Model-Based Autosynthesis of Time-Triggered Buffers for Event-Based Middleware Systems

Jonathan Sprinkle[1] and Brandon Eames[2]

October 25, 2009

THE UNIVERSITY OF ARIZONA® UtahState UNIVERSITY

[1]University of Arizona, sprinkle@ECE.Arizona.Edu
[2]Utah State University, beames@usu.engineering.edu

Outline I

Outline II

4 Conclusion
- Implementation Feasibility
- Impact on Existing Examples
- Future/Ongoing Work
- References

What domain is this anyway?

- Autonomous Ground Vehicles
  - Complex, cyber-physical systems
  - Robotics, control, software, and information experts required
- Component-based middleware
  - Networked, real-time and soft real-time components
  - High bandwidth and low bandwidth components
  - Simple, component model
- Effort
  - Many domain experts, few programming experts
  - Heterogeneous models of computation
  - Experience in *information only* domain does not directly translate

What domain is this anyway?

- Autonomous Ground Vehicles
  - Complex, cyber-physical systems
  - Robotics, control, software, and information experts required
- Component-based middleware
  - Networked, real-time and soft real-time components
  - High bandwidth and low bandwidth components
  - Simple, component model
- Effort
  - Many domain experts, few programming experts
  - Heterogeneous models of computation
  - Experience in *information only* domain does not directly translate

What domain is this anyway?

- Autonomous Ground Vehicles
  - Complex, cyber-physical systems
  - Robotics, control, software, and information experts required
- Component-based middleware
  - Networked, real-time and soft real-time components
  - High bandwidth and low bandwidth components
  - Simple, component model
- Effort
  - Many domain experts, few programming experts
  - Heterogeneous models of computation
  - Experience in *information only* domain does not directly translate

What domain is this anyway?

- Autonomous Ground Vehicles
  - Complex, cyber-physical systems
  - Robotics, control, software, and information experts required
- Component-based middleware
  - Networked, real-time and soft real-time components
  - High bandwidth and low bandwidth components
  - Simple, component model
- Effort
  - Many domain experts, few programming experts
  - Heterogeneous models of computation
  - Experience in *information only* domain does not directly translate

What domain is this anyway?

- Autonomous Ground Vehicles
  - Complex, cyber-physical systems
  - Robotics, control, software, and information experts required
- Component-based middleware
  - Networked, real-time and soft real-time components
  - High bandwidth and low bandwidth components
  - Simple, component model
- Effort
  - Many domain experts, few programming experts
  - Heterogeneous models of computation
  - Experience in *information only* domain does not directly translate

What domain is this anyway?

- Autonomous Ground Vehicles
  - Complex, cyber-physical systems
  - Robotics, control, software, and information experts required
- Component-based middleware
  - Networked, real-time and soft real-time components
  - High bandwidth and low bandwidth components
  - Simple, component model
- Effort
  - Many domain experts, few programming experts
  - Heterogeneous models of computation
  - Experience in *information only* domain does not directly translate

What domain is this anyway?

- Autonomous Ground Vehicles
  - Complex, cyber-physical systems
  - Robotics, control, software, and information experts required
- Component-based middleware
  - Networked, real-time and soft real-time components
  - High bandwidth and low bandwidth components
  - Simple, component model
- Effort
  - Many domain experts, few programming experts
  - Heterogeneous models of computation
  - Experience in *information only* domain does not directly translate

What domain is this anyway?

- Autonomous Ground Vehicles
  - Complex, cyber-physical systems
  - Robotics, control, software, and information experts required
- Component-based middleware
  - Networked, real-time and soft real-time components
  - High bandwidth and low bandwidth components
  - Simple, component model
- Effort
  - Many domain experts, few programming experts
  - Heterogeneous models of computation
  - Experience in *information only* domain does not directly translate

What domain is this anyway?

- Autonomous Ground Vehicles
  - Complex, cyber-physical systems
  - Robotics, control, software, and information experts required
- Component-based middleware
  - Networked, real-time and soft real-time components
  - High bandwidth and low bandwidth components
  - Simple, component model
- Effort
  - Many domain experts, few programming experts
  - Heterogeneous models of computation
  - Experience in *information only* domain does not directly translate

What domain is this anyway?

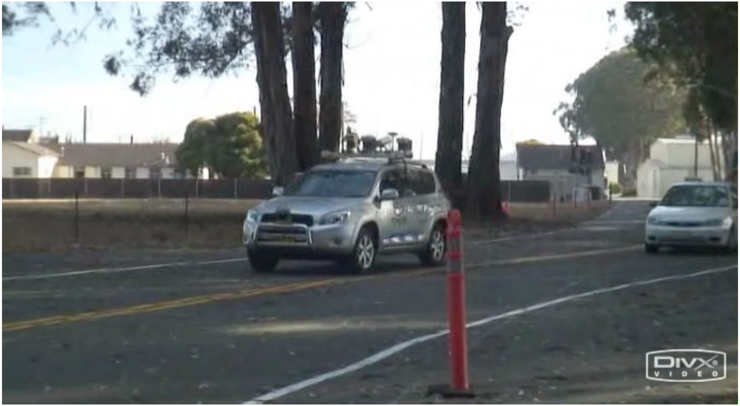
- Autonomous Ground Vehicles
  - Complex, cyber-physical systems
  - Robotics, control, software, and information experts required
- Component-based middleware
  - Networked, real-time and soft real-time components
  - High bandwidth and low bandwidth components
  - Simple, component model
- Effort
  - Many domain experts, few programming experts
  - Heterogeneous models of computation
  - Experience in *information only* domain does not directly translate

What domain is this anyway?

- Autonomous Ground Vehicles
  - Complex, cyber-physical systems
  - Robotics, control, software, and information experts required
- Component-based middleware
  - Networked, real-time and soft real-time components
  - High bandwidth and low bandwidth components
  - Simple, component model
- Effort
  - Many domain experts, few programming experts
  - Heterogeneous models of computation
  - Experience in *information only* domain does not directly translate

UA Autonomous Ground Vehicles

Sydney-Berkeley Driving Team



Link to online movie.

Domain Difficulties

- Robotics software in general
  - Individual task complexity and dynamic real-time nature [1]
  - Generalization of algorithms nontrivial
  - Large number of software contributors
  - Distributed, cross-platform computing environments are non-intuitive for domain experts
- Individual projects
  - Necessity of regression tests [2]
  - Simulation complexity increases dramatically when realistic simulations used

Domain Difficulties

- Robotics software in general
  - Individual task complexity and dynamic real-time nature [1]
  - Generalization of algorithms nontrivial
  - Large number of software contributors
  - Distributed, cross-platform computing environments are non-intuitive for domain experts
- Individual projects
  - Necessity of regression tests [2]
  - Simulation complexity increases dramatically when realistic simulations used

Domain Difficulties

- Robotics software in general
  - Individual task complexity and dynamic real-time nature [1]
  - Generalization of algorithms nontrivial
  - Large number of software contributors
  - Distributed, cross-platform computing environments are non-intuitive for domain experts
- Individual projects
  - Necessity of regression tests [2]
  - Simulation complexity increases dramatically when realistic simulations used

Domain Difficulties

- Robotics software in general
  - Individual task complexity and dynamic real-time nature [1]
  - Generalization of algorithms nontrivial
  - Large number of software contributors
  - Distributed, cross-platform computing environments are non-intuitive for domain experts
- Individual projects
  - Necessity of regression tests [2]
  - Simulation complexity increases dramatically when realistic simulations used

Domain Difficulties

- Robotics software in general
  - Individual task complexity and dynamic real-time nature [1]
  - Generalization of algorithms nontrivial
  - Large number of software contributors
  - Distributed, cross-platform computing environments are non-intuitive for domain experts
- Individual projects
  - Necessity of regression tests [2]
  - Simulation complexity increases dramatically when realistic simulations used

Domain Difficulties

- Robotics software in general
  - Individual task complexity and dynamic real-time nature [1]
  - Generalization of algorithms nontrivial
  - Large number of software contributors
  - Distributed, cross-platform computing environments are non-intuitive for domain experts
- Individual projects
  - Necessity of regression tests [2]
  - Simulation complexity increases dramatically when realistic simulations used

Domain Difficulties

- Robotics software in general
  - Individual task complexity and dynamic real-time nature [1]
  - Generalization of algorithms nontrivial
  - Large number of software contributors
  - Distributed, cross-platform computing environments are non-intuitive for domain experts
- Individual projects
  - Necessity of regression tests [2]
  - Simulation complexity increases dramatically when realistic simulations used

Domain Difficulties

- Robotics software in general
  - Individual task complexity and dynamic real-time nature [1]
  - Generalization of algorithms nontrivial
  - Large number of software contributors
  - Distributed, cross-platform computing environments are non-intuitive for domain experts
- Individual projects
  - Necessity of regression tests [2]
  - Simulation complexity increases dramatically when realistic simulations used

Middleware Solutions

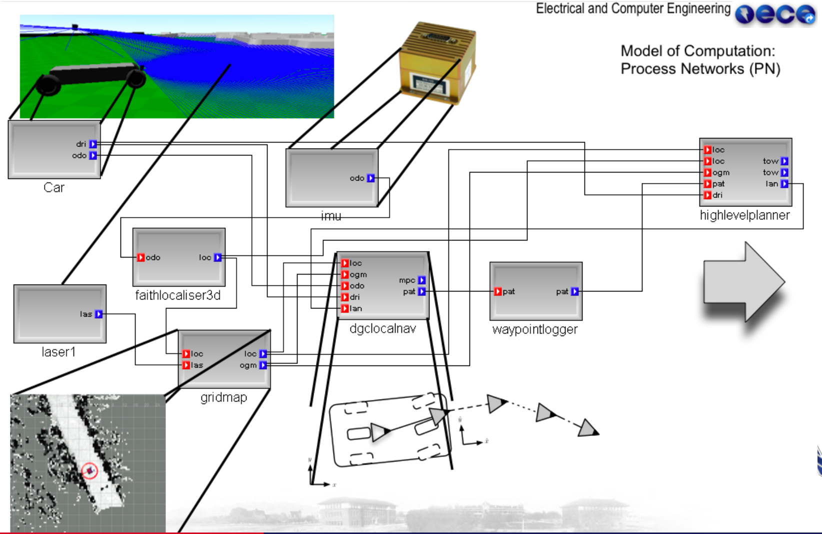
Middleware and component-based technologies facilitate the abstraction of communication, and location of computation.

- CORBA
- ICE

Distributed Real-Time Embedded Systems

- Composition of such systems a subject of significant effort by Schmidt et al.
- The CoSMIC Toolsuite [3] can
  1. model and analyze DRE application functionality and QoS requirements
  2. synthesize CCM-specific deployment metadata for end-to-end QoS (static and dynamic)

So what happens?



How is this problematic?

In a *Publish/Subscribe* framework, this design can be fragile, if continuous/control systems are involved.

Changes in the discrete execution may result in unstable dynamics.

Outline I

Outline II

4 Conclusion
- Implementation Feasibility
- Impact on Existing Examples
- Future/Ongoing Work
- References

Pub/Sub Overview

Publish/Subscribe is a common model of communication used for data interchange between components.

- A component c_1 will *subscribe* to another component's production of a particular data value
- Subscription uses services offered by the middleware.
- Middleware ensures that all subscribers receive the value once it is produced
- Event-based components generally execute *on new data*, or *after some time* if a token is not received

Explicit support for this model provided by Ice [4], DDS [5], and CORBA [6].

Why *Time* is Important

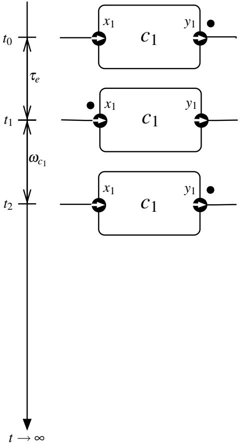


Figure: Messages (tokens) arrive before the timeout.

Why *Time* is Important

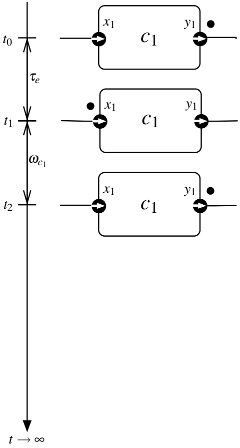


Figure: Messages (tokens) arrive before the timeout.

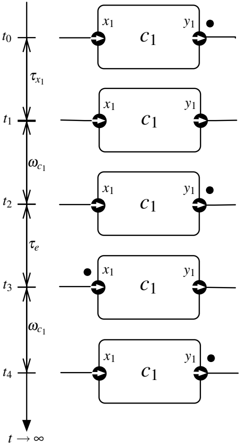


Figure: Behavior when no token arrives prior to timeout.

Message Chattering

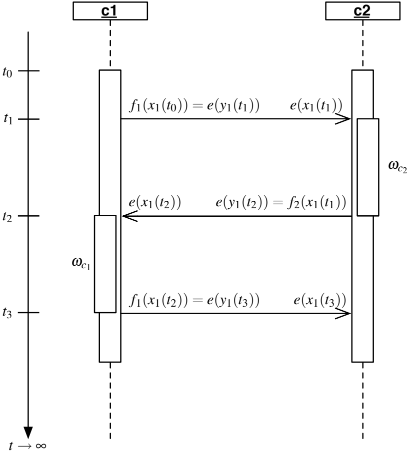


Figure: Proper timing results in expected behavior.

Message Chattering

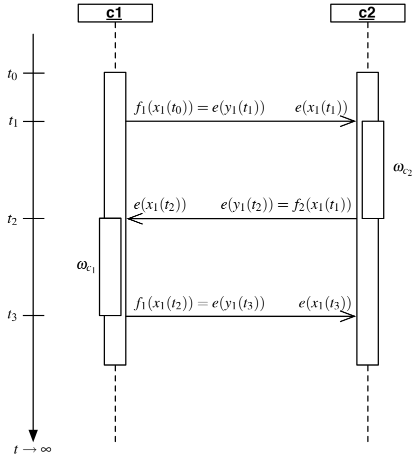


Figure: Proper timing results in expected behavior.

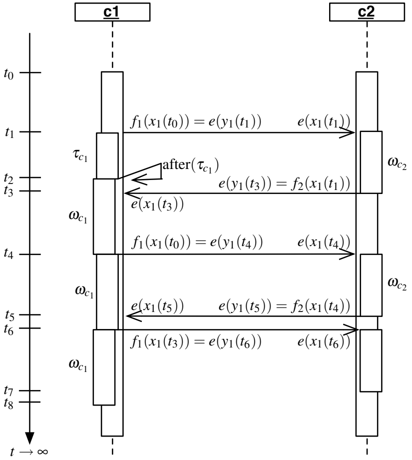


Figure: Improper timing results in "message chattering".

What are T-T Methods?

Time-Triggered execution means that stages of execution occur at particular *times*.

- Useful for distributed/embedded problems with *real-time* constraints
- Frameworks such as Giotto [7, 8] provide formalisms for capturing design issues
- Component models [9] support the development of these formalisms in RT systems

Foundation of the component models it the concepts of a *Logical Execution Time* (LET). Similar semantics are used by TTA [10] and TTP [11], but allows shared memory

Goals of T-T Methods

Time-Triggered methods enable rapid isolation of component or subsystem failures. With known timing of communication, potential failures can be identified in less than one clock cycle, and fault mitigation/isolation can begin.

TT methods *also* enable isolation of an algorithm from the platform on which it runs, enabling structured composition (at the price of some latency) [12].

Outline I

1. Introduction
   - The Domain
   - Autonomous Ground Vehicles
   - UA Autonomous Ground Vehicles
   - Issues and Solutions
2. Background
   - Publish/Subscribe Methods
   - Time-Triggered Methods
   - Domain Semantics
3. Approach
   - Triggers and Generators: Semantics
   - Transformation Results: How they will look
   - Semantics
   - The Transformation Definitions

Outline II

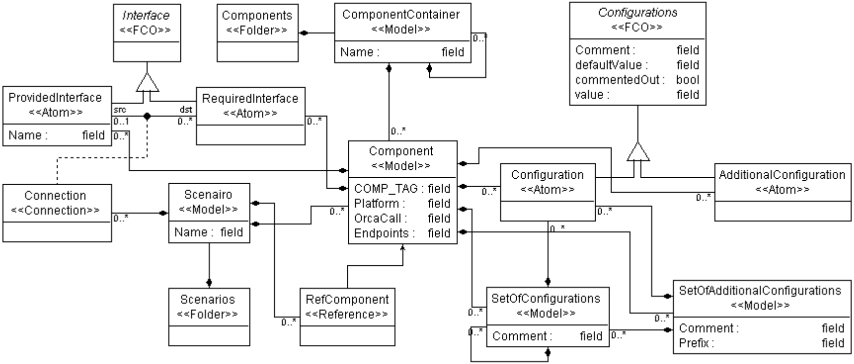
Our Contribution

This paper gives a generic algorithm that converts a standard, event-based distributed system into an event-based distributed system whose execution is “throttled” by time events.

This permits component software to remain *fully unchanged*, and execute in a more precise, more predictable, manner.

Language Design

We leveraged a simultaneously developed language [13] to capture data from the various configuration files, as well as the component interconnection, using the GME toolsuite [14].



A few comments

The language design bears some minor emphasis in a few points:

1. Connections between components are through strong types of provided/required interfaces
2. Directional associations restrict misconstructions
3. Components can be connected to *references* of other components (to permit reuse of all parameters)
4. The configuration space can be hierarchically managed
5. The execution platform can be specified in another aspect (not shown in this metamodel, for brevity)

A few comments

The language design bears some minor emphasis in a few points:

1. Connections between components are through strong types of provided/required interfaces
2. Directional associations restrict misconstructions
3. Components can be connected to *references* of other components (to permit reuse of all parameters)
4. The configuration space can be hierarchically managed
5. The execution platform can be specified in another aspect (not shown in this metamodel, for brevity)

A few comments

The language design bears some minor emphasis in a few points:

1. Connections between components are through strong types of provided/required interfaces
2. Directional associations restrict misconstructions
3. Components can be connected to *references* of other components (to permit reuse of all parameters)
4. The configuration space can be hierarchically managed
5. The execution platform can be specified in another aspect (not shown in this metamodel, for brevity)

A few comments

The language design bears some minor emphasis in a few points:

1. Connections between components are through strong types of provided/required interfaces
2. Directional associations restrict misconstructions
3. Components can be connected to *references* of other components (to permit reuse of all parameters)
4. The configuration space can be hierarchically managed
5. The execution platform can be specified in another aspect (not shown in this metamodel, for brevity)

A few comments

The language design bears some minor emphasis in a few points:

1. Connections between components are through strong types of provided/required interfaces
2. Directional associations restrict misconstructions
3. Components can be connected to *references* of other components (to permit reuse of all parameters)
4. The configuration space can be hierarchically managed
5. The execution platform can be specified in another aspect (not shown in this metamodel, for brevity)

Benefits of this design

With these points, the following benefits are enabled:

1. Data dependencies can be analyzed at design time
2. System startup order can be computed, rather than a design input
3. Execution platform can be changed without changing component definition/configuration

Benefits of this design

With these points, the following benefits are enabled:

1. Data dependencies can be analyzed at design time
2. System startup order can be computed, rather than a design input
3. Execution platform can be changed without changing component definition/configuration

Benefits of this design

With these points, the following benefits are enabled:

1. Data dependencies can be analyzed at design time
2. System startup order can be computed, rather than a design input
3. Execution platform can be changed without changing component definition/configuration

Example Transformation

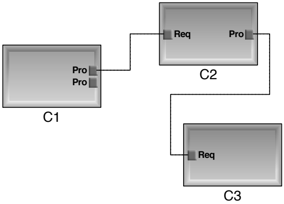


Figure: Example depends on internal timeouts, if no data are received.

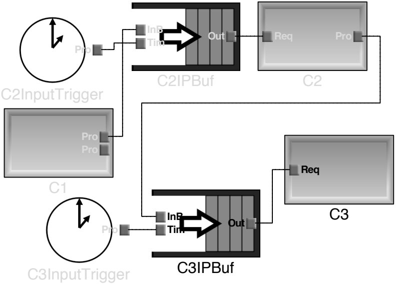


Figure: Time-triggered buffers inserted, instead of timeout values.

Example Transformation

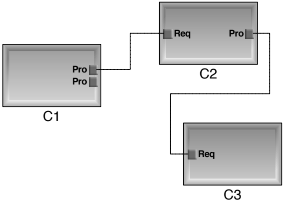


Figure: Example depends on internal timeouts, if no data are received.

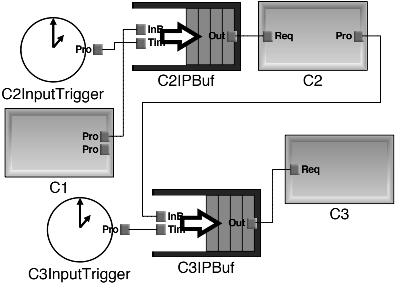


Figure: Time-triggered buffers inserted, instead of timeout values.

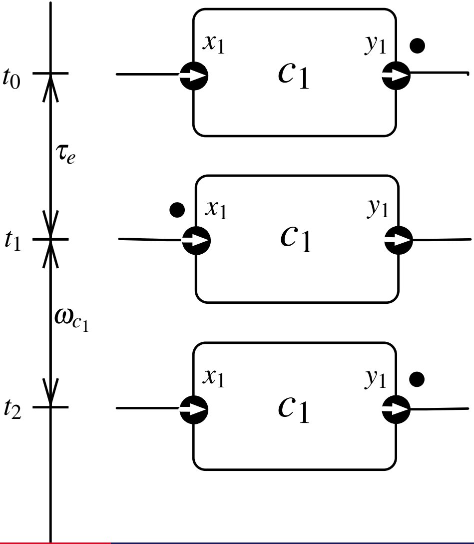
SISO

Consider a component, c_1, with a single input, x_1, and single output, y_1.

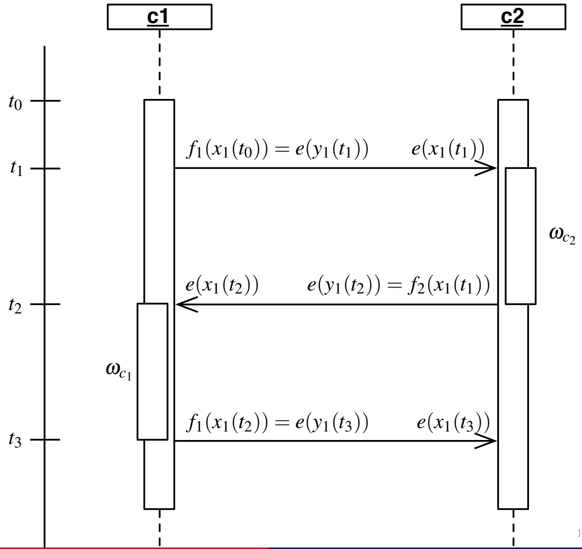
The value y_1 is obtained as the output of the functional behavior of the component, which may also be written in difference equation form as $y_1(k+1) = f(x_1(k))$, demonstrating the discrete notion of the software component, and our ability to encode y, x as signals in time (specifically, discrete time)[3].

[3] Of course, the internal state of an object can affect this outcome, but externally the interface is as presented.

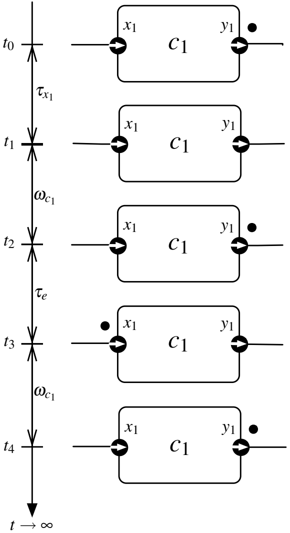
Simple SISO Execution



When Properly Coupled with another SISO Component



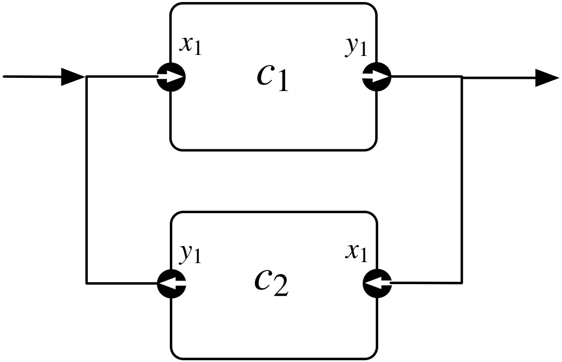
Simple SISO Execution w/ Timeout



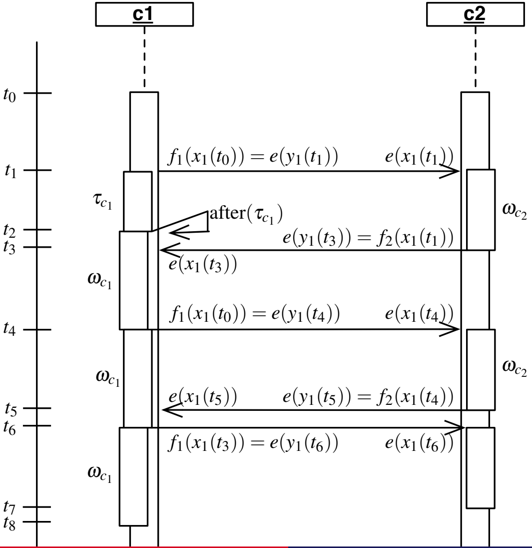
This is **fine** for standalone devices, such as those reading from a piece of hardware, or waiting for human input.

However, for *communicating* devices, this could be disastrous, if the timeouts are not properly set.

Coupled SISO Components

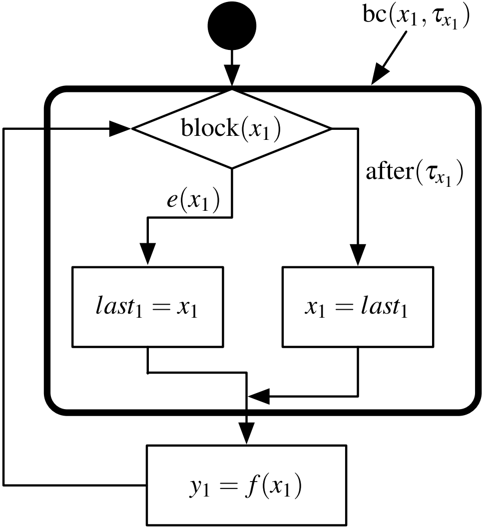


Improperly Timed SISO Coupling



Note that the *output* at time t_4 from component c_1 is the same output from time t_1!! Likewise, for t_3, t_6

Unifying Behavioral Model

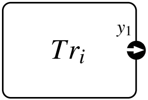


Where bc refers to "Block and Continue."

Trigger Generator

Definition

A *trigger component* produces, at specific *times* or *rates*, a special token whose data is the time at which the token was generated. The structure is a single output port.

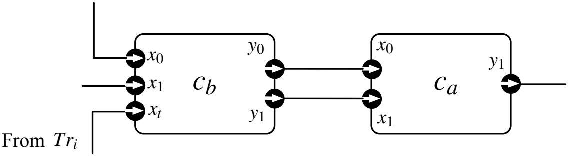


Tokens are produced on the port according to some internal parameters specified for the component, which include wait time, w, start modulus, m, and period, T. Usually, either w or m is specified, and once that time arrives a token is produced, and another token is produced every T seconds.

Buffer Components

Definition

A buffer component, C_b provides an integer number of outputs, j, with inputs $k = j + 1$. The j output ports match to the j inputs of some existing component being buffered, C_a. Values, when received by an input, are queued by C_b.

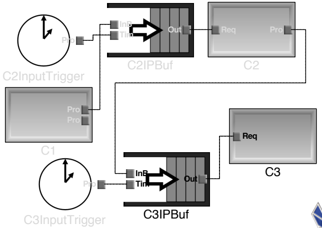


The input x_t subscribes to the single output port of some Tr_i component. When a token is received on x_t, the queued data values are sent to $y_{i \ldots N}$ such that they can be received by c_a.

Transformation: Event→Time

Our transformation modifies an existing graph, and permits existing components to execute with no behavioral changes. The only changes to the system are:

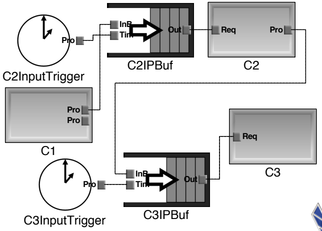
- the topological rewrite; and
- the insertion of new buffered components along with their time-based triggers.



Transformation: Event→Time

Our transformation modifies an existing graph, and permits existing components to execute with no behavioral changes. The only changes to the system are:

- the topological rewrite; and
- the insertion of new buffered components along with their time-based triggers.

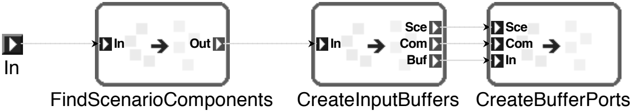


Methodology

Our methodology is as follows:

1. examine an existing component interconnection graph;
2. insert time-triggered buffer(s);
3. insert timed event-generator(s) for each buffer

The rewriting rules are trivial, when specified using the GReAT rewriting language [15].

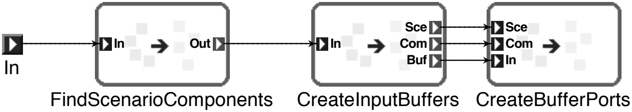


Methodology

Our methodology is as follows:

1. examine an existing component interconnection graph;
2. insert time-triggered buffer(s);
3. insert timed event-generator(s) for each buffer

The rewriting rules are trivial, when specified using the GReAT rewriting language [15].



2. Insert time-triggered buffer components

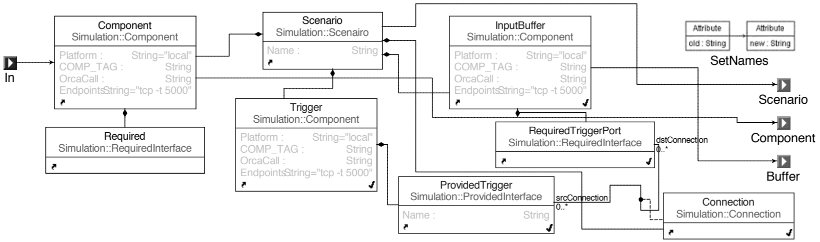


Figure: An example Transformation Specification (2).

3. Create Buffer Ports

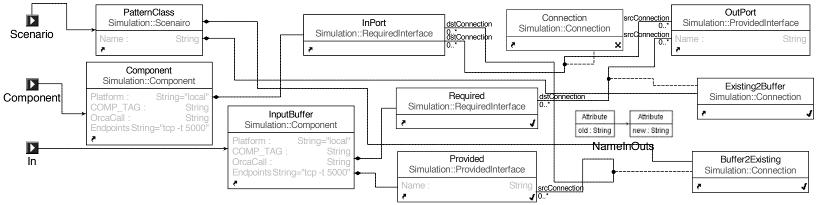
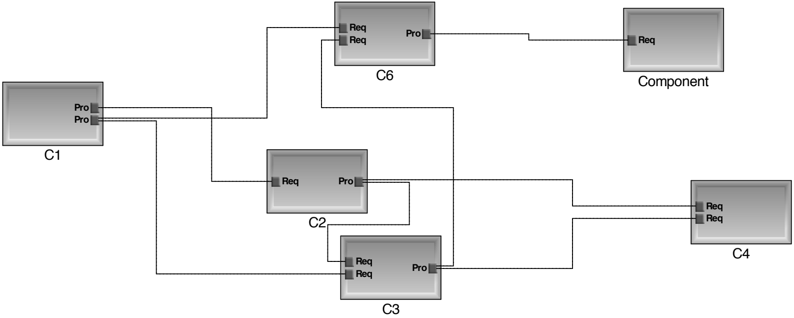
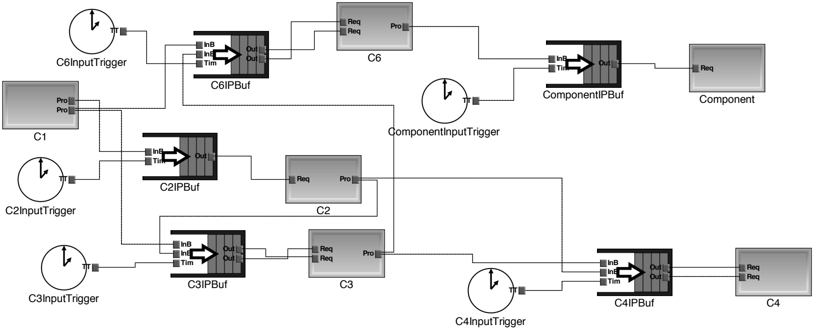


Figure: An example Transformation Specification (3).

A significant example



Transformed Results



Outline I

1. Introduction
   - The Domain
   - Autonomous Ground Vehicles
   - UA Autonomous Ground Vehicles
   - Issues and Solutions
2. Background
   - Publish/Subscribe Methods
   - Time-Triggered Methods
   - Domain Semantics
3. Approach
   - Triggers and Generators: Semantics
   - Transformation Results: How they will look
   - Semantics
   - The Transformation Definitions

Outline II

4. Conclusion
   - Implementation Feasibility
   - Impact on Existing Examples
   - Future/Ongoing Work
   - References

Real-Time Performance

Empirical results from our work shows that using a `pthreads` [16] enabled operating system[4] (but not a real-time OS) results in a variance in expected time generation of approximately 2-3 milliseconds. On a real-time OS[5], the variance is less than 1 ms

[4] Linux flavors Kubuntu and Gentoo were used.
[5] QNX was used for the RTOS.

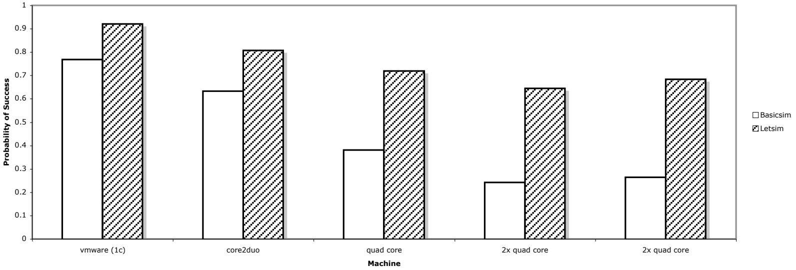


Figure: Related Example: System Failure for Autonomous Vehicle Technology

This example showed the feasibility of using TT buffers/triggers for only a single component. All this work was performed by hand (laborious), now we can move on to see whether a widespread use will equalize success regardless of processor.

Generalization to MIMO Systems

For brevity, we mention that the approach, and issues, for SISO systems can be generalized for MIMO systems (we showed this in the example).

Due to the scope of the workshop, we leave this discussion to future papers in the domain, rather than the language elements of our work.

Autogenerate Buffer Code

Buffers are added as `Component` objects, and the executable for this object can be generically synthesized based on the number of input/output ports, and the semantics chosen. We leave this detailed discussion to future papers, and concentrate instead on their insertion based on context.

Special Semantics for Buffers

In future work, we may provide a special semantics for buffers where the most recent value received (only) on each input port is passed to the output port.

Our current semantics requires that if more than one value (or no value at all) is received by the buffer between triggers, then C_a is responsible for determining whether to use all, or only the most recent, values.

This work supported by:

- Air Force Research Labs, under award #FA8750-08-1-0024, titled "MultiCore Hardware Experiments in Software Producibility."
- National Science Foundation CNS-0930919, "Physical Modeling and Software Synthesis for Self-Reconfigurable Sensors in River Environments"
- Air Force Office of Scientific Research, #FA9550-091-0519, titled "Modeling of Embedded Human Systems."

A. Makarenko, A. Brooks, and T. Kaupp, "On the benefits of making robotic software frameworks thin," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'07) Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware* (E. Prassler, K. Nilsson, and A. Shakhimardanov, eds.), November 2007.

J. Sprinkle *et al.*, "Model-based design: a report from the trenches of the DARPA urban challenge," *Software and Systems Modeling*, 2009.

D. Schmidt *et al.*, "CoSMIC: An MDA generative tool for distributed real-time and embedded component middleware and applications," in *OOPSLA 2002 Workshop on Generative Techniques in the Context of Model Driven Architecture, Seattle, WA*, 2002.

- M. Henning and M. Spruiell, *Distributed Programming with Ice*.
  3.3.1b ed., July 2009.
- Object Modeling Group, *Data Distribution Service for Real-Time Systems, Version 1.2*, formal/07-01-01 ed., January 2007.
- D. Schmidt, D. Levine, and S. Mungee, "The design and performance of real-time object request brokers," *Computer Communications*, April 1998.
- T. Henzinger, B. Horowitz, and C. Kirsch, "Giotto: a time-triggered language for embedded programming," *Proceedings of the IEEE*, vol. 91, pp. 84–99, Jan 2003.

- T. A. Henzinger, C. M. Kirsch, and S. Matic, "Composable code generation for distributed Giotto," *SIGPLAN Not.*, vol. 40, no. 7, pp. 21–30, 2005.
- E. Farcas, C. Farcas, W. Pree, and J. Templ, "Transparent distribution of real-time components based on logical execution time," in *LCTES '05: Proceedings of the 2005 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems*, (New York, NY, USA), pp. 31–39, ACM, 2005.
- H. Kopetz and G. Bauer, "The time-triggered architecture," *Proceedings of the IEEE, Special Issue on Modeling and Design of Embedded Software*, Oct. 2001.

H. Kopetz and G. Grunsteidl, "Ttp - a time-triggered protocol for fault-tolerant real-time systems," in *Proceedings of The Twenty-Third International Symposium on Fault-Tolderant Computing*, vol. FTCS-23, 1993.

S. Matic and T. A. Henzinger, "Trading end-to-end latency for composability," *Real-Time Systems Symposium, IEEE International*, vol. 0, pp. 99–110, 2005.

A. Schuster and J. Sprinkle, "Synthesizing executable simulations from structural models of component-based systems," in *3rd International Workshop on Multi-Paradigm Modeling*, October 2009.

- A. Ledeczi, A. Bakay, M. Maroti, P. Vőlgyesi, G. Nordstrom, J. Sprinkle, and G. Karsai, "Composing domain-specific design environments," *IEEE Computer*, vol. 34, pp. 44–51, November 2001.
- D. Balasubramanian, A. Narayanan, C. van Buskirk, and G. Karsai, "The graph rewriting and transformation language: GReAT," *Electronic Communications of the EASST*, vol. 1, 2006.
- B. Lewis and D. J. Berg, *Multithreaded Programming With PThreads*.
  Prentice Hall PTR, 1997.

We're always looking for good graduate students.

http://ece.arizona.edu/