Trading Technology Testing with th2

2023

BUILD SOFTWARE TO TEST SOFTWARE
exactpro.com

Content

Introduction

Active And Passive Test Methods

Test Oracle Implementations In Detail

Rule-Based Checking

Market Data Consistency Checks With Bookchecker

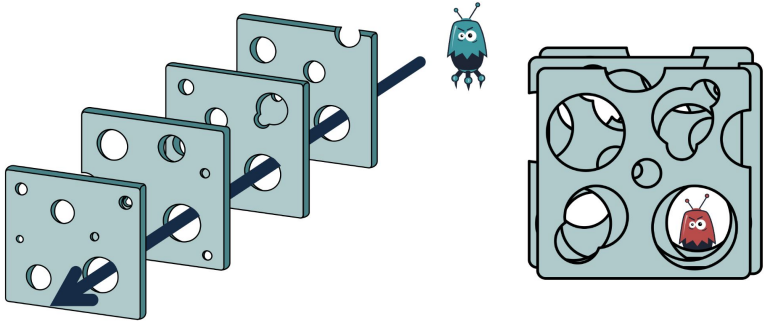
Data Services

Introduction

In testing high-performance distributed financial systems, concentrating on a limited set of tools or approaches **increases the chance of missing business-critical defects and causes more frequent visible outage incidents**. Defects may either not be covered by the testing strategy to begin with – as a result of a 'tunnel-vision' approach to testing – or hide at the intersection of functionalities/components integrated.

Using a variety of testing methods has proven to boost the sensitivity of defect detection. A strategy built on combining testing methods increases the chances of a defect missed by one approach to be picked up by another one.

This Case Study reviews a set of test methods that trading technology providers and their clients can benefit from while using th2. Combining these methods allows one to build a comprehensive and efficient test library and achieve exhaustive test coverage. The case study is based on our demo which you are welcome to watch below.



Active and Passive Test Methods

The testing methods can be subdivided into two types: active and passive ones. Using an active testing method means interacting with the system by sending it some data and verifying the response, while using a passive method implies gathering data and analysing it to get insights. Let's take a look at each method in more detail.

Active Testing

In the demo, we demonstrate a linear test scenario that describes a set of actions performed against the system. To check the responses from the system, we use two different types of test oracles:

- The first one is a **hardcoded expected result scripted by the software engineer**. In the script, we specify the messages that we expect to receive from the system in response to the actions in our scenario. The engineer has to describe the contents of every expected message.

- The second type of oracle that we use is based on the **algorithmic model of the system under test**. The algorithmic model written by software engineers generates the expected behaviour for a combination of inbound actions, and th2 components are comparing the observed messages from the system under test against the prediction of the algorithmic model.

In Active testing, the script interacts with the system in real time, sending requests and verifying responses.

Active and Passive Test Methods

Passive Testing

In contrast, the passive testing method uses data from other testing activities to monitor system behaviour after those activities have taken place.

Passive checks are at the centre of this approach which we illustrate with two examples:

- **BookChecker** – a tool that checks the consistency of the Market Data Stream and
- **Recon** (reconciliation) which allows us to perform various rules for the reconciliation of various outputs from the exchange.

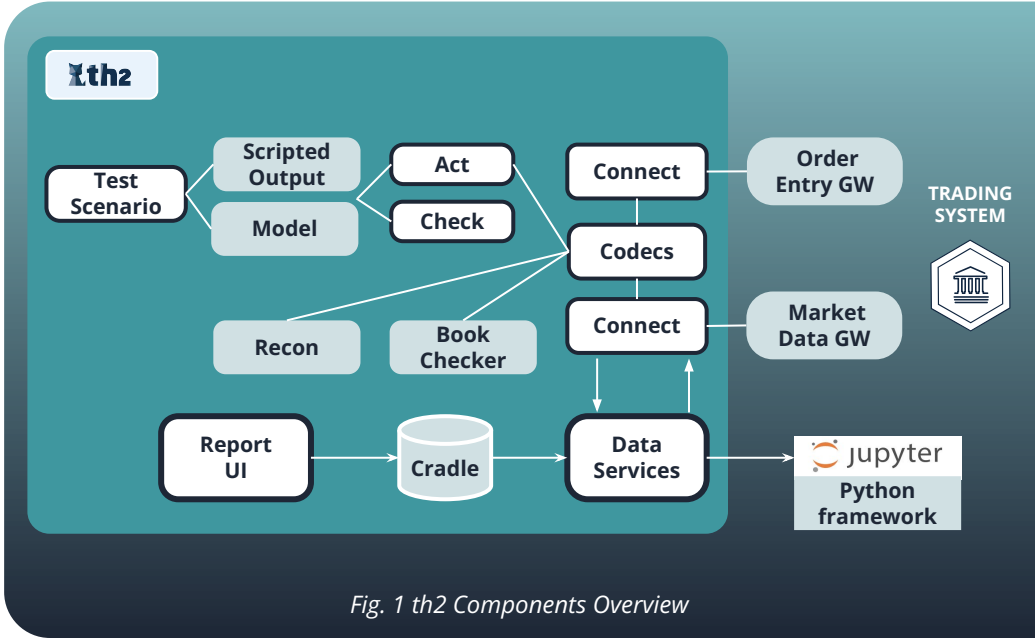


Fig. 1 th2 Components Overview

All the test data generated during test execution is collected and stored in the **Cradle database**. Modern data mining techniques can be used for efficient investigation of the test results of complex test libraries execution.

The overview schema in Fig. 1 illustrates the multiple th2 components allowing us to leverage different testing methods in order to create our test library.

Now, let us dive deeper into the actual implementations based on using each of these two types of test oracles.

Test Oracle Implementations in Detail

exactpro
EXITUS ACTA PROBAT

Human-scripted Expected Results

The th2 framework enables a flexible, fast, simple and efficient way for software engineers to script the expected outcomes.

The framework combines the power of modern programming languages (like Python or Java) and business-oriented API to perform actions/verifications via many connections simultaneously. In the demo, we define the logic of the test in "example_inputs.py", and in "data_for_test.py", we define various input parameters to test different permutations.

After the execution, test events, along with the verification results and the message flow will show up in the th2 GUI.

```
from linear_framework import Execution
from random_inputs import *

scenario = Execution(path_to_config="./linear_framework/framework_config.yaml")

""" ------------ create connection ------------ """
scenario.create_connectivity(users=users)

""" ---------------- begin execution ---------------------------------------- """
""" begin execution | (optional) desc: execution description, total: number of cases for progress bar"""
scenario.begin_set_execution(set_name="Demo test case", case_num=1)

""" ------------------ scenarios --------------------- """
    scenario.test_case_start(case_name=f"Demo_case")
    scenario.send_message(mc_1)
    scenario.verify_message(mcr_1)
    new1 = scenario.create_checkpoint()
    scenario.new_order(ord_1)
    scenario.verify_message([ord_1_er_pending_new, ord_1_er_new])
    scenario.amend_order(ocr_1)
    scenario.verify_message([ocr_1_er_replaced, ocr_1_er_pending_replace])
    scenario.cancel_order(oc_1)
    scenario.verify_message([oc_1_er_canceled, oc_1_er_pending_cancel])
    scenario.test_case_end()
    scenario.finish_set_execution()
""" ----------------- finish execution ----------------- """
scenario.finish_whole_execution()
```

Test Oracle Implementations in Detail

Model-generated Expected Results

This method uses a simplified model of the exchange capable of predicting the outbound messages based on the provided inbound actions. The model is written in Python and consists of several parts:

- The first part is the logic part, this is where all exchange rules can be described. These include: the matching algorithm, order validations, sessions, monitoring mechanisms.
- The second part is the Interface part which is related to a specific protocol (FIX, ITCH, Native or Binary, Downstream, etc.).
- The third part is Reference Data which contains information about different entities used in our tests (such as Users, Firms, Instruments and other specific parameters).

As input, the model can use a spreadsheet complete with data featuring a list of actions with the corresponding attributes.
As a result of model execution, th2 generates various reports for detailed analysis. The full content of your selected messages can be inspected in a separate tab within the test framework UI – the messages tab.

1													
2	TEST_CASE_START												
3	TC01. LimOrders. Passive Sell vs IOC Limit												
4	LCA_2022FEB17												
5													
6	Status	ID	PreviousID	Action	User	Side	OrderType	TIF	OrderQty	DisplayQty	Price	BidPx	BidSize
7		O1		New	fixoea01	Sell	Limit	DAY	1000000.00000001		9		
8		O2		New	fixoea01	Sell	Limit	DAY	1000000.00000002		8		
9		O3		New	fixoeb01	Buy	Limit	IOC	3000000.00000004		10		

Fig. 2 Model Input

Rule-Based Checking

Reconciliation Testing

The th2 framework can be used to perform reconciliation tests. **Reconciliation testing** (see Fig. 3) is used to ensure the consistency between two different transactional data streams. Its main goal is to validate that each request has its corresponding response with proper data. As input, we can use the results of different test activities – exploratory testing, functional and non-functional testing, random load, and others.

We are connecting to the system using the **Order Entry FIX** protocol and pushing various transactions to mimic participants on the trading platform. The data is being captured and then parsed by the related codec. When the data is parsed, it can be used by the **reconciliation (Recon)** component responsible for the rule-based checking method within the th2 framework.

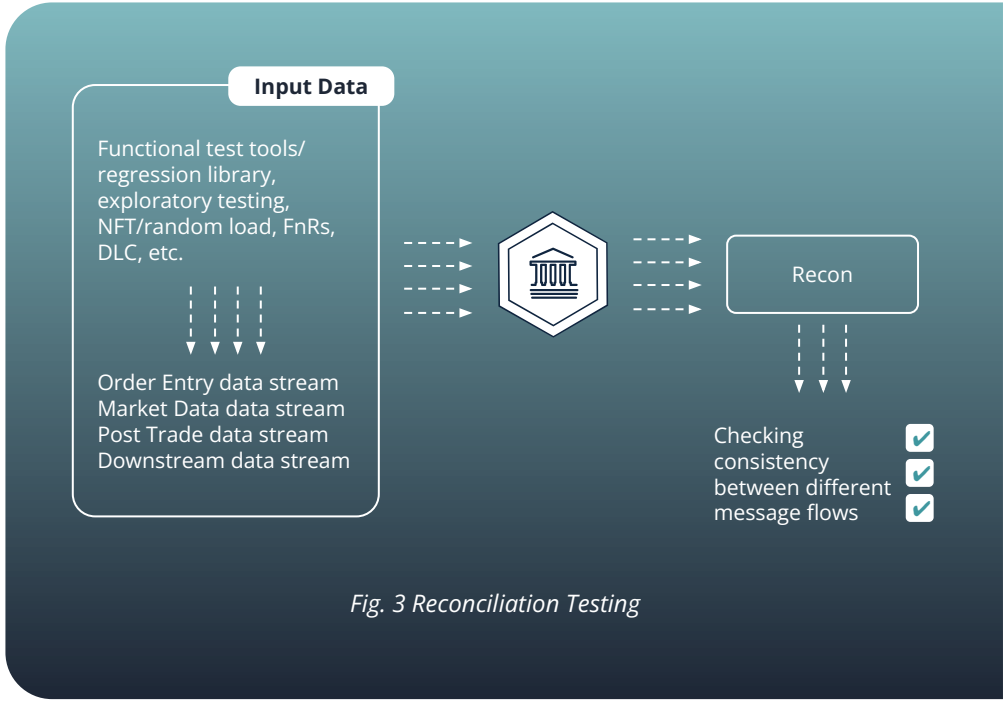


Fig. 3 Reconciliation Testing

Rule-Based Checking

The rules for the demo were written in Python. In our rule, we are checking if there is an acknowledgement for each New Order Single pushed to the system (i.e. if there's a corresponding Execution report). The messages will be matched by a set of unique criteria. In our case – **Client Order ID, Side and Symbol** (see Fig.4).

In addition, the test framework checks the content of these messages, verifying that the values in the responses from the system correlate with the respective values from the requests sent.

As a result, th2 generates various reports for detailed analysis. Report Viewer is a web-based UI for th2 Reports, it consists of events and messages that were stored in Cradle (our test data lake) during test executions. ‘Events’ are the logs of the test tools' work, and ‘messages’ are the logs of the low-level protocol interactions with the system under test.

The Recon component generates a root event corresponding to the reconciliation test, which will contain all the related events. Each event generated by the th2 Recon component consists of a pair of messages linked by a unique key (in the demo example, it is ClientOrderID). Just like before, the full content of the linked messages can be inspected separately in the 'Messages' tab.

```python
def get_description(self) -> str:
    return "Check all possible fields between NOS and ER."

def hash(self, message: ReconMessage, attributes: tuple, *args, **kwargs):
    session_alias = message.proto_message.metadata.id.connection_id.session_alias
    cl_ord_id = message.proto_message.fields['ClOrdID'].simple_value
    side = message.proto_message.fields['Side'].simple_value
    symbol = message.proto_message.fields['Symbol'].simple_value
    message.hash = hash(cl_ord_id+side+symbol+session_alias)
    message.hash_info['ClOrdID'] = cl_ord_id
    message.hash_info['Side'] = side
    message.hash_info['Symbol'] = symbol
```

Fig. 4 Message Matching Criteria

Market Data Consistency Checks with Bookchecker

It is very important to have consistent, proper and well-formed market data. Within th2, **BookChecker** is mainly focused on market data consistency checks.

To validate a market data flow, it builds the entire order book. This allows it to perform different tests related not only to specific messages, but also to the order book itself (checks for **crossed or locked books, order position checks, comparing market data snapshots**, etc.). Just like in the Recon case, BookChecker can verify messages generated by any test activity such as exploratory testing, functional and non-functional testing, random load, and others.

To get the Market data messages in our demo example, we are connecting to the Market Data gateway ITCH interface. Then, this information is decoded by the corresponding codec and can be used by the Bookchecker component.

As a result, Bookchecker can build the entire snapshot of the order book which can be used for our checks and provide corresponding statistics (see Fig. 5).

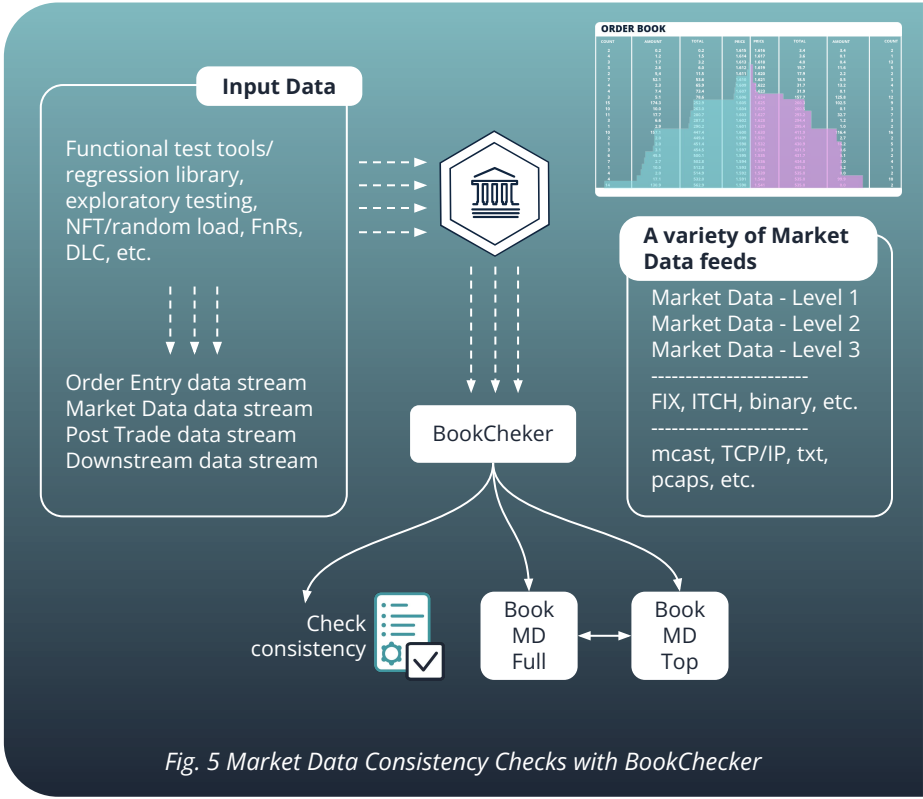


Fig. 5 Market Data Consistency Checks with BookChecker

Data Services

Thanks to th2's feature of keeping a unified storage for all test data, there is an ability to also analyse the results after the test execution happens. Using th2 data services, the data (both messages and events) stored in the th2 data lake can be additionally analysed.

For instance:

1. The data can be aggregated into the respective stats – graphs/charts – for easy test report generation;
2. Information about the available events (either passed or failed) can highlight if there are any gaps in the coverage of the executed tests.
3. Analysis can also provide information about the nature of inputs – permutations used in the tests, the number of messages, their types and so on.

For the purposes of the demo, we extracted all the events produced by the Linear test scripts to see which messages they process to analyse the coverage.

This ends the overview of the testing methods that can be used with th2.

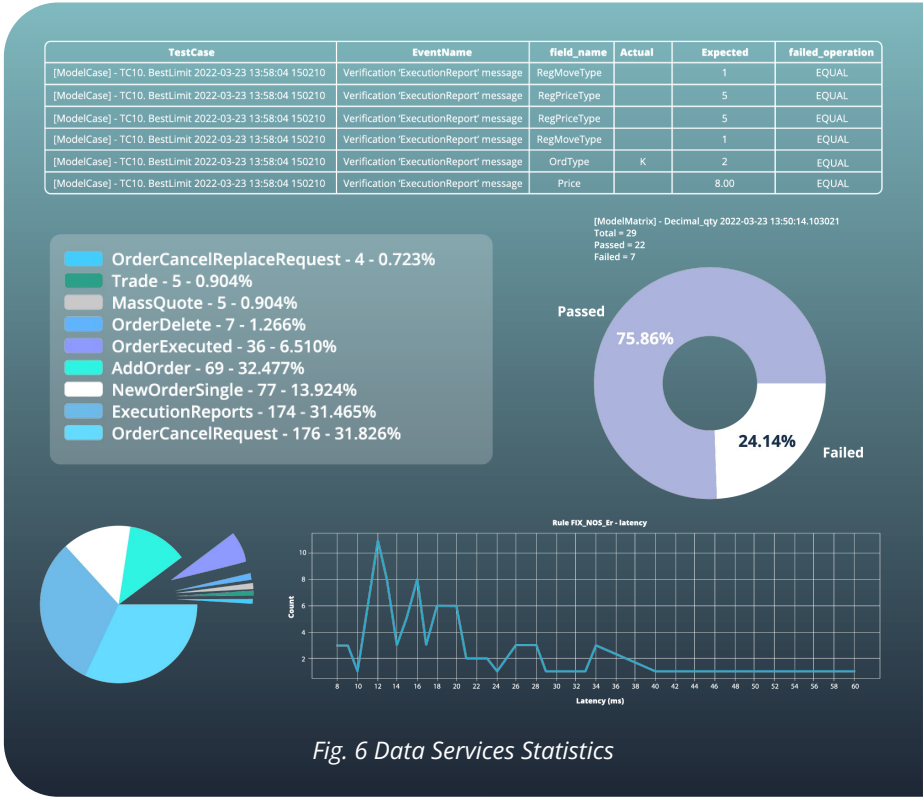


Fig. 6 Data Services Statistics

Thank You!

About Exactpro

Exactpro is an independent provider of AI-enabled software testing services for financial sector organisations. Our clients are exchanges, post-trade platform operators, and banks across 20 countries. Our area of expertise comprises protocol-based testing of matching engines, market data, market surveillance, clearing and settlement systems, payments APIs. We help our clients to decrease time to market, maintain regulatory compliance, improve scalability, latency and operational resiliency. Exactpro is involved in a variety of transformation programmes related to large-scale cloud and DLT implementations at systemically important organisations.

Founded in 2009, the Exactpro Group is headquartered in the UK and operates delivery centres in Georgia, Sri Lanka, Armenia, Lithuania and the UK and representative offices in the US, Canada, Italy and Australia.

Are you considering improving quality, time to market, regulatory compliance, reducing costs or latency? If so, visit us at exactpro.com.

Contact us