SCCharts in Motion
Interactive Model-Based Compilation for a Railway System

Christian Motika, Steven Smyth, and Reinhard von Hanxleden

Real-Time Systems and Embedded Systems Group
Department of Computer Science
Kiel University, Germany

SYNCHRON 2014
Aussois, 1 Dec. 2014
Reactive Embedded Systems

- Embedded systems often *safety-critical*
- *React* to inputs with computed outputs, *state based* computations
- Computations often exploit *concurrency*
  - Threads $\leadsto$ Non-Determinism
    - *Synchronous languages*: Lustre, Esterel, SCADE, SyncCharts
  - Sequentiality hard to model
    - *Sequentially Constructive Charts (SCCharts)*

**SCCharts well-suited for safety-critical systems**
Recall: Sequentially Constructive Charts – SCCharts

- André’s SyncCharts Syntax
- + Sequentially Constructive Semantics
- 1. Core features
- 2. Extended feat.
- Model transformations: Extended → ... → Core
SCCharts for Safety-Critical Systems

Language/semantics well-suited

... but that is not enough

- **Compiler** must be reliable
  (well structured, understandable, extensible, maintainable, ...)
- **Modeling**: Toolchain must facilitate building reliable models
  (abstraction mechanisms, support to understand language&models,
  simulations, optimizations, fine-tuning, ...)
- **Practicability**: Challenge real-life examples!

→ That’s what this talk is about!
Part I
Compiler

Original Model
Intermediate Models
Fully transformed Model

Part II
Modeling

Textual Modeling
Select Transformation

Part III
Practicability

Real-Life Evaluation

Single-Pass Language-Driven Incremental Compilation (SLIC)
Part I
Compiler

Part II
Modeling

Part III
Practicability

Single-Pass Language-Driven Incremental Compilation (SLIC)

Original Model → Intermediate Models → Fully transformed Model

Compiler

Textual Modeling

Select Transformation

Real-Life Evaluation

S → C → C

Init → Entry → ...

Implementation

Compilation Approach (SLIC)

Modeled Diagram

(Intermediate) Transformed Diagram / Code

Christian Motika

SCCharts in Motion

6 / 29
AO – A Simple SCChart

- Initially set $O$ to false
- Wait for input $A$ to become true
- Once $A$ is true:
  - Take transition $WA \rightarrow DA$
  - Set $O$ to true

Extended feature: Initialization
AO – Applying Transformations (→SYNCHRON ’13: ABRO)
AO – Applying Initialization Transformation

```
AO
input bool A
output bool O = false

[-] MainA
  WA
  ↓
  DA
A / O = true

AO
input bool A
output bool O
entry / O = false

[-] MainA
  WA
  ↓
  DA
A / O = true
```
AO – Applying Entry Transformation

AO
input bool A
output bool O
entry / O = false

[-] MainA

WA

A / O = true

DA

AO
input bool A
output bool O

[-] MainA

_Init

/ O = false

WA

A / O = true

DA
Initialization Transformation Implementation

```scala
def void transformInitialization(State state) {
  val initializedValuedObjects = state.valuedObjects.filter[initialValue != null]

  // Walk thru all initialized valuedObjects
  for (valuedObject : initializedValuedObjects) {
    // For every initialization: Create entry action
    val entryAction = state.createEntryAction

    // Copy the initial value to entry action assignment
    entryAction.addAssignment(valuedObject.assign(valuedObject.initialValue.copy))

    // Clear initialization (= no initialization any more)
    valuedObject.setInitialValue(null)
  }
}
```
SCCharts Extended Feature Compilation

- Sequence derived from dependencies: produces & not-handled-by
- *Single-Pass Language-Driven Incremental Compilation (SLIC)*
Single-Pass Language-Driven Incremental Compilation [ISOŁA’14]

- **Single-pass sequence** *derived* from dependencies produces & not-handled-by
- Requirement: No cycles
- Trade-off: More & simple ↔ less & complex
- SLIC Characteristic: *Intermediate results = valid models*

- Idea: Writing *simple* compiler, surprisingly also *very practical*
- *Discussion: Usable also for other languages/compilers?*
SCCharts Compilation - Advantages

Validation

- Each compilation step is simple → Understandable ✓
- Each transformation can be inspected/tested separately
- Intermediate results are valid models → Well structured ✓
- New extended features can be easily added → Extendable ✓
SCCharts Compiler Demo
Part I
Compiler

Part II
Modeling

Part III
Practicability

Single-Pass Language-Driven Incremental Compilation (SLIC)
Traditional Modeling & SW Synthesis User Story

1. User edits/draws model
2. Compiler parses model and synthesis code
3. User may inspect final artifacts

😊 Appropriate for advanced users
😊 But little guidance for beginners
😊 Compiler is black box
😊 Difficult for compiler writer
😊 Hardly allows to fine-tune and optimize the intermediate and/or resulting artifacts
😊 Hard to extend
SCCharts Modeling & Advantages

- View original and transformed model
  - Understanding language and models
  - Appropriate for advanced users and beginners
  - Facilitates validation for compiler writer
- View effects of intermediate transformations
  - Optimization & fine-tuning
SCCharts Interactive Modeling Details

Textual Modeling

Modeled Diagram

(Intermediate) Transformed Diagram / Code

Select Transformation

Compiler
SCCharts Modeling Demo
Part I
Compiler

Original Model → Intermediate Models → Fully transformed Model

Part II
Modeling

Textual Modeling → Select Transformation → Compiled Model

Part III
Practicability

Real-Life Evaluation

Single-Pass Language-Driven Incremental Compilation (SLIC)
Model Railway Project

- SCCharts student project (7 participants)
- Project size
  - States: 1,628 (modeled)
  - States: 135,000 (expanded)
  - Transition: 2,219 (modeled)
  - Transitions: 152,000 (expanded)
  - Concurrent Regions: 17,000 (expanded)
  - Generated C-Code: 650,000 lines
  - Compile time: 2-3 min, response time: <2ms

→ Medium-Size Example

http://rtsys.informatik.uni-kiel.de/confluence/display/SS14Railway

[from David Harel, Statecharts: A Visual Formalism for Complex Systems, 1984]
Project Results

- Improvements in efficiency, stability
  → **Maintainability ✓**
    - Compile Time (eAllContents)

- New extended features
  → **Extendability ✓**
    - Reference state expansion
    - Arrays
    - Hostcode function calls

- Results + **Evaluation Survey** → Technical Report
Survey – Language Evaluation

Deterministic Concurrency

Simplicity

Sequentiality

Language Preferences

perfect
good
ok
hardly usable

ecellent
good
bad
worst

Esterel  SyncCharts  SCCharts  Ptolemy  C  Java  Haskell

Esterel  SyncCharts  SCCharts  Ptolemy  C  Java  Haskell

Esterel  SyncCharts  SCCharts  Ptolemy  C  Java  Haskell

Esterel  SyncCharts  SCCharts  Ptolemy  C  Java  Haskell
Survey – Tooling Evaluation

Maintainability
- Excellent
- Good
- Bad
- Worst

Esterel SyncCharts SCCharts Ptolemy C Java Haskell

Debugging
- Excellent
- Good
- Bad
- Worst

Esterel SyncCharts SCCharts Ptolemy C Java Haskell

SCCharts Quality of Modeling
- Professional
- Advanced
- Ok
- Hardly usable

SCCharts Tooling Quality
- Professional
- Advanced
- Ok
- Hardly usable

Project Start Project End
Railway Project Contributors

- Karsten Rathlev
- Carsten Sprung
- Caroline Butschek
- Alexander Schulz-Rosengarten
- Niclas Flieger
- Nis Börge Wechselberg
- Stanislaw Nasin
Conclusions

- Model based compilation \((SLIC) \rightarrow \text{Reliable Compiler}\)
- Interactive modeling \(\rightarrow \text{Reliable Models}\)
- Practicability \(\rightarrow \text{Real-Life Models}\)

www.SCCharts.com
To Go Further

http://www.sccharts.com


SCCharts in Motion
Interactive Model-Based Compilation for a Railway System

Christian Motika, Steven Smyth, and Reinhard von Hanxleden

Real-Time Systems and Embedded Systems Group
Department of Computer Science
Kiel University, Germany

SYNCHRON 2014
Aussois, 1 Dec. 2014

That’s all folks! — Any questions or suggestions?
ABRO – The “Hello World” of the Synchronous World

- Initially set $O$ to $false$
- Concurrently wait for inputs $A$ and $B$ to become $true$
- Once both are $true$, take termination immediately and set $O$ to $true$
- Reset behavior with $R$
- Strong preempt emission of $O$ when $R$ is $true$

Extended features: (a) Strong Abort transition, (b) Entry action
ABRO – Applying Transformations

1. initialization

2. entry

AO
Sequentially Constructive MoC

- Natural sequencing prescribes deterministic scheduling
  - `stmt1; stmt2`
  - `trigger/effect`
- Only concurrent data dependencies matter
  - Sequential data dependencies do not lead to rejection
- Deterministic concurrent scheduling:
  Distinguish between relative and absolute writes
  - Absolute writes: `x = false`
  - Relative writes: `x = x | true`
  - Reads: `y = x`
  - (1) Absolute writes, (2) relative writes, (3) reads
- Sequentially Constructiveness fully subsumes
  Berry Constructiveness
Synchronous Program Classes

- Sequentially Constructive (S)
- Logically Correct (L)
- Pnueli-Shalev Constructive (P)
- Berry Constructive (B)
- Acyclic SC (A)

Static cycles
Dynamic scheduling
Sequences of values
Cycle of concurrent dependencies or concurrent writes
Out-of-order scheduling

Ineffective writes

Speculate on absence or presence
Speculate on absence

if (x) then y=z par
if (!x) then z=y

if (x) x=1

if (x) x=1
if (y) z=1 par
if (z) {y=1; z=1}

if (x and y) x=1

if (x) x=1 par
if (!x) x=1
if (!y) {x=1; y=1}

if (x) if (y) x=1
else {x=1; y=1}

if (x) y=1 par
if (y) {x=1; y=1}

if (x) y=1 par
if (y) (x=1; y=1)

if (x) y=1 par
if (!x) y=1 par
if (!y) x=1

if (x) then y=z par
if (y) y=1 par
if (z) {x=1; y=1}

if (y) (x=1; y=1)

if (y) x=1
if (!x) y=1 par
if (!y) x=1

if (y) (x=1; y=1)

if (y) x=1
if (!x) y=1
par
if (!y) {x=1; y=1}

if (y) (x=1; y=1)

if (y) x=1
if (!x) y=1
par
if (!y) {x=1; y=1}

if (y) (x=1; y=1)

if (y) x=1
if (!x) y=1
par
if (!y) {x=1; y=1}

if (y) (x=1; y=1)

if (y) x=1
if (!x) y=1
par
if (!y) {x=1; y=1}

if (y) (x=1; y=1)

if (y) x=1
if (!x) y=1
par
if (!y) {x=1; y=1}

if (y) (x=1; y=1)

if (y) x=1
if (!x) y=1
par
if (!y) {x=1; y=1}

if (y) (x=1; y=1)
SyncCharts

- **Statechart** dialect for specifying deterministic & robust concurrency
- SyncCharts:
  - Hierarchy, Concurrency, Broadcast
  - Synchrony Hypothesis
    1. Discrete ticks
    2. Computations: Zero time

[Charles André, Semantics of SyncCharts, 2003]

[Christian Motika, SCCharts in Motion]

[Christian Motika, SCCharts in Motion]
Causality in SyncCharts

- Rejected by SyncCharts compiler
- *Signal Coherence Rule*
- May seem awkward from SyncCharts perspective, but common paradigm
- Deterministic sequential execution possible using *Sequentially Constructive MoC* → *Sequentially Constructive Charts (SCCharts)*
Causality in SyncCharts (cont’d)

```
concurrent_causality
signal x
signal y

[·]
S1 !x / y S2
[-]
S3 !y / x S4
```
Concurrent with Threads

- Typical **observer pattern** implemented with Java Threads

```java
public class ValueHolder {
    private List listeners = new LinkedList();
    private int value;
    public interface Listener {
        public void valueChanged(int newValue);
    }
    public void addListener(Listener listener) {
        listeners.add(listener);
    }
    public void setValue(int newValue) {
        value = newValue;
        Iterator i = listeners.iterator();
        while(i.hasNext()) {
            ((Listener)i.next()).valueChanged(newValue);
        }
    }
}
```

E. A. Lee, The Problem with Threads, 2006

- Not thread safe! E.g., multiple threads call `setValue()`.
SCCharts Compilation Overview

- Extended feature compilation (1): *SLIC approach*
- Also further compilation:
  - Normalization (2), mapping to SCG (3), sequentialization, ...
Harel Wristwatch – Citizen Quartz Multi - Alarm III
Railway Installation
Track Layout
Project Overview - Controller Size

(taken from the final presentation of the railway project)
Tooling Evaluation - Compiler Performance

(taken from the final presentation of the railway project)