Playing with Time in Publish-Subscribe using a Domain-Specific Model Checker

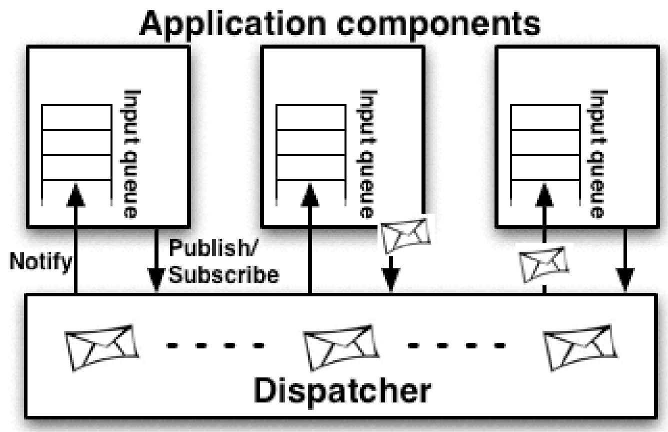
presented by Luca Mottola (mottola@elet.polimi.it)
joint work with Luciano Baresi, Giorgio Gerosa, and Carlo Ghezzi

Dipartimento di Elettronica ed Informazione
Politecnico di Milano, Italy

6th Int. Workshop on
Specification and Verification of Component-Based Systems (SAVCBS07)
Dubrovnik, (Croatia) - September 3rd, 2007

Publish-Subscribe Architectures

PubSub Paradigm



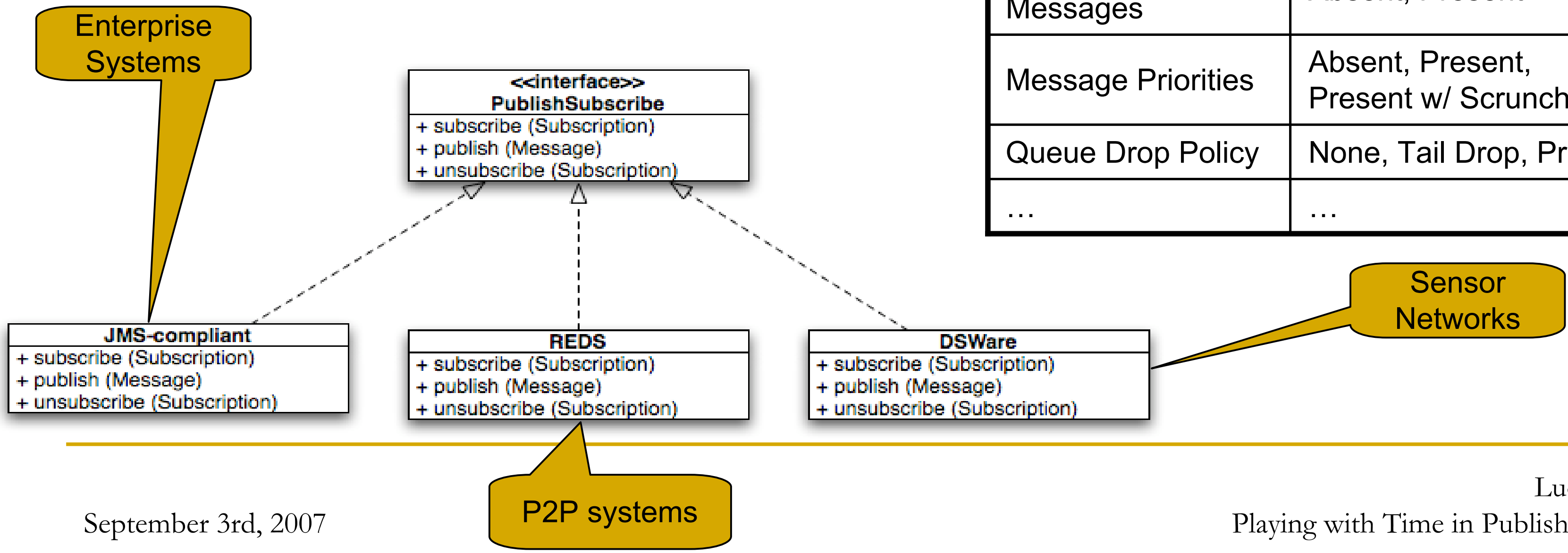
- Asynchronous communication mediated by a *dispatcher*
  - anonymous and multipoint
  - implicit addressing (e.g., content-based PubSub)
- Application components
  - *subscribe* to relevant message patterns
  - *publish* messages
- The *dispatcher* matches published messages against previously issued subscriptions
- Allows dynamic addition and removal of components
  - suited to distributed applications in dynamic environments

Publish-Subscribe Architectures

Different Flavors…

- PubSub is a *model* with many different *implementations*
  - from enterprise systems…
  - …to wireless sensor networks
- Different *guarantees* provided
- Difficult to verify the application behavior

Guarantee	Choices
Message Reliability	Absent, Present
Message Ordering	Random, Pair-wise FIFO, System-wide FIFO, Causal Order, Total Order
Filtering	Precise, Approximate
Real-time Guarantees	Absent, Present
Subscription Propagation Delay	Absent, Present
Repliable Messages	Absent, Present
Message Priorities	Absent, Present, Present w/ Scrunching
Queue Drop Policy	None, Tail Drop, Priority Drop
…	…



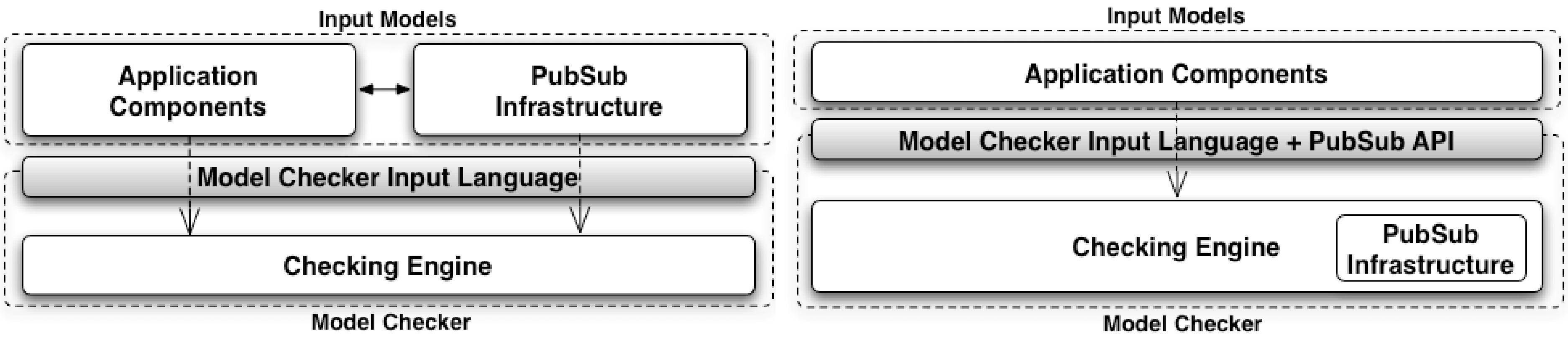
Domain Specific Model Checker

A Change of Perspective

- Model checking proposed to address the verification issue
  - standard tools (e.g., SPIN) used to model *both* the application and the PubSub infrastructure
  - *fine-grained* models unfeasible due to state space explosion
  - *parametric* models difficult due to little support for parameterization

A change of perspective:
embed the PubSub communication paradigm *within* the model-checker

L. Baresi, C. Ghezzi, and L. Mottola. On Accurate Automatic Verification of Publish-Subscribe Architectures. In Proc. Of the 29th Int. Conf. on Software Engineering (ICSE), 2007.



Domain-Specific Model Checker

PubSub APIs in Bogor

- Extend the Bogor model checker with a PubSub module
- Additional constructs used in developing BIR models
- PubSub operations are used in BIR to issue subscriptions, publish messages, …

PubSub extension API

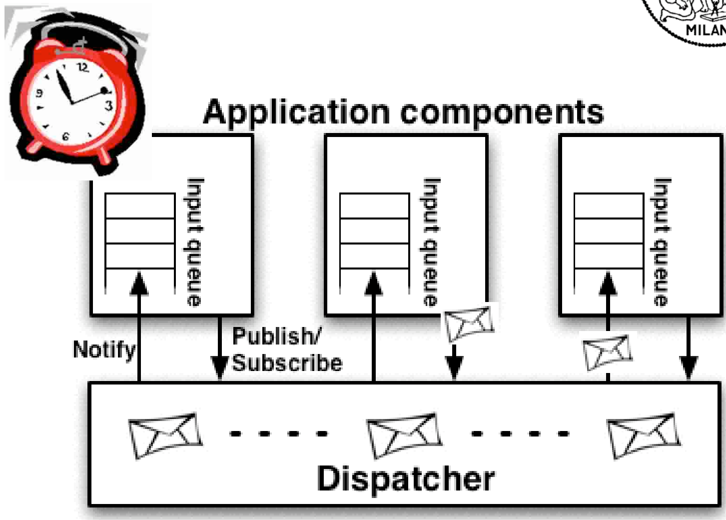
```
typealias MessagePriority int (0,9);
enum DropPolicy {TAIL_DROP, PRIORITY_DROP }

extension PubSubConnection for polimi.bogor.bogorps.PubSubModule {
  typedef type<'a>;

  expdef PubSubConnection.type<'a> register<'a>();
  expdef PubSubConnection.type<'a> registerWithDropping<'a>(int, DropPolicy);
  actiondef subscribe<'a>(PubSubConnection.type<'a>, 'a -> boolean);
  actiondef publish<'a>(PubSubConnection.type<'a>, 'a);
  actiondef publishWithPriority<'a>(PubSubConnection.type<'a>, 'a,
                                                    MessagePriority);
  expdef boolean waitingMessage<'a>(PubSubConnection.type<'a>);
  actiondef getNextMessage<'a>(PubSubConnection.type<'a>, lazy 'a);  }
```

Time Extension

Time Model

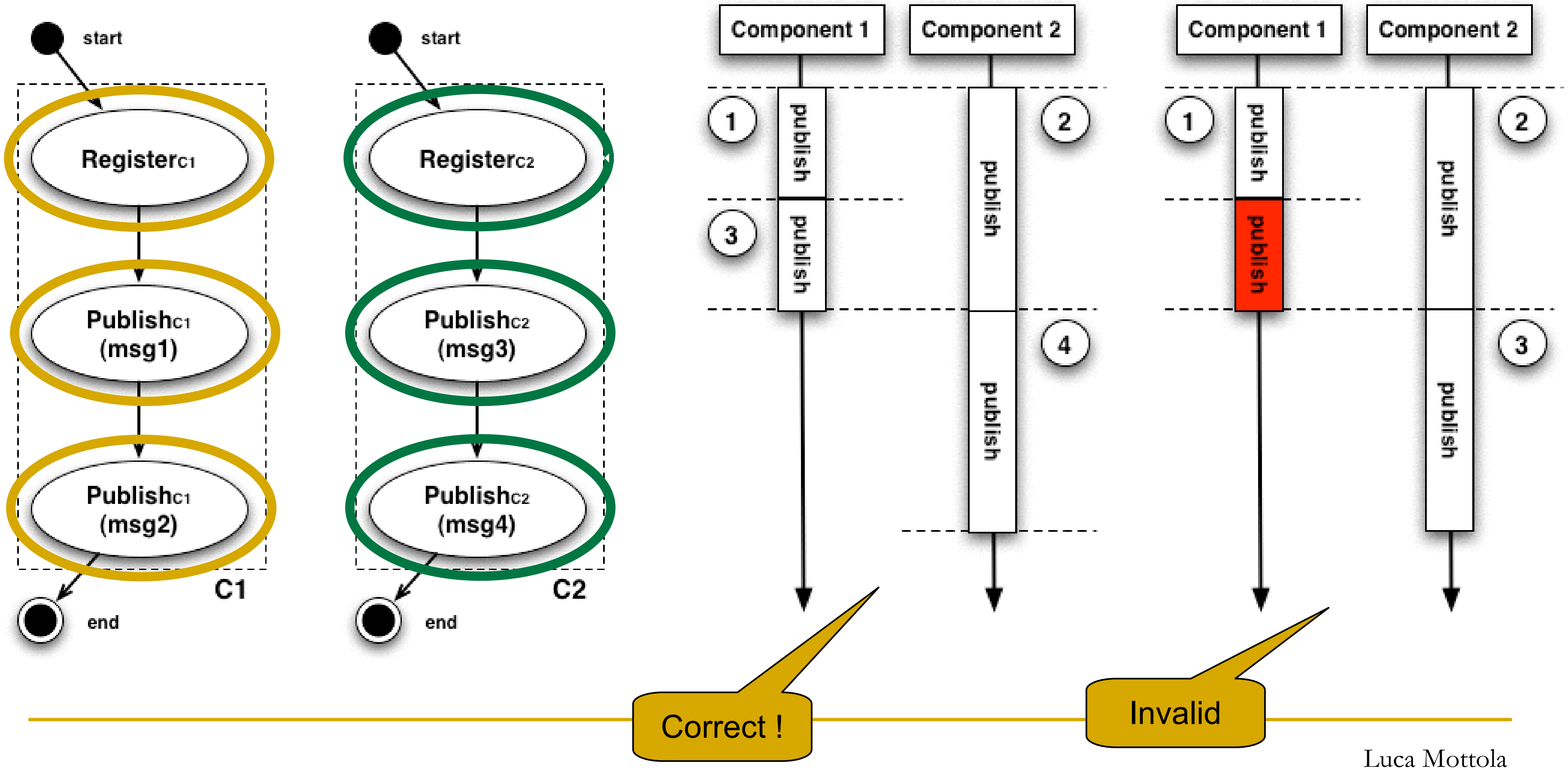


- No generic notion of time, rather:
  - suited to the dynamics of PubSub applications
  - enabling its *interplay* with other PubSub guarantees
- System evolution determined by:
  - *component execution rate* w.r.t. the PubSub dispatcher
  - (random) *message delays*
- Time alters the exploration of the state space, not the individual states
- Inspired by X. Deng, M. B. Dwyer, J. Hatcliff, and G. Jung. Model-checking middleware-based event-driven real-time embedded software. In Proc. of the 1st Int. Symposium on Formal Methods for Components and Objects, 2002

Time Extension

Time Model - Example

- C1 running at twice the execution rate of C2



Time Extension

Example Use

Message definition

Connection open

Configure time extension with exec rate 2, and random msg delay between 1 and 0

Guards control the components' interleavings

```
record MyMessage {int value;}
active thread Publisher() {
  MyMessage event = new MyMessage;
  PubSubConnection.type<MyMessage> ps;
  loc loc0:
  do { ps := PubSubConnection.register();
       PubSubConnection.configureTimeParams(ps,2,1,0); }
```

```
fun isGreaterThan(MyMessage event)
      returns boolean = event.value > 0;
active thread Subscriber() {
  PubSubConnection.type<MyMessage> ps;
  loc loc0:
  do { ps := PubSubConnection.register();
       PubSubConnection.subscribe<MyMessage>(ps,isGreaterThan);
       PubSubConnection.configureTimeParams(ps,2,1,0); }
  goto loc1;
  loc loc1:  // Message receive
  when (PubSubConnection.timedWaitingMessage(ps)==CAN_PROCEED) do {
       PubSubConnection.getNextMessage<MyMessage>(ps, receivedEvent); }
  when (PubSubConnection.timedWaitingMessage(ps)==QUEUE_EMPTY) do {
       // Do something else… }
  return; }
```

Subscription definition

Issues a (matching) subscription

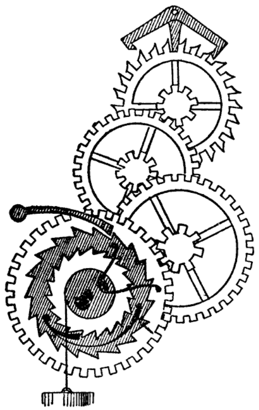
The component can proceed to receive a message

The component can proceed, but the queue is empty

Time Extension

Implementation

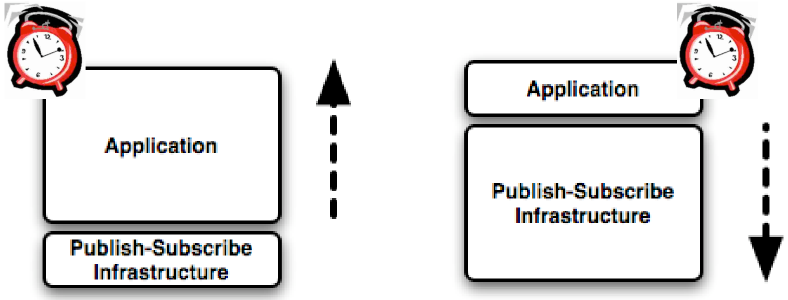
- Time divided in *frames*
  - equivalent to single operations in lowest priority component
- Generate all possible interleavings within a frame
- Take advantage of domain-specific semantics, e.g.,
  - with causal order delivery, check message ordering first and then run the time extension
  - not every single value in the message delay interval generates a different schedule
  - do not run the time extension if the component is already scheduled but the queue is empty

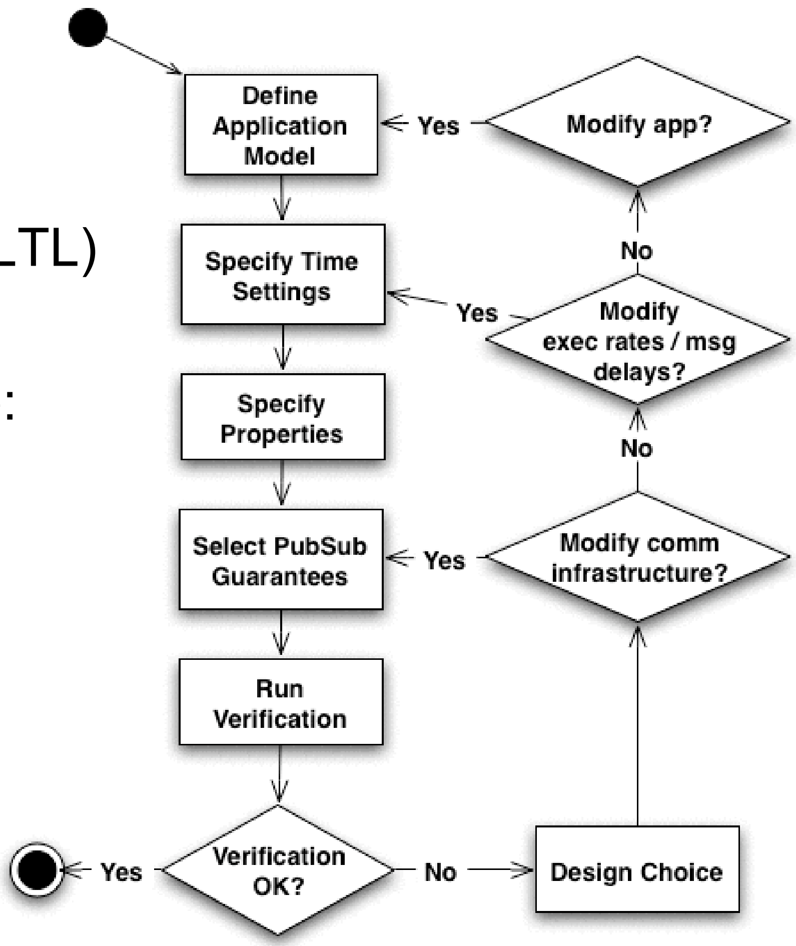


Case Study

Performing the Verification

- Specify the application model using the PubSub API
- Specify time settings
- Specify the properties to be checked (LTL)
- Select PubSub guarantees
- Depending on the verification outcome:
  - change time settings
  - modify the application model
  - change the guarantees selected





Case Study

A Telemedicine Scenario

- Remote monitoring of patients
- Several components involved:
  - variable number of *patients*
  - *medical lab*
  - *flying squad*
  - *hospital*
- Interactions expressed as PubSub operations:
  - sensors monitor a patient's status, and report to the medical lab
  - under *moderate danger*, the lab sends back corrective actions
  - in *emergency*, the lab informs the flying squad and notifies the hospital about an incoming patient
  - on the way to the hospital, the flying squad sends periodic reports to the hospital until the patient is handed over

Case Study

Requirements and Verification Outcome

- *R1: under moderate danger, any corrective action must be communicated within T1 time units*
  - fails due to dropped messages when
    - *finite queues* are assumed
    - the medical lab is not assigned an *execution rate* sufficient to handle multiple reports from different patients
- *R2: in emergency, the hospital must receive request for hospitalization within T2 time units*
  - fails for the same reason as above when the lab sends notifications to the hospital
- *R3: when a patient arrives, the hospital must have received the corresponding request for hospitalization*
  - requires causal ordering in general
  - verified also with different delivery orderings and constant message delays

Case Study

Performance

- 10 or 20 patients each publishing 10 messages
- Performances not affected by different combinations of PubSub guarantees

Requirement	Memory	States	Time
R1 - 10 patients	278.38	70234	≈16 min
R1 - 20 patients	312.31	123122	≈20 min
R2 - 10 patients	412.21	113213	≈22 min
R2 - 20 patients	502.75	209123	≈26 min
R3 - 10 patients	498.1	232123	≈30 min
R3 - 20 patients	591.1	289124	≈35 min

Verifying the Time Extension

Problem and Approach

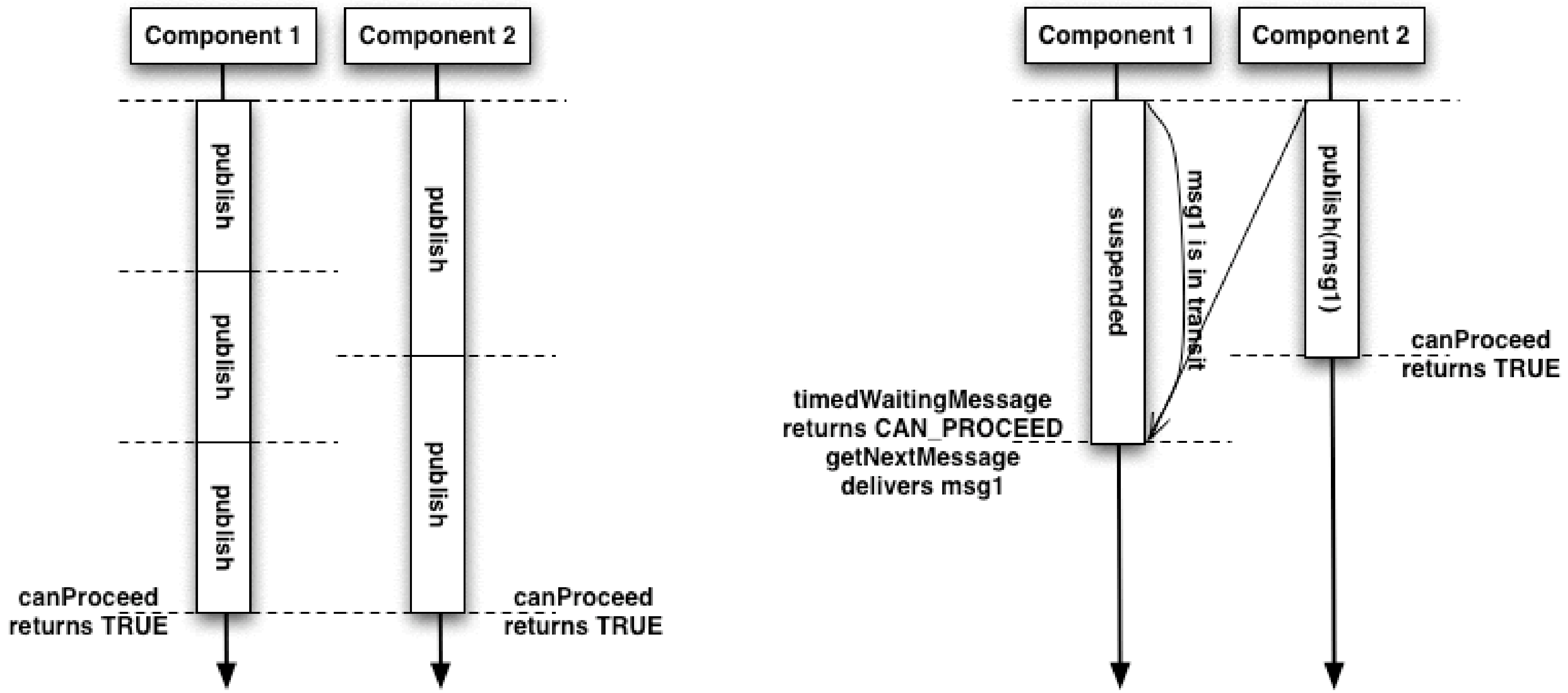
- Imperative to substantiate the correctness of the results obtained with our extension(s)
- Formal verification of our implementation
- Use Bandera, in turn based on Bogor !! 

- Unfortunately Bogor *and* PubSub extension as input to Bandera generate intractable models, however… 
  - the time extension alters the state space exploration, not the single states
  - we only need to check the values returned by the guards in all possible cases
- Manual slicing of Bogor to minimize the code input to Bandera
  - no Bogor parsers
  - no extension points
  - no reflection
  - …

Verifying the Time Extension

Generating all Possible Interleavings

- Only 2 components and 4 scenarios needed



- Discovered a bug in **`timedWaitingMessage`** due to uninitialized boolean variable !!

Conclusion and Future Work

- Embed domain specific mechanisms within a model checker
- Offer this functionality as primitive constructs of the modeling language
- Time adds the missing tile

- Better assessment through several cases studies
- Extend the formal verification to the whole PubSub extension

Bogor

An Extensible Model Checker

- *Bandera Intermediate Language* (BIR) as input
  - provides basic constructs similar to, e.g., Promela
  - function pointers, generic types, and dynamic threads
- Example: adding a non-deterministic choice requires
  - adding a new construct to BIR
  - implementing the required semantics

Indicates the Java class implementing the *choose* semantics

BIR

```
extension GenericRandom for polimi.genericRandom.GenericRandomModule {
  typedef type<'a>;
  expedef GenericRandom.type<'a>
    choose (GenericRandom.type<'a>, GenericRandom.type<'a>);}
```

Java

```
package polimi.genericRandom;
public class GenericRandom implements IModule {
  public IMessageStore connect (IBogorConfiguration bc) {
    // Retrieve Bogor hooks }
  public IValue choose (IExtArguments args) {
    // Implements the semantics for choose… }}
```

E.g., a reference to the state generation component in use

Generates two “next states“ to be explored corresponding to the two possible choices