Component-based Modeling of Real-Time Systems

LASER 2005

Joseph Sifakis
VERIMAG
sifakis@imag.fr

In collaboration with
Gregor Goessler (INRIA)
Ananda Basu, Marius Bozga, Susanne Graf (Verimag)
Modeling real-time systems - motivation

Modeling plays a central role in systems engineering
• Can profitably replace experimentation on actual systems
• Can provide a basis for rigorous system development and implementation (model-based approaches).

Modeling real-time systems
• Raises hard problems about concepts, languages and their semantics e.g. What is an architecture? What is a scheduler? How synchronous and asynchronous systems are related?
• Requires a deep understanding of basic system design issues such as development methodologies (combination of techniques and tools, refinement) and architecture design principles.
Model-based Development

Move from physical prototypes to virtual prototypes (models) with obvious advantages: minimize costs, flexibility, genericity, formal validation is a possibility.

Modeling and validation environments for complex real-time systems

- Libraries of Components
  - ex. HW, SW, Models of continuous dynamic systems
- Languages and tools for assembling components

Synthesize embedded software from domain-specific models
  - ex. Matlab, SystemC, UML, SDL.
Modeling real-time systems - objectives

Provide a rigorous and general framework for modeling,

- Based on a general concept of architecture as a means to organize computation (behavior, interaction, control)
- Encompassing heterogeneous description, specific styles and paradigms, e.g.
  - synchronous and asynchronous execution
  - heterogeneous interaction (strong, weak, event-driven, state-driven)
  - architecture styles e.g. client-server, blackboard architecture
- Equipped with rules for correctness-by-construction wrt. generic properties such as deadlock-freedom, liveness, safety.
- Providing a basis for automated support for component integration and generation of glue code meeting given requirements
Overview

• Modeling real-time systems
  – The problem
  – Heterogeneity
  – Component-based construction

• Interaction Models
  – Definition
  – Composition
  – Examples
  – Deadlock-freedom preservation

• Timed systems
  – Definition
  – Examples
Overview (2)

• Scheduler modeling
  – The role of schedulers
  – Control invariants
  – Scheduler specifications
  – Composability results

• Timed systems with priorities
  – Definition
  – Composition of priorities
  – Correctness-by-construction results

• Tools and applications
  – The IF toolset
  – The BIP framework

• General Discussion
Thesis: A Timed Model of a RT system can be obtained by “composing” its application SW with timing constraints induced by both its execution and its external environment.
### Modeling real-time systems - our approach

<table>
<thead>
<tr>
<th></th>
<th>Application SW</th>
<th>Timed model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DESCRIPTION</strong></td>
<td>Reactive machine (untimed)</td>
<td>Reactive machine + External Environment + Execution Platform</td>
</tr>
<tr>
<td><strong>TIME</strong></td>
<td>Reference to physical (external) time</td>
<td>Quantitative (internal) time Consistency pbs - timelocks</td>
</tr>
<tr>
<td><strong>TRIGGERING</strong></td>
<td>Timeouts to control waiting times</td>
<td>Timing constraints on interactions</td>
</tr>
</tbody>
</table>

#### Diagrams

- **Timeouts to control waiting times**
  - Reactive machine
    - **?e**
    - **TO(5)**
  - Timed model
    - **?e [0,6]**
    - **!e [0,4]**

**Actions**

- **No assumption about Execution Times Platform-independence**
- **Assumptions about Execution Times Platform-dependence**
Modeling real-time systems - our approach

1. Application SW
2. Platform Timed Model
3. Environment Timed Model
4. User Requirements
5. Composition/Synthesis
6. Code Generation
7. System Timed Model
8. Analysis
9. Implementation
10. Diagnostics

Component-based modeling
Modeling real-time systems – Taxys (1)

Environment

DSP

Esterel+C

Event handler

Deadline constraint

\[ t_{out} - t_{in} < D \]

Throughput constraint: no buffer overflow
Modeling real-time systems – Taxys (2)

- ESTEREL + C Data
  - SAXO-RT
  - C Code
  - Machine Description
  - SAXO
  - Target Machine executable code

- Event Handler Timed Model
- Timed Model
- IF/KRONOS
- Timing Diagnostics

- C2TimedC
  - Exec. Times

C Code

Target Machine executable code
Modeling real-time systems – Taxys(3)

Application = ESTEREL + Pragmas

Event Handler

Environment = ESTEREL + Pragmas

Instrumented C Code

SAXO-RT

KRONOS Algorithms and Data Structures

Instrumented C Code

IF/KRONOS

Timing Diagnostics

Target Machine Executable Code

SAXO

Exec.Times

QoS requ.
Overview

• Modeling real-time systems
  – The problem
  – Heterogeneity
  – Component-based construction

• Interaction Models
  – Definition
  – Composition
  – Examples
  – Deadlock-freedom preservation

• Timed systems
  – Definition
  – Examples
Heterogeneity – Abstraction Levels

Model
(requirements)

Application Software

System

Execution Platform
Heterogeneity - from application SW to implementation

Application SW

Matlab/Simulink
Lustre
Esterel
ADA
SDL
RT- Java
UML

C
C++

Implementation

DSP
μcontroller

TTA

CORBA

CAN

RTOS

OSEK
Heterogeneity - from application SW to implementation

Functional properties - logical abstract time
High level structuring constructs and primitives
Simplifying synchrony assumptions wrt environment

Non functional properties, involving time and quantities
Task coordination, scheduling, resource management,
Execution times, interaction delays, latency
Heterogeneity - synchronous vs. asynchronous execution

**Application SW**

- **Synchronous**
  - Lustre, Esterel, Statecharts
  - Non interruptible execution steps
  - Usually, single task, single processor
  - «Everybody gets something »

- **Asynchronous**
  - ADA, SDL
  - Event triggered
  - Multi-tasking - RTOS
  - Usually, static Priorities
  - «Winner takes all »

**Implementation**
Heterogeneity - interaction

Interactions can be
- strict (CSP) or non strict (SDL, Esterel)
- atomic (CSP, Esterel) or non atomic (SDL)
- binary (point to point as in CCS, SDL) or n-ary in general
Heterogeneity - example

A: Atomic interaction

B: Blocking interaction

Asynchronous Computation

Synchronous Computation

Lotos
CSP

Java
UML

SDL
UML

Lotos
CSP

Matlab/Simulink
VHDL/SystemC
Statecharts

A B
nonA B
A nonB
nonA nonB
Overview

• Modeling real-time systems
  – The problem
  – Heterogeneity
  – Component-based construction

• Interaction Models
  – Definition
  – Composition
  – Examples
  – Deadlock-freedom preservation

• Timed systems
  – Definition
  – Examples
Component-based construction - components

Build systems by **composition of components**

**Components** are building blocks composed of **behavior** and **interface**

**Behavior** is a transition system

**Interface** hides irrelevant internal behavior and provides some adequate abstraction for composition and re-use, e.g. set of action names (ports) and associated variables
Component-based construction – formal framework

Pb: Build a component $C$ satisfying a given property $P$, from
• $C_0$ a set of atomic components
• $GL = \{gl_1, \ldots, gl_i, \ldots\}$ a set of glue operators on components

- Components are terms of an algebra of terms $(C, \cong)$
  generated from $C_0$ by using operators from $GL$
- $\cong$ is a congruence compatible with operational semantics
Component-based construction – formal framework

Glue operators transform sets of components into components

Glue operators
• model mechanisms used for communication and control such as protocols, controllers, buses
• restrict the behavior of their arguments, that is
  \( gl(C_1, C_2, \ldots, C_n) \mid A_1 \text{ refines } C_1 \)
Component-based construction - requirements

Examples of existing frameworks:
- Sequential functions with logical operators and delay operators for building circuits
- Process algebras
- Distributed algorithms define generic $gl$ for a given property $P$ e.g. token ring, clock synchronization …

Pb: Find a set of glue operators meeting the following requirements:
- Expressiveness (discussed later)
- Incremental description
- Correctness-by-construction
1. Decomposition of $gl$

$$\approx \begin{align*}
\text{gl}_1 & \approx \text{gl}_2 \\
C_1 & \approx C_1' \\
C_2 & \approx C_2' \\
& \cdots \\
C_n & \approx C_n'
\end{align*}$$

2. Flattening of terms

$$\approx \begin{align*}
\text{gl}_2 & \approx \text{gl}_1 \\
C_1 & \approx C_1' \\
C_2 & \approx C_2' \\
& \cdots \\
C_n & \approx C_n'
\end{align*}$$

Flattening can be achieved by introducing an idempotent operation $\oplus$ such that $(\text{GL}, \oplus)$ is a commutative monoid and

$$\text{gl}(\text{gl}'(C_1, C_2, \ldots, C_n)) \approx \text{gl} \oplus \text{gl}'(C_1, C_2, \ldots, C_n)$$
Component-based construction - Correctness by construction: compositionality

Build correct systems from correct components

We need compositionality results about preservation of progress properties such as deadlock-freedom and liveness.
Component-based construction - Correctness by construction: composability

Make the new without breaking the old

\[ g^l \cup g'^l \] implies \[ P \wedge P' \]

Property stability phenomena are poorly understood
- feature interaction
- non composability of scheduling algorithms
Component-based construction - compositionality vs. composability

Integration/compositionality vs. Layering/composability
Overview

• Modeling real-time systems
  – The problem
  – Heterogeneity
  – Component-based construction

• Interaction Models
  – Definition
  – Composition
  – Examples
  – Deadlock-freedom preservation

• Timed systems
  – Definition
  – Examples
Component-based modeling – The BIP framework

Layered component model

Scheduler: dynamic priority rules
Interaction Model: Connectors + Interactions

Composition (incremental description)
Interaction models

**Connectors** are maximal sets of compatible actions.

**Interactions** are subsets of connectors; they are defined by using typing: \((\text{complete} \bigtriangleup, \text{incomplete} \odot)\). Either they are maximal or they contain some complete interaction.

Interactions:
{tick1, tick2, tick3}, {out1}, {out1, in2}, {out1, in3}, {out1, in2, in3}
Interaction models - examples

1. CN:{cl1,cl2} MI: ∅

2. CN:{out,in} MI: {out}

3. CN:{in1,out,in} MI: {out}
Interaction models - definition

Given a set of atomic components $K$ with disjoint action vocabularies $A_i$ for $i \in K$,

- A **connector** $\gamma$ is a non empty subset of $\bigcup_{i \in K} A_i$ such that $|\gamma \cap A_i| \leq 1$
- **Interactions** are non empty subsets of connectors
- An **interaction model** $im$ is a pair $im=(\Gamma, \Delta)$ such that
  - $\Gamma$ is a set of non comparable connectors
  - $\Delta$ is a set of **minimal complete** interactions with $\forall \delta \in \Delta \exists \gamma \in \Gamma. \delta \subseteq \gamma$.

The interactions of $im = \Gamma \cup \{ \alpha | \exists \delta \in \Delta. \delta \subseteq \alpha \subseteq \gamma \}$
Interaction models – operational semantics

CN: \{put, get\}, \{prod\}, \{cons\}
MI: \{prod\}, \{cons\}
Interaction models - composition

\[CN[P,C]: \{\text{put, get}\}\]
\[MI[P,C]: \emptyset\]

\[CN[P]: \{\text{put}, \text{prod}\}\]
\[MI[P]: \{\text{prod}\}\]

\[CN[C]: \{\text{get}, \text{cons}\}\]
\[MI[C]: \{\text{cons}\}\]

\[CN: \{\text{put, get}, \text{prod}, \text{cons}\}\]
\[MI: \{\text{prod}, \text{cons}\}\]
Interaction models - composition

**IM**[K1,K2]:
CN[K1,K2] : \{a1, a2, a3, a4\}, \{a11, a12\}
MI[K1,K2] : \{a1, a2, a3, a4\}, \{a11\}

**IM**[K1]:
CN[K1] : \{a1, a2\}, \{a5, a9\}, \{a6, a9\}
MI[K1] : a5, a6, a11

**IM**[K2]:
CN[K2] : \{a3, a4\}, \{a7, a10\}, \{a8, a10\}
MI[K2] : a10
Interaction models – composition (2)

\[ IM[K_1, K_2]: \]
\[ CN[K_1, K_2] : \{a_1, a_2, a_3, a_4\}, \{a_{11}, a_{12}\} \]
\[ MI[K_1, K_2] : \{a_1, a_2, a_3, a_4, a_{11}\} \]

\[ IM[K_1]: \]
\[ CN[K_1] : \{a_1, a_2\}, \{a_5, a_9\}, \{a_6, a_9\} \]
\[ MI[K_1] : a_5, a_6, a_{11} \]

\[ IM[K_2]: \]
\[ CN[K_2] : \{a_3, a_4\}, \{a_7, a_{10}\}, \{a_8, a_{10}\} \]
\[ MI[K_2] : a_{10} \]

\[ IM[K_1 \cup K_2] = IM[K_1] \otimes IM[K_2] \otimes IM[K_1, K_2] \]
\[ CN[K_1 \cup K_2] = \max \ CN[K_1] \cup CN[K_2] \cup CN[K_1, K_2] \]
\[ MI[K_1 \cup K_2] = \min \ MI[K_1] \cup MI[K_2] \cup MI[K_1, K_2] \} \]
Incremental commutative composition encompassing blocking and non-blocking interaction
Interaction models - mod8 counter
Interaction models-mod8 counter(2)
Interaction models - commitment protocol

\[ CN : \{vote\} \cup \{vote_i\}_{i \in I}, \{commit\} \cup \{commit_i\}_{i \in I}, \{yes\} \cup \{yes_i\}_{i \in I} \]

\[ MI: \text{abort, no, no}_i \text{ for } i \in I \]
Interaction models - commitment protocol (2)

\[ CN : \{\text{vote}\} \cup \{\text{vote}_i\}_{i \in I}, \{\text{commit}\} \cup \{\text{commit}_i\}_{i \in I}, \{\text{yes, yes}_i\}_{i \in I} \text{ for } i \in I \]

\[ MI: \text{abort, no, no}_i, \text{abort}_i \text{ for } i \in I \]
Interaction models - checking for deadlock-freedom

For a given system (set of components + interaction model), its dependency graph is a bipartite labeled graph with:

Nodes $N = \text{Set of components} \cup \text{Set of minimal interactions}$

Edges $E$
- $(\alpha, a, k) \in E$ if $\alpha$ is an interaction, $a \in \alpha$ is an incomplete action of $k$
- $(k_1, a_1, \alpha) \in E$ if $a_1 \in \alpha$ is an action of $k_1$

Blocking condition for an incomplete action $a$:
$$Bl(a) = \text{en}(a) \land \neg (\text{en}(a_1) \land \text{en}(a_2) \land \text{en}(a_3))$$
Theorem 1: A system is deadlock-free if its atomic components have no deadlocks and its dependency graph has a backward closed subgraph such that for all its circuits $\omega$

$$Bl(\omega) = \bigwedge_{a \in \omega} Inc(\omega) \land Bl(a) = false$$

where $Inc(\omega) = \bigwedge_{k \in \omega} Inc(k)$ with $Inc(k)$ the set of the states of $k$ from which only incomplete actions can be executed.

Interaction models - checking for deadlock-freedom (2)
Interaction models - checking for deadlock-freedom: example
Interaction models - checking for deadlock-freedom: example

\[ \omega_1 = (\text{producer}, n_1, \text{consumer}_2, n_2) \quad B(\omega_1) = \text{false} \]

\[ \omega_2 = (\text{producer}, n_2, \text{consumer}_1, n_1) \quad B(\omega_2) = \text{false} \]

\[ \omega_3 = (\text{consumer}_1, n_1, \text{consumer}_2, n_2) \]

\[ B(\omega_3) = \text{Inc}(\omega_3) \land \text{en}(\text{get}_1) \land \neg (\text{en}(\text{get}_2) \land \text{en}(\text{put})) \]
\[ \land \text{en}(\text{get}_2) \land \neg (\text{en}(\text{get}_1) \land \text{en}(\text{put})) \]
\[ = \text{Inc}(\omega_3) \land \text{en}(\text{get}_1) \land \text{en}(\text{get}_2) \land \neg \text{en}(\text{put}) \]

Deadlock-freedom if \( \text{Inc}(\text{producer}) \land \neg \text{en}(\text{put}) = \text{false} \)
Interaction models - checking for individual deadlock-freedom

Definition: A component of a system is individually deadlock-free if it can always perform some action.

Theorem 2: Sufficient condition for individual deadlock-freedom of a component $k$

- $k$ belongs to a backward closed subgraph of a dependency graph satisfying conditions of Theorem 1;
- In any circuit of this subgraph, all its components are controllable with respect to their outputs i.e. it is always possible by executing complete interactions, to reach states enabling all the output actions of the component;
- All the n-ary interactions for $n>2$ are strong synchronizations.
Interaction models - discussion

• The distinction interaction model / behavior is crucial or the model construction methodology. Layered description => separation of concerns => associativity

• Different from other approaches e.g. process calculi, which combine behavior composition operators and restriction/hiding operators at the same level.

\[ ((P1||P2)a ||P3)a' \quad \rightarrow \quad \begin{array}{c} a \oplus a' \\ \hline P1||P2||P3 \end{array} \]

• Framework encompassing strict and non strict synchronization
Overview

• Modeling real-time systems
  – The problem
  – Heterogeneity
  – Component-based construction

• Interaction Models
  – Definition
  – Composition
  – Examples
  – Deadlock-freedom preservation

• Timed systems
  – Definition
  – Examples
Timed systems – from untimed to timed systems

Methodology:

• Avoid over-specification which may lead to inconsistency

• Make explicit all the consequences of the constraints on interactions

• Define $\parallel_T$ so as to preserve properties such as well-timedness, and deadlock-freedom
Untimed system: A set of transitions

\[ \begin{array}{c}
S \quad a, g, f \quad S'
\end{array} \]

where

- \( S \) is a finite set of control states
- \( A \) is a set of actions
- \( \rightarrow \subseteq S \times A \times S \), a transition relation
- \( X \) a set of variables

Each transition is labeled with a guard and a transfer function

Operational semantics: A set of transitions

\[ (s,x) \xrightarrow{a} (s',f(x)) \]

where \( x \) is a valuation of \( X \) such that \( g(x) = true \)
Timed Systems - definition

Timed system: A set of transitions

\[ \phi_s \overset{a, g, u, f}{\rightarrow} \phi_{s'} \]

where

• **u is an urgency condition** such that \( u \Rightarrow g \)
• Each control state \( s \) is labeled with a function \( \phi_s \) such that \( \phi_s(x,t) \) is the valuation of state variables when time progresses by \( t \) from state \( (s,x) \).

Informal semantics:
• Discrete transitions as for untimed systems
• Notion of time progress: time can progress at \( s \) only if the urgency conditions of the transitions issued from \( s \) are false
A periodic process of period $T > 0$ and execution time $E$, $E \leq T$. 

$t = T$, $t = T$, $t := 0$, $t \leq T - E$, $t = T - E$, $x := 0$, exec

$t' = x' = 1$ at all states
Timed Systems - definition

A state is a pair \((s, x)\) where \(x\) is a valuation of \(X\)

**Discrete Transitions**

\[(s, x) - a_i \rightarrow (s_i, f_i(x)/x) \quad \text{if} \quad g_i(x) = \text{true}\]

**Time steps**

\[(s, x) - t \rightarrow (s, \phi_s(x, t)) \quad \text{if} \quad \forall t' < t \quad t_p S(\phi_s(x, t')) \quad \text{where} \quad t_p S = \neg(\bigvee_i u_i)\]

*Time can progress as long as no urgency condition is true*
Timed Systems - relating urgency and time progress

3 ≤ x ≤ 5
x = 5
4 < y ≤ 7
4 < y ≤ 7

3 ≤ x ≤ 5
4 < y ≤ 7

tp = x ≠ 5 ∧ (y ≤ 4 ∨ y > 7)
Timed Systems – urgency types

\[ b = (a, g, u, f) \]

- \( g \): a may be executed
- \( u \): a must be executed

\[ u \Rightarrow g \]

**Invariant:** If a cannot be executed then time can progress at \( s \)

- eager \((\varepsilon)\)
- delayable \((\delta)\)
- lazy \((\lambda)\)
Timed Systems: Urgency types

Replace urgency conditions by *urgency types* preserved by restriction of guards

- $g^\lambda$: lazy guard \((u=false)\)
- $g^\varepsilon$: eager guard \((u=g)\)
- $g^\delta$: delayable guard \((u=g\downarrow)\)

Any TS can be transformed into an equivalent one with urgency types
A periodic process of period \( T > 0 \) and execution time \( E, \) \( (E \leq T). \)
Timed systems – guard restriction

\[ g' \Rightarrow g \quad u \Rightarrow u' \]

TS’ simulates TS
Timed systems – guard restriction

\[ u = g \]

\[ (g, u) \]

\[ (g', u') \]

\[ (u', u) \]

\[ (g'', u'') \]

maximal urgency \[ u = g \]

Laser 2005 - Summer School on Software Engineering
Timed Systems as transition systems

$Q$: set of states

$\rightarrow \subseteq Q \times A \times Q$  \hspace{0.5cm} $q \rightarrow a \rightarrow q'$  \hspace{0.5cm} untimed transition

$\rightarrow \subseteq Q \times R_+ \times Q$  \hspace{0.5cm} $q \rightarrow t \rightarrow q'$  \hspace{0.5cm} time step

Property (time additivity)

$q_1 \rightarrow t_1 \rightarrow q_2$ and $q_2 \rightarrow t_2 \rightarrow q_3$ implies $q_1 \rightarrow t_1 + t_2 \rightarrow q_3$

A run is a maximal sequence of transitions from states

$q_0 \ q_1 \ \ldots \ q_i \ \ldots$ such that $q_i \rightarrow a \rightarrow q_{i+1}$ or $q_i \rightarrow t \rightarrow q_{i+1}$

\[
\text{time} \ [q_0, q_i] = \sum_{k \leq i} t_k
\]

$q_0 \ q_1 \ \ldots \ q_i \ \ldots$ is time divergent if $\forall k \in \mathbb{N} \ \exists i \ \text{time} \ [q_0, q_i] > k$

Important: Well-timed systems (only time divergent runs !)
Timed systems as transition systems - discrete vs. continuous

a TIMEOUT[2]b : execute a within 2 otherwise execute b

time unit 1

dense time
Timed systems as transition systems - discrete vs. continuous

\[ a \ (bc \ \text{TIMEOUT}[1] \ AL2) \ \text{TIMEOUT}[1] \ AL1 \quad \text{for time unit 1} \]

possible abc within 0

\[ \text{possible abc within 0} \]

Laser 2005 - Summer School on Software Engineering
Timed systems as transition systems - discrete vs. continuous

\[ a \ (bc \ \text{TIMEOUT}[1] \ AL2) \ \text{TIMEOUT}[1] \ AL1 \] for time unit 0.25

Possible abc within 1.75
Timed systems as Transition Systems - discrete vs. continuous

\[ a \ (bc \ \text{TIMEOUT}[1] \ \text{AL2}) \ \text{TIMEOUT}[1] \ \text{AL1} \quad \text{for dense time} \]

possible abc within <2
Overview

- Modeling real-time systems
  - The problem
  - Heterogeneity
  - Component-based construction
- Interaction Models
  - Definition
  - Composition
  - Examples
  - Deadlock-freedom preservation
- Timed systems
  - Definition
  - Examples
Timed systems – from untimed to timed systems

Methodology:

- Avoid over-specification which may lead to inconsistency
- Make explicit all the consequences of the constraints on interactions
- Define $\parallel_T$ so as to preserve properties such as well-timedness, and deadlock-freedom
Scheduler modeling - example

A periodic process of period $T$ and completion time $C$

- **Actions**
  - $a$: arrive (u)
  - $b$: begin (c)
  - $f$: finish (u)
  - $p$: preempt (c)
  - $r$: resume (c)

$t' = x' = 1$ at all states except stop ($x' = 0$)
Overview (2)

• Scheduler modeling
   – The role of schedulers
   – Control invariants
   – Scheduler specifications
   – Composability results

• Timed systems with priorities
   – Definition
   – Composition of priorities
   – Correctness-by-construction results

• Tools and applications
   – The IF toolset
   – The BIP framework

• General Discussion
Scheduler modeling - the role of schedulers

A scheduler is a controller restricting access to resources by triggering controllable interactions so as to respect timing constraints (state predicates) $K_0 = K_{SCH} \land K_{POL}$

- $K_{SCH}$ scheduling constraints (timing constraints on processes)
- $K_{POL}$ scheduling policy

Scheduler for $K_{SCH} \land K_{POL}$

*controllable interaction*

*state*

Interactions

Processes

QoS requirem

Timed Model

Laser 2005 - Summer School on Software Engineering
A control invariant $K \Rightarrow K_0$

- Control invariants are preserved by uncontrollable actions
- It is possible to maintain the system in $K$ by executing controllable actions
Scheduler modeling - restriction by a constraint

The restriction of \( TS \) by a constraint \( K \) is a timed system \( TS/K \)

\[
\begin{align*}
TS \quad s1 \quad a^c \\
\quad g \\
\quad s2
\end{align*}
\]

\[
\begin{align*}
TS/K \quad s1 \quad a^c \\
\quad g \wedge K \wedge \text{pre}_{12}(K) \\
\quad s2
\end{align*}
\]

In \( TS/K \), \( K \) holds right before and right after the execution of any controllable action

If \( K \) is a control invariant of \( TS \) then \( TS/K \), is the scheduled (controlled) system
Scheduler modeling – controller synthesis

There exists a scheduler maintaining $K_0$ if there exists a non empty control invariant $K$, $K \Rightarrow K_0$

For given $K_0$, the maximal control invariant $K$, $K \Rightarrow K_0$ can be computed as the result of a synthesis semi-algorithm $\text{SYNTH}(TS, K_0) = \lim_{i} \{K_i\}$ where

$$K_{i+1} = K_{i+1} \land \text{contr-pre}(K_i)$$

from $K_0$

*All states from which TS can be led to $K_i$ no matter how the environment behaves*
Scheduler modeling - invariants vs. control invariants

Def: $K$ is an invariant of $TS$ if it is preserved by the transition relation ($TS \text{ sat } \text{inv}(K)$)

- Any invariant is a control invariant
- $K$ is a control invariant of $TS$ if $K$ is an invariant of $TS/K$, that is $TS/K \text{ sat } \text{inv}(K)$
- $TS^u \text{ sat } \text{inv}(K)$ implies $TS/K \text{ sat } \text{inv}(K)$
Scheduler modeling – composability of control invariants

- Are control invariants preserved by conjunction?
- Is it possible to apply a composition principle by computing control invariants?

**Def:** A control invariant $K_1$ of $TS$ is **composable** if for all constraints $K_2$, $K_1$ is a control invariant of $TS/K_2$

- If $K_1$ is composable and $K_2$ is a control invariant of $TS/K_1$ then $TS/(K_1 \land K_2) \text{ sat inv } (K_1 \land K_2)$

- $K$ is composable iff $TS^u \text{ sat inv}(K)$
Scheduler modeling – composability of control invariants

TS1 ∪ TS2 / K_mutex

K_mutex = ¬ (e1 ∧ e2)

is a composable control invariant of TS1 ∪ TS2

Laser 2005 - Summer School on Software Engineering
Scheduler modeling – composability of control invariants

\[ TS_1 \cup TS_2 / K_{\text{mutex}} \]

\[ K_{df} = K_{df1} \land K_{df2} \] is a control invariant of \( TS_1 \cup TS_2 \)

\[ K_{df} \] is not a control invariant of \( TS_1 \cup TS_2 / K_{\text{mutex}} \)

Laser 2005 - Summer School on Software Engineering
Scheduler modeling – the scheduling constraint $K_{\text{SCH}}$

The scheduling constraint $K_{\text{SCH}}$ relates timing constraints of 3 different kinds

- *from the execution platform* e.g. execution times, latency times
- *from the external environment* about arrival times of triggering events e.g. periodic tasks
- *user requirements* e.g. QoS, which are timing constraints relating events of the real-time system and events of its environment e.g. deadlines, jitter
Scheduler modeling – the scheduling constraint $K_{SCH}$

Each shared resource induces a partition \{Sleep, Wait, Use\}.

- **Sleep**
  - arrive
  - $t := 0$
  - $T_{min} \leq t \leq T_{max}$

- **Wait**
  - begin
  - $x := 0$
  - $t \leq D - E_{max}$

- **Use**
  - $t \leq D$

- finish
  - $E_{min} \leq x \leq E_{max}$
  - $t \leq D$

**Arrival time ($t$)**
**Execution time ($x$)**
**Deadline ($D$)**
Scheduler modeling – the scheduling constraint $K_{SCH}$

$K_{SCH} = \bigwedge_i K^i_{SCH}$

where $K^i_{SCH}$ expresses the property that no timing constraint is violated in process $i$.

For timelock-free process models with bounded guards, schedulability boils down to deadlock-freedom of processes.

$K_{SCH} = s \land (t \leq T) \lor w \land (t \leq T-E) \lor u \land (x \leq E)$
Scheduler modeling – the scheduling policy $K_{POL}$

$K_{POL}$ is the conjunction of scheduling policies for the set $R$ of shared resources

$$K_{POL} = \bigwedge_{r \in R} K^r_{POL} \quad \text{where} \quad K^r_{POL} = K^r_{CONF} \land K^r_{ADM}$$

- $K^r_{CONF}$ says how conflicts for the acquisition of resource $r$ are resolved e.g. EDF, RMS, LLF
- $K^r_{ADM}$ says which requests for $r$ are considered by the scheduler at a state e.g. masking
Scheduler modeling – the scheduling policy $K_{POL}$

$K_{POL}$: scheduling policy

$K_{ADM}$: admission control

$K_{CONF}$: Conflict resolution

$r_1^1 K_{ADM}^{1}$

$r_i^i K_{ADM}^{i}$

$r_n^n K_{ADM}^{n}$

$r_1^1 K_{CONF}^{1}$

$r_i^i K_{CONF}^{i}$

$r_n^n K_{CONF}^{n}$
Scheduler modeling – the scheduling policy $K_{\text{POL}}$ : example

$K_{\text{POL}}$ for the Priority Ceiling Protocol

Admission control: “Process $P$ is eligible for resource $r$ if the current priority of $P$ is higher than the ceiling priority of any resource allocated to a process other than $P$”

Conflict resolution: “The CPU is allocated to the process with the highest current priority”
Scheduler modeling – composability results

• Any constraint $K_{pol}$ is a composable control invariant that is,
  \[ \text{SYNTH}(TS, K_{pol}) = TS/K_{pol} \]

• Decomposition of the global synthesis problem
  \[ \text{SYNTH}(TS, K_{sched} \land K_{pol}) = \text{SYNTH}(TS/K_{pol}, K_{sched}) \]

• Reduction to verification of $\text{SYNTH}(TS, K_{sched})$
  1. Choose a scheduling policy $K_{pol}$ such that the conflicts on controllable actions of $TS/K_{pol}$ are resolved
  2. Check $TS/K_{pol}$ sat $\text{inv}(K_{sched})$
Composability results - application

A scheduler design methodology supported by the Prometheus tool connected to Kronos

\[
\text{K:= K_sched;}
\]
\[
\text{while } \neg (TS/K \text{ sat inv(K) } ) \text{ do}
\]
\[
\quad \text{choose K_pol; K:= K_sched } \land \text{ K_pol}
\]
\[
\text{od}
\]
Overview (2)

• Scheduler modeling
  – The role of schedulers
  – Control invariants
  – Scheduler specifications
  – Composability results

• Timed systems with priorities
  – Definition
  – Composition of priorities
  – Correctness-by-construction results

• Tools and applications
  – The IF toolset
  – The BIP framework

• General Discussion
If $K$ is a constraint characterizing a set of deadlock-free states of $TS$ then there exists a set of priority rules $\text{pr}$ such that $\text{pr}(TS)$ preserves $K$

For any control invariant $K$ of $TS$ there exists a set of dynamic priority rules $\text{pr}$ such that the scheduled system $TS/K = \text{pr}(TS)$

Any feasible scheduling policy $K_{POL}$ induces a restriction that can be described by priorities
Timed Systems with priorities

Priority                      Strengthened guard
\[ a_1 \langle_0 a_2 \]        \[ g_1' = g_1 \land \neg g_2 \]
\[ a_1 \langle_5 a_2 \]        \[ g_1' = g_1 \land \neg \langle 5 \rangle g_2 \]
\[ a_1 \langle_{\infty} a_2 \]  \[ g_1' = g_1 \land \neg \langle \infty \rangle g_2 \]

Notation:  \[ \langle k \rangle g(X) = \exists \ t \leq k \ g(X+t) \] (= eventually g within time k)
Timed Systems with priorities

\[ a_1 \not<^k a_2 \text{ means that } a_1 \text{ is disabled when } a_2 \text{ will be enabled within time } k \]

Def: A priority order is a set of partial orders \( \langle \langle k \rangle \) equating partial order on \( A \) \( k \in \mathbb{R}^+ \) s.t.

\[ a_1 \not<^k a_2 \land a_2 \not<_m a_3 \Rightarrow a_1 \not<_{k+m} a_3 \quad \text{(transitivity)} \]

\[ g_i' = g_i \land \langle k \rangle \in \langle \bigwedge \langle a_i \not<_k a_m \rangle \rangle g_m \]
Timed Systems with priorities

A timed system with priorities is a pair \((TS, pr)\) where \(pr\) is a set of priority rules \(pr = \{C^i, \langle^i \rangle_i\}\) with

- \(\{C^i\}_i\) is a set of disjoint time invariant predicates
- \(\{\langle^i \rangle_i\}_i\) is a set of priority orders

\[
\begin{align*}
pr &= \{ C_i \rightarrow \langle^i \rangle_i \}_i \\
g_i' &= g_i \land \bigwedge C \rightarrow \langle \in pr \left( C \Rightarrow \bigwedge k \in \langle \left( \bigwedge ai k am \langle^k \rangle g_m \right) \right) \\\n\text{Activity Preservation Theorem: } \& i g_i &= \& i g_i'
\end{align*}
\]
Timed Systems with priorities - fixed priority policy

\[ w_1 \land w_2 \rightarrow b_1 \preceq_k b_2 \text{ for some } k \]
Timed Systems with priorities - FIFO policy

\[ t_1 \leq t_2 \rightarrow b_1 \prec_0 b_2 \quad t_2 \leq t_1 \rightarrow b_2 \prec_0 b_1 \]
L1 ≤ L2 → b2 \langle 0 b1 L2 ≤ L1 → b1 \langle 0 b2

where Li = Ti - Ei - ti,
Priority Systems - Composition of priorities

\[
\begin{align*}
\text{pr2} & \quad \text{pr1} \\
\text{pr1} & \quad \neq \\
\text{pr2} & \\
\end{align*}
\]
Priority Systems - Composition of priorities

We take:

\[
\begin{array}{c}
pr2 \\
pr1
\end{array}
\]

\[=\]

\[
\begin{array}{c}
pr1 \oplus pr2
\end{array}
\]

\[
\begin{array}{c}
\ldots
\end{array}
\]

pr1 \oplus pr2 is the least priority containing pr1 \cup pr2

Results:

- The operation \( \oplus \) is partial, associative and commutative
- \( pr1(pr2(B)) \neq pr1(pr2(B)) \)
- \( pr1 \oplus pr2(B) \) refines \( pr1 \cup pr2(B) \) refines \( pr1(pr2(B)) \)
- Priorities preserve deadlock-freedom
Timed Systems with priorities – mutual exclusion

\[ w_1 \land e_2 \rightarrow b_1 \prec \infty f_2 \quad w_2 \land e_1 \rightarrow b_2 \prec \infty f_1 \]

Idea: Give infinitely higher priority to the process using the resource
Timed Systems with priorities – mutual exclusion

The behavior after application of mutual exclusion constraints
Timed Systems with priorities – mutual exclusion

Mutex on $R'$: $b_1 \prec_\infty f_2$, $b_2 \prec_\infty \{ f_1, b_1' \}$

Mutex on $R$: $b_1' \prec_\infty \{ f_2, b_2 \}$, $b_2' \prec_\infty f_1$

Risk of deadlock: The composition is not a priority order!
Timed Systems with priorities – mutual exclusion + FIFO policy

\[ t_1 \leq t_2 \rightarrow b_1 \preceq_0 b_2 \quad t_2 \leq t_1 \rightarrow b_2 \preceq_0 b_1 \]

\[ w_1 \land e_2 \rightarrow b_1 \preceq_\infty f_2 \quad w_2 \land e_1 \rightarrow b_2 \preceq_\infty f_1 \]
The BIP framework - fixed priority preemptive scheduling (1)

\[ b_i \langle b_j, r_i \langle r_j, r_i \langle b_j, b_i \langle r_j \text{ (access to the resource – priority preserved by composition)} \]

\[ \{b_i, p_j\} \langle f_j, \{r_i, p_j\} \langle f_j, n \geq l > j \geq 1 \text{ (non pre-emption by lower pty tasks)} \]

CN: \( \{b_i, p_j\} \{r_i, p_j\} \) for \( n \geq i, j \geq 1 \)

MI: \( a_i, f_i, b_i \) for \( n \geq i \geq 1 \)
The BIP framework - fixed priority preemptive scheduling
The BIP framework - fixed priority preemptive scheduling (2)

\[ b_i < b_j, \quad r_i < r_j, \quad r_i < b_j, \quad b_i < r_j \] (access to the resource – pty inherited by composition)

\[ p_i < f_j, \quad \text{if } w_i \text{ or } e_i' \quad n \geq 1 > j \geq 1 \] (non pre-emption by lower pty tasks)

\[ \{b_i, r_i\} < \{f_j, p_j\} \quad n \geq 1, j \geq 1 \] (Mutual exclusion)
The BIP framework - traffic light for tramway crossing

signal     red  g2r  green
----------  ----  ----  ----------
out signal  r2g

{enter,r2g} {exit,g2r}
The BIP framework – run to completion

i1 \langle \{o1,i2\} \langle \{o2,i3\} \langle o3

CN: \{o1,i2\}, \{o2,i3\}  MI: i1,o3

Laser 2005 - Summer School on Software Engineering
Overview (2)

• Scheduler modeling
  – The role of schedulers
  – Control invariants
  – Scheduler specifications
  – Composability results

• Timed systems with priorities
  – Definition
  – Composition of priorities
  – Correctness-by-construction results

• Tools and applications
  – The IF toolset
  – The BIP framework

• General Discussion
Timed Systems with priorities – liveness

Run: a maximal sequence of successive transitions in a TS
\[ q_0 - t_0 \rightarrow q_0' - a_1 \rightarrow q_1 - t_1 \rightarrow q_1' - a_2 \rightarrow \ldots \quad \ldots \]
\[ q_i - t_i \rightarrow q_i' - a_i \rightarrow q_{i+1} - t_{i+1} \rightarrow \quad \ldots \]

Timelock: a run where the total time elapsed is bounded

Liveloack: a run where only a finite number of transitions occur

LIVE = Timelock-free + Liveloack-free
Timed Systems with priorities – structural liveness

Enforce liveness satisfaction by appropriate structural restrictions preserved by composition operators

3 structural properties easy to check

- structurally non-Zeno
- locally timelock-free
- locally livelock-free

\[ \text{timelock-free} \quad \text{livelock-free} \quad \text{structurally live} \]
Timed Systems with priorities – structural liveness

Structurally non-Zeno: any circuit of the control graph has some clock reset and tested against some positive lower bound

Locally Timelock-free: if time cannot progress then some action is executed (satisfied by construction)

Locally Livelock-free: if time can progress then some action will be executed

\[ \text{in}(s) \Rightarrow \Diamond \lor \bigvee_i \text{ui} \]

LTLF + SnZ \(\Rightarrow\) TLF \hspace{1cm} LTLF + LLLF \(\Rightarrow\) LLF

Structurally live = LTLF + SnZ + LLLF
Timed Systems with priorities – structural liveness

A periodic process of period $T > 0$ and execution time $E$, $(E \leq T)$.

This process is structurally live:

• *Timelock-free because* $\text{SnZ}+ \text{LTLF}$

• *Locally LLF because*

  $\text{in(wait)} = (t=0) \Rightarrow \Diamond (t=T-E) = t \leq T-E$

  $\text{in(exec)} = (x=0) \Rightarrow \Diamond (x=E) = x \leq E$

  $\text{in(sleep)} = (x=E) \Rightarrow \Diamond (t=T) = t \leq T$  ???
Timed Systems with priorities – structural liveness

A periodic process of period $T > 0$ and execution time $E$, $(E \leq T)$.

This process is structurally live:

• *Timelock-free because SnZ+ LTLF*

• *Locally LLF because*

  
  \[ \text{in(wait)} = (t=0) \Rightarrow \Diamond (t=T-E) = t \leq T-E \]

  \[ \text{in(exec)} = (x=0) \land (t \leq T-E) \Rightarrow \Diamond (x=E) \land (t \leq T) \]

  \[ \text{in(sleep)} = (x=E) \land (t \leq T) \Rightarrow \Diamond (t=T) = t \leq T \]

Laser 2005 - Summer School on Software Engineering
Theorem: Priorities preserve the 3 structural properties, thus they preserve structural liveness.

If TS is structurally live then (TS, pr) is structurally live too.
Flexible Composition

- All interactions are complete
- \( pr_{12} \) specifies maximal progress, that is an interaction has lower priority wrt all the interactions containing it e.g. \( \alpha_1 \preceq \alpha_1 \cup \alpha_2 \) \( \alpha_2 \preceq \alpha_1 \cup \alpha_2 \)
Flexible Composition - timed systems

For \( b_i=(\alpha_i, g_i^{\tau_i}, f_i) \), \( \{b_1,b_2\} = (\alpha_1 \cup \alpha_2, g_1^{\tau_1} \land g_2^{\tau_2}, f_1 \cup f_2) \) with 
\[ g_1^{\tau_1} \land g_2^{\tau_2} = (g_1 \land g_2)^{\tau} \quad \text{with} \quad \tau = \max\{\tau_1,\tau_2\} \]

\( \text{pr}_{12}(s_1,s_2) = \alpha_1 \langle_{\infty} \alpha_1 \cup \alpha_2, \alpha_2 \langle_{\infty} \alpha_1 \cup \alpha_2 \)

Theorem: Flexible composition preserves
- Structural liveness if the synchronization guards are not lazy
- Individual liveness if \( \Diamond \neg g_i \) for \( i=1,2 \)
Flexible Composition - timed systems: best effort synchronization

For $g_1 = (t_1 \leq T_1 - E_1)$, $g_2 = (t_2 \leq T_2 - E_2)$ we have:

- $g_1 \land g_2 = (t_1 \leq T_1 - E_1) \land (t_2 \leq T_2 - E_2)$
- $g_1' = (t_1 \leq T_1 - E_1) \land (t_2 > T_2 - E_2)$, $g_2' = (t_2 \leq T_2 - E_2) \land (t_1 > T_1 - E_1)$
Timed Systems with priorities – discussion

STRICT (NON FLEXIBLE) COMPOSITION
• Preserves urgency - risk of timelock
• Constraint - oriented

FLEXIBLE (NON STRICT) COMPOSITION
• Relaxes urgency to avoid timelock
• Design/Synthesis-oriented
Timed Systems with priorities – about priorities

• Priorities are a special kind of restriction used to resolve conflicts between actions

• Priorities are commonly used in systems for resource management and scheduling

• Their combination with behavior raises some problems e.g. associativity of composition

• Have often been considered as “low” level concept e.g. “What It Means for a Concurrent Program to Satisfy a Specification: Why No One Has Specified Priority” Leslie Lamport, POPL, 1984
Priorities are a powerful modeling tool

- they can advantageously replace static restriction
- for real-time systems they allow straightforward modeling of urgency, scheduling policies
- run to completion and synchronous execution can be modeled by assigning priorities to threads
Overview (2)

• Scheduler modeling
  – The role of schedulers
  – Control invariants
  – Scheduler specifications
  – Composability results

• Timed systems with priorities
  – Definition
  – Composition of priorities
  – Correctness-by-construction results

• Tools and applications
  – The IF toolset
  – The BIP framework

• General Discussion
The IF toolset: objectives

Model-based development of real-time systems

Use of high level modeling and programming languages
- Expressivity for faithful and natural modeling
- Cover functional and extra-functional aspects
- Openness

Model-based validation
- Combine static analysis and model-based validation
- Integrate verification, testing, simulation and debugging

Applications:
Protocols, Embedded systems, Asynchronous circuits, Planning and scheduling
The IF toolset: approach

Modeling and programming languages (SDL, UML, SCADE, Java …)

IF: Intermediate Format, based on a general and powerful semantic model

Transition systems

Optimisation and abstraction

simulation
test
verification1 verification2 verification3

state explosion
The IF toolset: challenges

Find an adequate intermediate representation

Expressiveness: direct mapping of concepts and primitives of high modeling and programming languages
- asynchronous, synchronous, timed execution
- buffered interaction, shared memory, method call …

Use information about structure for efficient validation and traceability

Semantic tuning: when translating languages to express semantic variation points, such as time semantics, execution and interaction modes
The IF toolset - IF notation: system description

Processes (Behavior)
- extended timed systems
  (non-determinism, dynamic creation)

Interactions
- asynchronous channels
- shared variables

Data
- predefined data types
  (basic types, arrays, records)
- abstract data types

Dynamic priorities

Laser 2005 - Summer School on Software Engineering
The IF toolset: - IF notation: the basic model (ACTA)

```plaintext
x,y: var
t: timer
?g,a
```
The IF toolset - IF notation: system description

• A process instance:
  – executes asynchronously with other instances
  – can be dynamically created
  – owns local data (public or private)
  – owns a private FIFO buffer

• Inter-process interactions:
  – asynchronous signal exchanges (directly or via signalroutes)
  – shared variables
const N1 = ...; // constants

type t1 = ...; // types

signal s2(t1, t2), // signals

// signalroutes
signalroute sr1(1) ... // route attributes from P1 to P3

// processes
process P1(N0)
... // data +
behaviour
endprocess;

... process P3(N3)
... 
endprocess;

The IF toolset - IF notation: system description
The IF toolset -IF notation: process description

Process = hierarchical, timed system

```
process P1(N1);
  fpar ... ;

  // types, variables, constants, procedures
  state s0 ... ;
    ... // transition t1
  endstate;

  state s1 #unstable... ;
    ... // transitions t2, t3
  endstate;

  ... // states s2, s3, s4
endprocess;
```

![Diagram of process P1(N1) with states and transitions labeled]

Laser 2005 - Summer School on Software Engineering
The IF toolset - IF notation: dynamic creation

- **process creation:**
  
  \[ p := \text{fork client (true)} \]
  
  - pid of the newly created instance
  - process name
  - parameters
  - a new instance is created

- **process destruction:**
  
  \[ \text{kill client(2)} \]
  
  \[ \text{kill p} \]
  
  - pid expression
  - the instance is destroyed, together with its buffer, and local data

- **process termination:**
  
  \[ \text{stop} \]
  
  - the “self” instance is destroyed, together with its buffer, and local data
transition = urgency + trigger + body

state s0
...

transition

urgent = eager
provided x!=10;
when c2 >= 4;
input update(m);
body ....
nextstate s1;
...
endstate;

statement list

statement = data assignment
message emission,
process or signalroute creation or destruction, ...

untimed guard
timed guard
signal consumption from the process buffer

sequential, conditional, or iterative composition

Laser 2005 - Summer School on Software Engineering
The IF toolset- IF notation: data and types

Variables:
- are *statically typed* (but *explicit conversions* allowed)
- can be declared *public* (= shared)

Predefined basic types: integer, boolean, float, pid, clock

Predefined type constructors:
- (integer) interval:  
  ```
  type fileno = range 3..9;
  ```
- enumeration:  
  ```
  type status = enum open, close endenum;
  ```
- array:  
  ```
  type vector = array[12] of pid
  ```
- structure:  
  ```
  type file = record f fileno; s status endrecord;
  ```

Abstract Data Type definition facilities …
The IF toolset - IF notation: interactions

**signal route** = connector = process to process communication channel with **attributes**, can be **dynamically** created

```
signalroute s1(1) #unicast #lossy #fifo
```

**attributes:**
- queuing policy: **fifo** | **multiset**
- reliability: **reliable** | **lossy**
- delivery policy: **peer** | **unicast** | **multicast**
- **delay policy:** **urgent** | **delay[l,u]** | **rate[l,u]**

**from** server **to** client **with** grant, fail;
The IF toolset - IF notation: interactions (delivery policies)

- Peer: to one specific instance
- Unicast: to a randomly chosen instance
- Multicast: to all instances
The IF toolset - IF notation: interactions (signal exchange)

Signal emission (non blocking):

- to a specific process: \texttt{output req (3, open) to server(2);}
- via a signalroute: \texttt{output req(3, open) via s0(1);}
- mixed: \texttt{output token via link(1) to client(k+1)\%N;}

Signal consumption (blocking):

\texttt{input req (f, s);}
The IF toolset - IF notation: System description (example)

const NS= ..., NC= ... ;
type file= ..., status= ..., reason= ... ;

signal stop(), req(file, status), fail(reason), grant(), abort(), update(data);

signalroute s0(1) #multicast
  from server to client with abort;

signalroute s1(1) #unicast #lossy
  from server to client with grant,fail;

signalroute s2(1) #unicast
  from client to server with req;

process server(NS) ... endprocess;
process client(NC) ... endprocess;
The IF toolset - IF notation: timed behavior

The model of time [timed systems]
- global time → same clock speed in all processes
- time progress in stable states only → transitions are instantaneous

![Diagram showing timed behavior]

System configuration:
- \( P_1, P_2, P_3, \ldots, P_k \)
- Time progression:
  - \( \text{time} = 0 \)
  - \( \text{time} = \delta_0 \)
  - \( \text{time} = \delta_0 + \delta_1 \)
The IF toolset - IF notation: timed behavior

- **operations on clocks**
  - set to value
  - deactivate
  - read the value into a variable

- **timed guards**
  - comparison of a clock to an integer
  - comparison of a difference of two clocks to an integer

```plaintext
state send;
  output sdt(self,m,b) to {receiver}0;
set t:= 10;
nextstate wait_ack;
endstate;

state wait_ack;
  input ack(sender,c);
  ...
when 10 < t < 20:
  ...
endstate;
```
The IF toolset - IF notation: dynamic priorities

- priority order between process instances p1, p2 (free variables ranging over the active process set)

\[
\text{priority\_rule\_name} : p1 < p2 \text{ if } \text{condition}(p1,p2)
\]

- semantics: only maximal enabled processes can execute

- scheduling policies
  - fixed priority: \( p1 < p2 \) if \( p1 \) instanceof T and \( p2 \) instanceof R
  - run-to-completion: \( p1 < p2 \) if \( p2 = \text{manager}(0).\text{running} \)
  - EDF: \( p1 < p2 \) if \( \text{Task}(p2).\text{timer} < \text{Task}(p1).\text{timer} (p1) \)
IF toolset - overall architecture

IF Exploration Platform

IF Description

IF Static Analyzer

Objecteering
- UML
- aml2if

Rational Rose
- RT/UML
- OMega
- uml2if

ObjectGeode
- SDL
- sdl2if

Objecteering
- ObjectGeode
- Rational Rose

TGV
- Test Generation
- Test Suites

model construction
- SPIDER
- CADP
- LTS

model checking
- guided simulation

guided simulation
- guided simulation

mincost path extraction
- schedules
IF toolset - Core components

IF description
- parser
- writer
IF AST
- syntactic transformation tools:
  - static analyser
  - code generator

C/C++ code
- application specific process code
- predefined modules (time, channels, etc.)

Compiler
- interaction model
- priorities (scheduling)
- state space representation
- LTS exploration tools
  - debugging
  - model checking
  - test generation
IF toolset - core components: exploration platform

Interaction model

priorities (scheduling)

active instances

process 1

I₁:P₁

I₂:P₁

process 2

I₁:P₂

I₂:P₂

process j

Iₖ:Pₖ

Time module

Time

execution control

run

step

run

step

run

step

create

output

set, reset

Succ?

Succ!

Laser 2005 - Summer School on Software Engineering
IF toolset - core components: exploration platform (time)

Dedicated module
- including clock variables
- handling dynamic clock allocation (set, reset)
- checking timing constraints (timed guards)
- computing time progress conditions w.r.t. actual deadlines and
- fires timed transitions, if enabled

i) discrete time
- clock valuations represented as varying size integer vectors
- time progress is explicit and computed w.r.t. the next enabled deadline

ii) continuous time
- clock valuations represented using varying size difference bound matrices (DBMs)
- time progress represented symbolically
- non-convex time zones may arise because of deadlines: they are represented implicitly as unions of DBMs

Two implementations for discrete and continuous time (others can be easily added)
IF toolset - case studies: protocols

SSCOP
Service Specific Connection Oriented Protocol

MASCARA
Mobile Access Scheme based on Contention and Reservation for ATM case study proposed in VIRES ESPRIT LTR

PGM
Pragmatic General Multicast
case study proposed in ADVANCE IST-1999-29082
IF toolset - ase studies: asynchronous circuits

timing analysis

functional validation
IF toolset - case studies: embedded software

Ariane 5 Flight Program
joint work with EADS Lauchers

K9 Rover Executive

IF toolset - Ariane-5 flight program
IF toolset - Ariane-5 flight program: the model

- built by reverse engineering by EADS-LV

- two independent views
  1. asynchronous
     - high level, non-deterministic, abstracts the whole program as communicating extended finite-state machines
  2. synchronous
     - low level, deterministic, focus on specific components …

- we focus on the asynchronous view
IF toolset - Ariane-5 flight program: architecture

OBC (On Board Computer)

- Regulation
  - engines/boosters
  - ignition/extinction

- Configuration
  - stage/payload
  - separation

Control
- Navigation
- Guidance
- Algorithms

Ground

OBC (Redundant)

~3500 lines of SDL code

Laser 2005 - Summer School on Software Engineering
initiate sequences of "regulation" commands at right moments in time:
- at $T_0 + \Delta_1$ execute action$_1$
- at $T_0 + \Delta_2$ execute action$_2$
  ...
- at $T_0 + \Delta_n$ execute action$_n$

if necessary, stopped at any moment

described as "sequential" processes, moving on specific, precise times
IF toolset - Ariane-5 flight program: configuration components

• initiates “configuration” changes depending on:
  – flight phase: ground, launch, orbit, …
  – control information: reception of some signal, …
  – time: eventually done in $[T_0+L, T_0+U]$

• described as processes combining signal and timeout-driven transitions
the opening action eventually happens between $T_{\text{early}}$ and $T_{\text{late}}$ moments, if possible, on the reception on the open signal.
IF toolset - Ariane-5 flight program: control components

• compute the flight commands depending on the current flight evolution
  – guidance, navigation and control algorithms

• abstracted over-simplified processes
  – send flight commands with some temporal uncertainty
IF toolset - Ariane-5 flight program: control components
(example)

time non-deterministic:
the firing signal can be sent between $T_0 + L$ and $T_0 + U$

```
init
  lazy
  \[ T_0 + L \leq \text{now} \text{ and } \text{now} \leq T_0 + U \]
  output firing to vulcain
  done
```

time deterministic:
the firing signal is sent exactly at $T_0 + K$

```
init
  eager
  \[ T_0 + K = \text{now} \]
  output firing to vulcain
  done
```
IF toolset - Ariane-5 flight program: requirements

- **general** requirements
  - e.g. no deadlock, no timelock

- **overall system** requirements
  - e.g. flight phase order
  - e.g. stop sequence order

- **local component** requirements
  - e.g. activation signals arrive eventually in some predefined time intervals
IF toolset - Ariane-5 flight program: validation (model exploration)

- test simple properties by random or guided simulation
- several **inconsistencies** because timing does not respect causality e.g., deadline missed because of $\Delta_1 > \Delta_2$

\[
\text{now} = T_0 + \Delta_1 \\
\text{output status}
\]

\[
\text{now} = T_0 + \Delta_2 \\
\text{output desactivation}
\]

Laser 2005 - Summer School on Software Engineering
IF toolset - Ariane-5 flight program: Validation (static analysis)

• Clock reduction
  1\textsuperscript{st} version: 143 clocks reduced to 41 clocks
  2\textsuperscript{nd} version: 55 clocks, no more reduction

• Live variable analysis
  20\% of all variables are dead in each state

• Dead code analysis
  eliminates passive processes (without outputs and requirements are independent of their events)
Some results (for 31 processes)

<table>
<thead>
<tr>
<th></th>
<th>time</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>deterministic</td>
<td>non-deterministic</td>
</tr>
<tr>
<td>- live reduction</td>
<td>n.a.</td>
<td>n.a.</td>
</tr>
<tr>
<td>- partial order</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ live reduction</td>
<td>2201760 st.</td>
<td>n.a.</td>
</tr>
<tr>
<td>- partial order</td>
<td>18796871 tr.</td>
<td></td>
</tr>
<tr>
<td>+ live reduction</td>
<td>1604 st.</td>
<td>195718 st.</td>
</tr>
<tr>
<td>+ partial order</td>
<td>1642 tr.</td>
<td>278263 tr.</td>
</tr>
</tbody>
</table>
Overview (2)

• Scheduler modeling
  – The role of schedulers
  – Control invariants
  – Scheduler specifications
  – Composability results

• Timed systems with priorities
  – Definition
  – Composition of priorities
  – Correctness-by-construction results

• Tools and applications
  – The IF toolset
  – The BIP framework

• General Discussion
The BIP framework
The BIP framework – the execution platform

Description in L

Component Meta-model

Interaction Meta-model

Dynamic priorities Meta-model

Execution kernel
Component: C
Ports: p1, p2, ...
Data: x, y, z, ....
Access: (p1, {x, y, z}), (p2, {x, u, v}),

Behavior:

state s1
  on p1 provided g1 do f1 to state s1'
  ................. ......
  on pn provided gn do fn to state sn'

state s2
  on ..... 

........

state sn
  on .. ..
run() {
    Port* p;
    int state = 1;
    while(true) {
        switch(state) {
            case 1: p = sync(a, g_a, d, g_d);
                if (p == a)
                    f_a; state = 2;
                else
                    f_d; state = 3;
                break;
            case 2: p = sync(b, g_b, e, g_e);
                ...
            case 3: ...
        }
    }
}
Connector: BUS={p, p', ..., }
complete()
Behavior:

  on $\alpha_1$ provided $g_{\alpha_1}$ do $f_{\alpha_1}$
  on $\alpha_2$ provided $g_{\alpha_2}$ do $f_{\alpha_2}$

Priorities: PR

  if C1 then $\{ (\alpha_1, \alpha_2), (\alpha_3, \alpha_4), ... \}$
  if C2 then $\{ (\alpha, ...), (\alpha, ...), ... \}$

  if Cn then $\{ (\alpha, ...), (\alpha, ...), ... \}$
The BIP framework - modulo-8 counter: atomic component

The diagram illustrates a modulo-8 counter using the BIP framework. The counter transitions between states based on inputs and ticks. The states are labeled as 'Zero', 'Zero\textsuperscript{'}', 'One\textsuperscript{'}', and 'One'. The transitions are labeled with guards (g) and actions (f) as follows:

- From 'Zero' to 'Zero\textsuperscript{'}':
  - g\textsubscript{flip}: X = 1
  - f\textsubscript{flip}: Y := 0

- From 'Zero\textsuperscript{'}' to 'One':
  - g\textsubscript{flip}: X = 1
  - f\textsubscript{flip}: Y := 1

- From 'One\textsuperscript{'}' to 'Zero':
  - g\textsubscript{flip}: X = 1
  - f\textsubscript{flip}: Y := 0

- From 'One':
  - g\textsubscript{flip}: X = 1
  - f\textsubscript{flip}: Y := 1

The counter transitions are triggered by 'tick' events.
The BIP framework - modulo-8 counter: architecture
The BIP framework - modulo-8 counter: the model

<table>
<thead>
<tr>
<th>tick $\langle$ flip$_0$, tick $\langle$ flip$_1$, tick $\langle$ flip$_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CN:</strong> tick = {tick$_0$, tick$_1$, tick$_2$}</td>
</tr>
<tr>
<td><strong>CI:</strong> $\emptyset$</td>
</tr>
<tr>
<td>Transfer: $X_1 := Y_0; X_2 := Y_1 \land Y_0$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CN: tick$_0$, flip$_0$</th>
<th>CN: tick$_1$, flip$_1$</th>
<th>CN: tick$_2$, flip$_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CI:</strong> flip$_0$</td>
<td><strong>CI:</strong> flip$_1$</td>
<td><strong>CI:</strong> flip$_2$</td>
</tr>
</tbody>
</table>

![Diagram showing the BIP framework for the modulo-8 counter]

Laser 2005 - Summer School on Software Engineering
The BIP framework – readers/writers: architecture

Writer

Resource

Reader_i

Laser 2005 - Summer School on Software Engineering

165
The BIP framework – readers/writers: resource model

Component: Resource

Port: wrReq, rdReq, rdRel, wrRel

Data: repository, rdID, rdData0, rdData1, …. rdData_n

Access: (wrRel, {repository}), (rdRel, {rdData0, rdData1, …. rdData_n}),
         (rdReq, {rdID})

Behavior: (State: Idle, RdBusy, WrBusy)

state Idle

on wrReq provided (true) do {Nop} to state WrBusy

on rdReq provided (true) do {rdData_{rdID} := get(repository), rdCount++}
                          to state RdBusy

state WrBusy

on wrRel provided (true) do {Nop} to state Idle

state RdBusy

on rdReq provided (true) do {rdData_{rdID} := get(repository), rdCount++} to state RdBusy

on rdRel provided (rdCount > 0) do {rdCount -- } to state RdBusy

on rdRel provided (rdCount == 0) do {Nop} to state Idle
The BIP framework – readers/writers: writer model

Component: Writer

Port: wrReq, wrRel
Data: file, quantum
Access: (wrRel, {quantum})
Behavior: (States: Idle, Writing)
  state Idle
  on wrReq provided (true) do { quantum:=get(file)} to state Writing

state Writing
  on wrRel provided (true) do { Nop} to state Idle
The BIP framework – readers/writers: reader model

Component: Reader
Port: rdReq, rdRel,
Data: file, quantum
Access: (rdRel, {quatum})
Behavior (States: Idle, Reading)

state Idle
on rdReq provided (true) do {Nop} to state Reading

state Reading
on rdRel provided (true) do {put(quantum, file)} to state idle
The BIP framework – Readers/Writers: connectors

Connector : \( Wr\text{Req} = \{\text{Writer}.wr\text{Req}, \text{Resource}.wr\text{Req}\} \)

\[ \text{Complete()} : \emptyset \]

**Behavior:**
On \( Wr\text{Req} \) provided (true), do \( \{\text{Nop}\} \)

Connector : \( Wr\text{Rel} = \{\text{Writer}.wr\text{Rel}, \text{Resource}.wr\text{Rel}\} \)

\[ \text{Complete()} : \emptyset \]

**Behavior:**
On \( Wr\text{Rel} \) provided (true), do \( \{\text{put}(\text{quantum},\text{repository})\} \)

Connector : \( Rd\text{Req}_i = \{\text{Resource}.rd\text{Req}, \text{Reader}_i.rd\text{Req}\} \)

\[ \text{Complete()} : \emptyset \]

**Behavior:**
On \( Rd\text{Req}_i \), provided (true), do \( \{\text{RdID} := i\} \)

Connector: \( \text{ReadRel}_i = \{ \text{Resource}.rd\text{Rel}, \text{Reader}_i.rd\text{Rel} \} \)

\[ \text{Complete()} : \emptyset \]

**Behavior:**
On \( \text{ReadRel}_i \), provided (true), do \( \{\text{Reader}_i.\text{quantum} := \text{RdData}_i\} \)
The BIP framework - Readers/Writers: state graph
The BIP framework – implementation: the kernel
The BIP framework – implementation: the kernel

- **init**: Launch atom’s threads
- **loop**: Wait all atoms
- **execute**: Notify involved atoms
- **choose**: Execute chosen interaction transfer
- **filter**: Choose among maximal
- **stable**: Compute legal interactions
- **ready**: Filter w.r.t. priorities

Diagram:
- init -> loop
- loop -> execute -> stable
- execute -> choose -> ready
- filter -> choose
- stable -> execute
- filter -> execute
- init -> filter
Overview (2)

• Scheduler modeling
  – The role of schedulers
  – Control invariants
  – Scheduler specifications
  – Composability results

• Timed systems with priorities
  – Definition
  – Composition of priorities
  – Correctness-by-construction results

• Tools and applications
  – The IF toolset
  – The BIP framework

• General Discussion
General discussion - the BIP framework

• Framework for component-based modeling encompassing heterogeneity and relying on a **minimal set of constructs and principles** e.g. interaction models + dynamic priorities

• Clear separation between behavior and structure.
  - Structure is a first class entity
  - Correctness-by-construction techniques for deadlock-freedom and liveness, based on sufficient conditions on structure (mainly)

• Applications at Verimag
  - IF toolset allows layered description of timed systems,
  - Methodology and tool support for generating scheduled code for real-time applications (work by S. Yovine et al.)
General discussion – work dealing with components

- Architecture Description Languages focusing on non-functional aspects or SW Design Description Languages

- Modeling languages: Statecharts, UML, Simulink/Stateflow, Synchronous languages, SystemC, Metropolis, Ptolemy

- Coordination languages (extensions of programming languages): Linda, Javaspaces, TSpaces, Concurrent Fortran, …

- Middleware standards: IDL, Corba, Javabeans, .NET

- Software development environments: PCTE, SWbuses, Softbench, Eclipse

- Process algebras and automata: Pi-Calculus, ORC, automata-based approaches
General discussion – related approaches

Vanderbilt’s Approach

Semantic Unit
Meta-model

Composition
Operators

Behavior

Operational
Semantics

ASML

.net

Metropolis

Semantic Domains

Quantity
Managers

Media

Behavior

Operational
Semantics

Platform

PTOLEMY

MoC
(Model of Computation)

Directors
Connectors

Behavior

Operational
Semantics

Platform

Laser 2005 - Summer School on Software Engineering
General discussion – future work

Develop a rigorous and general basis for architecture modeling and implementation:

• Study the concept of architecture as a means to organize computation (components, interaction, control)
• Define a meta-model for real-time architectures, encompassing specific styles, paradigms, e.g.
  - Modeling synchronous reactive systems
  - Event driven vs. state driven interaction (distinction event ports / state ports)
  - Hierarchical modeling
  - Timed systems
  - Distributed real-time systems, GALS
  - Architecture styles e.g. client-server, blackboard architecture

• Provide automated support for component integration and generation of glue code meeting given requirements
General discussion – future work: synchronous reactive systems

Data flow partial order

Connector:
s1,s2,s3,s4
transfer:
x2,x3:=y1;
x4:=y2,y3
Priority: lowest

Laser 2005 - Summer School on Software Engineering
Pb: coordination between fathers and children
General discussion – future work: expressiveness

Study Component Algebras \( CA = (B, GL, \oplus, \cong) \)

- \((GL, \oplus)\) is a monoid and \(\oplus\) is idempotent
- \(\cong\) is a congruence compatible with operational semantics

- Study classes of glue operators
- Focus on properties relating \(\oplus\) to \(\cong\)

Study notions of **expressiveness** characterizing structure

Given \( CA_i = (B, GL_i, \oplus_i, \cong_i), \quad i=1,2, \)

\( CA_1 \) is more expressive than \( CA_2 \) if \( \forall \ P \)

\[ \exists \ gl_2 \in GL_2 \; gl_2(B_1, \ldots, B_n) \; \text{sat} \; P \Rightarrow \exists \ gl_1 \in GL_1. \; gl_1(B_1, \ldots, B_n) \; \text{sat} \; P \]
Example: For given $B$, $IM$ and $PR$ which coordination problems can be solved?

**Notion of expressiveness different from existing ones which**
- Either completely ignore structure
- or use operators where separation between structure and behavior seems problematic e.g. hiding, restriction