

Message Trace Analysis Tool

MTAT is a tool designed to help an architect manage and understand events sent within an asynchronous event-based architecture. Our tool automates the process of capturing events sent within an architecture, uses heuristic rules to determine causality relationships between the captured events, and provides a means of visualizing the resulting causality relationships to cultivate greater understanding of the architecture.

Capturing Events:

Events are captured at the architectural level of an application. Our tool examines an architecture specified in xADL 2.0 and modifies it by interspersing "trace connectors" into the description. Trace connectors log messages that are exchanged between components into a database. Components remain unaware of any changes. This approach has proven to be applicable to different event-based architectural styles.

Determining Causality:

Causality between events is determined using heuristic rules specified in the architecture description using an extension to xADL 2.0. Rules specify required properties of events involved in causality relationships. When determining causality, rules are translated into database queries. Complete component behavioral specification is unnecessary since the heuristic

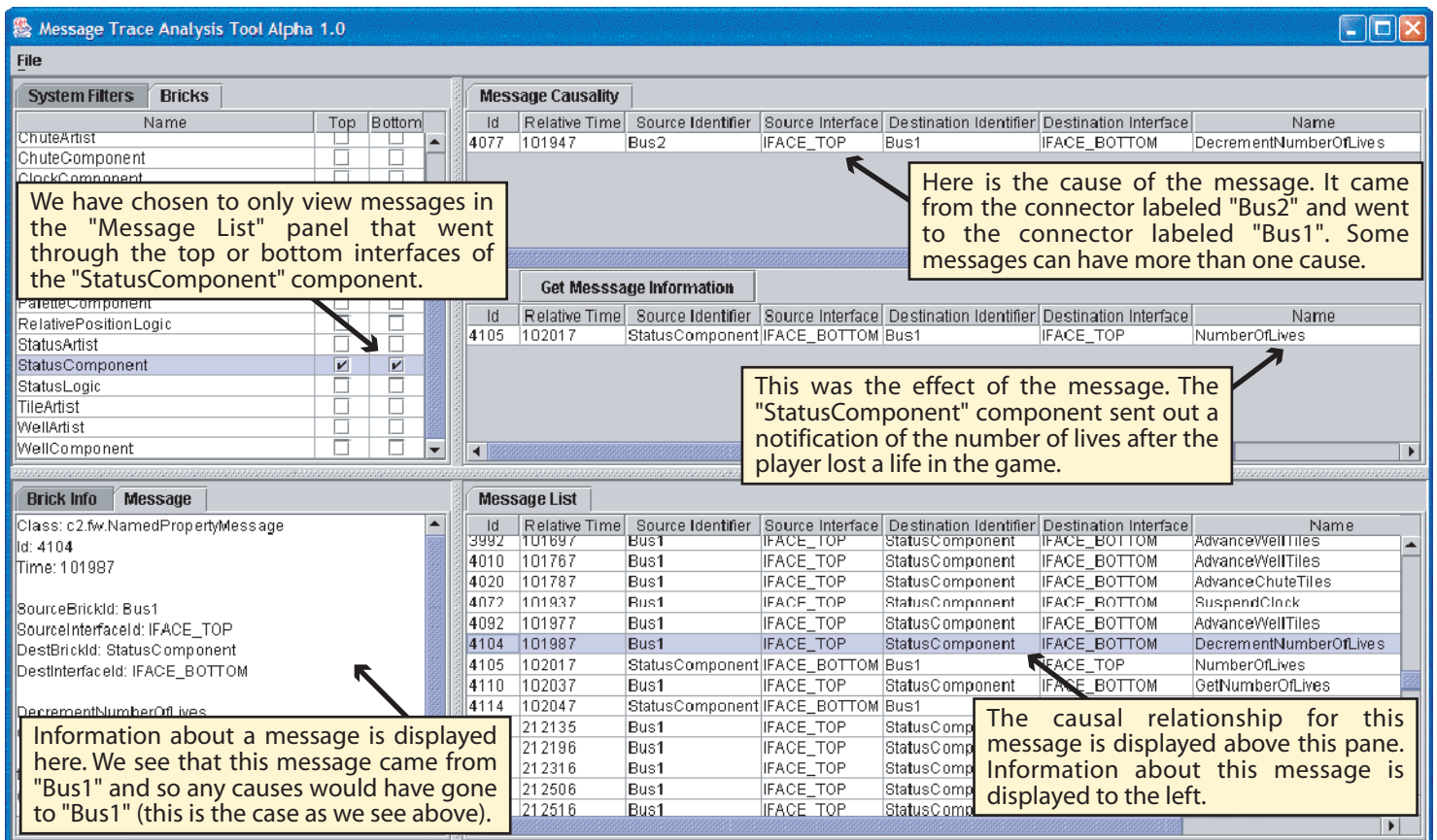
nature of the rules allows the tool to be applied to partial information. However, more specific rules produce more accurate results.

Visualizing Causality:

Our visualization tool, MTAT, allows a user to explore causality chains between captured events as well as visualizing, sorting, and categorizing the events themselves. Using MTAT one can visualize events as the architecture is executing or after the program has completed execution. Changes made to the rules specifying causality relationships will be immediately reflected in the results of the tool. Results can be verified using the visualization tool since the complete and correct set of captured events are at your fingertips.

Incremental Refinement:

Our approach facilitates incremental refinement of the bases by which causality is determined as an architect gains more insight into an architecture. That is, during analysis an architect may modify rules to better reflect the true causal relationships of a component. As new behavior is discovered (especially when dealing with black-box components), or incorrect behavioral specification is identified, rules can be added or corrected, analysis continued, and program understanding increased.



The screenshot displays the MTAT Alpha 1.0 interface. It features several panels: 'System Filters' and 'Bricks' on the left; 'Message Causality' and 'Get Message Information' in the center; and 'Brick Info' and 'Message List' at the bottom. The 'Message Causality' table shows a message (Id: 4077) originating from 'Bus2' and going to 'Bus1'. The 'Message List' table shows a sequence of messages, with Id: 4104 highlighted. Callouts explain the causal relationships and the effect of the messages.

Id	Relative Time	Source Identifier	Source Interface	Destination Identifier	Destination Interface	Name
4077	101947	Bus2	IFACE_TOP	Bus1	IFACE_BOTTOM	DecrementNumberOfLives
4105	102017	StatusComponent	IFACE_BOTTOM	Bus1	IFACE_TOP	NumberOfLives

Id	Relative Time	Source Identifier	Source Interface	Destination Identifier	Destination Interface	Name
3992	101897	Bus1	IFACE_TOP	StatusComponent	IFACE_BOTTOM	AdvanceWellTiles
4010	101767	Bus1	IFACE_TOP	StatusComponent	IFACE_BOTTOM	AdvanceWellTiles
4020	101787	Bus1	IFACE_TOP	StatusComponent	IFACE_BOTTOM	AdvanceChuteTiles
4077	101937	Bus1	IFACE_TOP	StatusComponent	IFACE_BOTTOM	SuspendClock
4092	101977	Bus1	IFACE_TOP	StatusComponent	IFACE_BOTTOM	AdvanceWellTiles
4104	101987	Bus1	IFACE_TOP	StatusComponent	IFACE_BOTTOM	DecrementNumberOfLives
4105	102017	StatusComponent	IFACE_BOTTOM	Bus1	IFACE_TOP	NumberOfLives
4110	102037	Bus1	IFACE_TOP	StatusComponent	IFACE_BOTTOM	GetNumberOfLives
4114	102047	StatusComponent	IFACE_BOTTOM	Bus1	IFACE_TOP	NumberOfLives

Distributed Systems:

MTAT works on both single-process and distributed systems. In distributed systems, connectors that bridge process boundaries are broken into two halves—one on each process they connect. Distributed connectors are treated the same way that single-process connectors are; a trace connector is placed on every link in every process. When tracing event causality through an architectural layer, distributed systems are viewed as large, unbroken architectures. Events that crossed multi-process connectors appear just as if they had crossed an in-process connector.

Heuristic Rules:

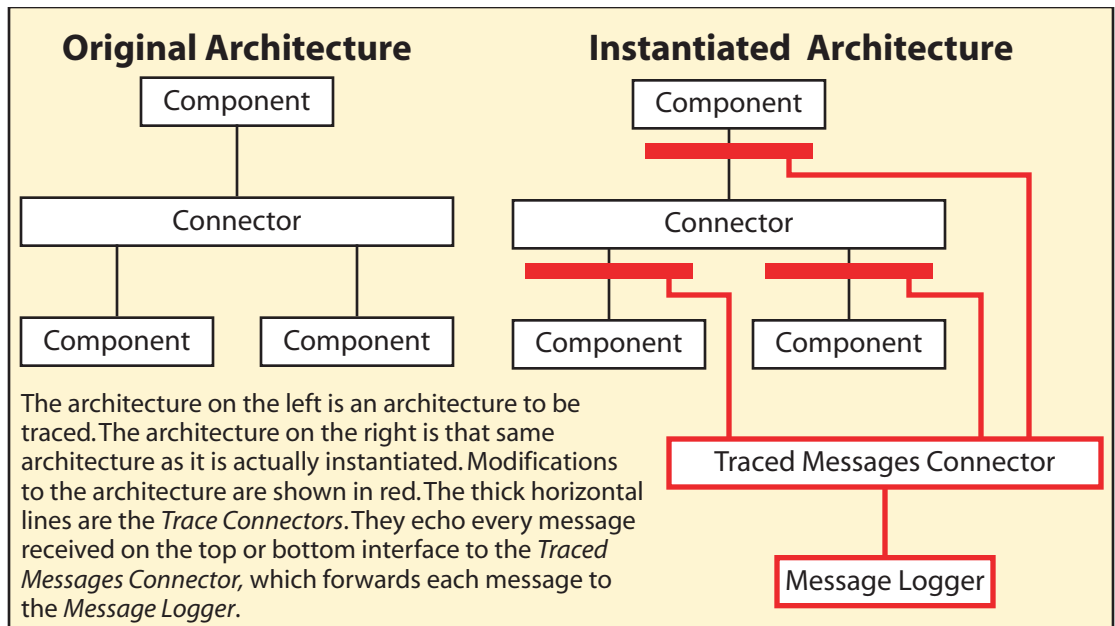
The rule set used to specify causal relationships between captured events is simple, intuitive, and usable, but not fully formal, and applies to systems where component source may not be available. Rules are used to guide an architect through the events of an architecture by limiting the possible causes and effects of a given event to those in which the architect is interested. Uninteresting events (those that are not addressed in rules) are ignored.

Rule Types:

Currently, there are two types of rules: *MostRecent* and *MatchingN*. The *MostRecent* rule type indicates that a component either responds to an event immediately or not at all. The *MatchingN* rule indicates that a component will always respond to a set of events. This type of rule is especially applicable to components that queue up requests or broadcast events.

Case Studies:

We applied our tool to two architectures. One, KLAX, is an interactive computer game written in the C2 architectural style. We annotated KLAX twice by using two different approaches. First we extracted rules for each component by looking only at the message log. Our tool was useful in this endeavor as it provided a means for us to manage the high volume of events and allowed us to start taking advantage of



the inferred rules immediately. The rules resulting from this approach were compared to those obtained by the second approach of extracting rules from the source code. The high degree of correlation between the rules indicated the usefulness of our tool.

The other architecture we applied our tool towards was the AWACS simulator. It is a distributed simulation of the software systems used on the US Air Force AWACS aircraft. We were interested in the effectiveness of our tool in a large-scale distributed architecture. We found that our tool is applicable to architectures with hundreds of components and connectors that produces tens of thousands of events. We also found that the performance of causality determination is directly related to query performance by the underlying relational database. This means that our approach can leverage, and scale up with, improving relational database technologies.

Why Use MTAT:

MTAT provides a significant improvement over manual tracing, and augments well other techniques for program understanding. It contributes a usable and viable approach to understanding complex event-based systems and features

- automated event capture,
- independence from a specific event-based architectural style,
- support for distributed systems,
- simple and intuitive rules, and
- a powerful visualization tool.

Contact Information

Professor Richard N. Taylor
Professor David F. Redmiles
Professor David S. Rosenblum
Professor Andre van der Hoek
Information and Computer Science
University of California
Irvine, California 92697-3425

{taylor, redmiles, dsr, andre}@ics.uci.edu
949-824-{6429, 3823, 6534, 6326}
949-824-1715 (fax)

For more information about this work
please contact:

Scott A. Hendrickson at:
shendric@ics.uci.edu
949-824-3100 (phone)
949-824-1715 (fax)

This material is based upon work sponsored by the Defense Advanced Research Projects Agency, and Rome Laboratory, Air Force Materiel Command, USAF, under contract number F30602-00-2-0607. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.