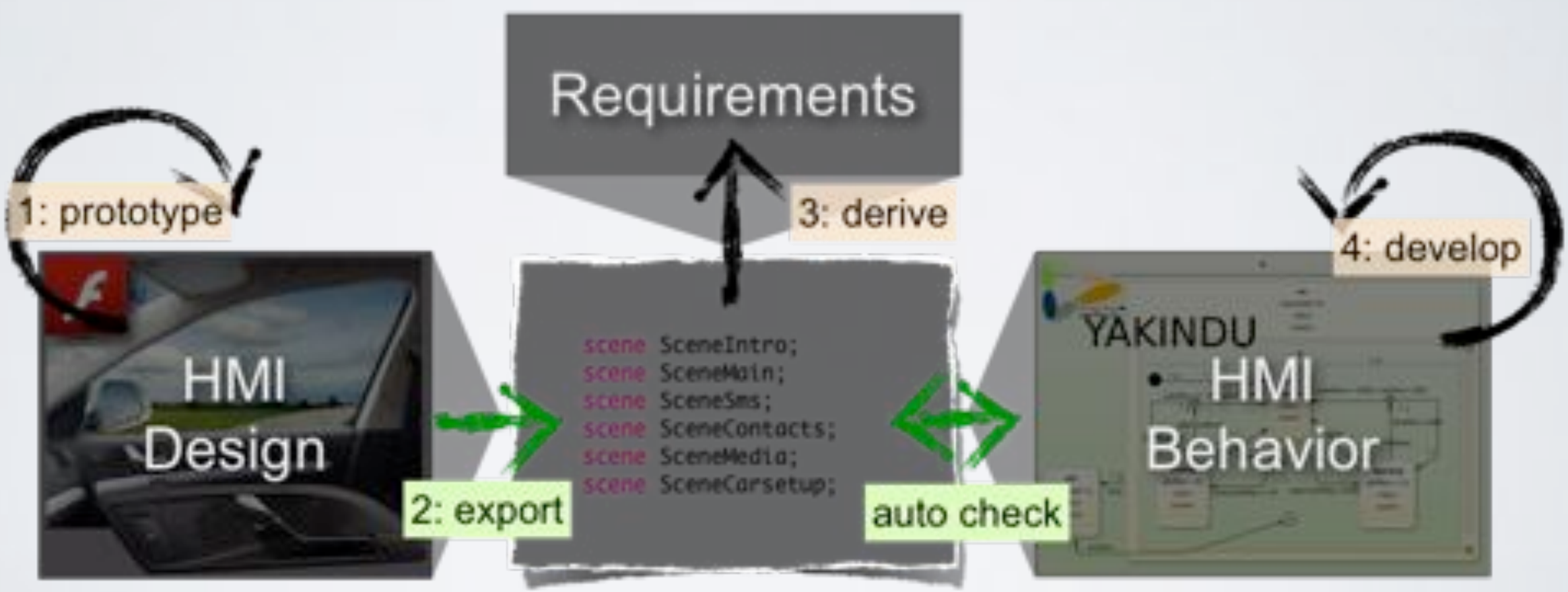
HMI Contract

in a requirements driven process



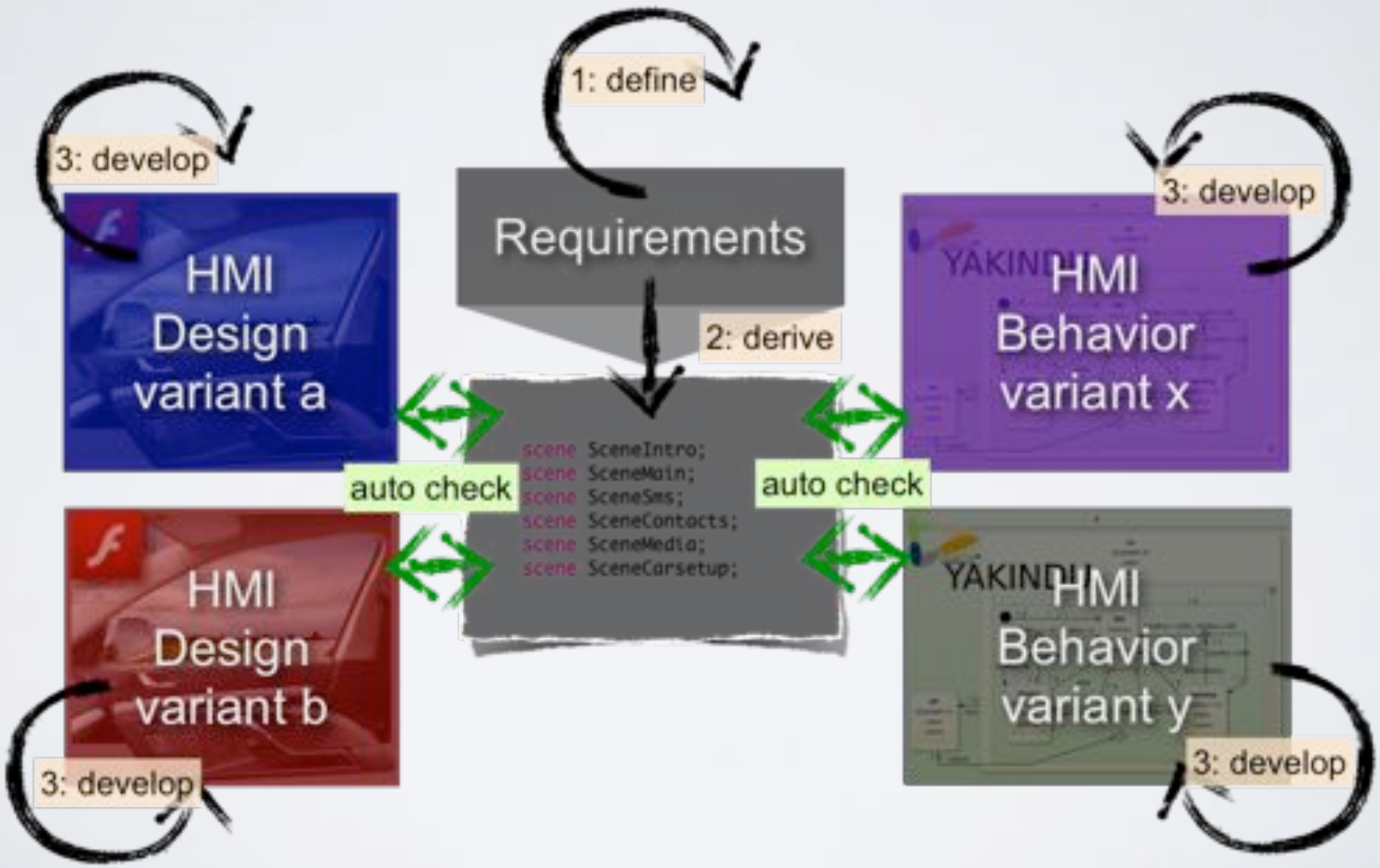
HMI Contract

in a design driven process



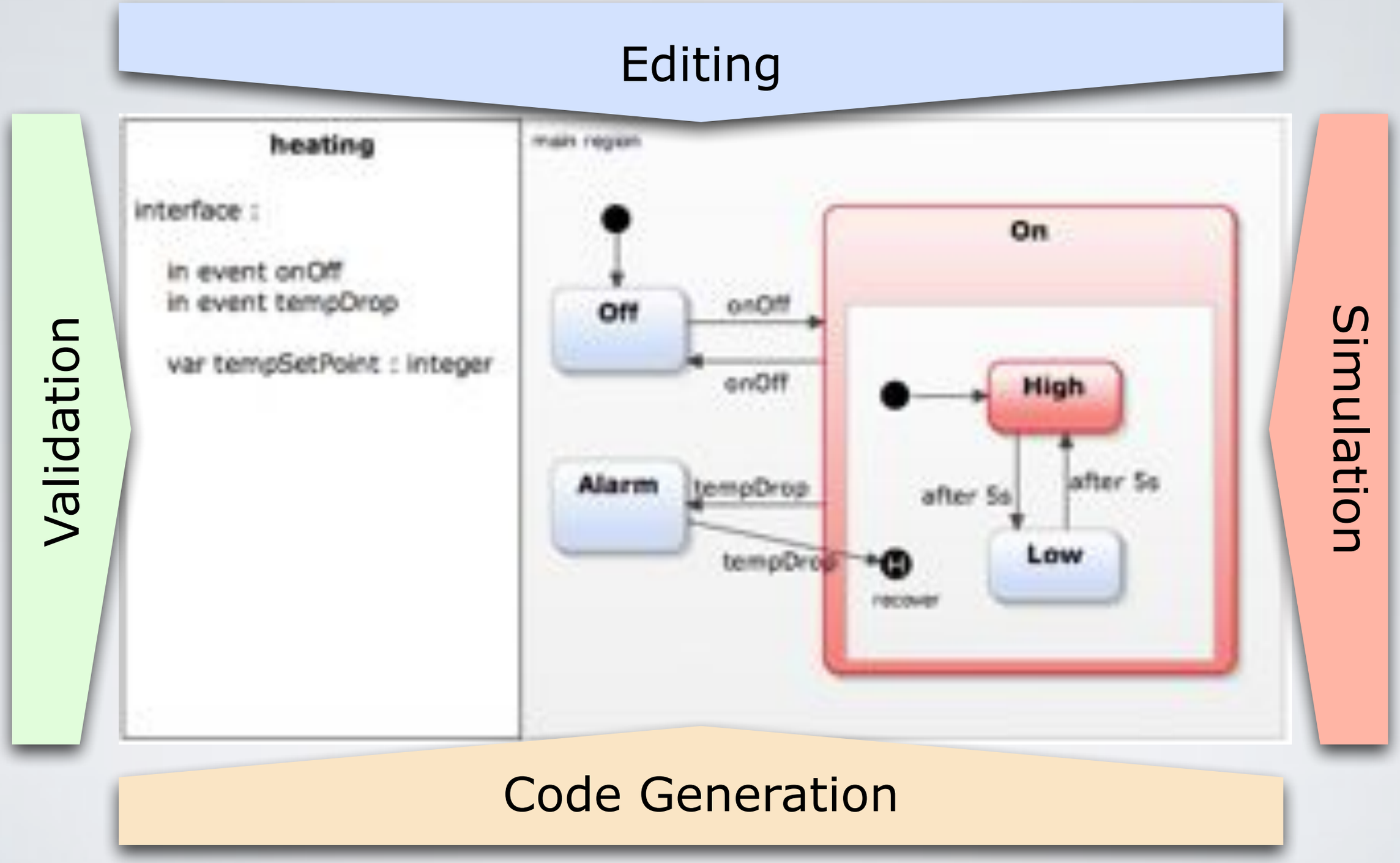
HMI Contract

support variants



Thank You! Questions?

YAKINDU Statechart Tools (SCT)



The Statechart Application Gap

State-based modeling
is useful
in many domains

Typically, statecharts
are independent
of any domain

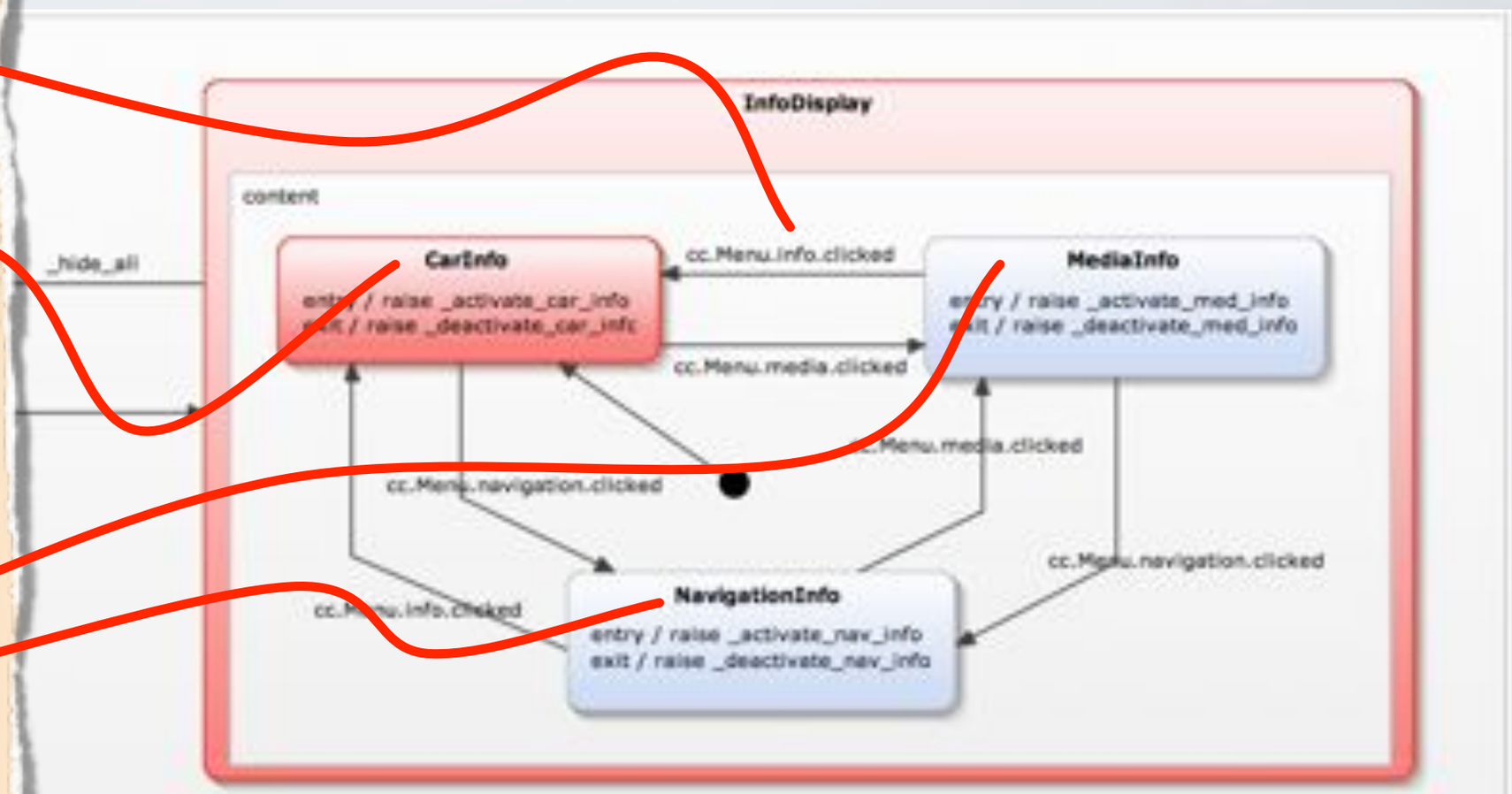
- How can statecharts be adopted to different domains?
- How can tools support this adoption?

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,...

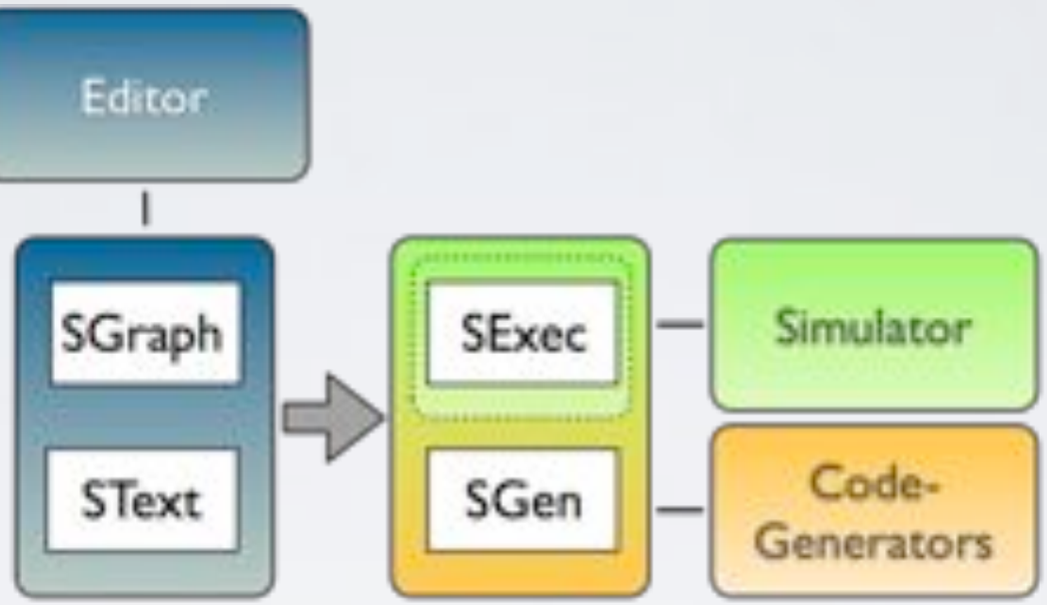
Integration of HMI Concepts

```
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

- 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

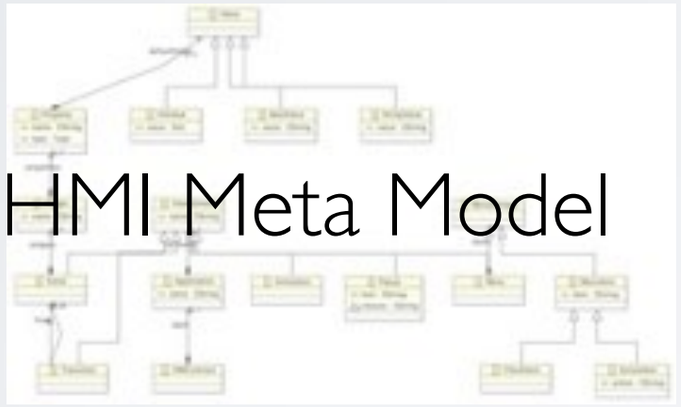
Built-In Extensibility

- 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

YAKINDU SCT Approach

specialization

Domain-Specific



HMI Meta Model

references

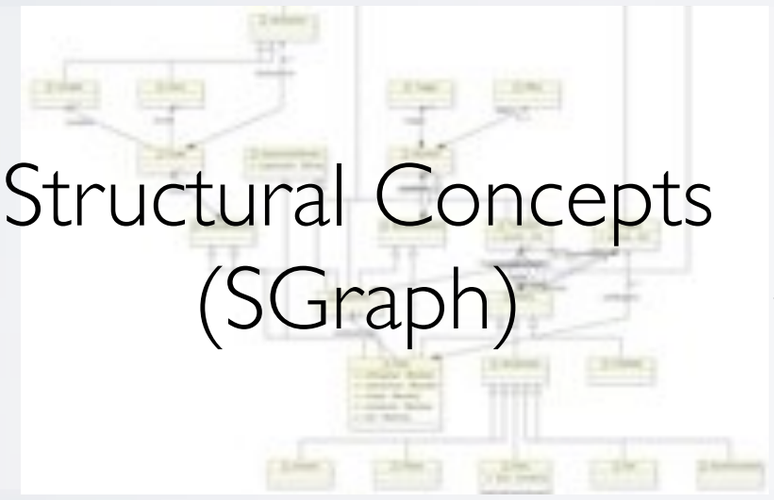
```
grammar com.yakindu.hmi.sctmodel.HMIText with  org.yakindu.sct.model.stext.SText

/* ---- root rules ----
These root rules are not relevant for the
into a single grammar.
*/
Root:
        (roots+=DefRoot)*;
Declaration returns sct::Declaration:
        EventDefinition | VariableDefinition | Clock | Operation
        | LocalReaction | Entrypoint | Exitpoint | HMIDeclaration;

HMIDeclaration:
        HmiScene | HmiPopup | HmiAnimation | HmiTransition;
HmiScene:
        'scene' scene=[contract::Scene|QID];
HmiPopup:
        'popup' popup=[contract::Popup|QID];
HmiAnimation:
        'animation' animation        ::Animation|QID];
```

HMI Declarations

extends

```
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} (declarations+=Declaration)*;
        // defines the poosible scopes for statecharts
StatechartScope returns sct::Scope:
        InterfaceScope | InternalScope;
InterfaceScope returns sct::Scope:
```

Declarations & Expressions (SText)



Structural Concepts (SGraph)

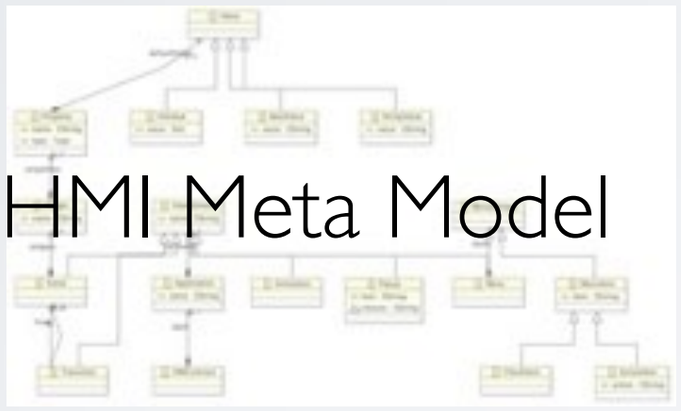
extends

Generic

YAKINDU SCT Approach

specialization

Domain-Specific



HMI Meta Model

references

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

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

Declaration returns sct::Declaration:
        EventDefinition | VariableDefinition | ClockI Operation
        | LocalReaction | Entrypoint | Exitpoint | HMIDeclaration;

HMIDeclaration:
        HmiScene | HmiPopup | HmiAnimation | HmiTransition;
HmiScene:
        'scene' scene=[contract::Scene|QID];
HmiPopup:
        'popup' popup=[contract::Popup|QID];
HmiAnimation:
        'animation' animation         ::Animation|QID];
```

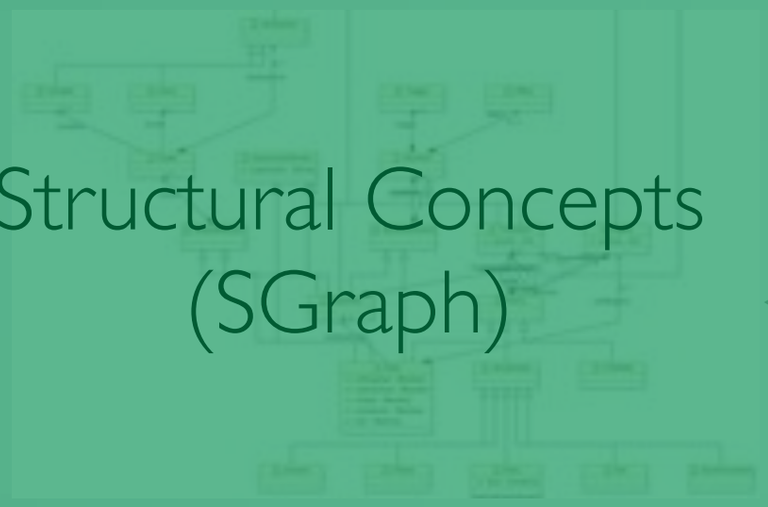
HMI Declarations

Domain Specific Statechart

```
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} (declarations+=Declaration)*;
        // defines the poosible scopes for statecharts
StatechartScope returns sct::Scope:
        InterfaceScope | InternalScope;
InterfaceScope returns sct::Scope:
```

Declarations & Expressions (SText)

extends



Structural Concepts (SGraph)

Generic

Yakindu SCT

- Open Source / EPL
- Hosted at EclipseLabs
- Eclipse-Proposal planned for 2013
  - Interested parties welcome!

- Important Links:
  - Project Site: http://statecharts.org
  - Eclipse Labs Site: http://code.google.com/a/eclipselabs.org/p/yakindu/

Thank You! Questions?