Model Checking Publish-Subscribe Systems

David Garlan

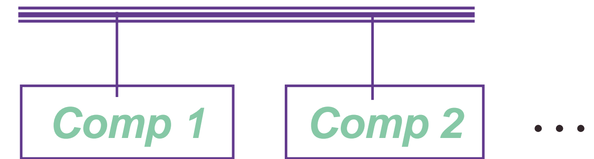
Jung Soo Kim

Carnegie Mellon University

Publish-Subscribe Systems

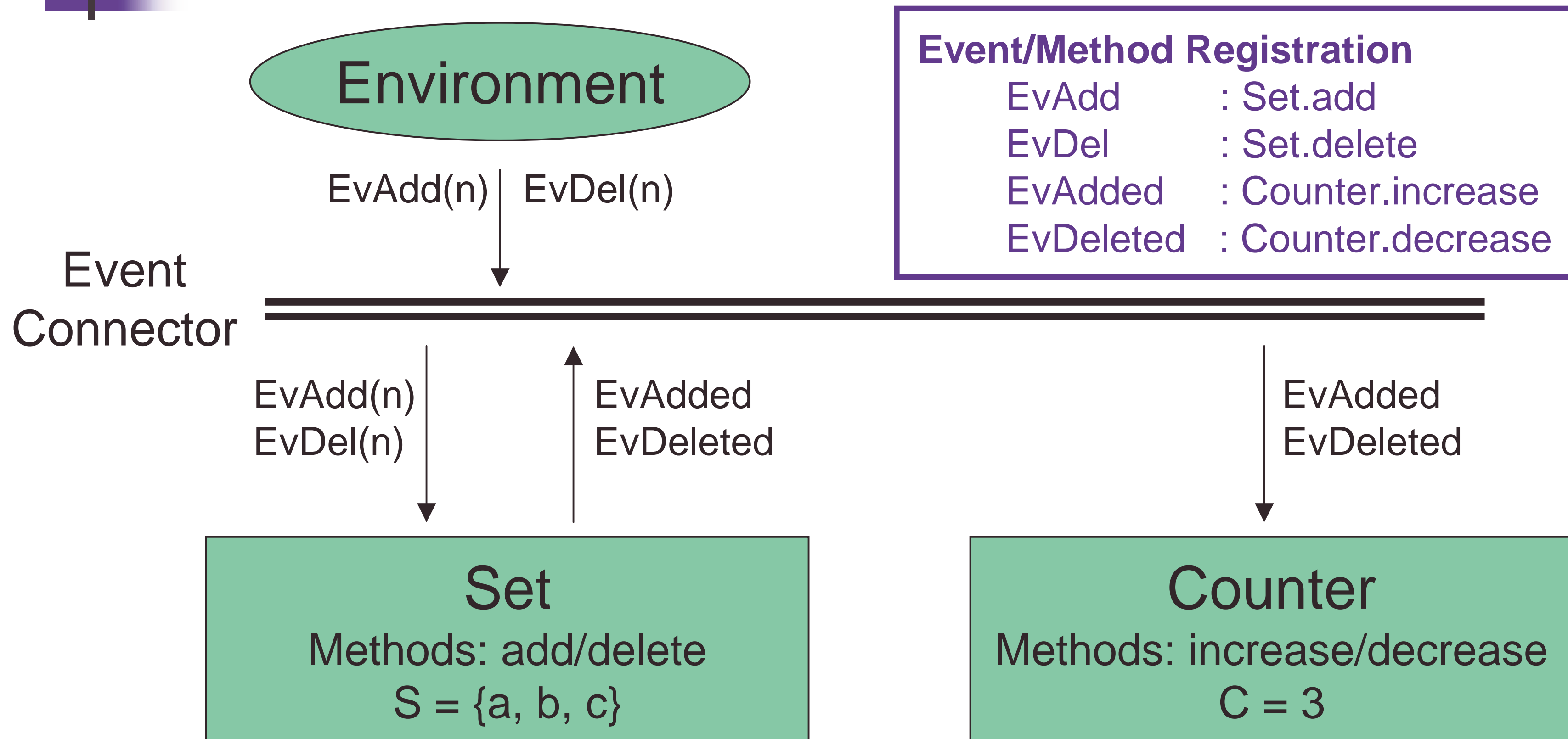
- The Problem:
  - Publish-subscribe systems are ubiquitous
    - CORBA, JMS, Visual Basic, MVC, …
  - But we don't have good ways to express properties or to reason about their satisfaction
- Approach:
  - Logical framework for reasoning about pub-sub systems
    - Rely-guarantee approach
    - Temporal logic used for property specifications
  - Model checking tool tailored to pub-sub systems
    - built-in checks for common properties
    - tailorable to specific variant of pub-sub infrastructure

Publish-Subscribe Model



- Components
  - Have local state and methods
  - Announce (publish) events
  - Register for (subscribe to) events by indicating a method to be invoked when event is announced
- Events
  - The unit of communication between components
  - May carry additional information as parameters
- Event Connector (Dispatcher)
  - Maintains event-method registrations
  - Invokes registered methods when an event is announced

Example: Set and Counter



Establish the invariant |S| = C

Advantages

- Loosely coupled components
  - A component that announces an event does not know (and does not need to know) the consequence of announcement
- Improves system maintainability
  - Easy to add and remove components
  - Easy to modify individual components

Disadvantages

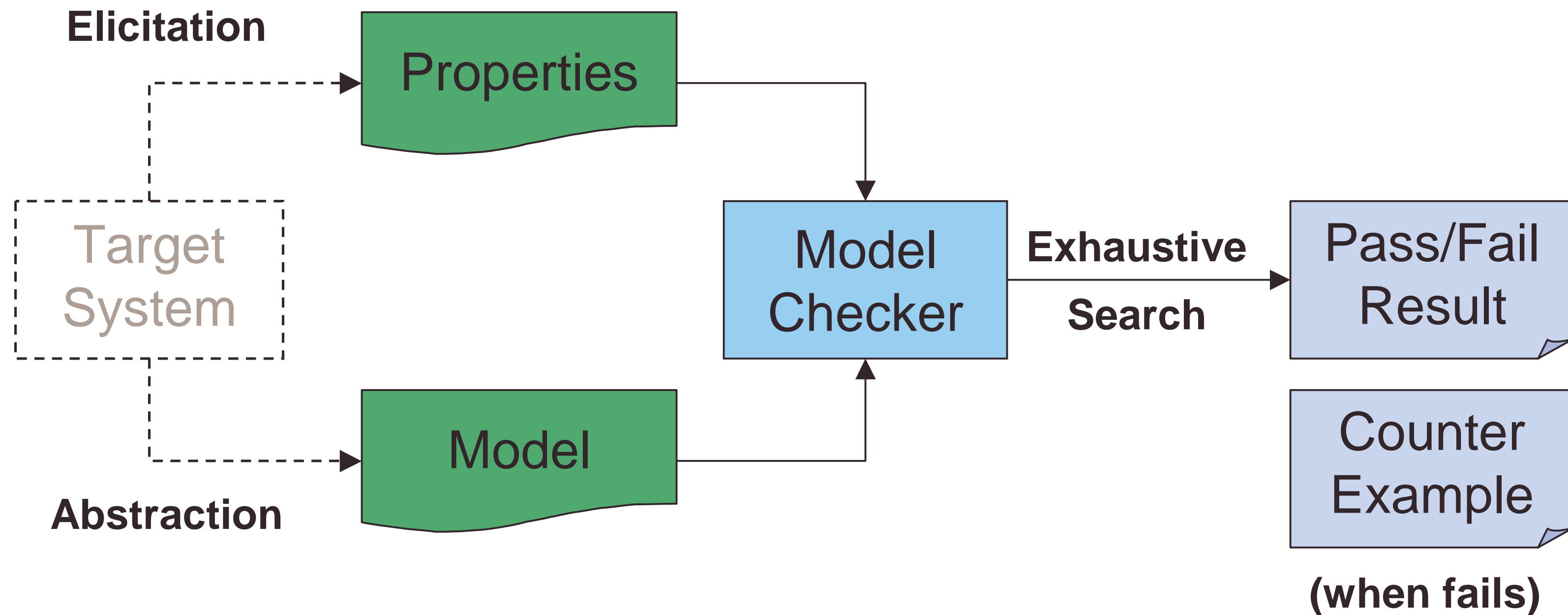
- Lots of inherent non-determinism
  - Order of events to deliver
  - Order of invocation of multiple event recipients
  - Timing of in-transit events
  - Order of completion of event handling
- Burden of correctness falls on system integrator
  - Difficult to guarantee intended system behavior
  - Difficult to choose the right event infrastructure
    - many possible dispatch policies, concurrency disciplines, synchronization schemes

Difficult Questions

- What do we want to say about such systems? What's an "invariant" and how to check it?
- Do the components announce the events that they should announce?
- What will be the effect of announcing a particular event?
- If a new component is added, will it break what is already there?
- Can a different event infrastructure be used without causing any problem?

Possible Solution: Model-Checking

Typical model-checking process

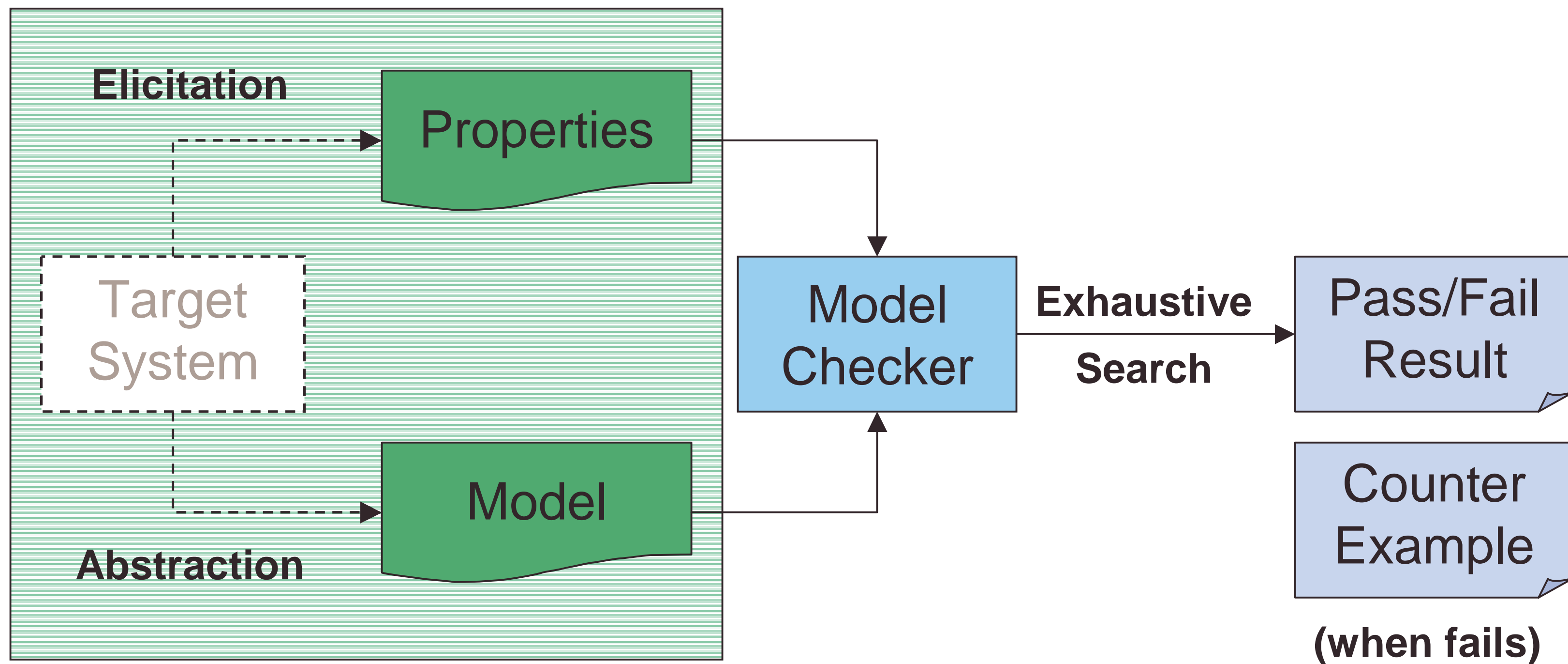


Pros and Cons

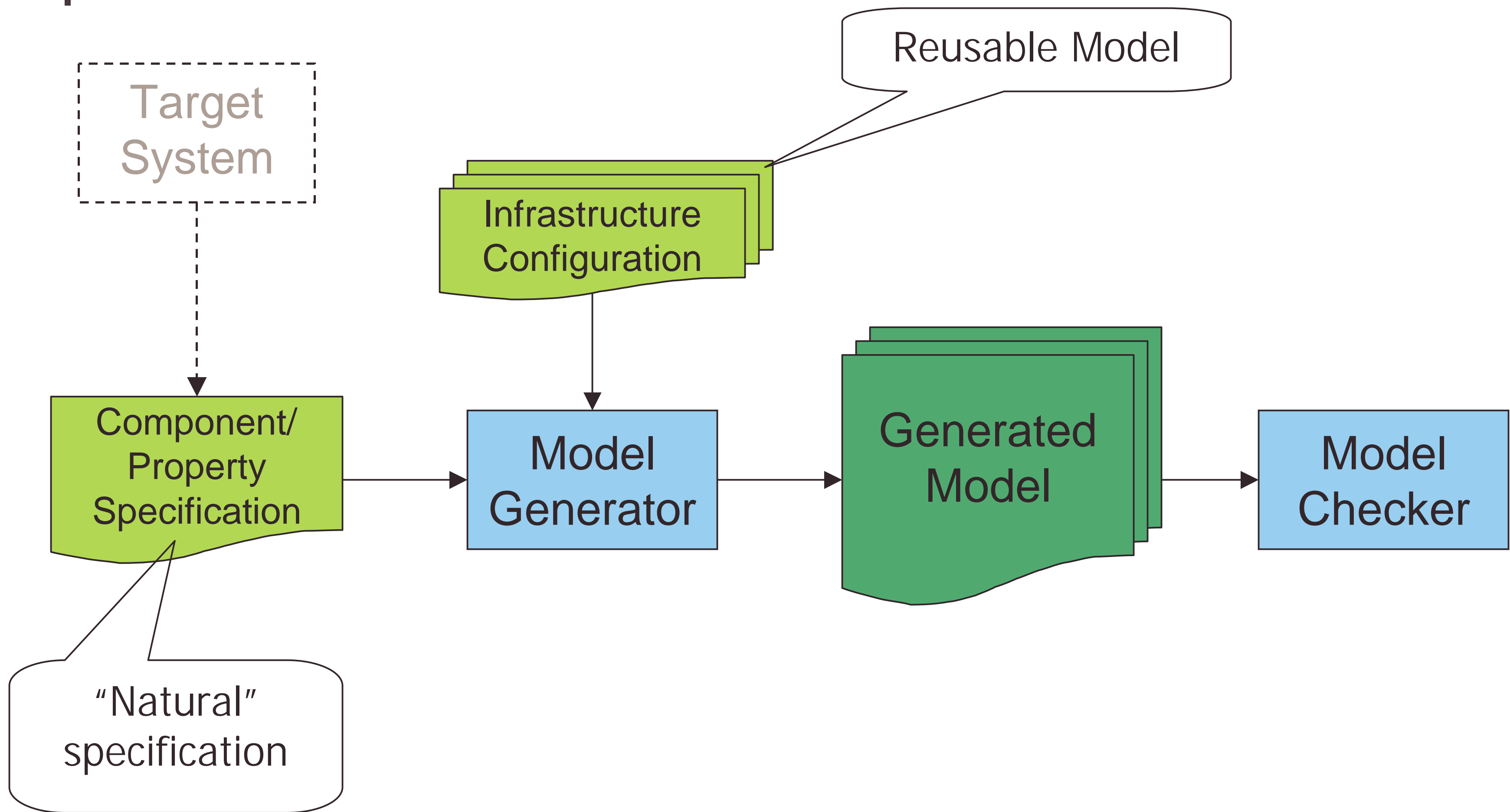
- The Good
  - Exhaustive search over the state space
  - Counter-example generation
  - Mature theoretical foundations for reasoning
- The Bad
  - State explosion
  - Steep learning curve
  - Hard to construct a good model
  - Hard to specify properties of interest
- The Ugly
  - "Pass" does not mean that everything is all right
  - Difficult to maintain and reuse the existing model

Focus of Research

Ease the burden of constructing models and properties by providing domain-specific model-checking front end.



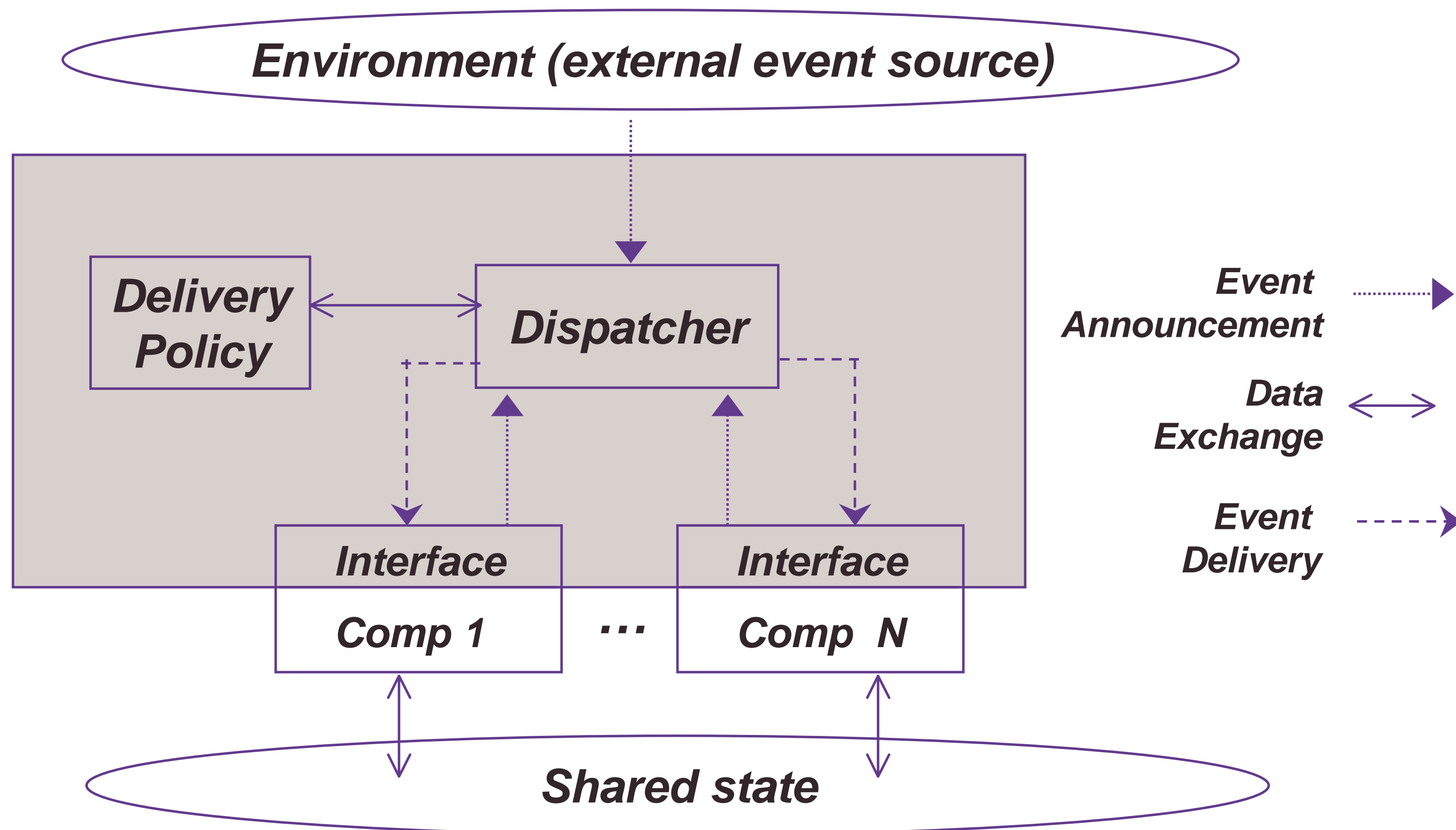
Approach



Innovative Features

- Automatic model generation of the pub-sub communication infrastructure
  - Reduces the cost of constructing models for publish-subscribe systems
  - Reduces model errors
- Parameterized communication infrastructure
  - Allows easy exploration of alternatives
- User-friendly component/property specification
  - Eases specification of component behavior

Reusable Infrastructure Model



Infrastructure Design Space

- Announcement options
  - Asynchronous: immediate return from announcement
  - Synchronous: wait for complete event handling
- Dispatch order
  - FCFS, Random, Prioritized
- Delivery options
  - Guaranteed, Lossy
- Startup
  - Synchronous, Random
- Concurrency options
  - Single thread of control
  - Separate threads of control
    - Single thread per component
    - Multiple threads per component
      - Concurrent invocation of different methods
      - Concurrent invocation of any method

Initial Results

- Experimented with several systems
  - Toy examples, such as set-counter
  - Distributed resource management
- Reduced effort for model generation
  - Typical reduction: 80% of the model automatically generated, although depends on number and size of components
  - E.g., for set-counter 147/184 lines

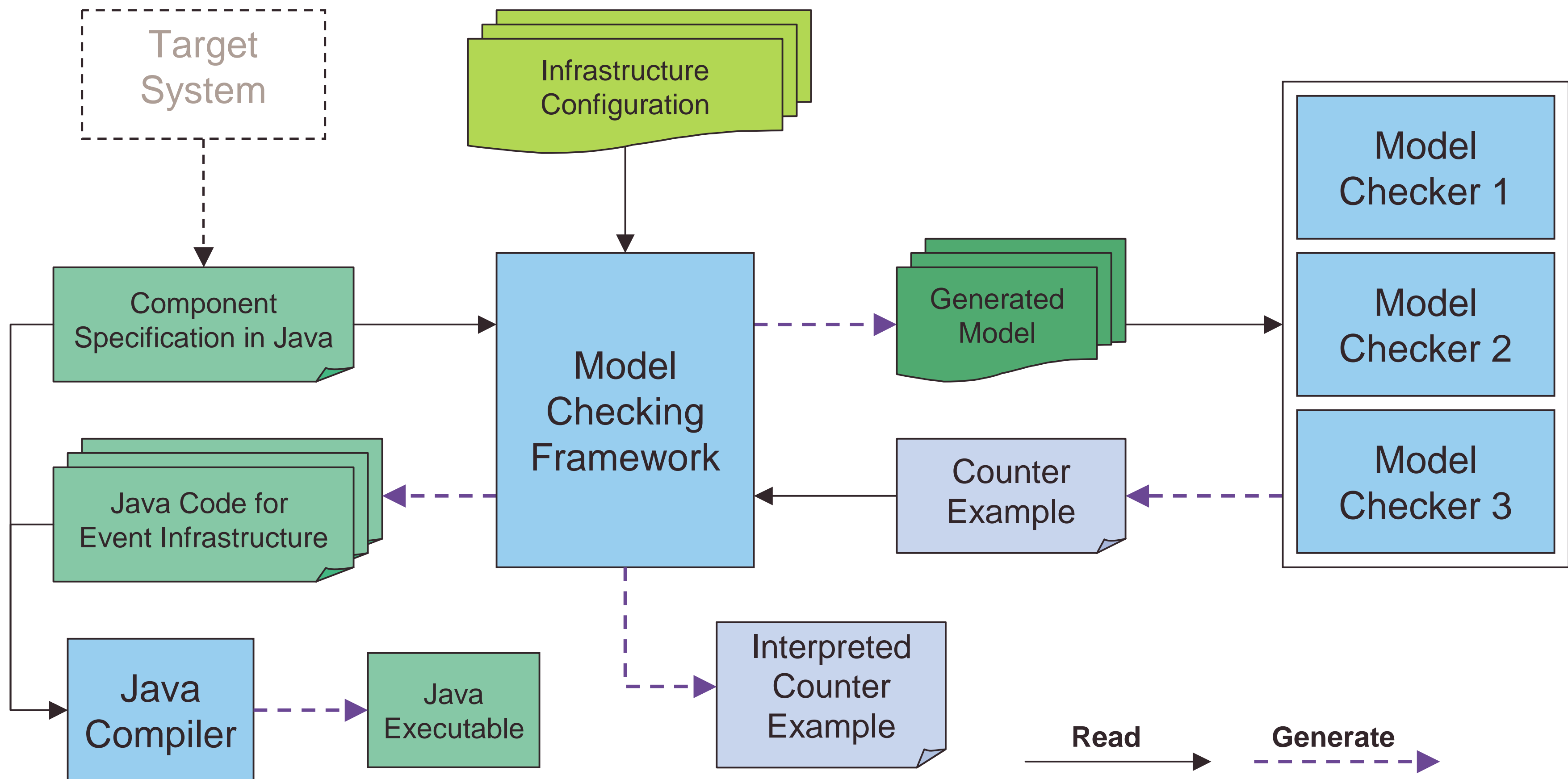
Limitations

- Component specification in XML
- Properties specified in LTL
- No support for dynamism
- No support for synchronous start up
- Subset of infrastructure options supported
- Counter examples in terms of lower-level model

Current Work

- Component specifications in stylized Java
  - More intuitive link to implementations
  - Can execute the specifications
- More complete enumeration of communication alternatives
  - Formal model of design space (Z & FSP)
- Support for dynamism
  - Add/remove components/registrations
- Support for alternative startup policies
- Retargeted to Spin
  - Better support for communication/dynamism

New Framework



Component Specification: Old

```
<component name = "Counter">
    <local-var name = "counter" type = "-2..5"> 0 </local-var>

    <method name = “increase">
      <statement>
        <assignment var-name = "counter"> counter + 1 </assignment>
      </statement>
    </method>

    <method name = “decrease">
      <statement>
        <assignment var-name = "counter"> counter - 1 </assignment>
      </statement>
    </method>
  </component>
```

Component Specification: Old

```
<event name = “EvAdded"/>
<event name = “EvDeleted"/>

<component-instance component-name = "Counter"
    instance-name = "theCounter"> />

<event-binding event-name = "EvAdded">
    <method-binding instance-name = "theCounter"
        method-name = “increase"/>
</event-binding>

<event-binding event-name = “EvDeleted">
    <method-binding instance-name = "theCounter"
        method-name = “decrease"/>
</event-binding>
```

Component Specification: New

```
class Example extends PubSub {

    class EvAdd   extends Event { int element; }
    class EvAdded extends Event {}
    ...

    class Set extends Component {
        boolean[] elements = new boolean[MAX_ELEMENT];

        void add (EvAdd ev) {
            if (element[ev.element] == false) {
                element[ev.element] = true;
                announce (“Added");
            }
        }
        ...
    }

    Example () {
        int cid;
        cid = create(“Set");
        subscribe(“EvAdd”, “add”, cid);
        ...
    }
}
```

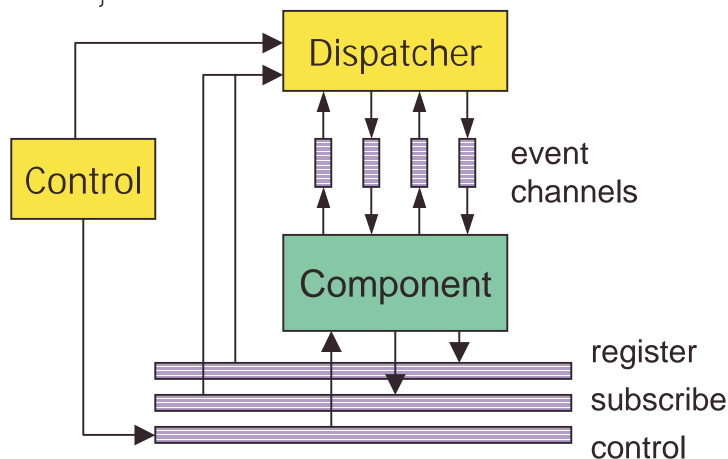
Generated SPIN Model

```
mtype = {ev_added, ev_removed, ...};

#define Counter_proc_increase_id 0
#define Counter_proc_decrease_id 1

inline Counter_proc_increase () {
    counter ++;
    receive_ack!param.eid;
}

inline Counter_proc_decrease () {
    counter --;
    receive_ack!param.eid;
}
```




```
proctype Counter (chan register, subscribe, control) {
    byte counter;

    chan announce_req = [1] of {mtype, attr};
    chan announce_ack = [1] of {int};
    chan  receive_req = [1] of {int, mtype, attr};
    chan  receive_ack = [1] of {int};

    mtype event;
    attr  param;

    register!ps_join (_pid, announce_req, announce_ack,
                      receive_req, receive_ack);

    control?ps_start;

    do
    ::  receive_req?Counter_proc_increase_id(event, param)
        -> Counter_proc_increase();
    ::  receive_req?Counter_proc_increase_id(event, param)
        -> Counter_proc_decrease();
    od;

    register!ps_leave (_pid, announce_req, announce_ack,
                       receive_req, receive_ack);
}
```

Sample Property: Old

Check the “invariant” $|S| = C$

```
ConsiderateEnvironment :
assert (G (~disp.evtBuffOverflow & ~updateInvQueue.error));

StoppingEnvironment :
assert(F G (~announceUpdt));

CounterCatchesUp :
assert(G F (set.setSize = counter.count));

using StoppingEnvironment, ConsiderateEnvironment
prove CounterCatchesUp;
```

Sample Property: New

Check the "invariant" $|S| = C$

```
invariant (quiescent() -> Set.size = Counter.counter);
```

Implementation Techniques

- Non-determinism
  - Workaround

```
switch (random(3)) {
    case 0: /* do something */ break;
    case 1: /* do something */ break;
    case 2: /* do something */ break;
}
```

- Operations for event communication
  - Reside in super classes

```
Class PubSub {
    Void subscribe (String, String, int) { ... }
    ...
}
```

In-Progress

- Property specification
- Counter example explanation
- Case studies
  - NASA MDS

Other Related Work (Posters)

- Architecture-based run time adaptation
  - Formal architectural models used to monitor and repair running systems
- Formal architecture design tools
  - Enforcing constraints of a style
  - NASA MDS case study
  - Ford Motor Company MSE project

The End

On-going Work

- Better linkage to implementation
  - Stylized use of programming language for specification
  - Generates executable system as well as a checkable model
  - Counter example explanation
- Property specification primitives and templates
  - Hide the details of generated model
  - Provide many of the common sanity checks
  - Move towards *push-button* tools

Current Work – (cont)

- New specification capabilities
  - Dynamic component creation and binding
  - Real-time properties

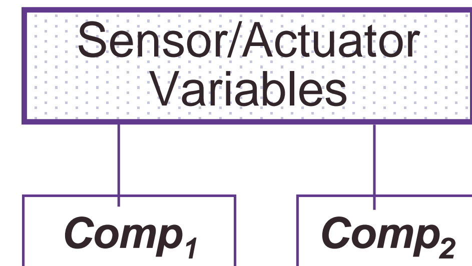
Examples

- Set-Counter
  - Set (S) has operations insert/delete
  - Counter (C) has operations inc/dec
  - Establish "invariant" |S| = C

- Distributed Simulation (HLA)
  - Arbitrary number of simulations publish values of objects that they simulate
  - Run-time infrastructure (RTI) maintains state (e.g., ownership of objects), mediates protocols of interaction
  - Many invariants (e.g., each object is owned by a single simulation)

More Examples (State-based duals)

- Shared-variable triggered systems
  - Aka "continuous query" systems
  - State changes trigger computations
  - Components read/write shared variables, but are otherwise independent



- Real-time periodic tasks
  - Tasks placed in periodically-scheduled buckets
  - Tasks consume values of certain variables; produce values of other variables
  - Tasks within bucket must complete before bucket period