High-level Supervision of Program Execution Based on Formal Specification

- Gergely PINTÉR
- István MAJZIK

Budapest University of Technology and Economics
Department of Measurement and Information Systems
Research goals

- Run-time fault-detection architecture based on the abstract specification
  - Behavioral models (e.g. statecharts)
  - Communication protocols (e.g. live sequence charts, sequence diagrams)
- Configurable granularity of observation
  - Selection of key aspects of the specification (e.g. by Temporal Logic Formulae)
- Supporting safety-critical SW architectures (e.g. EN-50128)
Run-time verification against formal models

- Specification
- Specification to be checked
- Source code
- Instrumented application
- Monitor

Instrumentation
Self-checking architecture
Run-time verification against formal models

**Specification**
- Basis of implementation
- Reference information

**Specification**

**Source code**

**Instrumented application**

**Instrumentation**

**Monitor**

**Self-checking architecture**
Run-time verification against formal models

Run-time verification
- Detection of programming bugs, misunderstood specification etc.
- Independent monitor component

- Specification
- Source code
- Instrumented application

Self-checking architecture

Instrumentation

Specification to be checked
Run-time verification against formal models

**Specification**
- Specification
- Specification to be checked

**Source code**
- .c
- .cc
- .java

**Instrumentation**
- .cc

**Implementation**
- Manual or automatic
- Pattern-based approach

**Self-checking architecture**
- Monitor
Run-time verification against formal models

Configurable granularity
- Selection of key modeling aspects (e.g., behavioral specification from the entire software model)

Specification

Instrumentation

Instrumented application

Monitor

Source code

Specification to be checked

Self-checking architecture
Run-time verification against formal models

- Specification
  - Source code
  - Instrumentation
- Reference information
  - Automatically derived from the specification
- Monitor

Self-checking architecture

- Specification to be checked
Run-time verification against formal models

Instrumentation
- Systematic, transparent
- Pattern-based approach

Specification

Specification to be checked

Source code
- .c
- .cc
- .java

Instrumented application

Self-checking architecture

Monitor
Run-time verification against formal models

Run-time observation

Object_A

Object_B

Monitor

ECL

Communication middleware

Specification to be checked

Monitor
Run-time verification against formal models

Run-time observation

Intra-object behavior
- Internal event-driven behavior should correspond to the behavioral specification
- Fault detection by embedded component
- Reference information: UML statecharts
Run-time verification against formal models

Run-time observation

Inter-object communication
- Monitor component on the communication bus
- Detection of communication protocol violations
- Reference information: Sequence diagrams, Life sequence charts

Object_A

Object_B

Monitor
Run-time verification against formal models

Run-time observation

- Detach the faulty object after fault detection
- Fault silent behavior or more advanced schemes

Error Confinement Layer

Self-checking architecture

Instrumented application

Specification to be checked

Monitor

Communication middleware

Object $A$

Object $B$

Monitor

Run-time observation
Run-time verification against formal models

Run-time observation

Object_A

Communication middleware

Object_B

Monitor

Specification to be checked
Run-time verification against formal models

Run-time observation

Specification to be checked

Monitor

Self-checking architecture

Communications middleware

Object A

Object B

ECL

ECL
Run-time verification for self-checking architecture

Focus and Contribution
- Run-time observation of *internal behavior*
- Advanced *monitoring mechanism*: Based on UML statecharts
- *Instrumentation method*: Aspect-oriented approach

Specification to be checked

Object\textsubscript{A}

Object\textsubscript{B}

Communication middleware

Instrumentation

ECL

Self-checking architecture

Monitor
Abstract, high-level control-flow fault detection

Reference information:
- Automatically derived from the behavioral specification
- Capable of expressing state hierarchies, concurrent operation, etc.

Implementation of the monitor:
- Based on the operational semantics of the behavioral model
- Run-time checking of the behavior on the basis of the abstract reference model

Implementation of the instrumentation:
- Providing information to the monitor about the internal behavior
- Configurable, transparent and automatically applied
Abstract, high-level control-flow fault detection

Reference information:
• Automatically derived from the behavioral specification
• Capable of expressing state hierarchies, concurrent operation, etc.

Implementation of the monitor:
• Based on the operational semantics of the behavioral model
• Run-time checking of the behavior on the basis of the abstract reference model

Implementation of the instrumentation:
• Providing information to the monitor about the internal behavior
• Configurable, transparent and automatically applied
Reference information of the internal behavior

- Extended Hierarchical Automata (EHA)
  - Clear structure:
    - Sequential automata:
      Containing any number of states
    - Non-composite states:
      Refined to any number of sequential automata
    - Non-interlevel transitions:
      Source restriction and target determination sets
  - Well elaborated formal semantics
  - Automatically derived from UML statecharts
Reference information of the internal behavior
Reference information of the internal behavior

UML statechart:
- Source parent
  - s1 (source child)
  - s2 → s3 → s3
- Target parent
  - t2

 EVT: 
- Source parent: evt SrcRest: {source child}
- Target parent: evt
- Target child: TrgDet: {target child}

EHA:
- Source parent
  - s1 (source child)
- Source child
  - s2 → s3 → s4
- Target child
  - t1
Reference information of the internal behavior

UML statechart

source parent

s1 → source child

s2 → s3 → s3

target parent

target child → t2

EHA

srcRest: \{source child\}

trgDet: \{target child\}

source parent

s1 → source child

s2 → s3 → s4

target child → t1
Reference information of the internal behavior

UML statechart:
- Source parent: s1
- Source child: s2, s3
- Target parent: t2
- Event: evt

EHA:
- Source parent: s1
- Source child: s2, s3
- Target parent: t1
- Event: evt
- SrcRest: {source child}
- TrgDet: {target child}
Reference information of the internal behavior

**UML statechart**
- **Source parent**: s1
- **Source child**: s2, s3
- **Target parent**: t2
- **Event (evt)**: From s1 to s2, s2 to s3, s3 to s3, and from s2 to t2.

**EHA**
- **Source parent**: s1
- **Event SrcRest**: {source child}
- **Target parent**: t1
- **Event TrgDet**: {target child}
Reference information of the internal behavior

UML statechart:
- Source parent: s1
- Source child: s2, s3
- Target parent: t2
- Events: evt
- Exit: exit

EHA:
- Source parent: s1
- Source child: s2, s3
- Target child: s4
- Target parent: t1
- Events: evt SrcRest: {source child}
- TrgDet: {target child}
- Exit: exit
Reference information of the internal behavior

UML statechart:
- Source parent: s1
- Source child: s2, s3
- Target parent: t2
- Event: evt
- Exit: exit

EHA:
- Source parent: s1
- Source child: s2, s3, s4
- Target parent: t1
- Event: evt SrcRest: {source child}, TrgDet: {target child}
- Exit: exit
Reference information of the internal behavior

![Diagram showing UML statechart and EHA with states and transitions labeled as s1, s2, s3, s4, t1, t2, and events evt. The diagram illustrates the association between source and target parents and children.]
Reference information of the internal behavior

UML statechart:
- Source parent: s1
- Source child: s2, s3, s3
- Target parent: t2
- Event: evt

EHA:
- Source parent: s1
- Source child: s2, s3, s4
- Target child: t1
- Event: evt SrcRest: {source child}, TrgDet: {target child}

Entry points:
- Source parent: entry
- Target parent: entry

Diagram shows the reference information with state transitions and event triggers.
Reference information of the internal behavior

UML statechart:
- Source parent
  - s1
  - s2
  - s3

Target parent
- t1
- t2

Source child
- s1
- s2
- s3

Target child
- s3

Evt
- SrcRest: {source child}
- TrgDet: {target child}

EHA:
- Source parent
- s1
- source child

Target parent
- entry
- t1

Target child
- entry
- t2
Abstract, high-level control-flow fault detection

Reference information:
• Automatically derived from the behavioral specification
• Capable of expressing state hierarchies, concurrent operation, etc.

Implementation of the monitor:
• Based on the operational semantics of the behavioral model
• Run-time checking of the behavior on the basis of the abstract reference model

Implementation of the instrumentation:
• Providing information to the monitor about the internal behavior
• Configurable, transparent and automatically applied
Abstract, high-level control-flow fault detection

Reference information:
- Automatically derived from the behavioral specification
- Capable of expressing state hierarchies, concurrent operation, etc.

Implementation of the monitor:
- Based on the operational semantics of the behavioral model
- Run-time checking of the behavior on the basis of the abstract reference model

Implementation of the instrumentation:
- Providing information to the monitor about the internal behavior
- Configurable, transparent and automatically applied
Checking the internal behavior

- Structural decomposition:
  - Run-to-completion and transition contexts
- Specification of contexts:
  - Protocol state machines (statecharts)

Diagram:
- ObservedApp
- MessageQueue
- Monitor
- RTCContext
- TransitionContext
- Run-time Monitor
Checking the internal behavior

- Structural decomposition:
  - Run-to-completion and transition contexts
- Specification of contexts:
  - Protocol state machines (statecharts)

Run-to-completion context
- Initialization
- Start and finish of event processing
- Dispatching messages to transition contexts

Run-time Monitor
- ObservedApp
- MessageQueue
- Monitor
- RTCContext
- TransitionContext
Checking the internal behavior

- Structural decomposition:
  - *Run-to-completion* and *transition contexts*

- Specification of contexts:
  - Protocol state machines (statecharts)

---

**Transition context**
- Atomic actions within a single transition (state entry and exit)
Checking the internal behavior

- Structural decomposition:
  - Run-to-completion and transition contexts
- Specification of contexts:
  - Protocol state machines (statecharts)

![Diagram](image-url)
Checking the internal behavior

- Structural decomposition:
  - Run-to-completion and transition contexts
- Specification of contexts:
  - Protocol state machines (statecharts)
Run-to-completion context

- Uninitialized
  - initStarting
  - initEntry [ieOK]
  - initFinishing [ifOK]
- Initialization
- Stable
  - evtProcStarting [epsOK]
  - evtProcFinishing [epfOK]
- Transient
  - trStarting [tsOK] / createTrCtx
  - dispatch
Before initialization
- The state configuration of the observed application is inconsistent, none of the states is active.
The observed application has started entering the initial configuration.
During initialization
- The state configuration of the observed application is inconsistent, some states of the initial configuration are active.
Entry during initialization
- The observed application has entered a state.
- **Guard**: (i) the state belongs to the initial configuration, (ii) is currently inactive and (iii) all parent states are active.
**Run-to-completion context**

**Initialization finished message**
- The observed application has finished entering the initial configuration.
- **Guard**: all states of the initial configuration are active.

**Stable**

**Transient**
**Run-to-completion context**

Stable configuration
- The observed application is in a consistent state configuration, no event is being processed.

- **Uninitialized**
- **InitStarting**
- **InitFinishing**
- **Stable**
  - **Stable configuration**
  - Starting [tsOK] / createTrCtx
  - [epsOK]
  - evtProcFinishing [epfOK]
- **Transient**
- **Dispatch**
Event processing started
- The observed application has received an event from the environment and started processing it.
Run-to-completion context

- **Uninitialized**
  - initStarting
  - initFinishing[ifOK]

- **Initial**

- **Stable**
  - evtProcStarting[epsOK]

- **Transient**
  - Transient state
    - The state configuration of the observed application is inconsistent, some transitions are being performed.
  - evtProcFinishing[epfOK]
  - dispatch

- **Transient state**
  - dispatching [tsOK] / createTrCtx
Transition selected for firing
- The observed application has selected a transition to be fired during the event processing.
- **Guard**: (i) triggered by the currently processed event, (ii) source state and ones in the source restriction set are active and (iii) does not violate the priority relations.
- **Action**: create a new transition context.
Dispatch message to transition context
• The observed application has performed an activity that is to be checked by a transition context.

Run-to-completion context

- Uninitialized
- Initialization
- Stable
- Transient

Dispatch

evtProcStarting[epsOK]
evtProcFinishing[epfOK]

initStarting
initFinishing[ifOK]
Event processing finished
- The observed application has finished the processing of the event.
- **Guard**: all transitions have been finished.
Run-to-completion context

- Uninitialized
  - initStarting
  - initEntry [ieOK]
- Initialization
  - initFinishing [ifOK]
- Stable
  - evtProcStarting [epsOK]
- Transient
  - trStarting [tsOK] / createTrCtx
  - dispatch
  - evtProcFinishing [epfOK]
Fault detected
- Any guard evaluated to false.
- Any protocol violation.
Checking the internal behavior

- Structural decomposition:
  - Run-to-completion and transition contexts
- Specification of contexts:
  - Protocol state machines (statecharts)
Checking the internal behavior

- Structural decomposition:
  - Run-to-completion and transition contexts
- Specification of contexts:
  - Protocol state machines (statecharts)
Transition context

Exiting states

-退出状态
- markInactive

Entering states

- 进入状态
- markActive

transition contexts

- trAssociated [taOK]
- trFinishing [tfOK]
- exitState [xsOK] / markInactive
- enterState [esOK] / markActive
Exiting states

- The observed application is in the first phase of performing a transition: leaving the source state and all active states refining it.

Entering states

- exitState [xsOK] / markInactive
- enterState [esOK] / markActive
State left
- The observed application has left a state during a transition.
- **Guard:** (i) the state is the source of the transition or a refinement of it, (ii) it is active and (iii) none of its refinements are active.
- **Action:** update the configuration observation.

Exiting states
- trAssociated [taOK]

Entering states
- trFinishing [tfOK]

actions:
- exitState [xsOK] / markInactive
- enterState [esOK] / markActive
Transition context

Associated action performed
• The observed application has finished the first phase of performing a transition (leaving source states) and has performed the action associated to the transition.
• **Guard**: the source and all states refining it have already been left.

Exiting states

Entering states

- trAssociated [taOK]
- enterState [esOK] / markActive
- trFinishing [tfOK]
Transition context

- Exiting states
  - $\text{trAssociated}\ [\text{tsOK}]$

- Entering states
  - $\text{enterState}\ [\text{esOK}] / \text{markActive}$
  - $\text{trFinishing}\ [\text{tfOK}]$

Entering states

- The observed application is in the second phase of performing a transition: entering the target state and the ones in the target determination set.
State entered
- The observed application has entered a state during a transition.
- **Guard**: (i) the state is the target of the transition or member of the target determination set, (ii) it is inactive and (iii) all of its parents are active.
- **Action**: update the configuration observation.
Transition context

- Exiting states
  - exitState [xsOK] / markInactive

- Entering states
  - trAssociated

Transition finished
- The observed application has finished performing the transition.
- Guard: the target and all states in the target determination set have been entered.
Transition context

Exiting states

- exitState [xsOK] /
  markInactive

- trAssociated [taOK]

Entering states

- enterState [esOK] /
  markActive

Transition context closed

- The transition context is closed, the observed application has legally performed the transition.
Exiting states

- exitState [xsOK] / markInactive

Entering states

- enterState [esOK] / markActive

Transition context
Transition context

Exiting states
- trAssociated [taOK]
- trFinishing [tfOK]

Entering states
- enterState [esOK] / markActive
- enterState [!esOK]

Fault detected
- Any guard evaluated to false.
- Any protocol violation.

Fault detected
- exitState [!xsOK]
- trAssociated [!taOK]
- trFinishing [!tfOK]
Checking the internal behavior

- Structural decomposition:
  - Run-to-completion and transition contexts

- Specification of contexts:
  - Protocol state machines (statecharts)

```
ObservedApp

MessageQueue

Monitor

RTTContext

TransitionContext
```

Run-time Monitor
Abstract, high-level control-flow fault detection

**Reference information:**
- Automatically derived from the behavioral specification
- Capable of expressing state hierarchies, concurrent operation, etc.

**Implementation of the monitor:**
- Based on the operational semantics of the behavioral model
- Run-time checking of the behavior on the basis of the abstract reference model

**Implementation of the instrumentation:**
- Providing information to the monitor about the internal behavior
- Configurable, transparent and automatically applied
Abstract, high-level control-flow fault detection

**Reference information:**
- Automatically derived from the behavioral specification
- Capable of expressing state hierarchies, concurrent operation, etc.

**Implementation of the monitor:**
- Based on the operational semantics of the behavioral model
- Run-time checking of the behavior on the basis of the abstract reference model

**Implementation of the instrumentation:**
- Providing information to the monitor about the internal behavior
- Configurable, transparent and automatically applied
Instrumentation

- Systematic transparent instrumentation:
  - Explicit message transfer to the monitor
  - Modification of the data model and the behavior
  - Case study: Aspect-Oriented Programming

StatechartBase
+ dispatchEvent
...

ObservedApp
- fireTransition
...

Implementation pattern
- Abstract base class: Fundamental facilities
- Derived class: Implements the behavior
Instrumentation

- Systematic transparent instrumentation:
  - Explicit message transfer to the monitor
  - Modification of the data model and the behavior
  - Case study: Aspect-Oriented Programming

```
StatechartBase
+ dispatchEvent
...

MessageQueue
+ sendTrStarting
...

ObservedApp
- fireTransition
...

msgq
```

Message queue
- Accessible from the base class
- Methods for message transfer to the monitor
Adding a member variable (Java AOP)

```java
public aspect BehavioralMonitoring {
    // Add a member variable to the base class
    protected MessageQueue StatechartBase.msgq;
    ...
}
```

```
StatechartBase
+ dispatchEvent
...
```

```
ObservedApp
- fireTransition
...
```

```
MessageQueue
+ sendTrStarting
...
```

```
dispatchEvent
```

```
sendTrStarting
```
Instrumentation

- Systematic transparent instrumentation:
  - Explicit message transfer to the monitor
- Modification of the statechart
- Case study: Aspect-Oriented Programming

StatechartBase
  + dispatchEvent
  ...

ObservedApp
  - fireTransition
  ...

Original behavior (Firing a transition)
  Recursively leave the source state
  Associated action
  Enter target states
Systematic transparent instrumentation:
- Explicit message transfer to the monitor
- Modification of the data model and the behavior

Case study: Aspect-Oriented Programming

Instrumented behavior (Firing a transition)
- Send the "Starting transition" message
- Recursively leave the source state
- Associated action
- Enter target states
- Send the "Transition finished" message

StatechartBase
+ dispatchEvent
...

ObservedApp
- fireTransition
...
Instrumented behavior (Firing a transition)

1. Send the “Starting transition” message
2. Recursively leave the source state
3. Enter target states
4. Send the “Transition finished” message

Case study: Aspect-Oriented Programming

```
StatechartBase
+ dispatchEvent
...

ObservedApp
- fireTransition
...
```

Transient

```
trStarting [tsOK] / createTrCtx
```

dispatch

```
dispatch
```

```
MessageQueue
+ sendTrStarting...
```

```
msgq
```

```
Instrumented behavior
(Firing a transition)
```

```
ObservedApp
```

```
StatechartBase
```

```
+ dispatchEvent
```

```
...
```

```
Associated action
```

```
Enter target states
```

```
Send the “Transition finished” message
```

```
Recursively leave the source state
```

```
Send the „Starting transition” message
```

```
Transient
```

```
trStarting [tsOK] / createTrCtx
```

```
dispatch
```

```
MessageQueue
+ sendTrStarting...
```

```
msgq
```

```
Instrumented behavior
(Firing a transition)
```

```
ObservedApp
```

```
StatechartBase
```

```
+ dispatchEvent
```

```
...
```

```
Associated action
```

```
Enter target states
```

```
Send the „Transition finished” message
```

```
Recursively leave the source state
```

```
Send the „Starting transition” message
```

```
Transient
```

```
trStarting [tsOK] / createTrCtx
```

```
dispatch
```

```
MessageQueue
+ sendTrStarting...
```

```
msgq
```

```
Instrumented behavior
(Firing a transition)
```

```
ObservedApp
```

```
StatechartBase
```

```
+ dispatchEvent
```

```
...
```

```
Associated action
```

```
Enter target states
```

```
Send the „Transition finished” message
```

```
Recursively leave the source state
```

```
Send the „Starting transition” message
```

```
Transient
```

```
trStarting [tsOK] / createTrCtx
```

```
dispatch
```

```
MessageQueue
+ sendTrStarting...
```

```
msgq
```

```
Instrumented behavior
(Firing a transition)
```

```
ObservedApp
```

```
StatechartBase
```

```
+ dispatchEvent
```

```
...
```

```
Associated action
```

```
Enter target states
```

```
Send the „Transition finished” message
```

```
Recursively leave the source state
```

```
Send the „Starting transition” message
```

```
Transient
```

```
trStarting [tsOK] / createTrCtx
```

```
dispatch
```

```
MessageQueue
+ sendTrStarting...
```

```
msgq
```

```
Instrumented behavior
(Firing a transition)
```

```
ObservedApp
```

```
StatechartBase
```

```
+ dispatchEvent
```

```
...
```

```
Associated action
```

```
Enter target states
```

```
Send the „Transition finished” message
```

```
Recursively leave the source state
```

```
Send the „Starting transition” message
```

```
Transient
```

```
trStarting [tsOK] / createTrCtx
```

```
dispatch
```

```
MessageQueue
+ sendTrStarting...
```

```
msgq
```

```
Instrumented behavior
(Firing a transition)
```

```
ObservedApp
```

```
StatechartBase
```

```
+ dispatchEvent
```

```
...
```

```
 Associated action
```

```
Enter target states
```

```
Send the „Transition finished” message
```

```
Recursively leave the source state
```

```
Send the „Starting transition” message
```

```
Transient
```

```
trStarting [tsOK] / createTrCtx
```

```
dispatch
```

```
MessageQueue
+ sendTrStarting...
```

```
msgq
```

```
Instrumented behavior
(Firing a transition)
```

```
ObservedApp
```

```
StatechartBase
```

```
+ dispatchEvent
```

```
...
```

```
 Associated action
```

```
Enter target states
```

```
Send the „Transition finished” message
```

```
Recursively leave the source state
```

```
Send the „Starting transition” message
```

```
Transient
```

```
trStarting [tsOK] / createTrCtx
```

```
dispatch
```

```
MessageQueue
+ sendTrStarting...
```

```
msgq
```

```
Instrumented behavior
(Firing a transition)
```

```
ObservedApp
```

```
StatechartBase
```

```
+ dispatchEvent
```

```
...
```

```
 Associated action
```

```
Enter target states
```

```
Send the „Transition finished” message
```

```
Recursively leave the source state
```

```
Send the „Starting transition” message
```

```
Transient
```

```
trStarting [tsOK] / createTrCtx
```

```
dispatch
```

```
MessageQueue
+ sendTrStarting...
```

```
msgq
```

```
Instrumented behavior
(Firing a transition)
```

```
ObservedApp
```

```
StatechartBase
```

```
+ dispatchEvent
```

```
...
```

```
 Associated action
```

```
Enter target states
```

```
Send the „Transition finished” message
```

```
Recursively leave the source state
```

```
Send the „Starting transition” message
```

```
Transient
```

```
trStarting [tsOK] / createTrCtx
```

```
dispatch
```

```
MessageQueue
+ sendTrStarting...
```

```
msgq
```

```
Instrumented behavior
(Firing a transition)
```

```
ObservedApp
```

```
StatechartBase
```

```
+ dispatchEvent
```

```
...
```

```
 Associated action
```

```
Enter target states
```

```
Send the „Transition finished” message
```

```
Recursively leave the source state
```

```
Send the „Starting transition” message
```

```
Transient
```

```
trStarting [tsOK] / createTrCtx
```

```
dispatch
```

```
MessageQueue
+ sendTrStarting...
```

```
msgq
```

```
Instrumented behavior
(Firing a transition)
```

```
ObservedApp
```

```
StatechartBase
```

```
+ dispatchEvent
```

```
...
```

```
 Associated action
```

```
Enter target states
```

```
Send the „Transition finished” message
```

```
Recursively leave the source state
```

```
Send the „Starting transition” message
```

```
Transient
```

```
trStarting [tsOK] / createTrCtx
```

```
dispatch
```

```
MessageQueue
+ sendTrStarting...
```

```
msgq
```

```
Instrumented behavior
(Firing a transition)
```

```
ObservedApp
```

```
StatechartBase
```

```
+ dispatchEvent
```

```
...
```

```
 Associated action
```

```
Enter target states
```

```
Send the „Transition finished” message
```

```
Recursively leave the source state
```

```
Send the „Starting transition” message
```

```
Transient
```

```
trStarting [tsOK] / createTrCtx
```

```
dispatch
```

```
MessageQueue
+ sendTrStarting...
```

```
msgq
```

```
Instrumented behavior
(Firing a transition)
```

```
ObservedApp
```

```
StatechartBase
```

```
+ dispatchEvent
```

```
...
```

```
 Associated action
```

```
Enter target states
```

```
Send the „Transition finished” message
```

```
Recursively leave the source state
```

```
Send the „Starting transition” message
```

```
Transient
```

```
trStarting [tsOK] / createTrCtx
```

```
dispatch
```

```
MessageQueue
+ sendTrStarting...
```

```
msgq
```

```
Instrumented behavior
(Firing a transition)
```

```
ObservedApp
```

```
StatechartBase
```

```
+ dispatchEvent
```

```
...
```

```
 Associated action
```

```
Enter target states
```

```
Send the „Transition finished” message
```

```
Recursively leave the source state
```

```
Send the „Starting transition” message
```

```
Transient
```
Instrumentation

- Systematic transparent instrumentation:
  - Explicit message transfer to the monitor
  - Modification of the data model and the behavior
- Case study: Aspect-Oriented Programming

StatechartBase

Instrumented behavior (Firing a transition)

- Send the „Starting transition“ message
- Recursively leave the source state
- Associated action
- Enter target states
- Send the „Transition finished“ message

Entering states

enterState / markActive

trFinishing [tfOK]
Add code around function call (Java AOP)

```java
public aspect BehavioralMonitoring {
    // Define pattern matching calls to fireTransition
    pointcut firingTransitionPattern
        call (StatechartBase+.fireTransition(Transition t));

    // Define instrumentation to be applied
    around(): firingTransitionPattern() {
        msgq.sendTrStarting();
        proceed();
        msgq.sendTrFinishing();
    }
}
```
Reference information:
• Automatically derived from the behavioral specification
• Capable of expressing state hierarchies, concurrent operation, etc.

Implementation of the monitor:
• Based on the operational semantics of the behavioral model
• Run-time checking of the behavior on the basis of the abstract reference model

Implementation of the instrumentation:
• Providing information to the monitor about the internal behavior
• Configurable, transparent and automatically applied
Summary

- Monitoring of the dynamic behavior
  - Verification against abstract specification
  - Pattern-based instrumentation scheme
- Prototype implementation
  - Benchmark experiment: bit-inversion faults in the statechart implementation (C++ version)
    - HW: 40%, monitor: 21.5%, SW: 18.5%
  - Instrumentation case study (Java, AspectJ):
    - Run-time overhead: 10.9%
Future work

Event queue

Statechart implementation
Future work

Statechart reference

Event queue

Run-time monitor

Temporal logic

Statechart implementation
Future work

- Run-time verification of behavior
- Statechart reference
- Event queue
- Verification against SC-LTL formulae
- Temporal logic
- Statechart implementation
- Instrumentation by AOP
- Exception handling in statecharts
- Assessment by fault injection
  EUROMICRO 04
Future work

- Statechart reference
- Event queue
- Run-time monitor
- Statechart implementation
- Temporal logic
- Formal reference
- Temporal logic fitted to statecharts
- Exception handling
- No manual modification on the implementation (AOP)
- Original event dispatcher interface

Run-time fault detection of statechart implementations

Event queue

Run-time monitor

Temporal logic

Statechart implementation