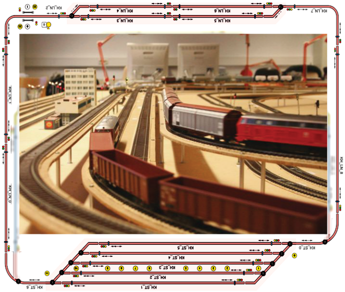
SCCharts in Motion

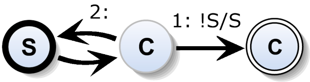
Interactive Model-Based Compilation for a Railway System

Christian Motika, Steven Smyth, and Reinhard von Hanxleden

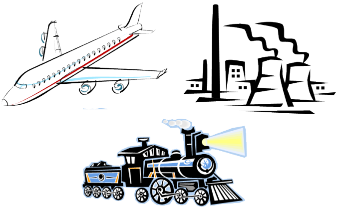
Real-Time Systems and Embedded Systems Group
Department of Computer Science
Kiel University, Germany

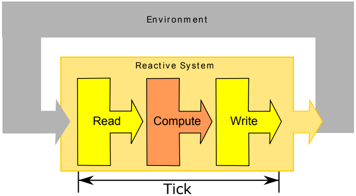


SYNCHRON 2014
Aussois, 1 Dec. 2014



Reactive Embedded Systems



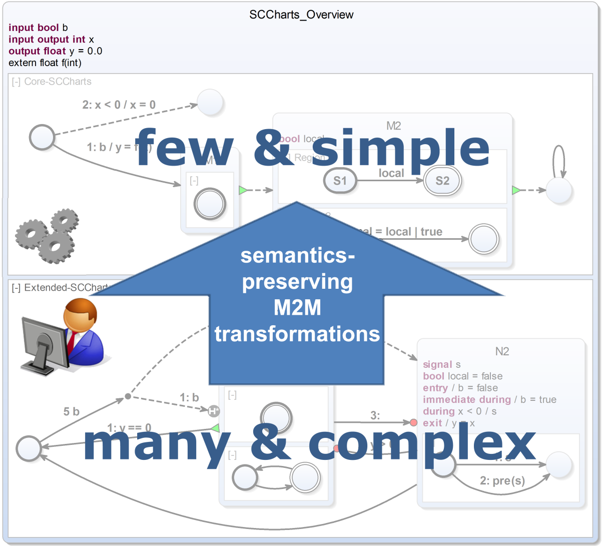


[SEUS 2013]

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

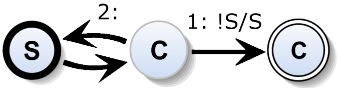
SCCharts well-suited for safety-critical systems

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



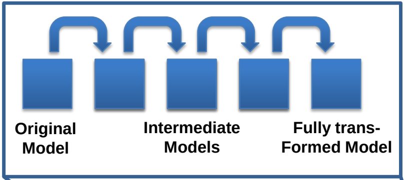
SCCharts well-suited for safety-critical systems

- ☺ 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!

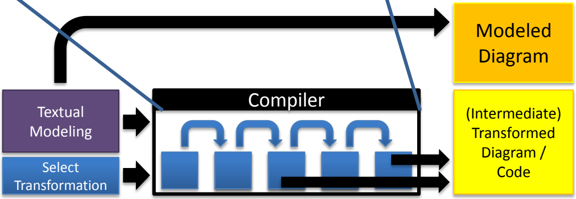
→ **That's what this talk is about!**

Single-Pass **L**anguage-Driven **I**ncremental **C**ompilation (SLIC)

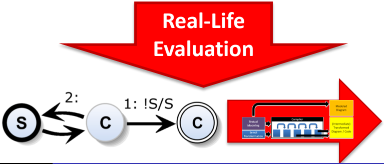
Part I
Compiler

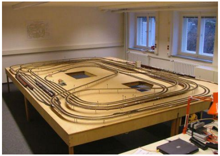


Part II
Modeling



Part III
Practicability

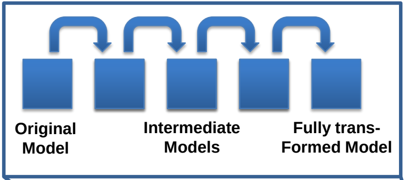




Single-Pass **L**anguage-Driven **I**ncremental **C**ompilation (SLIC)

Part I

Compiler



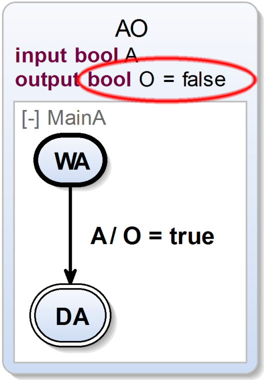
Part II

Modeling

Part III

Practicability

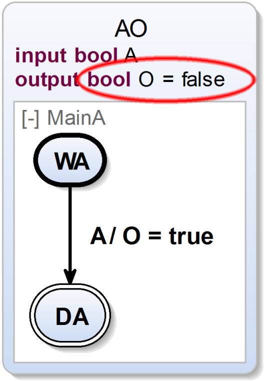
AO – A Simple SCChart

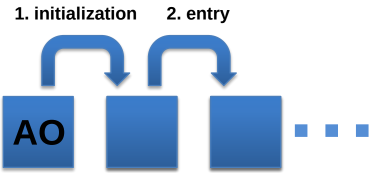


- Initially set *O* to *false*
- Wait for input *A* to become *true*
- Once *A* is *true*:
  - Take transition *WA* → *DA*
  - Set *O* to *true*

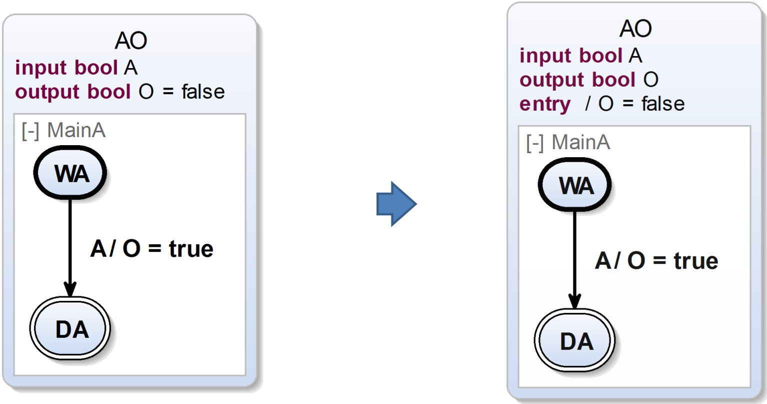
Extended feature: *Initialization*

AO – Applying Transformations *(→SYNCHRON '13: ABRO)*

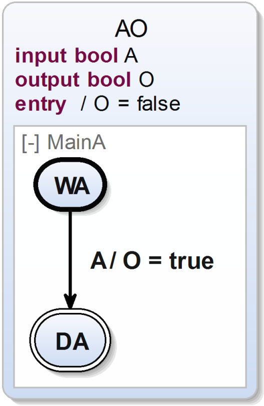


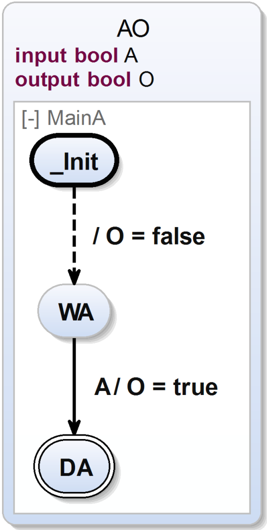


AO – Applying Initialization Transformation



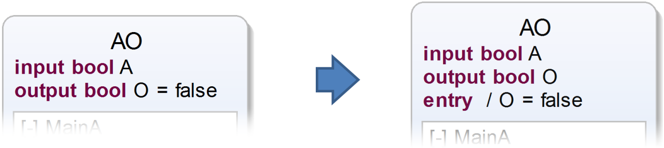
AO – Applying Entry Transformation



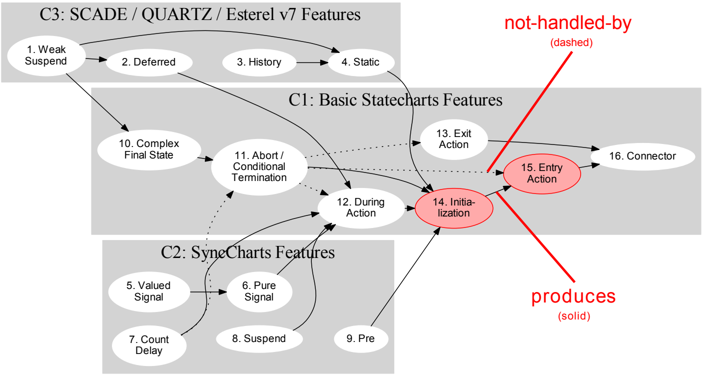


Initialization Transformation Implementation

```
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)
   }
}
```

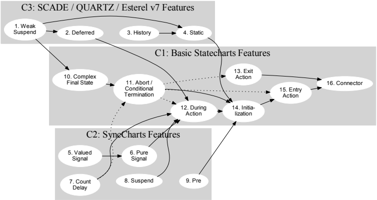


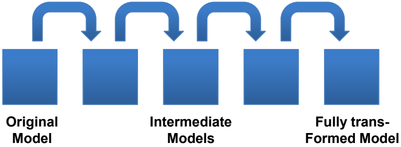
SCCharts Extended Feature Compilation



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

Single-Pass Language-Driven Incremental Compilation [ISoLA'14]

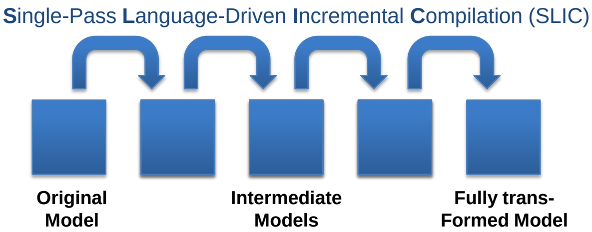




- **Single**-pass sequence *derived* from dependencies produces & not-handled-by
- Requirement: No cycles
- Trade-off: More & simple ↔ less & complex
- SLIC Characteristic: *Intermediate results = valid models*

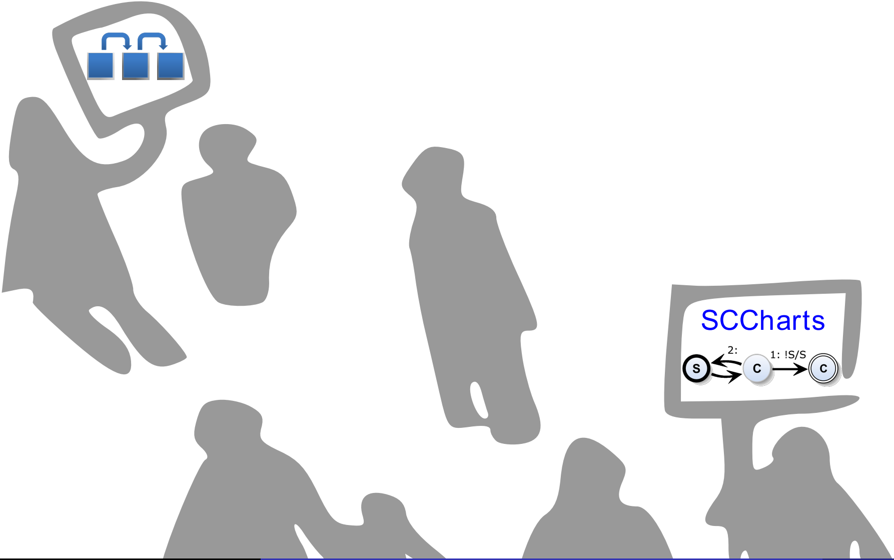
- Idea: Writing *simple* compiler, surprisingly also *very practical*
- *Discussion: Usable also for other languages/compilers?*

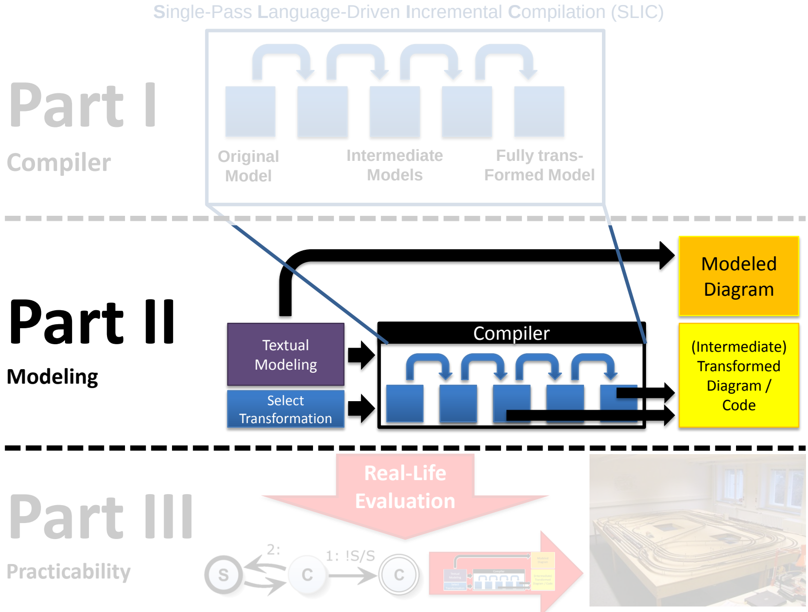
SCCharts Compilation - Advantages



- 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



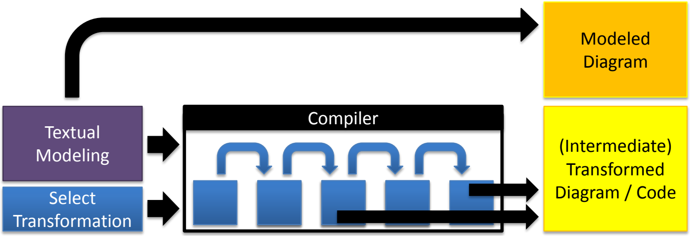

Single-Pass Language-Driven Incremental Compilation (SLIC)
Original Model
Intermediate Models
Fully trans-Formed Model
Part I
Compiler
Part II
Modeling
Textual Modeling
Select Transformation
Compiler
Modeled Diagram
(Intermediate) Transformed Diagram / Code
Part III
Practicability
Real-Life Evaluation
2:
1: !S/S
S
C
C

Traditional Modeling & SW Synthesis User Story

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

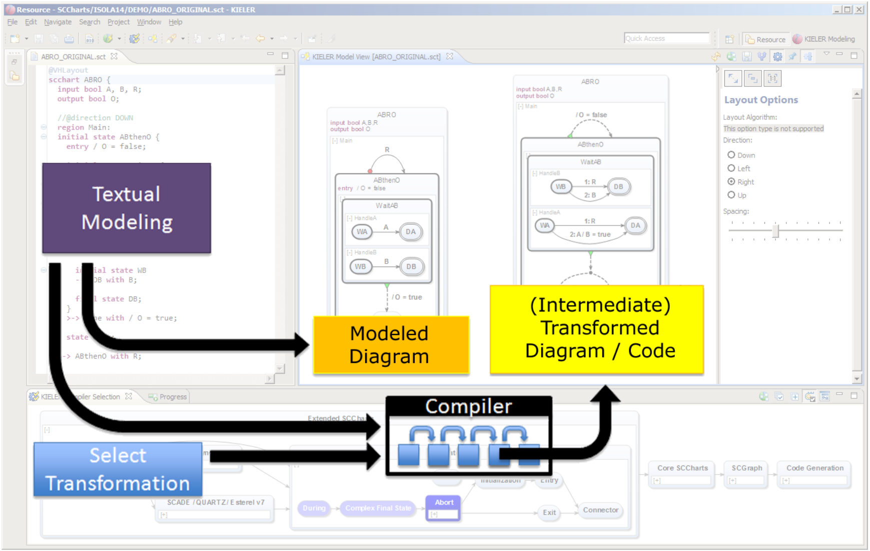
- ☺ Appropriate for advanced users
- ☹ But little guidance for beginners
- ☹ Compiler is black box
- ☹ Difficult for compiler writer
- ☹ Hardly allows to fine-tune and optimize the intermediate and/or resulting artifacts
- ☹ Hard to extend

SCCharts Modeling & Advantages

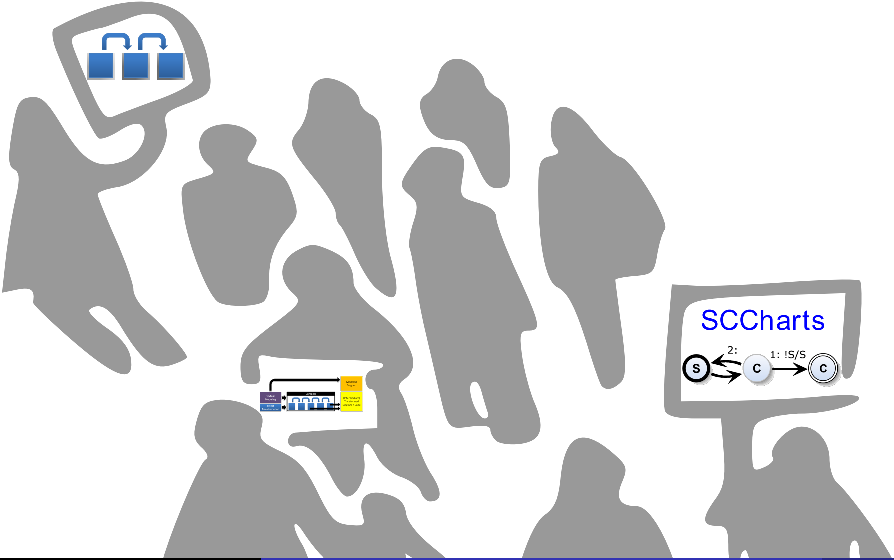


- View original and transformed model
  → *Understanding language and models* ✓
  - Appropriate for advanced users *and beginners*
  - Facilitates validation for *compiler writer*
- View effects of *intermediate* transformations
  → *Optimization & fine-tuning* ✓

SCCharts Interactive Modeling Details



SCCharts Modeling Demo

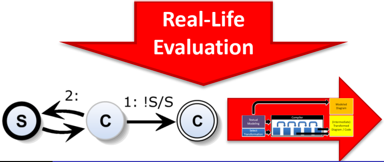


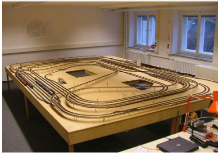
Single-Pass Language-Driven Incremental Compilation (SLIC)

Part I
Compiler

Part II
Modeling

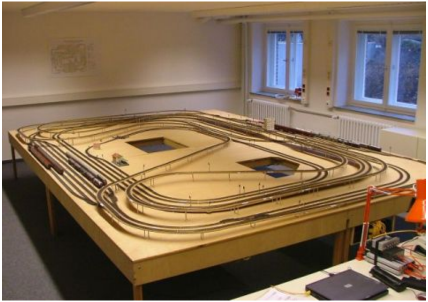
Part III
Practicability





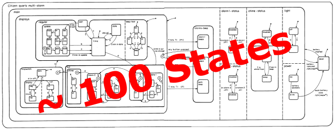
Model Railway Project

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



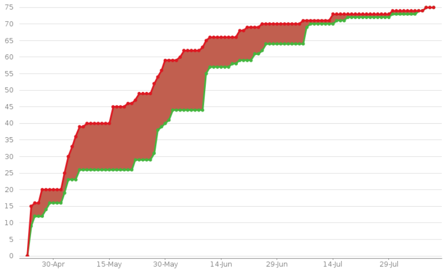
- SCCharts student project (7 participants)
- Project size
  - States: 1,628 (modeled)
    States: 135,000 (expanded)
  - Transition: 2,219 (modeled)
    Transitions: 152,000 (expanded)
  - Concurrent Regions: 17,000 (expanded)
  - Generated C-Code: 650,000 lines
  - Compile time: 2-3 min, response time: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

$\rightarrow$ *Medium-Size Example* ✓

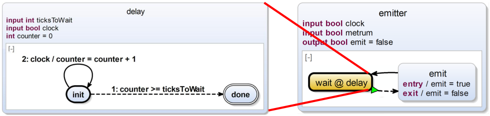


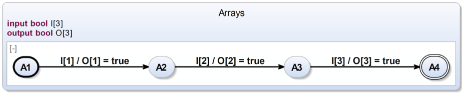
[from David Harel, Statecharts: A Visual Formalism for Complex Systems, 1984]

Project Results

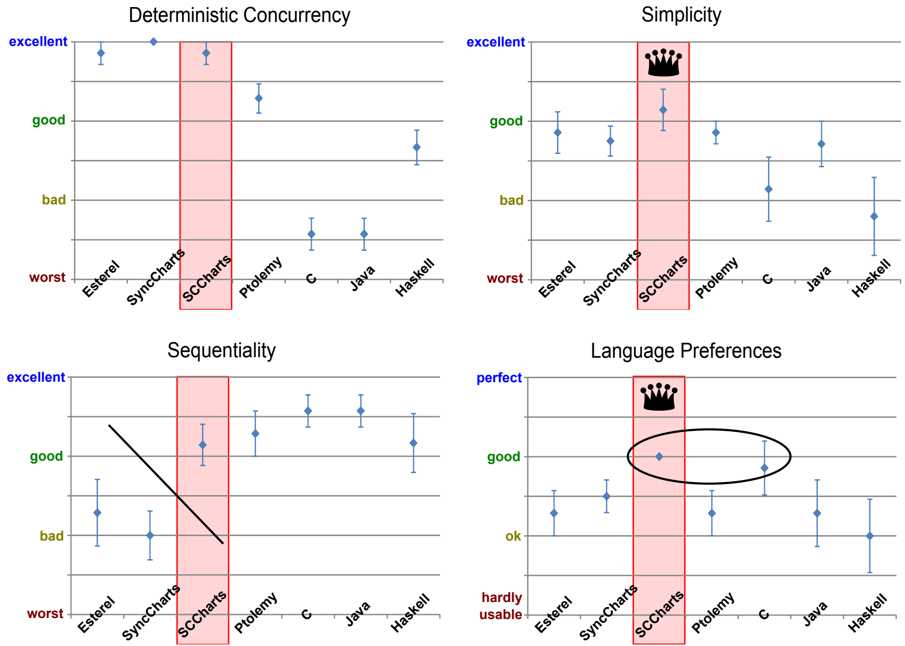


- Improvements in efficiency, stability
  → *Maintainability* ✓
  - Compile Time (`eAllContents`)
- New extended features
  → *Extendability* ✓
  - Reference state expansion
  - Arrays
  - Hostcode function calls
- Results + **Evaluation Survey** → Technical Report

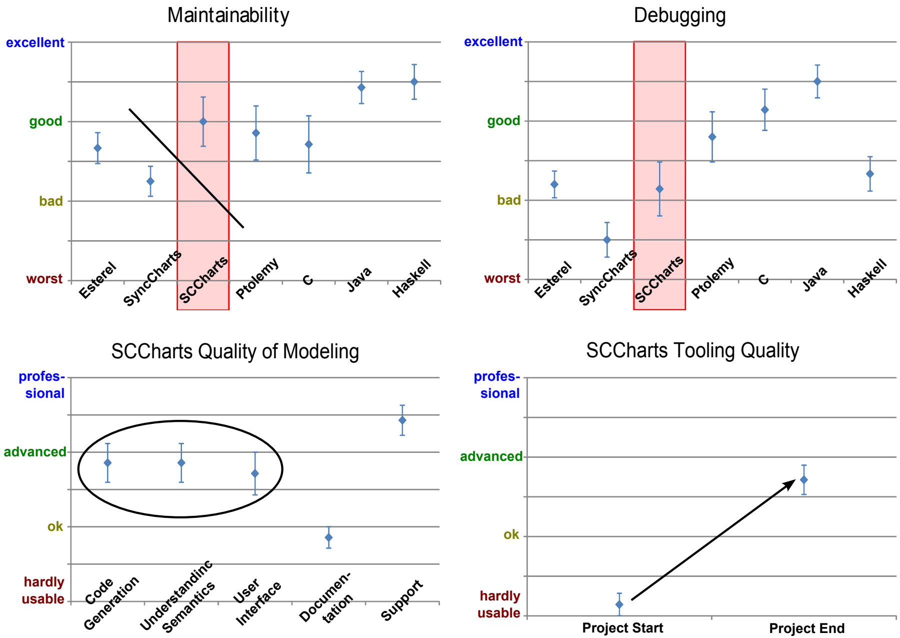




Survey – Language Evaluation



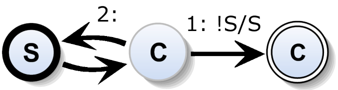
Survey – Tooling Evaluation



Railway Project Contributors

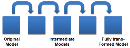
- Karsten Rathlev
- Carsten Sprung
- Caroline Butschek
- Alexander Schulz-Rosengarten
- Niclas Flieger
- Nis Börge Wechselberg
- Stanislaw Nasin

Conclusions

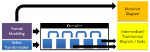


www.SCCharts.com

+ Model based compilation *(SLIC)* → *Reliable Compiler* ✓

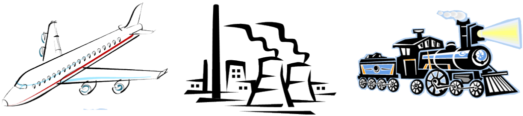


+ Interactive modeling → *Reliable Models* ✓



+ Practicability → *Real-Life Models* ✓

$\leftrightarrow$



To Go Further

- http://www.sccharts.com

- C. Motika, S. Smyth, and R. von Hanxleden. *Compiling SCCharts – A Case-Study on Interactive Model-Based Compilation*. 6th International Symposium On Leveraging Applications of Formal Methods, Verification (ISoLA'14), Corfu, Oct 2014.

- R. von Hanxleden, B. Duderstadt, C. Motika, S. Smyth, M. Mendler, J. Aguado, S. Mercer, and O. O'Brien. *SCCharts: Sequentially Constructive Statecharts for Safety-Critical Applications*. Proc. ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI'14), Edinburgh, Jun 2014.

- R. von Hanxleden, E. A. Lee, C. Motika, and H. Fuhrmann. *Multi-view modeling and pragmatics in 2020 – position paper on designing complex cyber-physical systems*. In Proceedings of the 17th International Monterey Workshop on Development, Operation and Management of Large-Scale Complex IT Systems, LNCS (Oxford, UK, Dec. 2012), vol. 7539.

- Charles André. *Semantics of SyncCharts*. 2003.

- Gérard Berry. *The Estrel v5 Language Primer*. 2000.

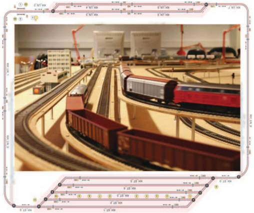
That's all folks! — Any questions or suggestions?

SCCharts in Motion

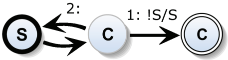
Interactive Model-Based Compilation for a Railway System

Christian Motika, Steven Smyth, and Reinhard von Hanxleden

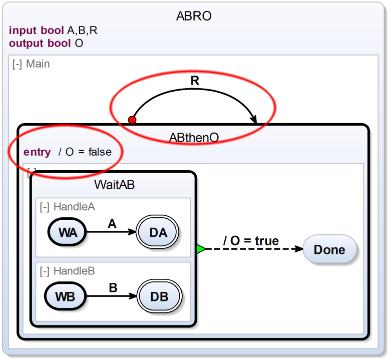
Real-Time Systems and Embedded Systems Group
Department of Computer Science
Kiel University, Germany



SYNCHRON 2014
Aussois, 1 Dec. 2014



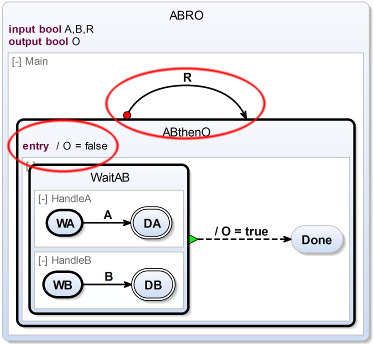
ABRO – The "Hello World" of the Synchronous World

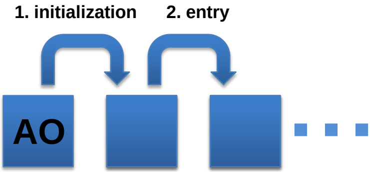


- Initially set *O* to *false*
- Concurrently wait for inputs *A* and *B* to become *true*
- Once both are *true*, take termination immediately and set *O* to *true*
- Reset behavior with *R*
- Strong preempt emission of *O* when *R* is *true*

Extended features: (a) Strong Abort transition, (b) Entry action

ABRO – Applying Transformations (→SYNCHRON '13)

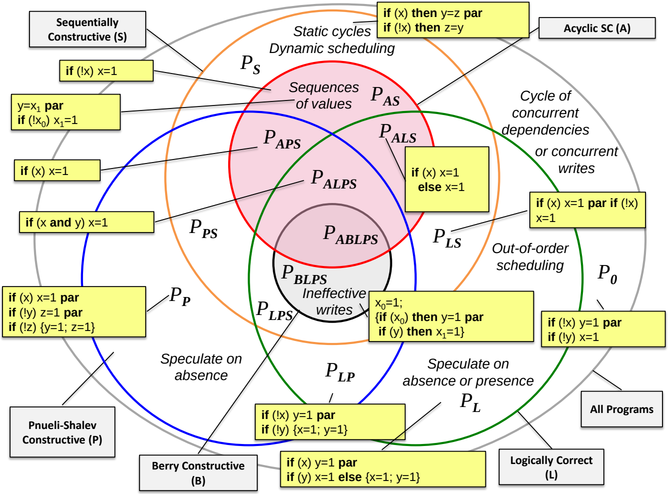




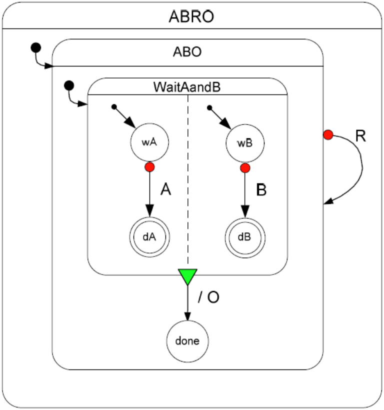
Sequentially Constructive MoC

- 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

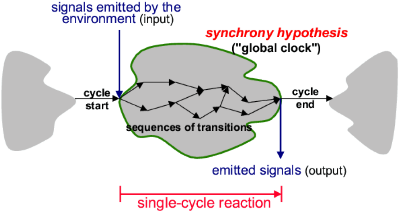


SyncCharts



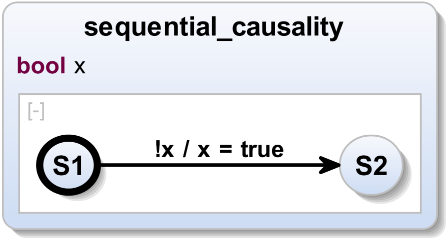
[Charles André, Semantics of SyncCharts, 2003]

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



[Gerald Lüttgen, 2001]

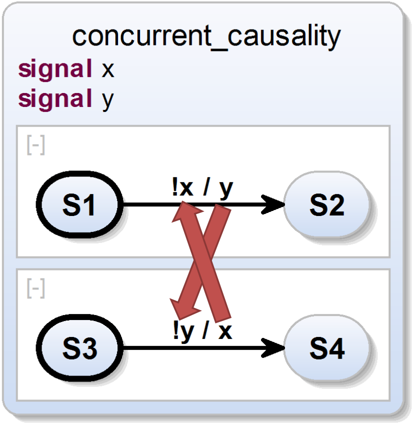
Causality in SyncCharts




```
if (!done) {
   ...
   done = true;
}
```

- Rejected by SyncCharts compiler
- *Signal Coherence Rule*
- May seem awkward from SyncCharts perspective, but common paradigm
- Deterministic sequential execution possible using *Sequentially Constructive MoC*
  $\rightarrow$ **Sequentially Constructive Charts (SCCharts)**

Causality in SyncCharts (cont'd)



Concurrency with Threads

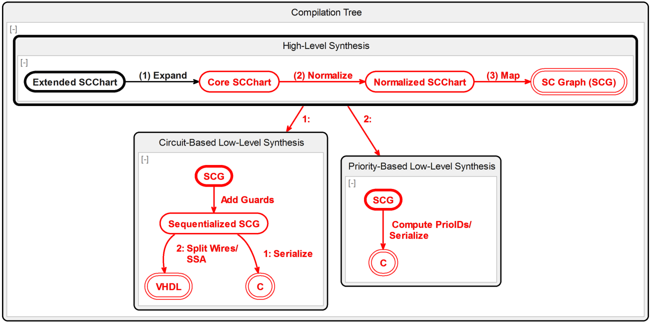
- Typical *observer pattern* implemented with Java Threads

```
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

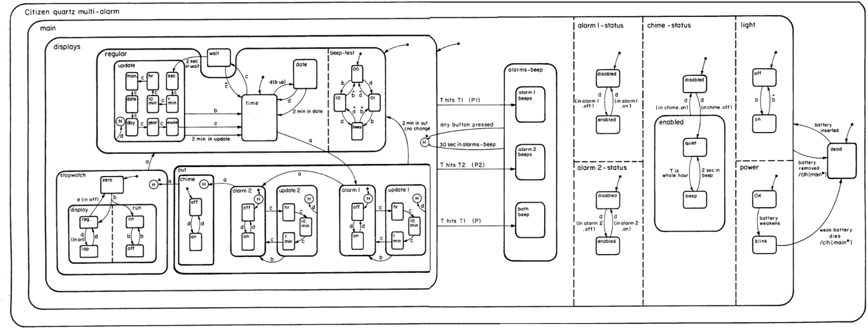
- Not thread safe! E.g., multiple threads call `setValue()`.

SCCharts Compilation Overview

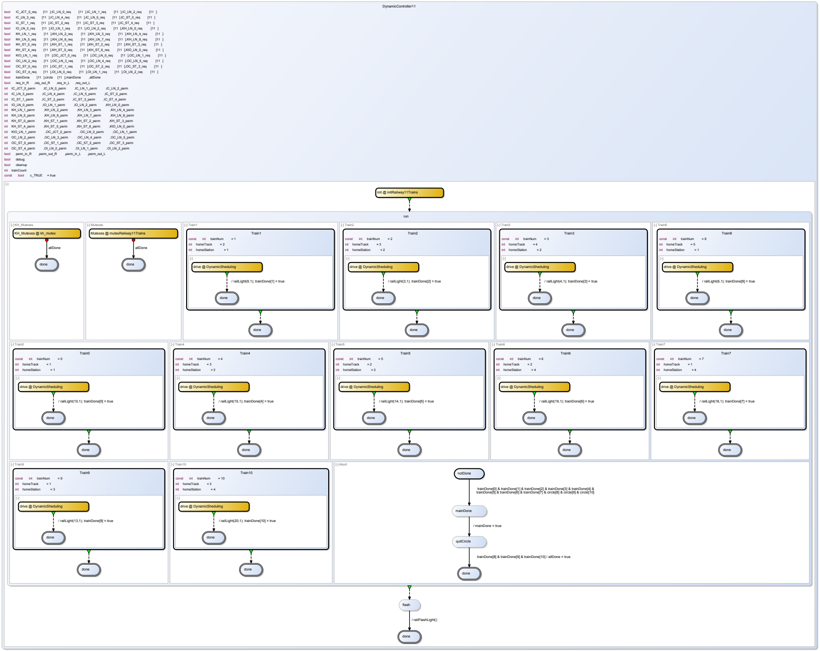


- Extended feature compilation (1): *SLIC approach*
- Also further compilation:
  - Normalization (2), mapping to SCG (3), sequentialization, ...

Harel Wristwatch – Citizen Quartz Multi - Alarm III

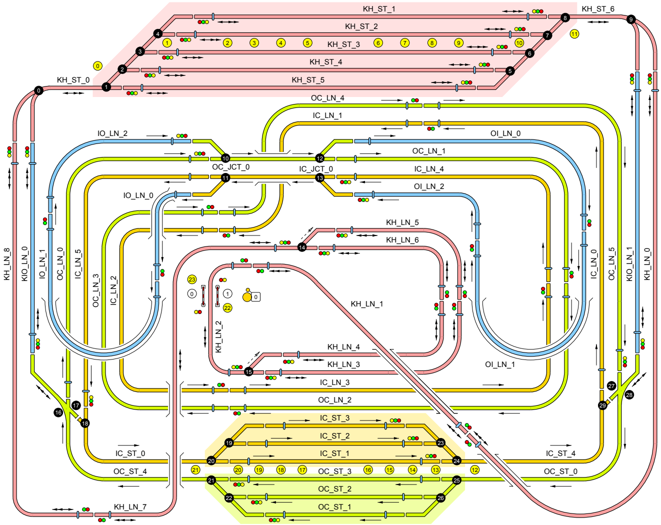


Railway Project – Dynamic Controller

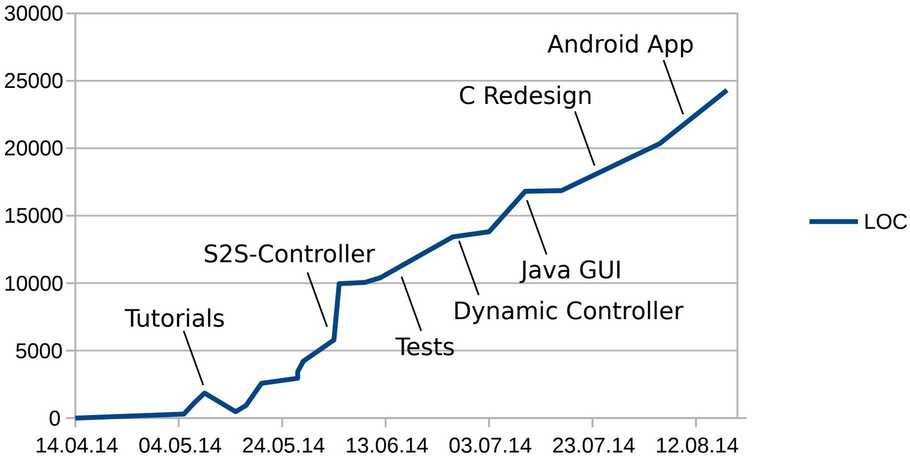


Railway Installation

Track Layout

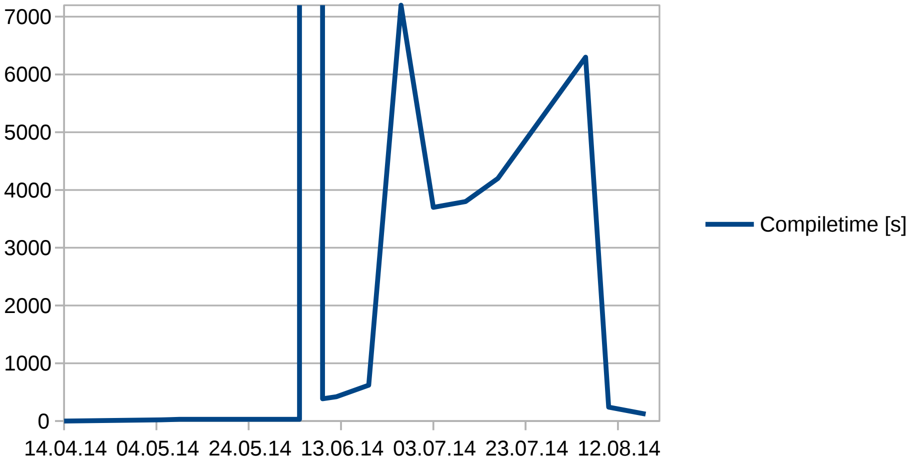


Project Overview - Controller Size



(taken from the final presentation of the railway project)

Tooling Evaluation - Compiler Performance



(taken from the final presentation of the railway project)