Improving and Evaluating the Automatic Analysis of Implicit Invocation Systems

Jeremy Bradbury

bradbury@cs.queensu.ca

Juergen Dingel

dingel@cs.queensu.ca

Applied Formal Methods Group

Software Technology Laboratory

Queen's University, Kingston, Canada

http://www.cs.queensu.ca/~stl/afmg/

(ESEC/FSE 2003)

SUPPORTED BY NSERC CRSNG

Overview

- Background
  - Architectural Styles
  - Implicit Invocation
- Research Problem
- Existing Analysis Approach
- Improvements
- Evaluation
- Conclusions

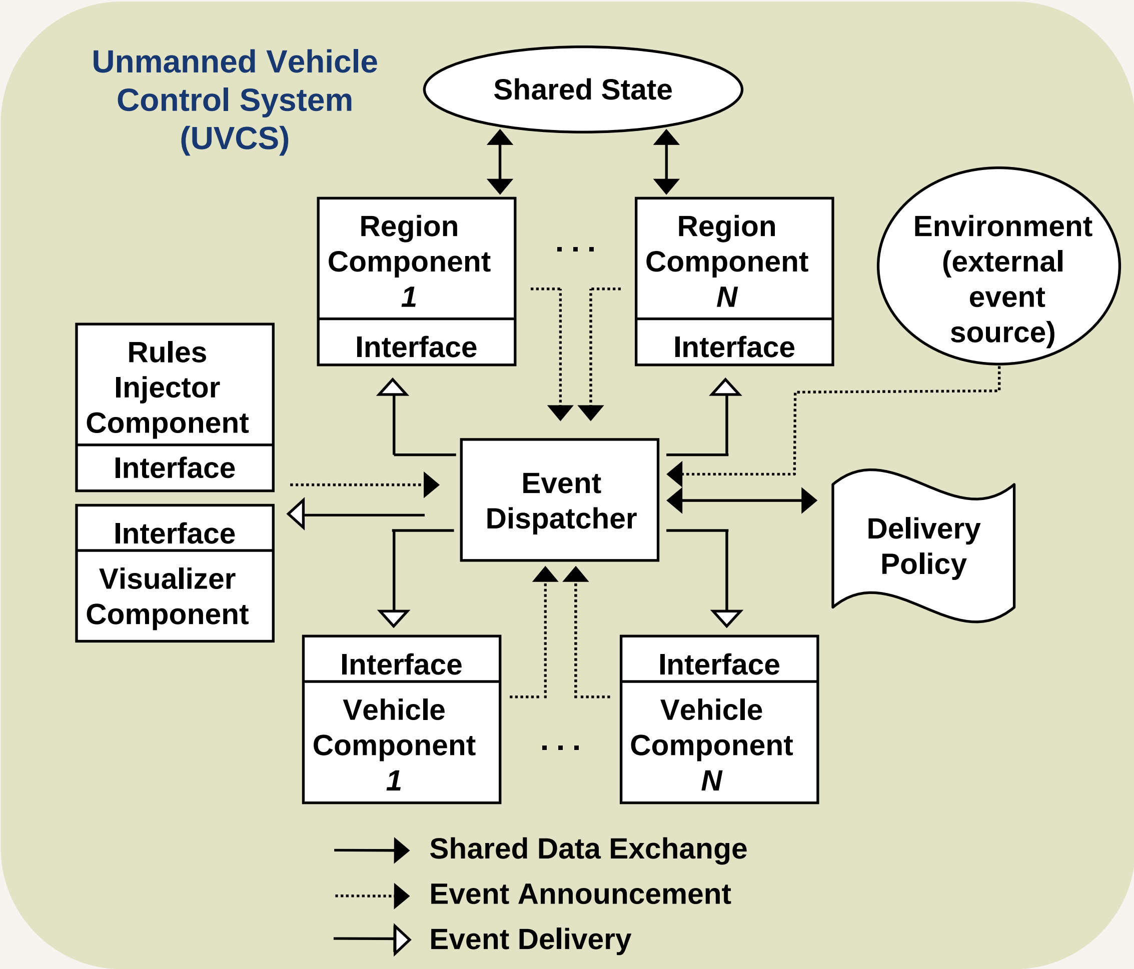
Architectural Styles

- A common framework consisting of:
  - *Components:* often encapsulate information or functionality
  - *Connectors:* describe the communication between components

- Specifically we focus on the event-based architectural style – Implicit Invocation (II)
  - Popular architectural style that is becoming more widely used
  - Challenging to reason about
  - Challenging to model as a manageable finite state machines

Implicit Invocation

- Implicit invocation systems consist of 6 parameters [GK00]:
  - Components
  - Events
  - Event-method bindings
  - Event delivery policy
  - Shared variables
  - A concurrency model



[GK00] D. Garlan and S. Khersonsky. Model checking implicit-invocation systems. In *Proc. of the 10th Int'l Workshop on Software Specification and Design*, Nov. 2000.

Past Work

- Research Problem: the application of model checking to software
  - Semantic gap
  - State explosion problem

- Garlan & Khersonsky address this problem in the context of implicit invocation [GK00], [GKK03].
  - Reduced semantic gap by developing an XML-based translation framework.

[GK00] D. Garlan and S. Khersonsky. Model checking implicit-invocation systems. In *Proc. of the 10th Int'l Workshop on Software Specification and Design*, Nov. 2000.

[GKK03] D. Garlan, S. Khersonsky, and J. S. Kim. Model checking publish-subscribe systems. In *Proc. of the 10th Int'l SPIN Workshop on Model Checking of Software*, May 2003.

Past Work

1. Modeling
   - Model the system as a finite state machine.

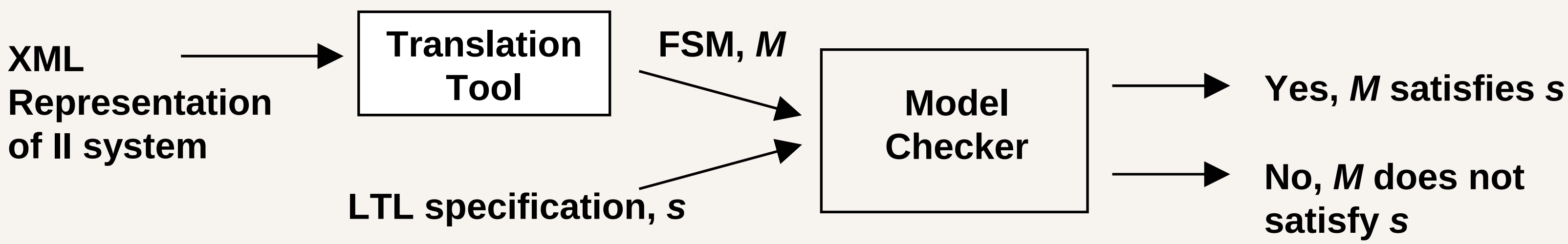
   G & K Approach

2. Specification
   - Express the specification that the system should satisfy as a temporal logic statement.

   Linear Temporal Logic (LTL)

3. Verification
   - Input the model and the specification to a model checker.

   Cadence SMV Model Checker



Past Work

- **Components**
- **Events**
- **Event-Method Bindings**
- **Event Delivery Policy**
- **Shared State**
- **Concurrency Model**

Components consist of:

- Local variables
- Accessible global variables
- Announced events
- Methods
  - Conditionals
  - Assignment statements
  - Event announcement

Past Work

- **Components**
- **Events**
- **Event-Method Bindings**
- **Event Delivery Policy**
- **Shared State**
- **Concurrency Model**

A list of identifiers

Past Work

- Components
- Events
- Event-Method Bindings
- Event Delivery Policy
- Shared State
- Concurrency Model

A set of pairs (e_i, m_i) where event e_i invokes method m_i

Past Work

- **Components**
- **Events**
- **Event-Method Bindings**
- **Event Delivery Policy**
- **Shared State**
- **Concurrency Model**

A short list of fixed policies:

- Immediate invocation of subscriber methods
- Random delay before announcement to subscriber components

Past Work

- **Components**
- **Events**
- **Event-Method Bindings**
- **Event Delivery Policy**
- **Shared State**
- **Concurrency Model**

A set of pairs (id_i, t_i) where a shared variable represented as identifier id_i has type t_i

Past Work

- **Components**
- **Events**
- **Event-Method Bindings**
- **Event Delivery Policy**
- **Shared State**
- **Concurrency Model**

One of a finite set of models:

- Single thread of control for all components
- Separate threads of control

Our Work

Specific Contributions

- **Improvements**
  - Enhanced events
  - Expression of more complex delivery policies
  - Dynamic event-method bindings
- **Evaluation**
  - Used extended approach on optimized systems with “real-world” significance

Enhanced Events

- Events parameter extended to allow for the inclusion of data.
- That is an event is no longer declared as an identifier *e* but by an identifier and a set of parameters.

```
VehicleInfoEvent
    •vehicleID, 1..2
    •xPosition, 1..5
    •yPosition, 1..5
```

Dynamic Delivery Policies

- Propositional logic policies of the form:

$$guard_1 \Rightarrow deliveryExpr_1,$$
$$guard_2 \Rightarrow deliveryExpr_2,$$
$$. \; . \; .$$
$$guard_n \Rightarrow deliveryExpr_n$$

- In an II system, the dispatcher executes the delivery policy by
  - determining the smallest value of i such that guard_i is true
  - making state changes to ensure that the corresponding delivery expression, deliveryExpr_i, is true.

Dynamic Delivery Policies

Examples:

1. **An immediate policy**:
   - If an *envImmediate* event is announced then *VehicleInfo* and *RulesInfo* events are delivered at the same time.

envImmediateAnnounced
⇒ deliver VehicleInfo ∧
deliver RulesInfo

2. **A priority-based queue policy:**
   - *VehicleInfo* events are delivered before *RulesInfo* events.

VehicleInfoAnnounced
⇒ (deliverVehicleInfo ∨
¬deliver VehicleInfo)
∧ ¬deliver RulesInfo,

RulesInfoAnnounced > 0
⇒ ¬deliver_VehicleInfo
∧ (deliver_RulesInfo ∨
¬deliver_RulesInfo)

Dynamic Event-Method Bindings

- A dynamic binding is a binding that can be modified at run-time.
- In our approach:
  - Dynamic bindings include an additional boolean attribute to specify whether it is active or inactive.
  - The status of a binding can be changed by the component whose method is invoked by the binding.

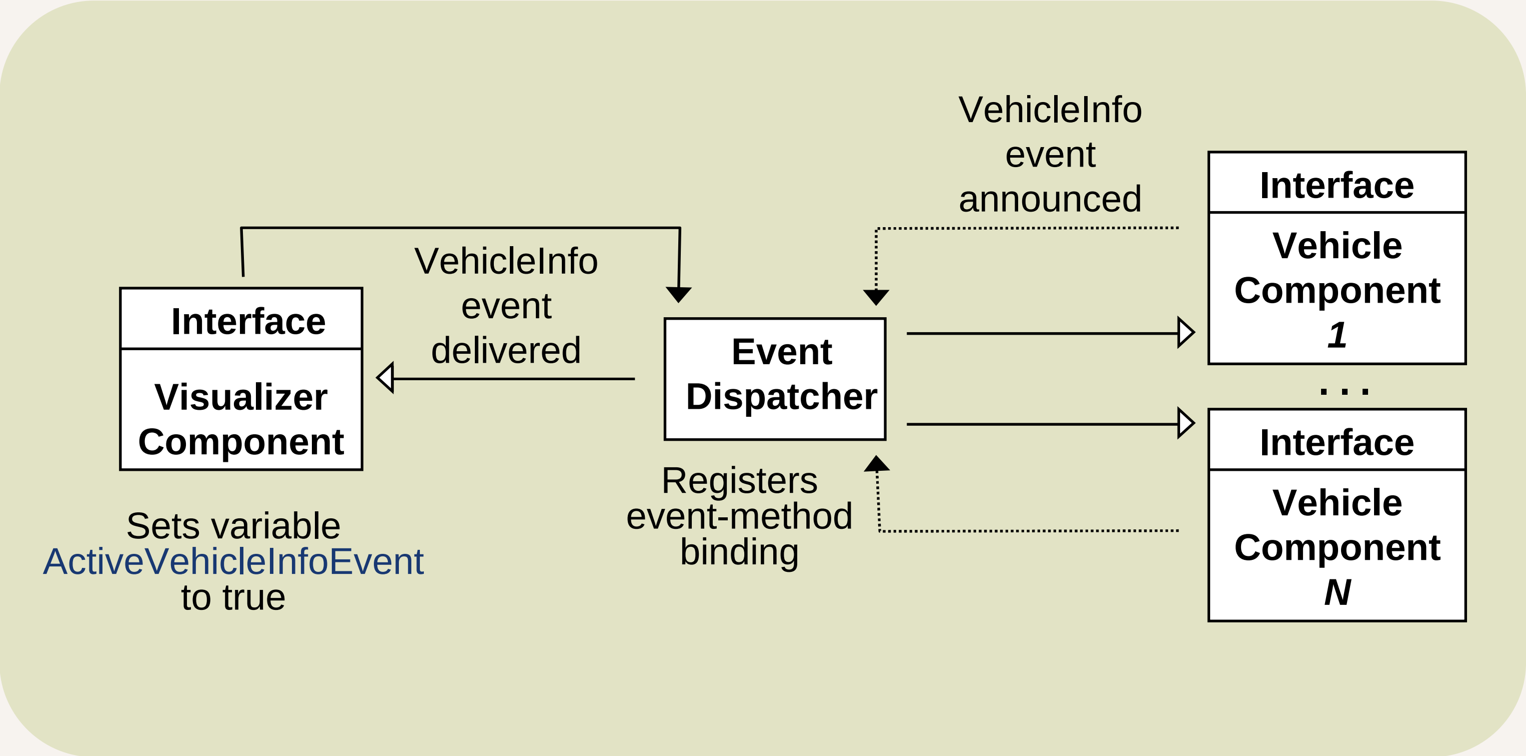
Dynamic Event-Method Bindings

- Modeling of the 4 basic types of dynamic change:

1. **Component Addition**
   A component receiving no events activates bindings to receive events.
2. **Component Removal**
   A component deactivates the bindings for all of the events to which it subscribes.
3. **Connector Addition**
   A component activates the binding for an event.
4. **Connector Removal**
   A component deactivates the binding for an event.

Dynamic Event-Method Bindings

<u>Example</u>: Connector addition to allow *Visualizer* component to receive *VehicleInfo* events.



Evaluation

- Used model checking in standard way to check liveness and safety properties.
- Modeled II systems that have high availability or are safety-critical.
- Evaluated variations of three systems:
  - Set-counter example[1]
  - Active Badge Location System (ABLS)
  - Unmanned Vehicle Control System (UVCS)

[1]Set-counter example not included in ESEC/FSE 2003 Paper

Evaluation

- Unmanned Vehicle Control System (UVCS)
  - Verified **4** LTL properties using **3** different delivery policies.
    - Collision avoidance guarantee

      G (~(Vehicle1.currRegion = Vehicle2.currRegion)
      | ~(Vehicle1.xpos = Vehicle2.xpos)
      | ~(Vehicle1.ypos = Vehicle2.ypos))

    - Achievement of a long term goal

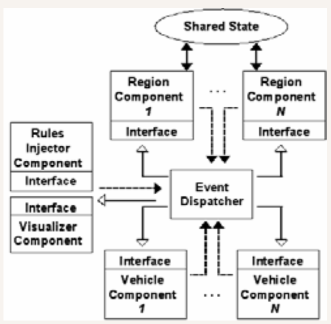
      F ((Vehicle1.currRegion = Vehicle1.destRegion)
      & (Vehicle1.xpos = Vehicle1.destxpos)
      & (Vehicle1.ypos = Vehicle1.destypos))

Evaluation

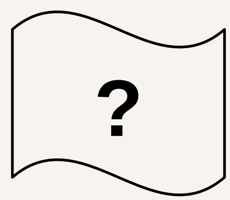
- Optimizations
  - Cone of influence reduction
  - Data abstraction
  - Reduction of non-determinism
  - Correctness preserving transformations (e.g. combining components or events)
- Optimized model separately for each property

Evaluation

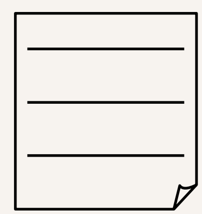
- Used iterative model checking to decide the appropriate delivery policy for a given system
  - Able to uncover and fix conflicts between the requirements and the delivery policy.



+



satisfies



Architecture

Delivery Policy

Requirements

Conclusions

- Improvements to event mechanisms were used to model a more interesting class of systems.
- Evaluation demonstrated that these systems can be modeled more naturally in this approach.
- However, even with exhaustive optimizations model checking in this context is still limited.
- Our solution to this problem was using partial system models.
    - II systems partition naturally across component boundaries.

Future Directions

Further Improvements…

- Currently exploring the model checker Bogor[1] as an alternative to Cadence SMV.
  - High-level language
  - More customization possible
- Generalize approach to include publish-subscribe systems that have, for example:
  - Multiple distributed dispatchers

Further Evaluation…

- Examine use of iterative analysis in determining the appropriate definition of system parameters.

---

[1]Developed at Kansas State University

Improving and Evaluating the Automatic Analysis of Implicit Invocation Systems

Jeremy Bradbury
bradbury@cs.queensu.ca

Juergen Dingel
dingel@cs.queensu.ca

Applied Formal Methods Group
Software Technology Laboratory
Queen's University, Kingston, Canada
http://www.cs.queensu.ca/~stl/afmg/

(ESEC/FSE 2003)

SUPPORTED BY NSERC CRSNG