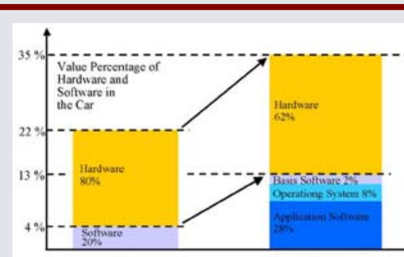
Composable Code Generation for Distributed Giotto

Tom Henzinger
Christoph Kirsch
Slobodan Matic

Chess Review
May 11, 2005
Berkeley, CA

Motivation

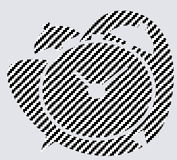
- Automotive software
  - Suppliers develop sw components, Manufacturer integrates
  - Mass production : optimality
- Aircraft software
  - Federated approach replaced by Integrated Modular Avionics
- Compositional design
  - Scale down problem
  - Reuse components
    - Preserve desired properties by composition

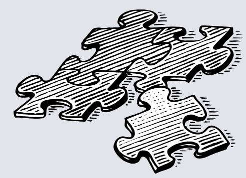


[HKK04]

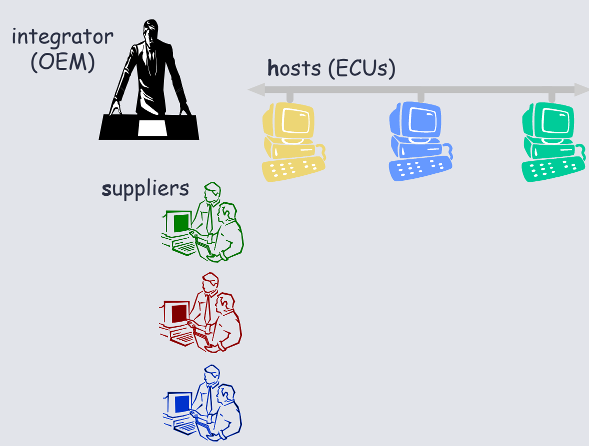
Real-time + Composability

- Distributed platform by distributed compilation
- Giotto concurrency abstraction
  - Logical Execution Time
- Verification
  - Efficient
  - Automatic
- Purely software time-triggered paradigm
  - Compilation
  - Program analysis



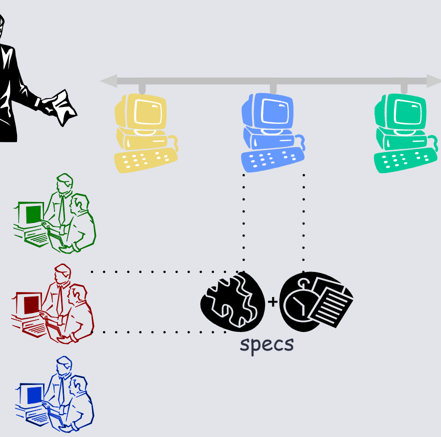


Distributed Code Generation Model



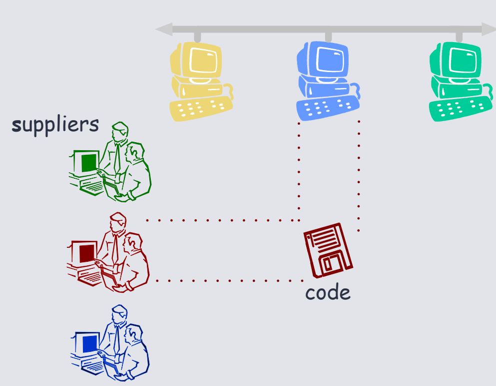
Distributed Code Generation Model

1



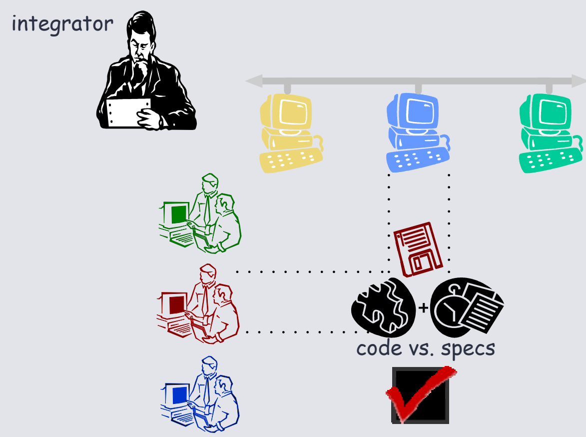
Distributed Code Generation Model

2

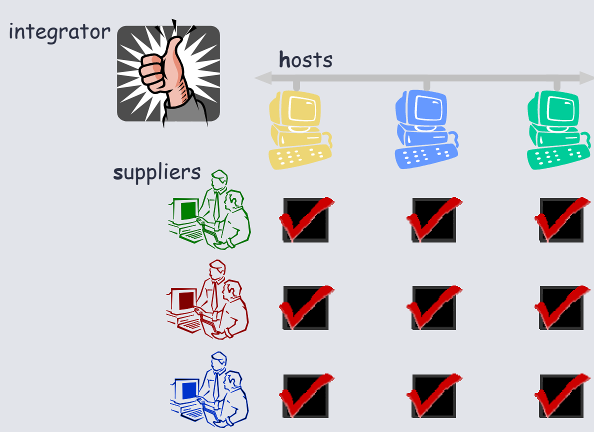


Distributed Code Generation Model

3

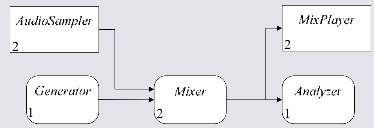


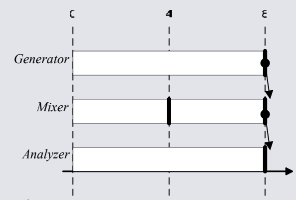
Distributed Code Generation Model



Giotto Framework

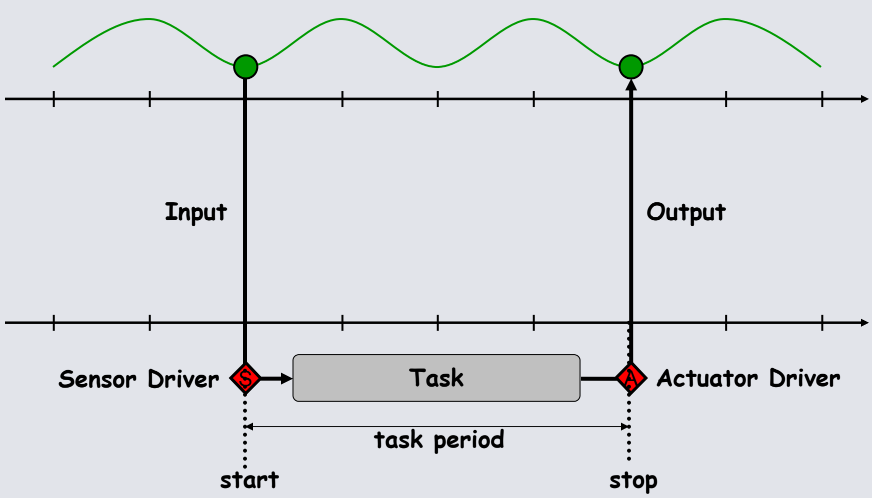
```
mode m1 () period 8 {
  actfreq 2 do MixPlayer();
  taskfreq 1 do Analyzer (Mixer);
  taskfreq 2 do Mixer(Generator);
  taskfreq 1 do Generator();
}
```



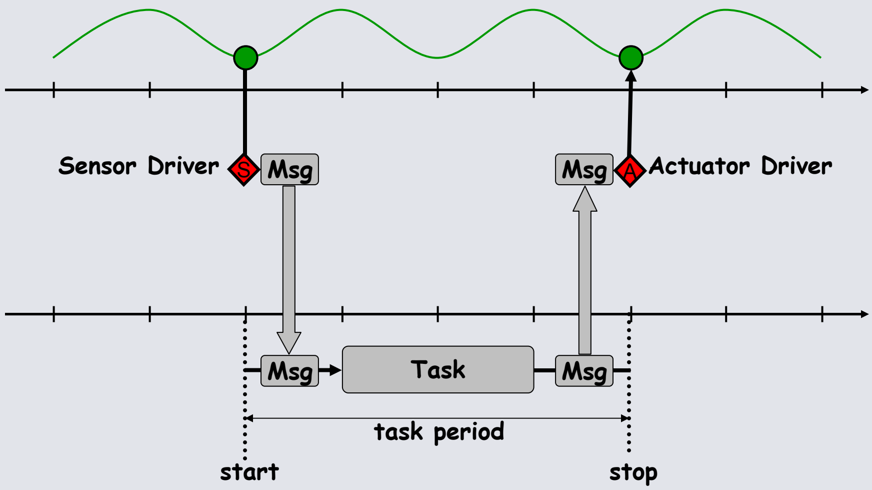


- Task instance
  - Start and stop times defined by period
  - Output available at stop time
- Unit delay
  - Deterministic timing and functional behavior
  - Easy multi-modal schedulability test
  - Temporal composability

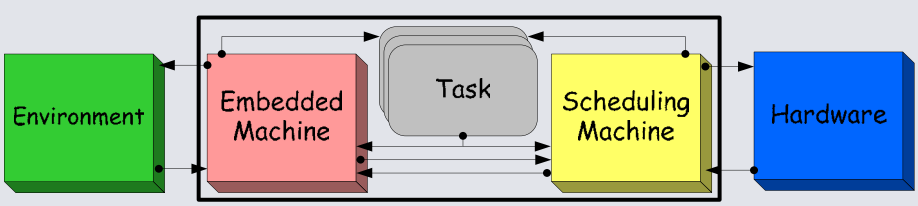
Giotto Abstraction



Giotto Implementation

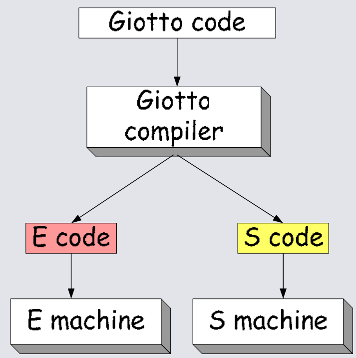


E and S Machine



- **Embedded** Machine - E code
  - environment interaction
  - task release
- **Scheduling** Machine - S code
  - task execution
  - communication schedule

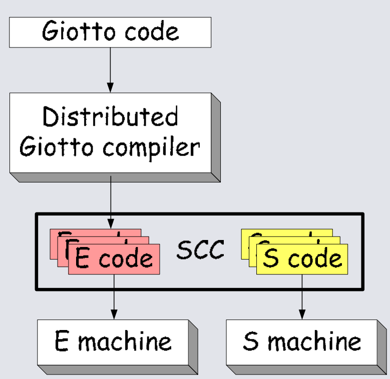
E and S Machine



- environment interaction
- task release

- task execution
- communication schedule

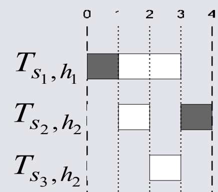
Schedule-Carrying Code



System Specification

- Supplier s on host h :
  - Component specification
    - E code module $E_{s,h}$
  - Timing interface:
    - set of time intervals $T_{s,h}$
      - where s may use h □
      - where s may send ■
- Integrator ensures interface *feasibility*

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



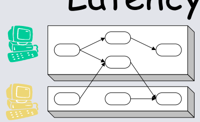
Schedulability

- S code module $S_{s,h}$

  even with interfaces EDF optimal

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

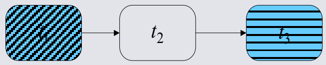
- Latency optimal

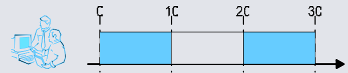


multiple processors + intertask communication ⇒ NP-complete

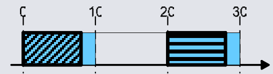
- With LET assumption
  - Task dependency and distribution not hard

LET and Temporal Partitioning



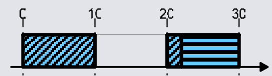


	t_1	t_2	t_3
wcet	8		8

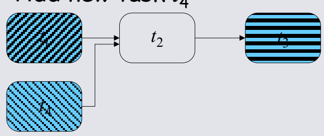


- Increase execution time of t_1

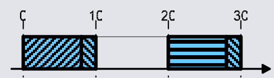
	t_1	t_2	t_3
wcet	12		8



- Add new task t_4



	t_1	t_2	t_3	t_4
wcet	8		8	4



SCC Properties

- SCC module
  - is time-safe if

  no driver accesses a *released* task before *completion*
  - complies with timing interface if

  all tasks are *executed* in time intervals
    - Platform dependent properties (*wcet*)
    - Deadlines specified in the E code

- SCC module - state transition system
  - Two properties – safety properties

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**

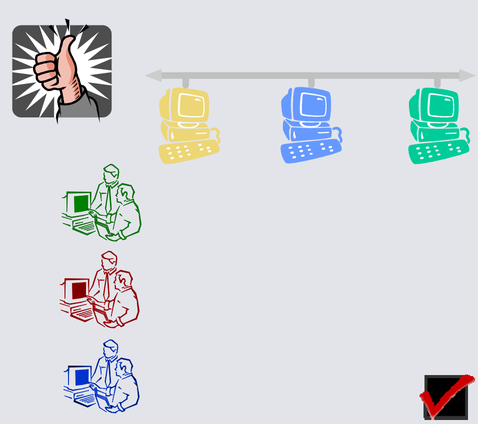
  If a given $P_{s,h}$ complies to $T_{s,h}$ and is time-safe can be checked in

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

Verification

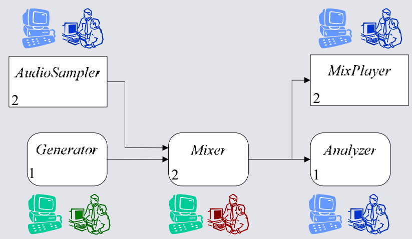
- Module modification
  - task invocation, interaction - $E_{s,h}$
  - schedule - $S_{s,h}$
  - execution time - *wcet*

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



Implementation

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



- PCs running Real-time 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