Monitoring Intra-Process Events Overview

SOAtest can visualize and trace the intra-process events that occur as part of the transactions triggered by the tests and dissect them for validation. This enables you to identify problem causes and validate multi-endpoint, integrated transaction systems—actions that are traditionally handled only by specialized development teams.

Why Monitor Events?

Many things can go wrong in a test environment: messages may be routed to the wrong system, transformed incorrectly, or the target system may not perform a required action.

If a test succeeds, how do you know that every step was executed correctly and the appropriate updates were performed? If it fails, how do you know what went wrong, where, and why?

Once you identify the cause of an error, how can you isolate it and design the right tests around the isolated parts so the problem can be resolved and then keep the test so that the problem can be detected again in the future?

Just sending a message into a system and validating the response coming out is not sufficient to address these challenges. Having the visibility and control inside these systems—especially ESBs that serve as the heart of today's business transactions—is critical to success.

Monitoring Events with SOAtest

Parasoft SOAtest can monitor messages and events inside ESBs, Java applications, databases, and other systems in order to provide validation and visibility into test transactions. SOAtest also provides a framework and an API to provide this level of internal visibility within almost any system.

In addition to sending an initial message and then validating the response (and possibly validating various changes that take place as a result of the transaction), SOAtest can also monitor the intermediate messages and events. For example, it may monitor the initial messages, the message after it is transformed, messages where a service calls another service and a response comes back, and the message that reaches the destination system—then also monitor the steps through the entire route that the response messages take back.

If the transaction executes correctly, you can easily define the assertions to automate validation of these intermediate messages/steps in all subsequent test runs. If it does not execute correctly, you can determine what went wrong and where. You can then apply the available tool set to validate whether messages satisfy functional expectations.

Using this functionality, you can ensure that tests aren’t marked as successful unless the test transaction executes exactly as intended.