Model Checking Implicit-Invocation Systems: An Approach to the Automatic Analysis of Architectural Styles

Jeremy Bradbury
M.Sc. Thesis Defense
Department of Computing
and Information Science
Queen's University
May 15, 2002

Supervised by Juergen Dingel

Overview

- Architectural Styles
- Implicit-Invocation
- Model Checking
- Motivation
- Our Approach
- Evaluation of Our Approach
- Contributions, Conclusions and Future Work

Architectural Styles

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

- **"...a vocabulary of components and connector types, and a set of constraints on how they can be combined."**

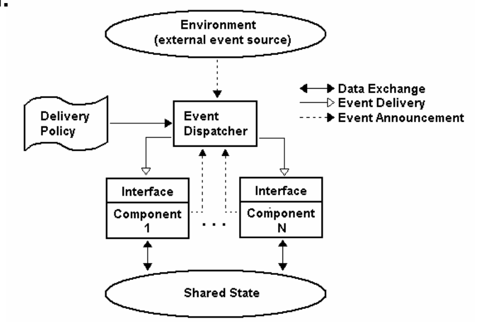
  \- D. Garlan & M. Shaw

- **"…a collection of rules that constrain the topology of an architecture and often also the behaviour of its components."**

  \- D. Jackson

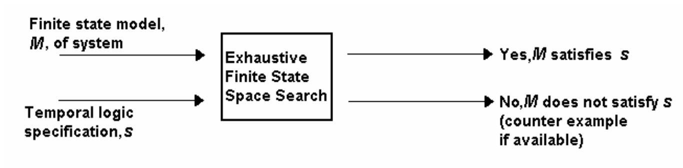
Implicit-Invocation

- **Implicit-invocation systems consist of 6 parameters: components, shared variables, events, event-method bindings, an event delivery policy, and a concurrency model.**



Model Checking

1) **Modeling.**
   – Model the system as a finite state machine.
2) **Specification.**
   – Express the specification that the system should satisfy as a temporal logic statement.
3) **Verification.**
   – Input the model and the specification to a model checker.



Model Checking

- **The state space in the context of model checking is a Kripke structure or a Labeled Transition System (LTS).**
- **An LTS is a four tuple $M = (S, S_0, R, L)$ where**
  – S is the finite set of states in the system
  – S_0 is the set of initial states
  – R is a total transition relation that defines all transitions between states in S. The relation is total because for every s in S, there exists a t in S such that $R(s,t)$ where $R \subseteq S \times S$.
  – $L: S \rightarrow 2^{AP}$ is a labeling function for every s in S. Each state is labeled with the atomic propositions (AP) that are true in that state. Specifically, for every p in AP, s in S we have p in $L(s)$ if and only if p is true in s.

Model Checking

- **Linear Temporal Logic (LTL) is a linear-time modal logic**
- **In LTL, operators describe events along a single computation path**

LTL Operator	Definition
X φ	In the next state φ holds.
G φ	In all future state φ holds. φ holds globally.
F φ	In some future state φ holds. φ holds eventually.
φ_1 U φ_2	φ_1 holds at least until φ_2 does.

Motivation

Why study formal methods (specifically model checking) in the context of software systems?

– As software systems become more integrated into our daily lives, our tolerance for failure decreases – in many cases failure has become unacceptable.

– Software is now widely used in safety-critical systems (nuclear power plants, air traffic control systems, medical instruments, weaponry, embedded systems running in aircraft or automobiles).



Motivation

Why has model checking not been successful when applied to software?

1) Semantic Gap: There is a large gap between the artifacts produced by software developers and the artifacts that are accepted by model checkers.

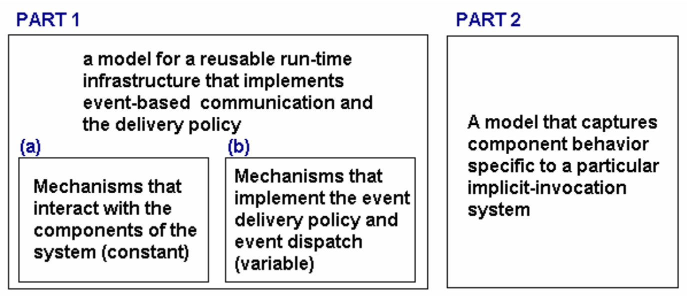
2) State Explosion Problem: Variables often range over infinite or large domains. The state space grows exponentially with the number of parallel processes in the system.

Why implicit-invocation systems?

– Popular architectural style that is becoming more widely used
– Challenging to reason about
– Challenging to model as finite state machines

Our Approach

- **Development of a reusable parameterized model.**

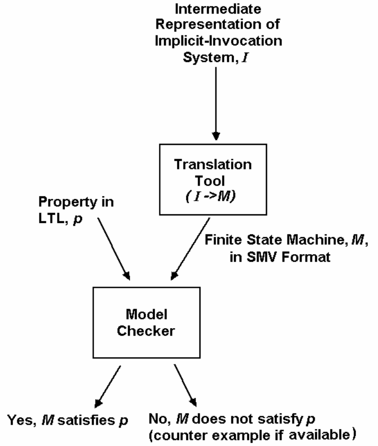


Our Approach

Enhancements:

1) **Representation of event data**
   – Allow events to encapsulate data

2) **Modified event delivery policy representation**
   – Increased level of expressiveness and flexibility



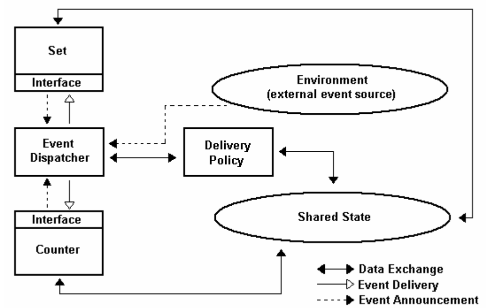
Evaluating Our Approach

- **Criteria for models of implicit-invocation systems**
  – Significant size
  – Real-world applicability
  – Interesting behaviour

- **Criteria for properties**
  – LTL is an expressive representation
  – Discuss in terms of safety-liveness taxonomy:
    - *Safety*: something “bad” never happens during execution.
    - *Liveness*: something “good” happens during execution (Could happen infinitely often, once, always, etc.)

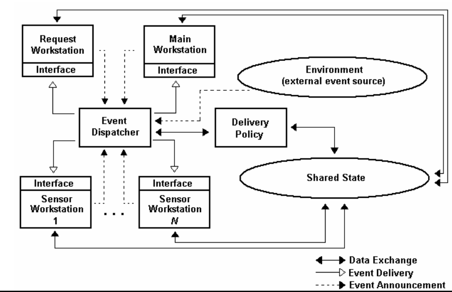
Evaluating Our Approach: Set and Counter Example

- **Relatively small - 2 components and 4 event types**
- **Primary example used by Garlan and Khersonky to test their finite model building technique**
- **Included to provide a comparison/contrast with Garlan and Khersonsky technique.**



Evaluating Our Approach: Active Badge Location System (ABLS)

- **An electronic tagging system for locating people in a localized setting**
- **Innovative alternative to conventional pager system**
- **Contains 3 types of processes: Active Badges, sensors, and a main workstation**
- **Supports 5 commands: Find, With, Look, Notify, History.**



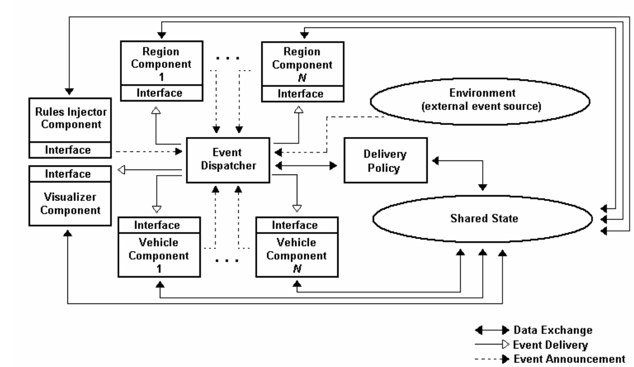
Evaluating Our Approach: Active Badge Location System (ABLS)

- ***Find Event Correctness In Next State:***
  - Liveness property to analyze how variations in delivery policies affect the timing of event delivery.
  - Additionally, answers questions such as “Does a given delivery policy allow us to provide guarantees about the timing of the delivery of a given event?”

G((((Master.state = sendFindResults) & X(Master.database[2][0] = 1))
-> X X(Request.invoke_receiveFindResult_via_FindResult.locationID = 1))
&(((Master.state = sendFindResults) & X(Master.database[2][0] = 0))
-> X X(Request.invoke_receiveFindResult_via_FindResult.locationID = 0))
&(((Master.state = sendFindResults) & X(Master.database[2][0] = -1))
-> X X(Request.invoke_receiveFindResult_via_FindResult.locationID = -1)))

Evaluating Our Approach: Unmanned Vehicle Control System (UVCS)

- **Originally designed for use with unmanned vehicles in the Maasvlakte port system in Rotterdam.**



Evaluating Our Approach: Unmanned Vehicle Control System (UVCS)

- ***Collision Avoidance:***
  - Safety property to verify that the two vehicles moving in the same region will never crash. In this context, crash is defined as both vehicles occupying the same x and y position on the grid.
  - Specifically, we check that Vehicle1 and Vehicle2 will never both be in the same region with the same x and y position.

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

  - This property should hold if there is no delay in the delivery of events.

Model Optimizations

- **Need optimizations – otherwise models are often too big to verify!**
- **Utilize optimization techniques such as:**
  - Cone of Influence Reduction
  - Data Abstraction
  - Reduction of Non-Determinism
- **Explored the use of architectural style specific optimizations**
  - Combinational Correctness Preserving Transformations

Contributions

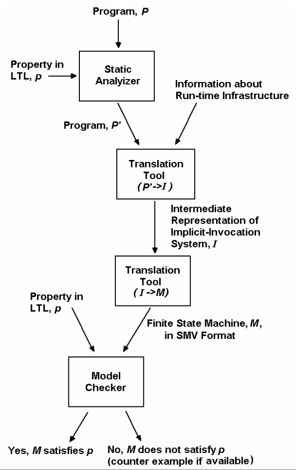
- **Extended an existing approach proposed in by Garlan and Khersonsky [GK00]**
  - Event data representation
  - Delivery policy representation
- **Evaluated extended approach using simplified "real-world" implicit-invocation systems**
  - Determined viability and usefulness of extended approach
- **Identified optimization techniques in context of architectural styles – specifically for use with implicit-invocation system**
  - Helped control "state explosion problem"

Conclusions

- **Model checking is a viable method of analysis for small implicit-invocation systems.**
- **The size of models of "real-world" systems requires state-of-the-art computers and a large quantity of patience and expertise!**
- **Although some large system are not feasible to model check in their entirety a compositional approach is a viable alternative**
  - The loose coupling of implicit-invocation components provides natural partitions for developing partial systems.
- **We have provided contributions to alleviate some of the problems that have traditionally limited software model checking.**
  - Semantic gap between artifacts and the state explosion problem.
- **Additional research is needed before model checking will become readily used outside of hardware and safety-critical software systems.**

Future Work

- **Alternative Intermediate Representation**
  - Is XML the best intermediate representation?
- **Optimization Techniques**
  - Model
    - Architecture specific optimizations?
  - Model checker tool
    - Parallel model checking?
- **Complete Automation of Model Generation**
  - Bridging the "semantic gap" between artifacts
- **Extension of Technique to Other Architectural Styles**
  - How can we take this approach and generalize it?



Questions