Composable Code Generation for Distributed Giotto

Tom Henzinger, Christoph Kirsch, and Slobodan Matic

presented by Rainer Trummer

Compositionality Seminar WS 2007

Department of Computer Sciences
University of Salzburg

Motivation

- Automotive software
  - Suppliers develop software components
  - Manufacturer integrates components
    - Mass production: optimality
- Compositional design
  - Scale down problem
  - Reuse components
    - Preserve desired properties by composition

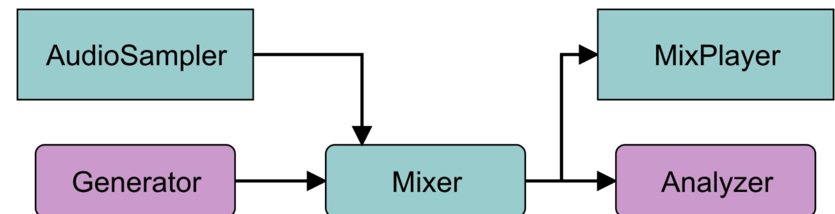
Real Time + Composability

- Giotto framework
  - Purely software time-triggered paradigm
  - Concurrency abstraction: Logical Execution Time
  - Enables compositional design of hard real-time systems
- Distributed platform
  - Realized by distributed compilation of components
  - Individually compiled components merged to final program
- Merge & Verification
  - Automatic check if components meet specification

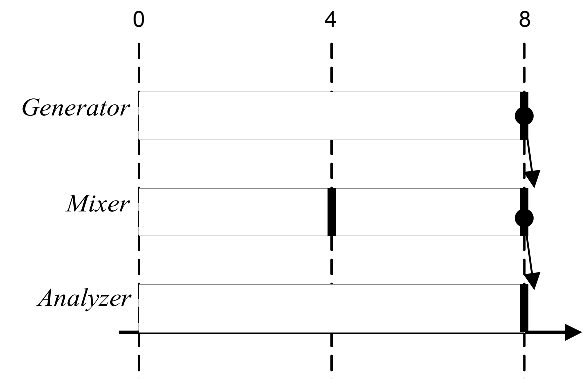
Giotto Framework

- Giotto program
  - Executes a *periodic* set of LET tasks
  - Set of tasks and periods may change upon *mode switches*

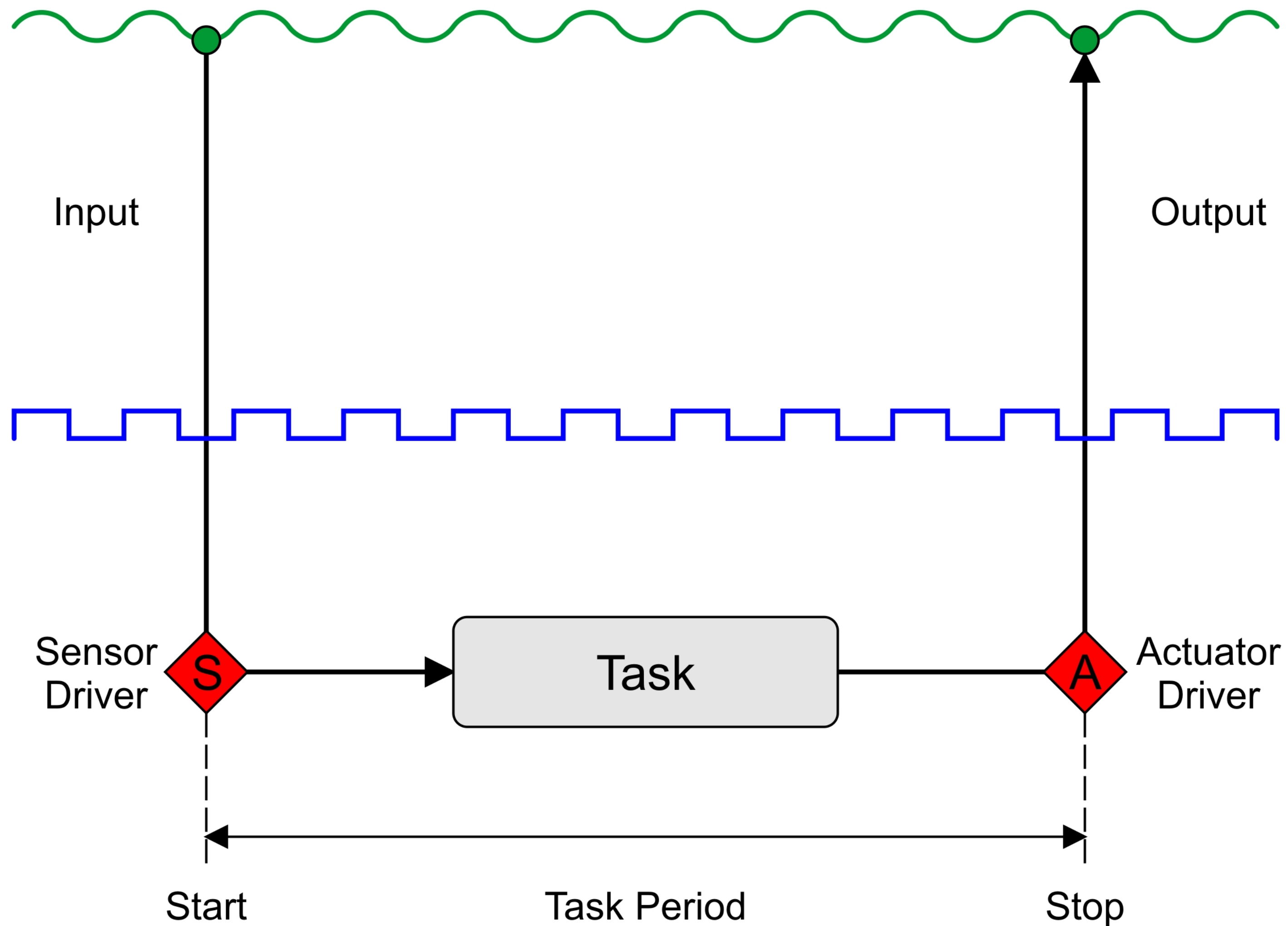
```
mode m1() period 8
{
  actfreq  2 do MixPlayer();
  taskfreq 1 do Analyzer( Mixer );
  taskfreq 2 do Mixer( Generator );
  taskfreq 1 do Generator();
}
```



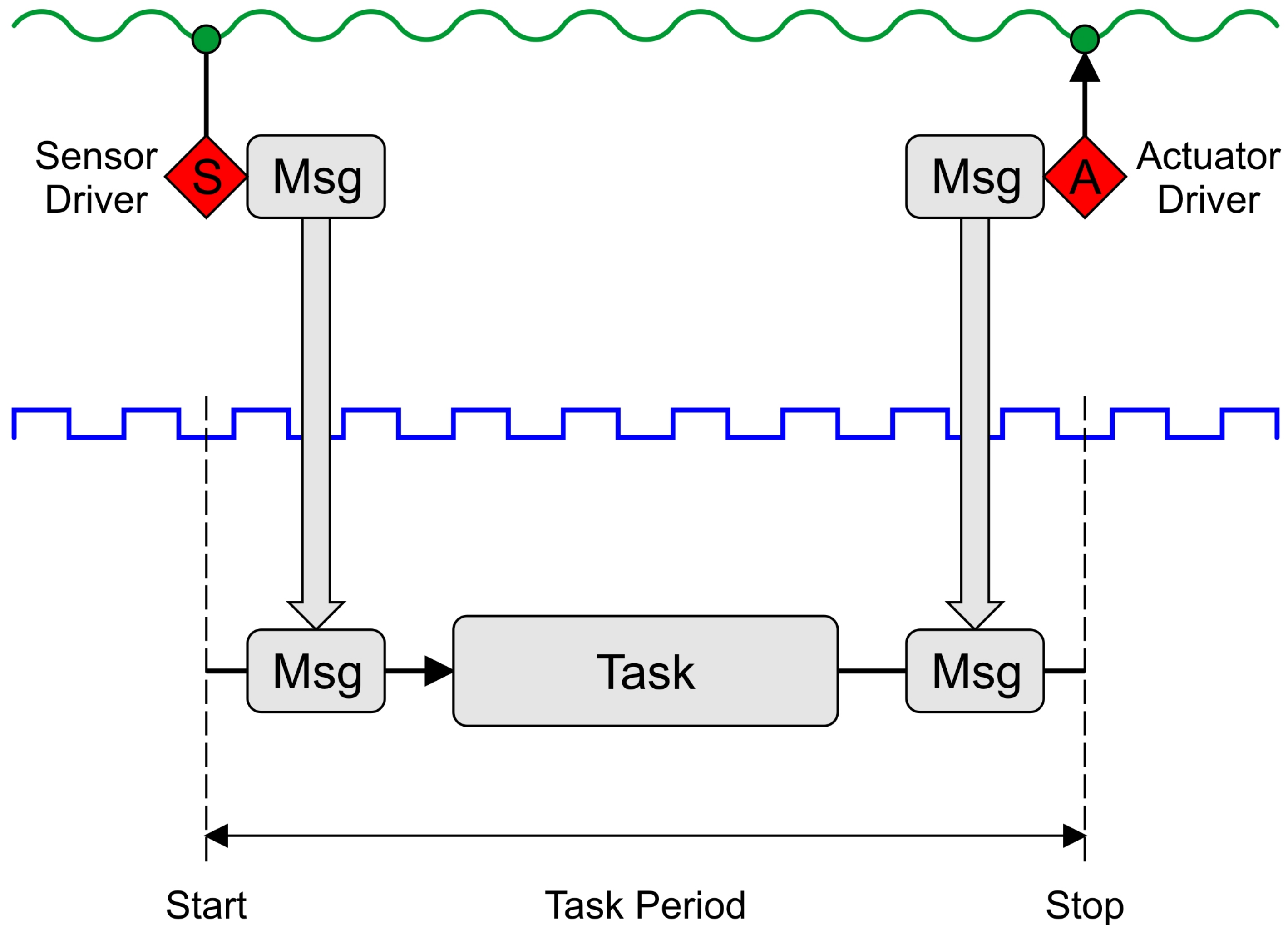
- Task instance
  - Period defines start and stop times
  - Output available at stop time



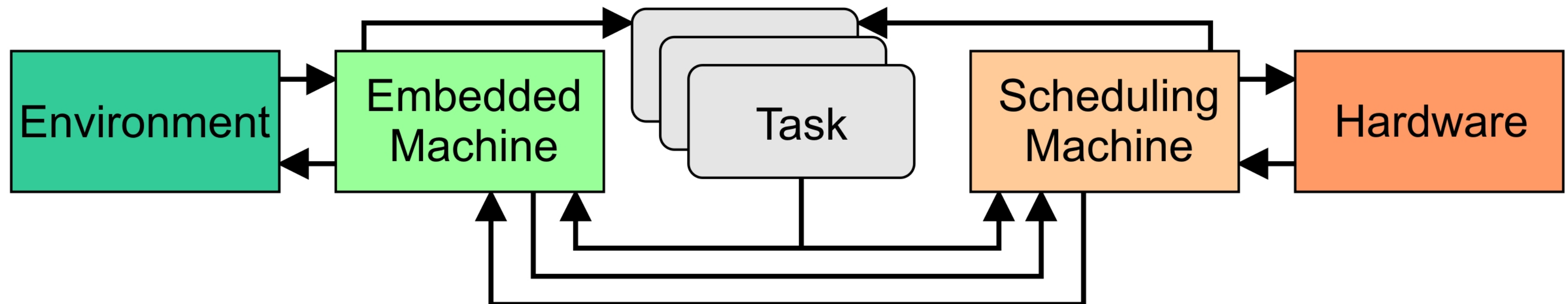
Giotto Abstraction



Giotto Implementation



E and S Machine



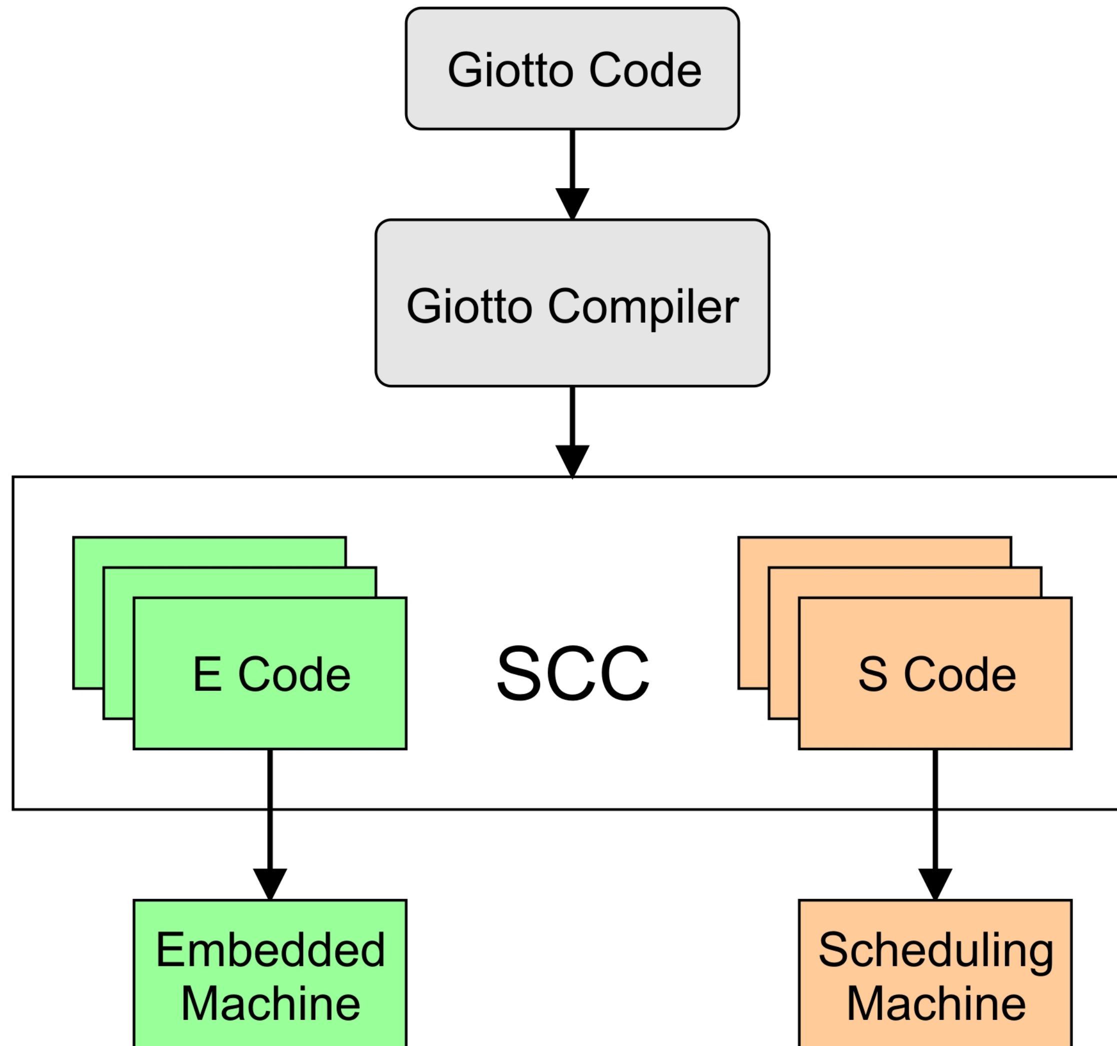
- **Embedded** Machine - E code
  - Environment interaction
  - Task release

```
E_s,h( m_1, 0 ):
call( copy[MixSound] )
call( copy[StringSound] )
release( 1; Mixer; 1 )
release( 1; [MixSound] )
future( 4, E_s,h( m_1, 1 ) )
```

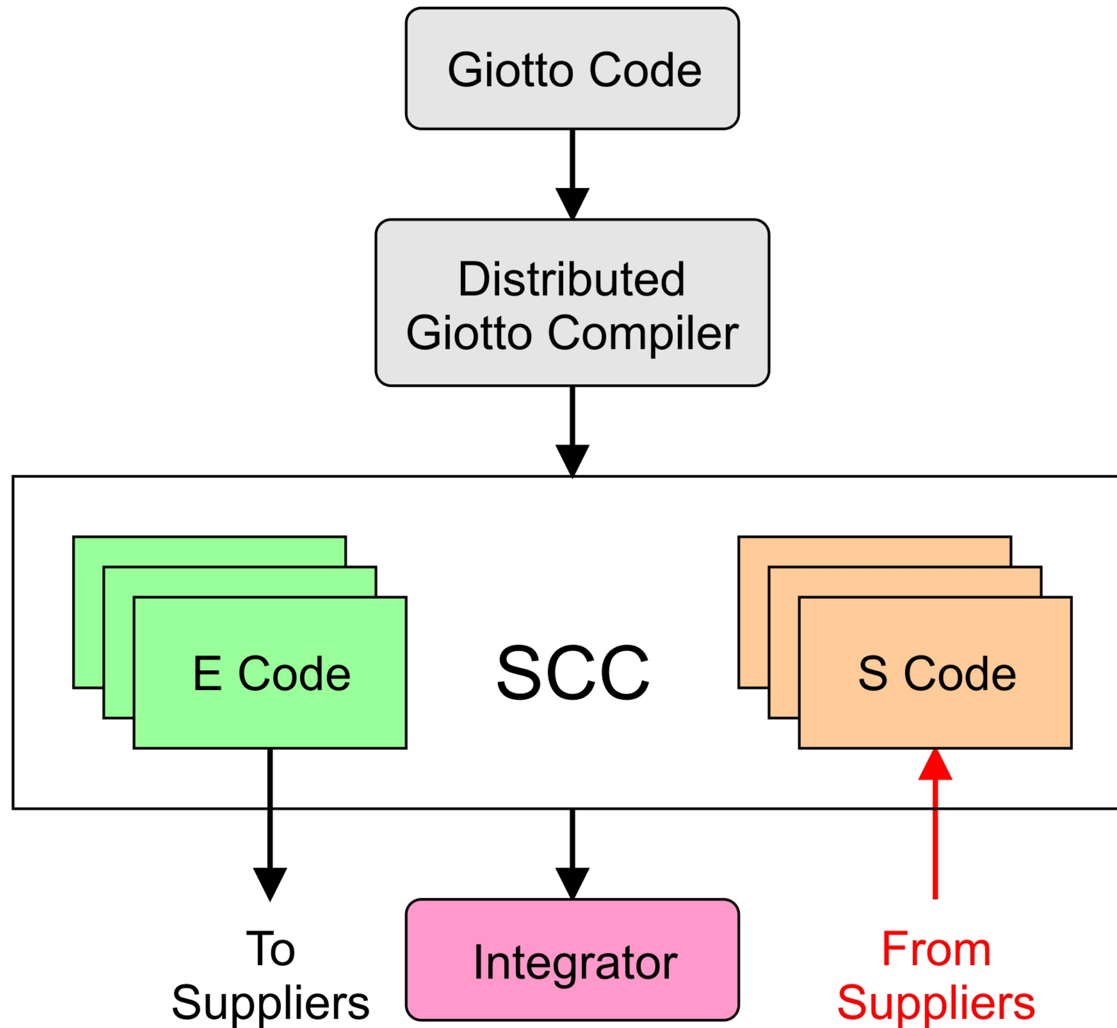
- **Scheduling** Machine - S code
  - Task execution
  - Communication schedule

```
S_s,h( m_1, 0 ):
idle( 1 )
call( InDrv2 )
dispatch( Mixer; 2 )
idle( 3 )
dispatch( [MixSound]; 4 )
```

Schedule-Carrying Code



Distributed Compilation



Distributed Code Generation

Integrator

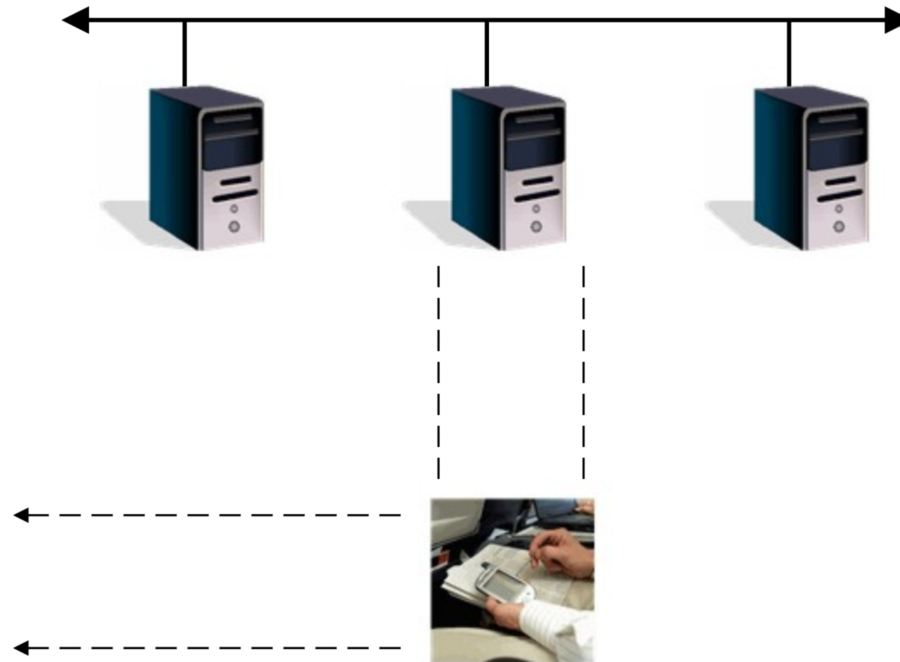
Hosts

Suppliers

Distributed Code Generation

Step 1

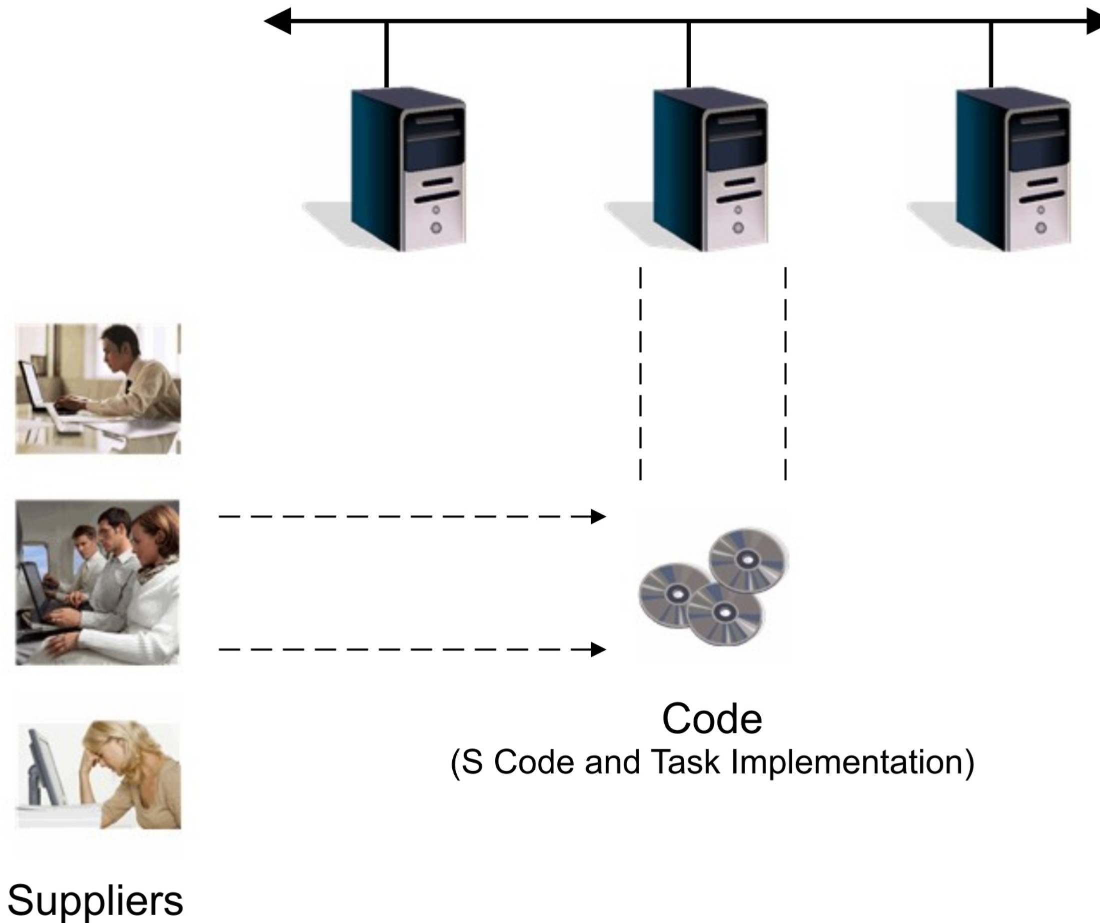
Integrator



Specification
(E Code Module and Timing Interface)

Distributed Code Generation

Step 2



Distributed Code Generation

Step 3

Integrator

Verification
(S Code vs. Specification)

Distributed Code Generation

Integrator

Hosts

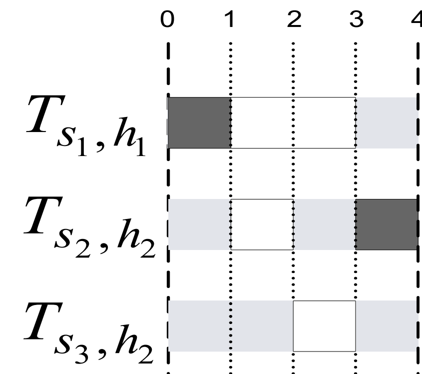
Suppliers

Specification

- Supplier ***s*** on host ***h*** gets
  - Component specification
    - E code module $E_{s,h}$

```
E_{s,h}( m_1, 0 ):
call( copy[MixSound] )
call( copy[StringSound] )
release( 1; Mixer; 1 )
release( 1; [MixSound] )
future( 4, E_{s,h}( m_1, 1 ) )
```

  - Timing interface
    - Set of time intervals $T_{s,h}$
      - where ***s*** may use ***h***
      - where ***s*** may send



- Integrator ensures interface *feasibility*

Integration

- Integrator receives
  - S code module $S_{s,h}$
    - Even with interfaces EDF optimal
  - Task Implementation
    - Usually written in different language
  - Merged SCC module
    - Time-safe if no driver accesses a *released* task before *completion*
    - Complies with timing interface if all tasks *executed* in time intervals

```
S_{s,h}( m_1, 0 ):
idle( 1 )
call( InDrv2 )
dispatch( Mixer; 2 )
idle( 3 )
dispatch( [MixSound]; 4 )
```

Verification

- Giotto program G
  - n : bound on all numbers in G
  - $g_{s,h}$: size of Giotto component implemented by supplier s on host h
- **Correctness**

  To check if a distributed SCC program P correctly implements Giotto program G it is enough to check if each $P_{s,h}$ complies to $T_{s,h}$ and is time-safe
- **Complexity**

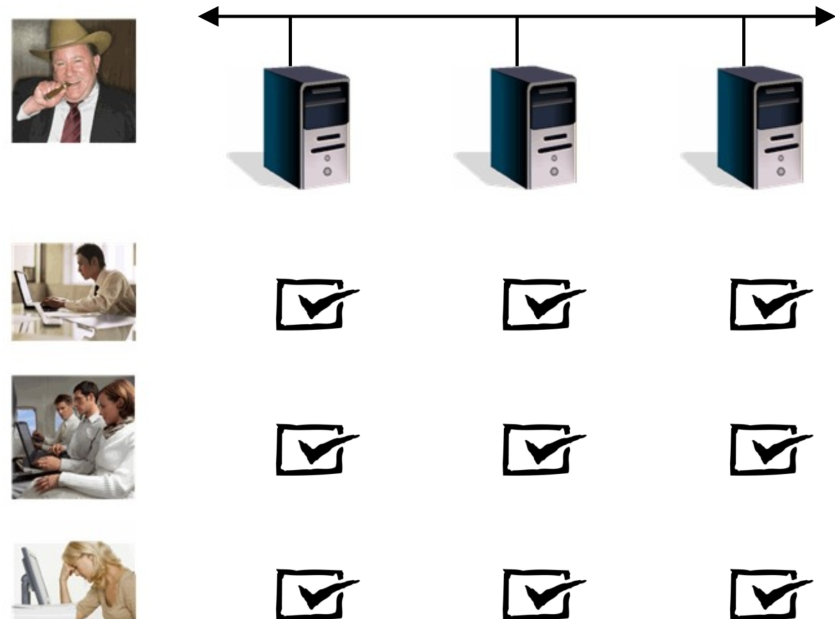
  If a given $P_{s,h}$ complies to $T_{s,h}$ and is time-safe can be checked in

$$O(g_{s,h}\, n) \text{ time}$$

Verification

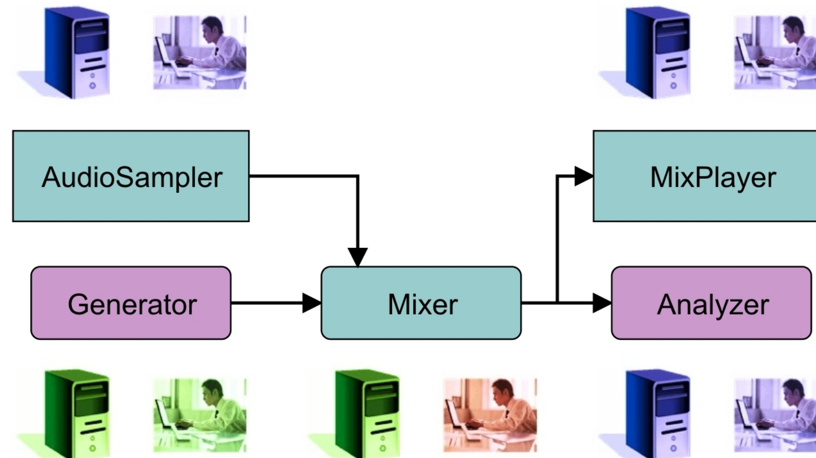
- Module modification
  - Interaction - $E_{s,h}$
  - Schedule - $S_{s,h}$
  - Duration - $wcet$

$O(g_{s,h}\ n)$



Implementation

- Distributed audio mixer application
  - File read, processed, analyzed, and reproduced
  - Two hosts and three suppliers



- PCs running RT-Linux, Ethernet
  - TDMA on top of software-based synchronization, 2.86Mb/s
  - Every 4ms 44 samples (11kHz) processed and transmitted
  - Overhead 3.7%: synchronization 25µs, virtual machine 12µs

Conclusions

- Timing interfaces
  - Used to *distribute code generation* for Giotto programs and distributed target platforms
- Component integration
  - Performed by individually checking *interface compliance* and *time safety* of each component
- Timing requirements
  - *Guaranteed* without solving scheduling problem: burden is shifted to generation of timing interfaces