



Intelligent Verification/Validation for XR Based Systems

Research and Innovation Action

Grant agreement no.: 856716

D5.4 – Project validation report

iv4XR – WP5 – D5.4

Version 1.1

December 2022



Project Reference	EU H2020-ICT-2018-3 - 856716
Due Date	31/12/2022
Actual Date	30/12/2022
Document Author/s	Joseph Davidson (GA), Fernando Pastor Ricós (UPV), Wishnu Prasetya (UU), Jean-Yves Donnart (THA-AVS), Marta Couto (INESC-ID), Tanja Vos (UPV), Jason Lander (GWE)
Version	1.1
Dissemination level	Public
Status	Final

This project has received funding from the European Union's Horizon 2020 Research and innovation programme under grant agreement No 856716



Document Version Control			
Version	Date	Change Made (and if appropriate reason for change)	Initials of Commentator(s) or Author(s)
0.1	05/10/2022	Initial document structure and contents	JD
0.2	02/11/2022	Change to structures, some details for GoodAI	JD
0.3	16/11/2022	More details for GoodAI	JD
0.4	25/11/2022	Adding Explorative TESTAR section	FP
0.5	13/12/2022	Add spatial coverage section	FP
0.6	16/12/2022	Finish Pilot parts	JD, JYD, JL
0.7	17/12/2022	Executive Summary and Conclusion	JD
1.0	30/12/2022	Reviewer comments, final polish	JD, JYD, VG, JL
1.1	30/12/2022	Final arrangements for submission	RP

Document Quality Control			
Version QA	Date	Comments (and if appropriate reason for change)	Initials of QA Person
0.7	22/12/2022	Review and Edits	AS
0.7	26/12/2022	Review and minor edits	RP

Document Authors and Quality Assurance Checks		
Author Initials	Name of Author	Institution
JD	Joseph Davidson	GA
FP	Fernando Pastor	UPV
WP	Wishnu Prasetya	UU
JYD	Jean-Yves Donnart	THA-AVS
MC	Marta Couto	INESC-ID
AS	Angelo Susi	FBK
JL	Jason Lander	GWE
RP	Rui Prada	INESC-ID

TABLE OF CONTENTS

1 Executive Summary	1
Acronyms and Abbreviations	1
2 Introduction	2
3. Space Engineers	3
3.1 SE Requirements for the Project	3
3.2 Metrics for Evaluation	4
3.3 Realisation of the Project	5
3.4 Evaluation of the Project	14
3.5 Future Work	18
4 Maev (the Nuclear Plant Intrusion Simulation)	18
4.1 Requirements for the MAEV use-case	18
4.2 Description of the use-case and current tests	19
4.3 Metrics for Evaluation	20
4.4 Realisation of the Project	21
4.5 Evaluation of the Project	24
4.6 Conclusions and Future Work	29
5 LiveSite	32
5.1 Introduction	32
5.2 Reading Verification and Analysis	32
5.3 Integration Overview	33
5.4 Train Moving Over Structure	34
5.5 Multi-Sites	35
5.6 Integration with LiveSite	36
5.7 Validation	37
6 Conclusion	39
7 Bibliography	40

1 EXECUTIVE SUMMARY

The iv4XR project, which aims to use artificial intelligence to automate the testing of software, was evaluated through three pilot studies involving the game Space Engineers, the simulation program MAEV, and the real-time instrumentation and monitoring system LiveSite. The results of these pilot studies indicate that the iv4XR project has been successful in improving the software testing processes for these systems.

The project has brought tangible results in terms of time saved for the testing processes of both MAEV and Space Engineers. The MAEV case uses reinforcement learning to replace human operators when testing how penetrable a security system is. For Space Engineers, a mix of scripted and unscripted agents have currently reduced the workload of the game testers by around 10.5 hours, with only 12.5% of the test cases currently automated.

For LiveSite, the value of the iv4XR project has been identified through discovering new errors in buildings and infrastructure. These errors are often indicative of structural defects and the efficacy of discovering them is sometimes a matter of safety for those that use the structures.

The pilot study has shown that iv4XR has thus far delivered tangible benefits to the industrial partners. Some of the partners have planned work to further develop and integrate aspects of the iv4XR framework and agents into their workflows.

ACRONYMS AND ABBREVIATIONS

XR	Extended Reality
GA	GoodAI
SE	Space Engineers
CGE	Computer Generated Entities
CGF	Computer Generated Forces
FTA	Functional Test Agent
SETA	Socio-Emotional Test Agent
RL	Reinforcement Learning
MBT	Model Based Testing
GUI	Graphical User Interface

2 INTRODUCTION

Work Package 5 in the iv4XR project is concerned with making use of the technologies and results of the other work packages in the project. In this project there are three industrial partners who each have applications for piloting agent-based testing. The pilots are a 3D game, an in depth simulation of the security systems of a nuclear power plant, and a real-time monitoring system for smart structures.

This deliverable concludes Work Package 5 by evaluating the results of the project in their application to the pilots. For each of the pilots, we discuss their requirements for an agent-based testing system, identify quantitative and qualitative metrics, and evaluate the realisation of the project with respect to those metrics.

Given the differences between the pilots and their subsequent requirements of the project, the rest of this report is split into sections with each section evaluating one of the pilots.

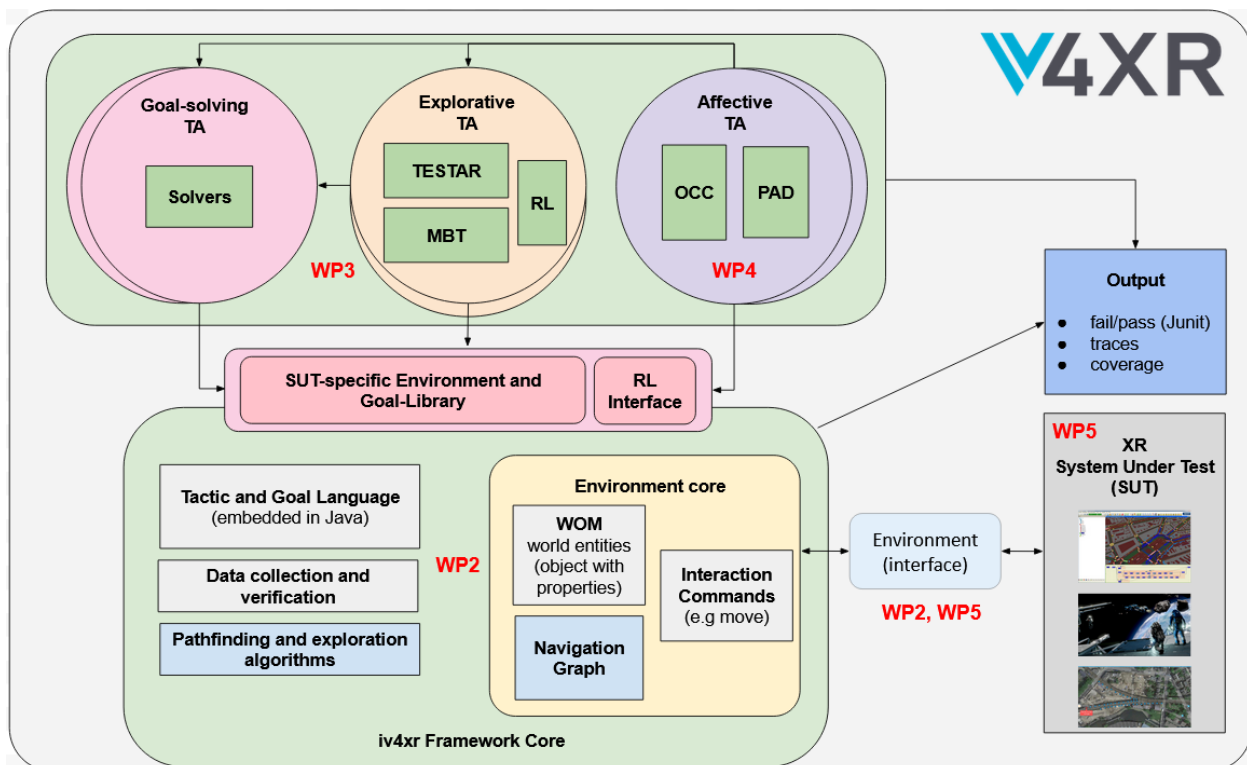


Figure 1: An overview of the iv4XR system. WP5 is on the right hand side and encapsulates the SUTs and interfaces.

3. SPACE ENGINEERS

3.1 SE REQUIREMENTS FOR THE PROJECT

Space Engineers is a game in which multiple players can interact in a simulated world. In this world, the players can find and extract resources, refine those resources into construction materials, and create tools, weapons, or “blocks” out of them. A block is the atomic component of construction and can be as simple as a platform, to as complex as an engine or a terminal which executes user created programs.

Given this complexity, there are a huge number of interactions which can happen not only between players, but also between blocks themselves. Despite this, the development team tests the game manually with no automation. What the testers have produced is a database of tests which cover the most important generic behaviours such as player movement and shared block attributes (can be placed, has some mass, durability, etc.), and tests which cover the specific attributes of blocks like maximum thrust or maximum program length.

Regression and Feature Testing

Typically, during a development cycle, in-progress features are tested, and feedback is provided to the designers. This test-feedback loop iterates until the features are in a stable and complete state. Before a major release containing these features, the above test database is used to complete a regression test of the game. For smaller hotfix releases, the testers curate a smaller set of tests focusing on the features being addressed in the hotfix, but those hotfix tests may not necessarily make it into the regression suite lest the suite grow too large.

The testers are a key part of the development cycle for releases of the game, however they are time pressured primarily through their loop of new feature testing and release testing. Other responsibilities for the testing team include the processing of user generated bug reports, analytics testing and performance metric testing.

The amount of time that the testing team has is the limiting factor on their activities. Therefore one of the aims of the project is to develop testing tools to save time. The testers have a deep knowledge of the systems from a players point of view allowing them to intuit the sources of bugs from a systems perspective. A lot of that knowledge is wasted in regression testing, so methods for automating these tests and providing coverage would enable the testers to put their knowledge to use in more challenging scenarios.

Scriptless Exploratory Testing

The regression tests cover the essential functionality of players, blocks, and tools. However, the interactions between these entities are a huge complex space. Bugs found outside of regression and feature testing often concern interactions between nearby blocks. It is, however, prohibitively expensive to extensively probe the interaction space between all blocks and create tests for each of them.

TESTAR is a tool which has been integrating its capabilities as Functional Test Agents (FTAs), which can provide exploratory testing to cover these complex interaction spaces. TESTAR is an FTA that attempts to map state spaces in an online fashion logging state transitions and monitoring logs, error reporting, and oracles so that events that are considered errors can be reported.

Model Based Testing

Model Based Testing (MBT) uses a similar approach where the interaction spaces are expressed as an extended finite state machine and goals are articulated. The MBT system probes the state machine to find multiple paths to the goals which it then executes. In doing this, the coverage of the scenario is calculated, and the feature coverage obtained is a function of the complexity of the scenario.

Work Package 3 goes into both exploratory and model based testing agents in detail.

Affective Testing

Another aspect of the project is in the prototypical testing of scenarios that may be functionally sound, but where there are unknowns in terms of players reaction - does the game portray information legibly? What is the anticipated player reaction in the encounter design?

While Socio-Emotional Test Agents (SETAs) testing could be a tool for developers in the future, it is hard to find useful cases for SETA given the current state. We have been working with the SETA researchers to facilitate the development of their agents on the Space Engineers platform. See Work Package 4 for more details on the SETA development.

3.2 METRICS FOR EVALUATION

Time

As discussed above, a major indicator for the success of the project is in how much tester time can be saved by introducing automation into the testing process. Currently, a large time sink for the testers is in the regression tests, so one key metric will be in the time that is saved through investing effort into automating the tests and executing them versus manually testing the release each time.

Bugs/Coverage

These tests are for regression testing, however, where the functionality is expected to work. We can evaluate more exploratory testing afforded by agents and methods such as TESTAR and MBT by constructing scenarios in which we know there are bugs (as confirmed by user reports) and seeing if they are able to find those bugs.

Another angle to consider is the coverage of the code. The above scripted regression tests are comprehensive in the functionality that they test, but that functionality is often isolated from other functionalities. For example, a test of movement will have a flat plane for the agent to

walk/run/sprint on, but it is an error if they clip through an obstruction. The separate block tests will check clipping issues with one block, but may not cover the myriad of cases where the block is not placed “conventionally”.

Code coverage can be calculated from the regression tests and the exploratory test from TESTAR and MBT agents. Since FTAs have different types of capabilities, which can be leveraged more effectively in different SE scenarios, we expect to see and anticipate that the techniques are complementary. Where the regression tests cover some functional features, the exploratory FTAs cover others. These expectations are based on the demonstration with empirical evidence about the complementarity between scripted and scriptless approaches for Graphical User Interfaces (GUI) systems [5, 6].

3.3 REALISATION OF THE PROJECT

The Plugin

To connect Space Engineers (SE) to the iv4XR aplib¹, an interface² was developed. This interface is a plugin for SE which features the communication layer with the game and two clients (written in Kotlin and Python) that wrap the functionality of the communication layer and provide a collection of higher level APIs which provide control to the developers.

The Kotlin client³ is the main interaction method with the game in the context of the iv4XR project, the agent, and any other testing tools. Kotlin [2] is a JVM based language developed by JetBrains that is designed to be fully interoperable with Java.

The Python client⁴ is a thin client that exposes the essential functionality for the game and is not as fully featured as the Kotlin version. It was created to demonstrate how stakeholders could work with the game through a python interface and contains some examples such as rendering png files in game as a block construction or generating mazes.

The APIs⁵ in the plugin covers most aspects of the game that are relevant for testing. Some highlights include:

- `spaceEngineers.controller` - Controlling the game
 - `Character` - Character movement and actions
 - `Blocks` - Placing or removing blocks
 - `Observer` - Getting observations of the environment
 - `Session` - Loading/Saving, Connecting/Disconnecting
- `spaceEngineers.movement` - Structures used for moving the character around.
- `spaceEngineers.model` - Definitions of blocks that can be found in the game

¹ <https://github.com/iv4xr-project/aplib>

² <https://github.com/iv4xr-project/iv4xr-se-plugin>

³ <https://github.com/iv4xr-project/iv4xr-se-plugin/tree/main/JvmClient>

⁴ <https://github.com/iv4xr-project/iv4xr-se-plugin/tree/main/PythonClient>

⁵ <https://iv4xr-project.github.io/iv4xr-se-plugin/index.html>

- `spaceEngineers.iv4xr.goal` - aplib goals and tactics
- `spaceEngineers.iv4xr.navigation` - aplib navigation routines

Regression Tests

Given the requirements of the regression tests and the current capabilities of the plugin and models, there is an estimate that around 75-80% of the regression tests can be automated using iv4XR. Tests that cannot be currently automated and require more work are tests such as installing the game, making sure that graphical renderings are correct and that sounds are played.

The agents that execute the regression tests follow a script and are placed into an environment that is designed for their testing (Figures 3-5). These testing environments have been hand crafted by a level designer and have separate stations which test a small part of the functionality of the game.



Figure 3: The environment for scripted tests is arranged into sections where each section tests some functionality, a section may have multiple stations for more fine-grained unit testing.

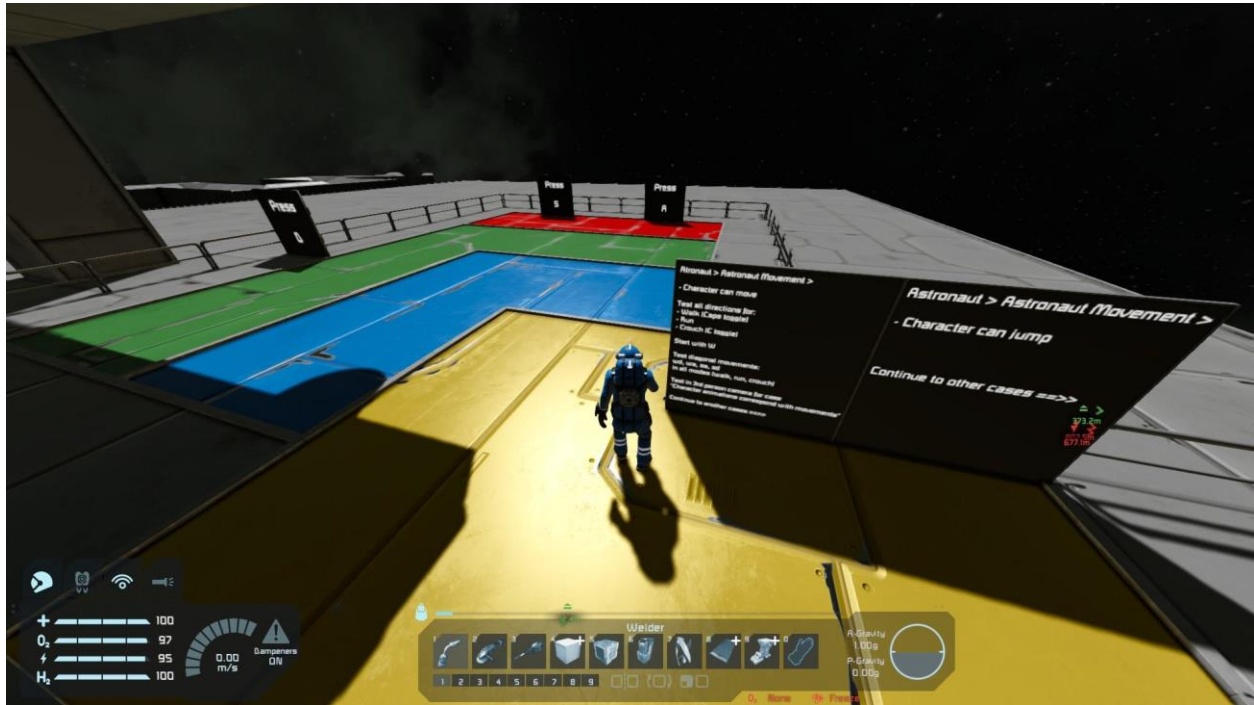


Figure 4: The environment for scripted tests. A movement station is pictured.

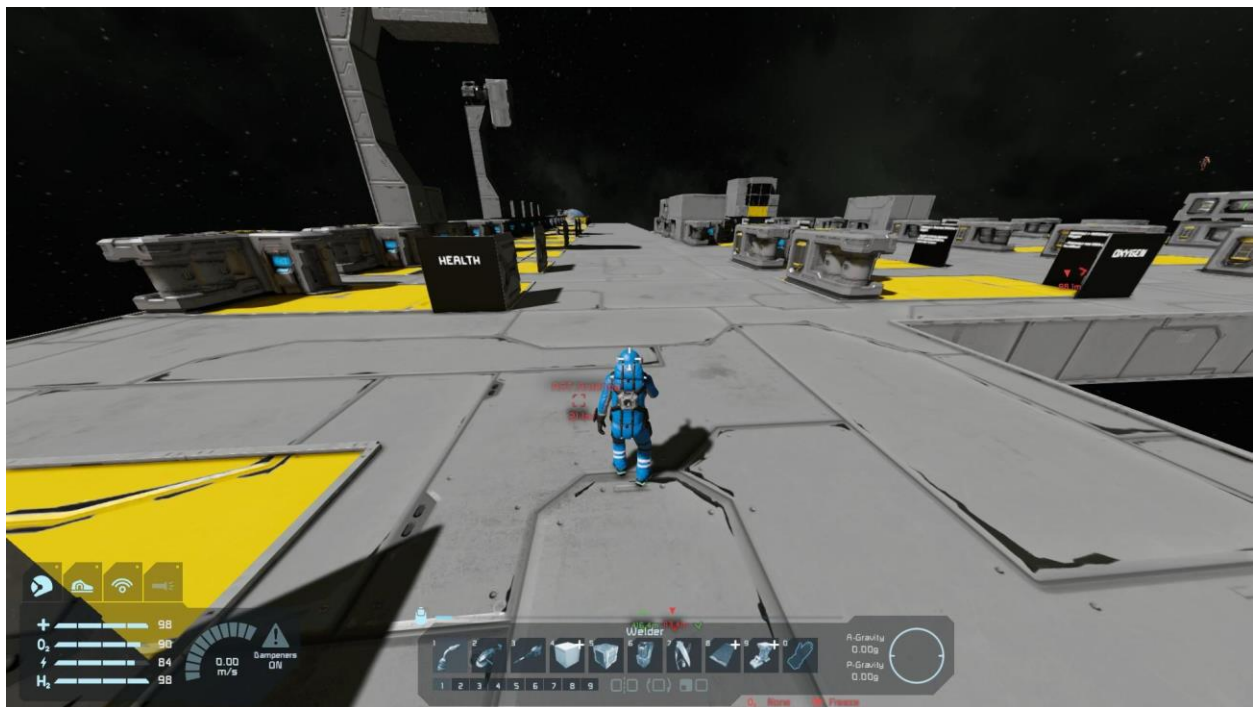


Figure 5: The environment for scripted tests. The Oxygen and Health sections are pictured.

The agents follow a script which is expressed in Behaviour Driven Development (BDD) tests. These tests are written in a clear English-like language using a given-when-then syntax (see Figure 6).

Feature: C197544 Mass of the object is added to the overall grid mass (an update of the grid has to happen to see this)

Background:

Given Scenario used is "simple-place-grind-torch-with-tools".

Scenario Outline: C197544 Mass of the object is added to the overall grid mass

Given Block type "<blockType>" has mass <mass>.

And Observed grid mass is <gridMassBefore>.

When Character selects block "<blockType>" and places it.

Then Observed grid mass is <gridMassAfter>.

Examples:

blockType	mass	gridMassBefore	gridMassAfter
LargeBlockCockpitSeat	1768	50000	51768

Figure 6: A listing of a BDD test which checks that the mass of a block is added to the collection of existing blocks (known as a “grid”)⁶

depicts one such test. The language and syntax used is Gherkin⁷ and the client maps phrases and terms such as `Character selects ...` and `grid mass` to explicit behaviours in the game. In using the Gherkin language and correctly structuring the mapping of phrases, testers can implement cases with little technical expertise. On the occasions that they do need to create a new mapping, someone with suitable expertise will be required, but the mappings themselves are concise and can be made reusable and generic (Figure 7).

```
@When("Character selects block {string} and places it.")
fun character_places_selects_block_and_places_it(blockType: String) = mainClient {
    val toolbar = items.toolbar()
    val toolbarLocation =
        toolbar.findLocation(blockType) ?: error("cannot find $blockType in toolbar")
    val definitionId =
        toolbar.items.first { it?.id?.type == blockType }?.id ?:
            error("Cannot find $blockType
in toolbar")
    items.setToolbarItem(definitionId, toolbarLocation)
    items.equip(toolbarLocation)
    delay(150)
    blocks.place()
    items.equip(ToolbarLocation(9, 0))
}
```

Figure 7: A listing of Kotlin code which maps the above highlighted “Character selects block...” to API calls⁸

⁶ <https://github.com/iv4xr-project/iv4xr-se-plugin/blob/main/CucumberTester/src/main/resources/features/C197544.feature>

⁷ <https://cucumber.io/>

⁸ <https://github.com/iv4xr-project/iv4xr-se-plugin/blob/main/CucumberTester/src/main/kotlin/bdd/CharacterActions.kt>

The automated regression tests are being implemented in a methodical fashion - of the 8806 separate regression tests there are 1322 that have been selected for implementation in this pilot. These tests cover the movement, actions and vital statistics of the astronauts. Tests that are not to be implemented at this juncture are tests of specific blocks, menus, installation of game, mods, scenarios and graphical checks.

Of the 1322 planned tests for the evaluation, 12.4% (164) of these have been implemented at the time of writing. We are interested here in the comparison between running these tests manually versus the time taken for the automation.



Figure 8: An excerpt of the regression report.

The results of the running tests are compiled into a report (Figure 8) and are stored into the company internal test tracking tool TestRail⁹. The report contains information about which tests fail, the reason why they have failed, the step that they have failed on, and a screenshot of the game at the time that the test has been completed.

TESTAR FTA

The TESTAR tool is an agent that explores the environment in a scriptless approach and does not follow specific tactics, goals, or crafted models to test the System Under Test (SUT). The underlying principle of the TESTAR *scriptless* exploration consists on: generate test sequences of (state, action)-pairs by connecting to the SUT in its initial state and continuously selecting an action to bring the SUT to another state while the TESTAR agent checks oracles to determine if the SUT contains a failure.

These oracles can be customised to examine the State or SE logs to detect suspicious messages, analyse the functional features of the SE blocks (e.g., integrity increases/decreases after grinding or welding) and the character (e.g., health is restored after interacting with a medical room) responds adequately to the TESTAR actions, and that random constructions work correctly according to existing materials.

⁹ <https://www.gurock.com/testrail/>

As the TESTAR agent explores the SE system, the FTA uses the observed information to collect and generate different types of reports such as a State Model that maps the observed states and executed transitions, an HTML report that contains visual and textual information, and a spatial map image that represents the space explored by the agent. Additional details regarding the State Model or the reports can be found in **Deliverable 3.5**.

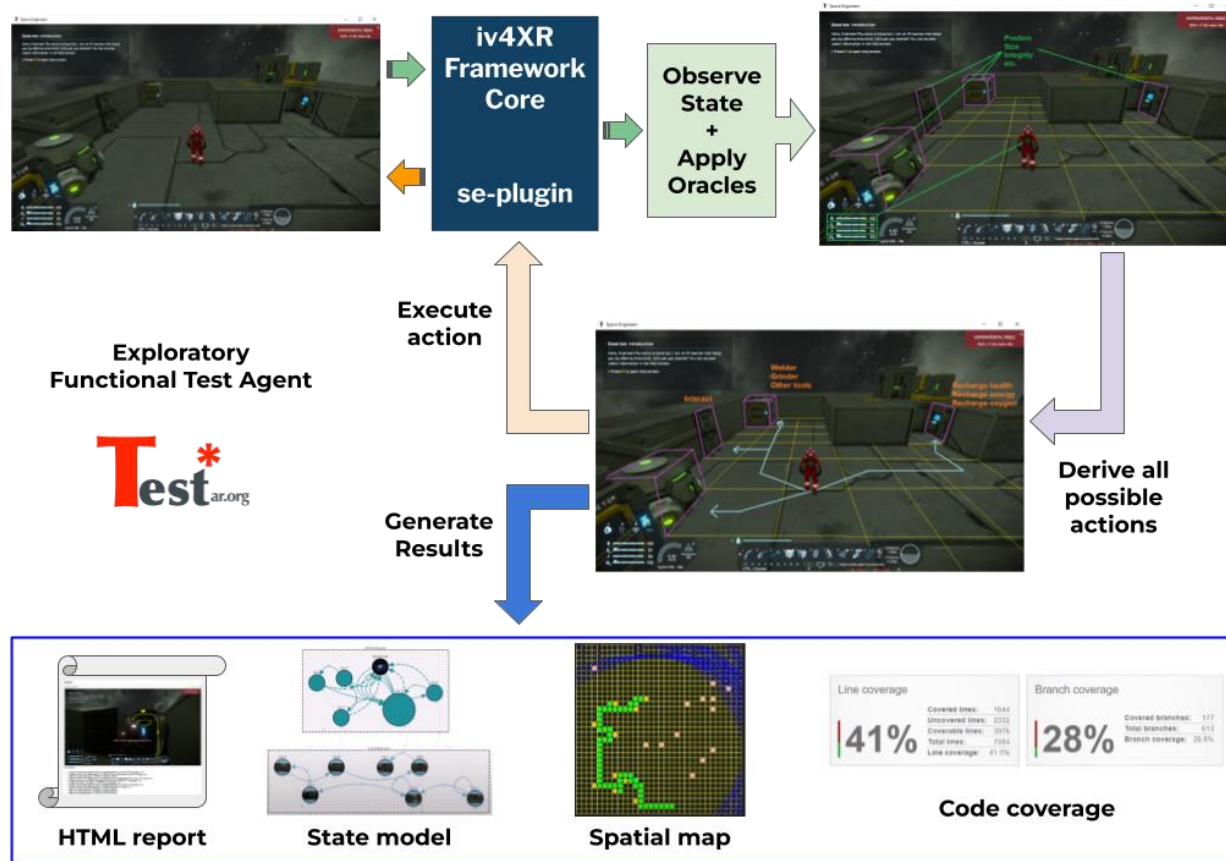


Figure 9: TESTAR explorative process with SE

TESTAR Action Selection Mechanism

The TESTAR agent uses an Action Selection Mechanism (ASM) to make runtime decisions about which action should be selected next as it observes and discovers the different types of blocks in the SE environment. The default ASM implemented in TESTAR was based on selecting a completely random action among the entire set of actions available in each state.

However, this random selection can be inefficient in XR environments where many exploratory movements can be executed because the agent can omit the interaction with different SE blocks. Also, when the TESTAR agent selects the following exploratory action, it does not consider the previous positions and can select to explore the same one.

An algorithm that prioritises selecting actions that interact with newly observed entities and tries to avoid exploring the same positions has been implemented to reduce the degree of randomness. This ASM improves the effectiveness of the TESTAR exploration in terms of

interacting with different functional blocks and covering a more significant amount of SE space. **Table 2** shows the improvement this new *Prioritise Interactions* algorithm brings to TESTAR.

TESTAR results

Along with developing the iv4XR framework, TESTAR has been iteratively extended to fit as an exploratory FTA, helping to test and improve the robustness of the SE plugin when the first explorations were executed. Throughout the project's last stage, TESTAR created a navigation grid on the fly using the geometry information of the observed entities to calculate the path of nodes that it can follow to reach a position and interact with an entity. Using this feature in previous versions of SE with documented bugs, the partners were able to demonstrate that the exploratory TESTAR can find functional failures, such as perceiving that the jetpack of the agent activates automatically after interacting with a ladder or detecting that the integrity of the blocks does not decrease in creative mode.

TESTAR links

The exploratory capabilities of the TESTAR agent and the generated reports can be found in these videos:

- [TESTAR exploring and testing Space Engineers functional features](#)
- [TESTAR calculating the navigable path to interact with the Space Engineers blocks](#)
- [Scriptless testing of TESTAR with v199 of SE to reproduce and detect a jet-pack bug](#)

The integration process in the TESTAR tool to become an FTA that explores XR systems and a fuller description of this iv4XR agent capabilities can be found in Deliverable 3.5, and in the wiki section of the GitHub repository: [TESTAR iv4xr](#)

EvoMBT

The tool EvoMBT combines model-based testing (MBT) with search algorithms for the automatic generation of test cases for generic XR systems. EvoMBT instantiates model-based testing (MBT) cycle (see Section 4 of D3.5) for Space Engineers, as depicted in Figure 10.

The MBT process starts from a specific set of features of the system under test (SUT) that are of interest to the tester. For the iv4XR project, we consider a maze-like scenario involving a player interacting with surrounding entities. A maze consists of several rooms connected by doors. Doors are opened/closed by pressing one or more buttons. It is however not necessary that all buttons be connected to doors, i.e., some buttons may not be connected to any door. A game level is created by the game designer through a customised level notation and saved as comma-separated value (csv) files. The SE plugin exposes the API used by EvoMBT to load a csv level and create the corresponding maze into SE.

The tests generated from the model are abstract and they need to be concretized before being executed on the actual game under test. EvoMBT is integrated into the iv4XR framework and translates each transition into an iv4XR Goal Structure (see D2.4 for a detailed description of the framework). In particular, exploiting the SE plugin API, we define a set of Tactics and Goals that allow an autonomous agent to navigate a map, press a button, and check the status of a door. SE Tactics include:

- `Tactic navigateToButton(String buttonId)`
The agent navigates to button `buttonId`
- `Tactic navigateToDoor(String doorId)`
The agent navigates to door `doorId`
- `Tactic pressButtton(String buttonId)`
The agent presses button `buttonId`

Tactics are used in SE Goal Structures:

- `GoalStructure buttonInCloseRange(String buttonId)`
The goal is satisfied if the distance between the agent position and the button is less than a given value. The goal uses tactics `navigateToButton`
- `GoalStructure doorInCloseRange(String doorId)`
The goal is satisfied if the distance between the agent position and the door is less than a given value. The goal uses tactics `navigateToDoor`
- `GoalStructure buttonIsInteracted(String buttonId)`
The goal is satisfied if the agent position is in interaction distance from the button. The goal uses tactics `pressButton`
- `GoalStructure doorIsOpen(String doorId, TestAgent agent)`
The goal is satisfied if the agent checks that the door is open. Here, the goal returns a verdict that will be used to build the test oracles.

Concretization, i.e., translation from EFSM transition to SE Goal Structures is straightforward. Moving to a door (button) becomes `doorInCloseRange` (`buttonInCloseRange`), interacting with a button is the sequence of `buttonInCloseRange` and `buttonIsInteracted`, and traversing a door uses `doorIsOpen` to check if the door is open and the action can be completed. Goal `doorIsOpen` defines also a test oracle because, when translating to a goal structure, the system knows the precise sequence of buttons pressed and therefore if a door should be open or not.

The last step in the MBT cycle involves executing concretized test cases on the actual SUT. The SE plugin integrates iv4XR framework and allows defining an autonomous agent that executes iv4XR goals into the game.

EvoMBT collects coverage information during both abstract test cases generation and concrete test cases execution. Abstract test cases generation is driven by coverage criteria meaning that EvoMBT searches for a set of test cases that satisfy the given criteria. EvoMBT supports different notions of coverage defined on EFSM, including state coverage, transition coverage, and k-

transition coverage. In the SE maze-like settings, state coverage corresponds to requiring that the autonomous agent can reach all the entities in the map (i.e., buttons and doors). Transition coverage requires that all the possible actions are performed (e.g., all buttons are pressed). Finally, k-transition is equivalent to demanding that all the sequences of k-actions are performed by the agent. Then, when running concrete test cases in SE, EvoMBT collects detailed information about every GoalStructure being executed. Moreover, when executing test cases in SE, code coverage can be collected.

For more information on the EvoMBT model, test generation and execution, see D3.5.

3.4 EVALUATION OF THE PROJECT

The metrics for the quantitative evaluation are time and various degrees of coverage. To collect time, we take the benchmark time for the testers to run their regression tests manually, then we measure the time taken by the agents to run their regression tests.

There are multiple definitions of coverage. For our purposes, statement coverage is defined as the proportion of statements in the code that are executed during a test. We gather the statements that are covered by the various approaches and determine which sets of statements are uniquely covered by the testing approach. In Table 1 below, we can see that all testing approaches have some small set of unique statements associated with them.

Additionally, there is the concept of model coverage which a tool such as TESTAR will report. A level in SE can be defined by a model of states connected through actions, model coverage in this sense is therefore how much of the model has been explored by TESTAR.

For statement coverage, we create `.pdb`¹⁰ files for Space Engineers which are files that hold debugging information for the game. We run agents or tests referencing these files using the coverage program dotCover which then produces a report providing statistics for which source statements are executed during the testing process.

Time

In a typical development cycle for Space Engineers, the full regression suite of 8806 tests is run once a release candidate build is ready. The running of this candidate takes around 380 hours. Of that, the subset of 1322 tests that we are evaluating takes approximately 90 hours due to setting up scenarios, entering results and so on, and so the 12.4% implemented takes around 10.5 hours to execute and report for a single person.

The automated tests take around 47-52 minutes to execute, with the TESTAR method being faster at around 15 minutes, and MBT method taking longer at around 70-72 minutes. For the regression tests, this results in a saving of approximately 10.5 hours of manual testing and reporting activity.

¹⁰ https://en.wikipedia.org/wiki/Program_database

Variance in the reported times for the regression tests are often due to retests of particular functionalities. While the testing is reliable, the game is not fully deterministic so some tests are repeated a number of times before they pass. TESTAR itself has some variance due to action selection also as the exploratory methods employed are not necessarily deterministic either.

The reported saving of 10.5 hours of tester time at this point is encouraging. As mentioned above there is also a generated report which can serve to highlight tests which have failed for more investigation. We have found that this reporting has streamlined the process of verifying that tests have been completed correctly and that tests which have failed have either failed for known reasons or require more detailed investigation.

Statement Coverage

We obtain a baseline of statement coverage by opening the game, loading the movement testing scenario, then closing it. This resulted in a coverage of 18.7% which is to say that 18.7% of the statements that comprise the game are executed when doing this. The coverage report also presents figures for iv4XR related code and the major third party library Havok¹¹ which are removed from these results.

	Baseline	Movement tests	Action tests	TESTAR Prioritise Interactions	MBT
Coverage (%)	18.7%	29.4%	30.5%	25.9%	25.2%
Unique statements covered	222	4481	10847	7709	7200
Slowest Time (min)	4	32	20	15	72
Cumulative coverage (L to R)	18.7%	29.9%	32.7%	34.1%	35.2%

Table 1: Statement coverage statistics of various approaches

In Table 1 we see the raw coverage percentages from each testing approach, as well as the number of unique statements, relative to the combined coverage of all the other tests. The cumulative coverage depicts the overall coverage of the codebase as the test reports are merged from left to right.

Introspection of the various tests reveal the broad areas of their unique coverage:

- Baseline: saving and exiting the game, clearing logs, error checking.
- Movement: more varied movement types like sneaking, sprinting, and using the jetpack.

¹¹ <https://www.havok.com/havok-physics/>

- Actions: Drilling for minerals, using emotes, dampeners, astronaut oxygen.
- TESTAR: Interactions with ladders, cryo chambers, placing blocks, damaging blocks with weapons.
- MBT: Interactions with buttons and opening doors.

In addition to these behaviours a lot of the statements are to do with reading attributes which are used by the oracles for checking the consistency of the world.

The regression tests have high overall coverage mainly due to their specialisation within their functional set. When designing the tests, all types of movement or actions are considered. The EvoMBT and TESTAR approaches require that these functional sets are available to the models or agents in order to be used. So while TESTAR and MBT have a broader range of action sets, the depths of those action sets are shallower.

At the time of writing, the TESTAR and MBT methods provide coverage of statements that are not as yet covered by the automated scripted testing due to their consideration of more functional sets. The scope of the scripted tests are continually increasing, but the intention is to also develop the TESTAR and EvoMBT sets to give them more depth in the sets of functions which they can perform.

The creation of the regression tests is a labour intensive task. The original descriptions of the test need to be broken down into more “machine friendly” steps, a testing level should be designed with stations to test each aspect of the behaviour, and the scripts need to be written. Each of these tasks take expertise in the game and familiarity with programming, so it is often the job of multiple people. However, once these tests are automated only minor maintenance will be required to keep them working.

Spatial Coverage

The SE system employs an XML file to store the information of the existing scenarios on which the agents execute the testing process. This file contains information regarding the relative position of the blocks representing the environment's floor and the coordinates of the existing functional blocks with which the agent can interact. It is possible to use this information at runtime to create a 2D map to obtain *spatial coverage* metrics that indicate the number and percentage of observed and interacted blocks and explored positions.

Figure 11 shows the spatial information obtained from an SE level. The yellow squares represent the unobstructed floor on which the agent can perform 2D navigation. The information regarding the navigated positions (green) is stored as the agent moves to interact with an entity or to explore. As the FTA moves through the SE level and the observation range changes, the information regarding the positions of the observed and the interacted entities are stored. Finally, the position of all the functional entities is extracted from the XML file to obtain metrics that indicate the percentage of interactions.

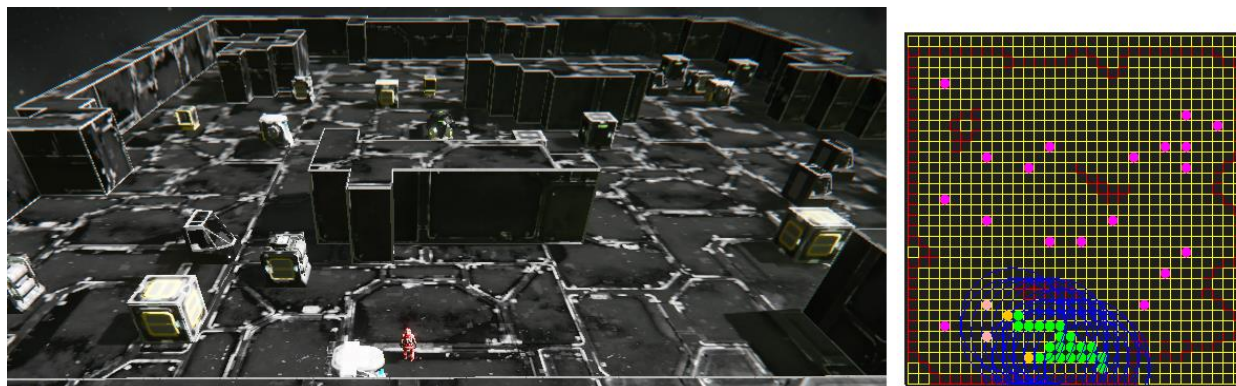


Figure 11: Spatial Coverage map created by using the XML scenario information

Table 2 is an example of the usage of the spatial coverage metrics to show a small-scale comparison between TESTAR Random and Prioritise Interactions ASMs with an agent observation of 4 blocks distance. The SE level was generated automatically, which created 23 functional blocks and 837 floor positions. While the random exploration offers weak results regarding interacted entities (grinder) and navigated space, more sophisticated ASM can allow exploratory tools to improve the effectiveness of the testing process.

TESTAR ASM	Random	Random	Prioritise Interactions	Prioritise Interactions
Executed Actions	50	100	50	100
Functional Blocks	23	23	23	23
Functional Observed	7	15	16	23
Interacted Functional	1	5	16	23
% Observed	30,43 %	65,22%	69,57%	100%
% Interacted	4,35 %	21,74%	69,57%	100%
Floor Positions	837	837	837	837
Explored Positions	68	151	119	278
% Explored	8,12%	18,04%	14,22%	33,21%
Average action time	5,12 seconds	5,81 seconds	7,82 seconds	7,9 seconds
Total time	289 seconds	635 seconds	424 seconds	848 seconds

Table 2: Spatial coverage using different TESTAR ASMs

3.5 FUTURE WORK

At present, only ~12% of the planned regression tests have been completed, so further work is to continue in this automation effort. The current estimate for total feasible automation is around 75-80% of the current set, with the remaining 20-25% are more difficult to automate tests like installing the game, checking graphics or sounds. While the system can determine which textures or sound files should be used, it is more challenging to verify that the usage is actually reaching the player.

For the graphical side, one method to automate these tests suggested in previous deliverables is to have the agent perform a “photoshoot” in which it collects screenshots of blocks and can compare those screenshots with reference photos to flag any potential texture issues for human checking. However, there are still some significant technical challenges to overcome before it can be used.

The exploratory testing of agents like TESTAR have shown that they provide extra statement coverage of the code base. As more tests are implemented, further work will be to empower the exploratory procedure to give it the capabilities and oracles to allow it to find errors in more open-ended scenarios.

We plan to continue improving the spatial coverage map by adding information regarding the different functional block types and distinguishing between the actions performed. Also, to prepare an empirical study with multiple executions at the SE level, to extend the results of **Table 2**, and to obtain significant evidence regarding the ASM effectiveness.

4 MAEV (THE NUCLEAR PLANT INTRUSION SIMULATION)

4.1 REQUIREMENTS FOR THE MAEV USE-CASE

The "Nuclear Plant Intrusion Simulation" has been designed as a proof of concept of the use of Artificial Intelligence techniques to automate the testing of simulation environments integrating dynamic scenarios where human operators are involved.

Thales AVS, in its “Training & Simulation” unit, generally builds this type of environment to simulate real situations for training or study purposes:

- In the case of a training, the objective of the tests is to replace the future students by operators who will test all the possible behaviours that these people could exhibit during an exercise and to verify that a specific behavioural sequence will not lead to a bias in their training.
- In the case of a study simulation, the objective for the testers is to evaluate a given strategy or tactic by imagining all the possible ways to counter this strategy or tactic.

Today, we know how to model in a dedicated simulation the procedural or doctrinal behaviour of real actors who follow very precise rules and which are, therefore, easily reproducible and

testable by artificial agents. On the contrary, it is much more difficult to model the behaviour of students in a training course or of a human adversary who tries to defeat the strategy or tactics set up in a study simulation. This is why we still are using many human testers to validate the simulations where man will be in the loop or where human behaviour is not easy to reproduce.

This use of real operators entails, of course, a human cost (in terms of number of operators) but also a time cost since the tests can only be done, at least, in real time. Moreover, it is difficult to prove the complete coverage of the tests because of the uncertainty on the behavioural choices of the future trainee or of a real human adversary.

Therefore, the requirements for the MAEV use-case should answer to the first objective declared for the evaluation and validation of the iv4XR project, which is the development of test agents:

1. that have the ability to solve test goals that human testers require little (e.g., 95%) or medium (e.g., 60%) effort to solve;
2. that are able to replace at least 50% of the manual testing (reduction in human effort);
3. and that are able to cover a much larger part (e.g. 10 times) of the XR interaction space than existing manual testing (improved verification strength).

4.2 DESCRIPTION OF THE USE-CASE AND CURRENT TESTS

The "Nuclear Plant Intrusion Simulation", developed by Thales AVS, is a use-case simple enough to verify that test agents may be able to replace human testers in the more costly tasks, and close enough to a real example to allow the extrapolation of the results to more complex application.

In this use-case, we want to test in simulation the defence security of a nuclear power plant against a possible intruder. In the worst case, this intruder may be a terrorist, in a more optimistic scenario, he may be a simple activist who wants to demonstrate the danger of this kind of infrastructure.

Assuming that the number of cameras and guards is limited, we want to automatize the tests of each defence strategy proposed by the security officer and verify that this use of test agents will lead to an effective reduction in human effort and time spent, and to a much better coverage of the possible tests.

The "Nuclear Plant Intrusion Simulation" has been modelled from the configuration of a real French nuclear plant (site of Percheville) with additional buildings in order to increase the difficulty of the problem.

This simulation was modelled with the Thales AVS constructive simulation MAEV which allows the 2D and 3D modelization of any natural or urban environment, the simulation of specific devices (e.g. the surveillance cameras) and the animation of dynamic actors (e.g. the patrols of guards).



Figure 12: The nuclear plant of Percheville

In our current testing process, the intruder is played by one or several human testers that tries to defeat the defence strategy put in place by the security officer by seeking to reach the core of the plant.

When one tester finds a way to reach the core of the plant, the solution is analysed to decide if a modification of the defence strategy may prevent the intrusion without implying another breach in this defence, or if additional surveillance cameras or guards may be needed to secure the site.

The problems of these tests is that they are obviously very time and resources consuming and that it may be very difficult to certify that there is no solution to the problem. The only thing we can expect is that if there is an obvious solution for the intruder, it will be found by the human testers.

Note: This security problem will not occur with an unlimited number of cameras or guards, or by concentrating the existing means around the place to defend. However, the security officers must be able to work with limited means and to ensure the security of the whole site.

4.3 METRICS FOR EVALUATION

The evaluation of the security of a critical infrastructure is a very difficult task, especially when the resources available to defend the site are limited. Using the simulation to test the defence strategy allows the test of scenarios difficult to reproduce in a real environment and allows a certain form of parallelization of the tests with several experts mandated to discover the possible breaches in the defence strategy.

Using test agents will obviously help to reduce the number of operators needed to test the defence strategy proposed by the security officer. We only need to identify the part of the tests that can not be automated, especially during the development of the test agents.

We would also expect a reduction of the time spent to perform the tests because these ones don't need to be performed in real time with the simulation. On the other hand, the use of Artificial Intelligence and, especially, of machine learning may also be time consuming.

And, finally, we need to quantify the coverage of the tests and their performances. However, the comparison of this criterion with the corresponding coverage and performances of human testers may be difficult to evaluate.

4.4 REALISATION OF THE PROJECT

When you want to reproduce the behaviour of real humans in a simulation, you may use several Artificial Intelligence techniques :

- The first option is to implement scripted behaviours that represent the common behaviour of real actors in a predefined scenario. This kind of modelling is especially adapted to the modelisation of procedural or doctrinal behaviours where every reaction can be formalised.
- A more realistic behaviour modelisation can be achieved by integrating goal directed behaviour and human factors (such as motivations and emotions) in order to enhance the autonomy of the artificial agents and exhibit human behaviour.
- Finally, the modelisation of the cognitive functionalities of real humans, such as planning and decision making, will increase the quality of the behaviour and its representativity.

In our case-study, the “Nuclear Plant Intrusion Simulation”, the behaviour of the guards can be implemented using models related to the first category (scripted behaviours) since this kind of behaviour is very procedural.

In Thales AVS, we usually use a simulation component called “Computer Generated Entities” (CGE) to animate the behaviour of artificial actors in our simulation and perform all the interactions of these actors with the other objects or elements of this simulation. The doctrinal behaviour of the guards in the nuclear plant can be easily implemented with the mission generator of MAEV as is the detection capacities of the cameras and the displacement of all the actors.

The modelisation of the behaviour of the intruder is more complex. It should not only represent the realistic behaviour of a real intruder but rather the behaviour of a human tester who tries to challenge the defence strategy implemented by the security officer.

This behaviour can be implemented as a goal-oriented problem and, more precisely a planning and optimization problem with static and dynamic constraints: “Trying to reach the core of the plant by avoiding the detection of the surveillance cameras and the patrols of guards”.

For the proof of concept, we have chosen to use the Deep Reinforcement Learning algorithms, integrated by our partner Thales SIX in the iv4XR Framework, to optimise the behaviour of the artificial intruder. The use of such a framework will allow in further projects to use other test

agents for this kind of case studies or to use such AI algorithms to test other kinds of applications.

To build the pilot, we have carried out several modifications to our simulation components and, especially, to the CGE (Computer Generated Entities) MAEV. In our constructive simulations, the role of MAEV is to model the physical and decisional behaviour of all the actors or objects of the simulation, which correspond mainly in our case study in the modelisation of the guards or of the surveillance cameras.

Note that the decisions of the artificial intruder are delegated by MAEV to the AI algorithms in charge of the tests, unlike its physical behaviour (movements and perceptions) which remains managed by MAEV.

These modifications fall in two categories:

- The acceleration of MAEV in order to perform the tests faster than the real time;
- The opening of the MAEV interfaces to allow extended interactions with external tools:
 - Control of the basic functions of the simulation (creation of actors, launch and stop of the scenario, management of simulation time, etc.);
 - Control of the behaviour of one or more actors in the simulation (in this case only the artificial intruder);
 - Transfer of extended information on the state of the actors (positions, perceptions, etc.).

We needed more open interfaces to be able to transfer data between MAEV and the AI algorithm. We also had to develop a specific software gateway to connect MAEV to the iv4XR Framework and benefit from the iv4XR Agent Model to assure this transfer of information.

Note that the source code of this gateway is available on the site GitHub of the project, unlike the executable and source code of MAEV for obvious Intellectual Property reasons.

Regarding simulation acceleration, the modifications allow the possibility to run the MAEV simulation about 10 times faster than real time in our use-case scenario. While this speedup rate might be sufficient for many testing tools in the industry, Reinforcement Learning algorithms usually require a higher ratio (e.g. 100 or 1000 times faster than real time) to perform the tests in reasonable times.

In order to bypass the problem, we used, with our partner Thales SIX, an incremental learning methodology, often referred to as "Curriculum Learning". This methodology consists in carrying out the first phases of the learning using a simpler version of the target simulation to orient the learning more quickly towards the most promising solutions before transferring the learned model to the target simulation where the learning can be finalised in a more realistic environment with finer models.

Thus, the first phases of the learning of the behaviour of the test agent have been carried out with a simpler simulation (from Thales SIX) called SE-STAR. This simulation implements a simplified version of the environment, and of the perception and movement of the test agent, that allow each simulation to run 50 times faster than the real time. Even if this acceleration factor is lower than the requirements for the practical use of Reinforcement Learning algorithms, it seems sufficient to experiment the concept of transfer learning between a simple simulation and a more complex one.

As described in the following picture, it is important to note that the use of the iv4XR Framework facilitated the concrete implementation of transfer learning.

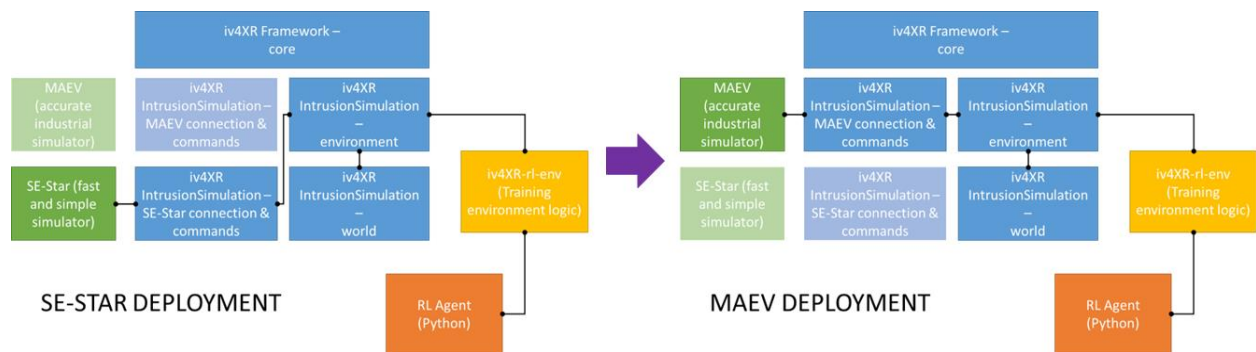


Figure 13: Transfer learning between SE-STAR and MAEV simulations (architecture)

The two simulations share the same World Model and use the same interfaces to the Reinforcement Learning tools. Only the gateway between the simulation and the framework is specific for each simulation.

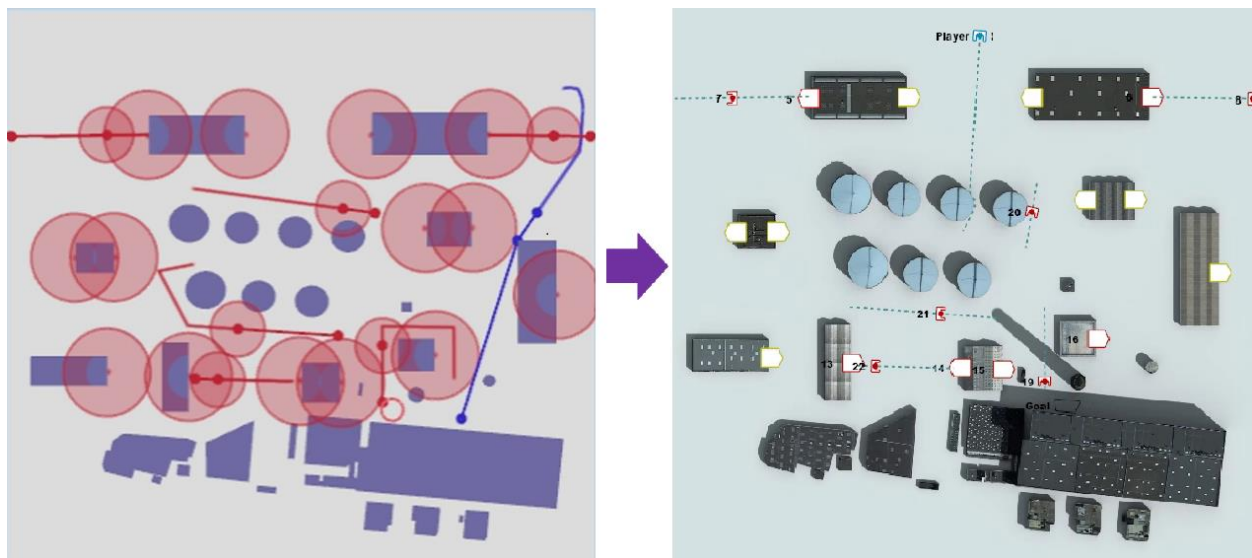


Figure 14: Transfer learning between SE-STAR and MAEV simulations (execution)

4.5 EVALUATION OF THE PROJECT

The project has been evaluated with the objective to verify the possibility to automatise our testing procedure using a framework such as iv4XR, and especially the practical use of Reinforcement Learning tools to model Test Agents that will partially replace the human testers in the future.

This evaluation use the following metrics that have been defined above :

- The reduction of the number of human operators;
- The reduction of the time spent to perform the tests;
- The augmentation of the coverage of the test.

Besides these metrics, we want to analyse what will be the future software and hardware constraints to practically use the iv4XR framework for our customer projects.

Description of the tests with the iv4XR framework

The testing procedure we want to implement in our future projects using the iv4XR framework is the following:

1. Definition of a defence strategy by the security officer using the current sets of surveillance cameras and guards available.
2. Connection of the simpler simulation (SE-STAR) to the iv4XR framework, with the defence strategy, designed by the security officer, implemented.
3. Running of the tests with a Test Agent that learns from experience in the “Nuclear Plant Simulation” thanks to the iv4XR Reinforcement Learning (RL) tools. Each trial starts from a randomly chosen position which is on the map, and ends when the test agent reaches its goal (the core of the plant). This learning process continues as long as performance increases (several millions of trials).
4. Evaluation the behaviour model of the intruder (the RL policy learned with SE-STAR) by the validation tests: calculation of the pourcentage of performances of the intruder trying to reach the core of the plant without being detected from a series of starting positions selected around the site.
5. Connection of the target simulation (MAEV) to the iv4XR framework in replacement of SE-STAR, with the defence strategy, designed by the security officer, implemented.
6. Transfer of the behaviour model of the intruder (the RL policy learned with SE-STAR) in MAEV and evaluation of this behaviour model with the same validation tests: calculation of the percentage of performances of the intruder trying to reach the core of the plant without being detected with the same series of starting positions selected around the site.
7. Continuation of the learning process with MAEV instead of SE-STAR. The learning process continues as long as the performances increase in MAEV.

8. Final evaluation of the defence strategy using the same validation tests as above (reaching the core of the plant without being detected from a series of starting positions selected around the site) with the optimised behaviour learned with MAEV.
9. The results of the test are given to the security officer in order to identify the gaps in the defence strategy that has been tested.
10. If needed, the security officer may modify its defence strategy (modification of the position of the camera, of the patrol of the guards or addition of cameras and guards) and continue the tests (return to step 1 with an already learned intruder model).

This test procedure has been experimented during the iv4XR project from step 1 to step 6. Future works will experiment with the rest of the procedure and the industrialization of this procedure, especially in terms of machine learning efficiency (cf. next section).

Comparison with the human testing procedure

The procedure with human testers is quite similar to the one implemented with the Test Agent although simpler:

1. Definition of a defence strategy by the security officer using the current sets of surveillance cameras and guards available.
2. Implementation of this defence strategy directly in the target simulation (MAEV).
3. Evaluation of the defence strategy by one or several human testers: calculation of the percentage of the performances of the intruder, played by a tester, trying to reach the core of the plant without being detected with a series of starting positions selected around the site.
4. The results of the test are given to the security officer in order to identify the gaps in the defence strategy that has been tested.
5. If needed, the security officer may modify its defence strategy (modification of the position of the camera, of the patrol of the guards or addition of cameras and guards) and renew the tests.

Both testing procedures need the presence of the security officer to design the defence strategy and an operator to implement this defence strategy in MAEV, and also in SE-STAR for the automatic testing procedure.

The difference in terms of human resources can be found in step 3 of both testing procedures where the testers or the test agent have to find a solution to reach the core of the plant without being detected from a set of starting positions selected around the plant. For each of these positions, a human tester will need to spend several hours to find a solution to reach the core of the plant without being detected by the surveillance cameras which positions are unknown at the beginning of the test and by the guards that are more difficult to avoid in first-person view, especially because the testers don't know the patterns of their patrols.

The process is completely automatic when using the test agent implemented in the iv4XR framework. However, Machine Learning still needs an AI expert to develop the intruder behaviour model since he will have to define for each step of the learning process the fitness function that will be used to update the agent policy. Hopefully, the work of the AI expert is performed only one time during this process to parametrize the algorithms.

Calculation of the metrics

There is an obvious correlation between the number of operators, the time spent and the coverage of the tests when we use human testers. It is not as much the case when you use automatic test agents.

With the human testing procedure, the coverage of the test is not only proportional to the number of starting positions selected around the plant (each starting position corresponds to a new planning problem for the tester), but also to the maximum time allocated to each tester to find a solution to the planning problem.

Assuming that each trial costs, in real time condition, an average of 10 minutes and that a human tester will need at least 100 trials to challenge the defence strategy, it means that the testing of each initial position will require at least 2 days of work.

If we assume that we need 100 starting positions (selected around the plant, on the left, right and upper border of the plant) to perform the coverage, it means that the total amount of time spent for the tests will be 200 days, which corresponds to 40 weeks of working days for a single human tester or 4 week of working days for a team of 10 human testers, which is quite prohibitive.

In comparison, when the machine learning process is achieved, the corresponding tests are far much quicker. With MAEV, running 10 times faster than the real time, only one trial will be needed to test each initial position because this simulation is deterministic. It means that all the tests can be performed in only 100 minutes.

Of course, the total time spent depends hardly on the time consumed during the machine learning phase. The first phase of the learning, with SE-STAR, has been achieved in 1 week of calculation on a single Laptop PC (CPU 3Ghz; GPU 2080TI) running a total of 10 millions trials.

The corresponding learning time with MAEV would have been 5 weeks of work but we expect that the finalisation of the machine learning with MAEV using the model already learnt with SE-STAR will only need an additional week of calculation.

We clearly see the great advantages of the approach, since we are able to reduce the number of human operators, the time spent or both, if we use the Test Agent of the iv4XR framework to perform the tests.

The advantages of using the iv4XR framework is even more important for test coverage. We have made the strong assumption that only 100 initial positions will be necessary to cover the tests and that only 100 trials will be needed for a human tester to find a solution to reach the core of the plant if a solution exists, but there is no real metric to demonstrate that these parameters will be sufficient.

Using the iv4XR approach for testing, an augmentation of the number of starting positions or the necessity to test several times each starting position if the simulation becomes less deterministic for instance (probability of detection by the guards for instance) will not significantly increase the duration of the validation tests.

Results of the Transfer Learning

These interesting results are counterbalanced by the fact that the Machine Learning process is still very time consuming even if we succeed, during the project, to transform a simulation (MAEV) designed to operate in real time into a simulation capable of running 10 times faster than the real time without modifying the level of details of its models.

The use of a more simple simulation SE-STAR helps a lot by allowing us to use a quicker simulation that can run 50 times faster than the real time. As pointed above, the optimization problem was solved using the Reinforcement Learning tools of iv4XR in only one week with a total of 10 million trials.

The following figure presents the learning curve of the test agent in SE-STAR (step 3 of the testing procedure described above with test agent), that is the percentage of success, over the time, of reaching the core of the plant for the intruder.

One can notice that the learning curve evolves very quickly but, then, fluctuates around 50% of successful intrusions, even after 10 millions of trials. The main reason is that the learning methodology implies the creation of impossible scenarios for the intruder where the starting points, randomly chosen all over the map, are directly detected by a camera or a guard, or easily detected after a while.

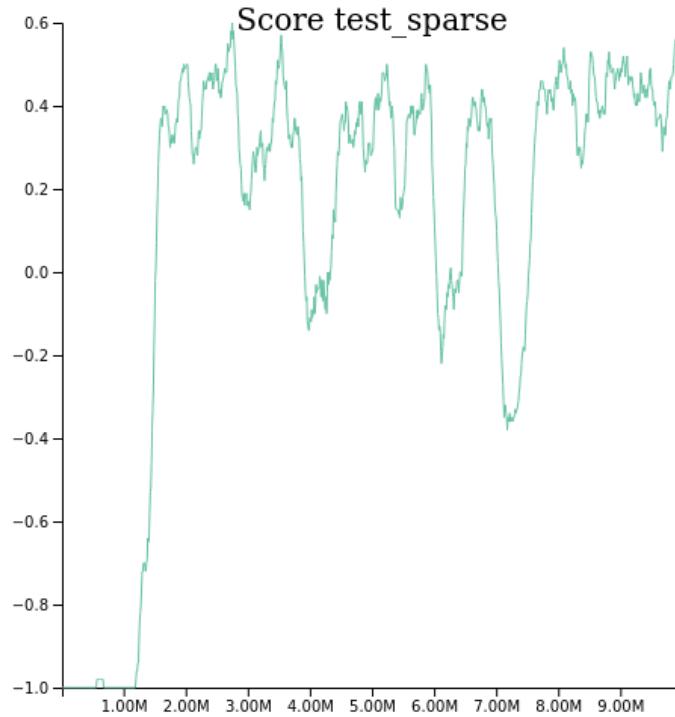


Figure 15: Learning curve of the test agent in SE-STAR

The next figure shows the performance curve of the test agent in SE-STAR (step 4 of the testing procedure described above with test agent), that is the percentage of success, over the time, of reaching the core of the plant for the intruder when it starts from a point situated on the borders left, right or up of the map.

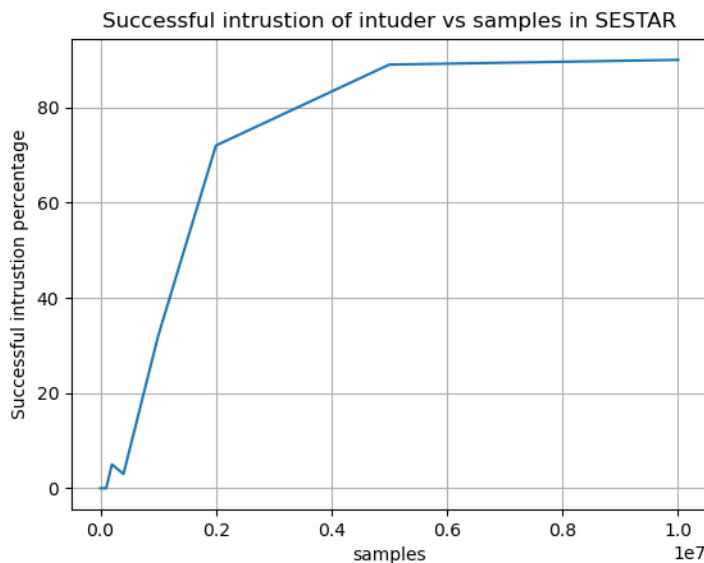


Figure 16: Performance curve of the test agent in SE-STAR

One can notice that the percentage doesn't exceed 90% of successful intrusions, even after 10 million trials. This means that there are some initial positions that are more difficult for the intruder to allow it to reach its goal. This could be good information for the security officer to distinguish the areas of the plant that are more secure than others.

Finally, the next figure shows the performance curve of the test agent when the intruder model is transferred to MAEV (step 6 of the testing procedure described above with test agent).

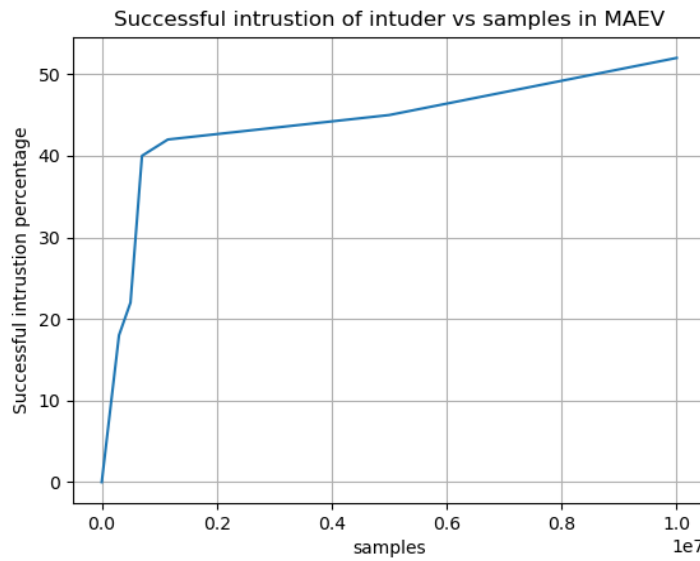


Figure 17: Performance curve of the test agent in MAEV (with no additional learning)

As expected, the results show a similar but lower performance curve, which means that the transfer learning was successful but also that it will be necessary to achieve another phase of machine learning directly in MAEV to enhance the performances of the test agent (step 7 of the testing procedure described above). We plan to do this task in future works after the end of the project.

4.6 CONCLUSIONS AND FUTURE WORK

The work done during the project allows us to validate the automating testing procedure proposed by the iv4XR framework which allows the use of test agents instead of human testers.

During the validation phase of the tests, we found a significant reduction of the number of human operators and of the time spent for the test. We also observed the possibility to have a much larger coverage of the test to validate the results.

However, this interesting approach relies on the strong assumptions:

1. That we will be able to construct for each specific application a test agent capable of replacing the human testers in most of their testing tasks;
2. And that the construction of the model of this test agent will not be prohibitive in time and human resources.

The first assumption has been proven for the "Nuclear Plant Intrusion Simulation" which is a case study close enough to the real applications we develop for our customers. In the future, we intend to experiment the use of the iv4XR framework in customer projects involving similar or quite different problem-solving tasks.

The second assumption was only partially demonstrated. The problem lies in the practical use of Reinforcement Learning, and Machine Learning in general, to develop a complex model for a customer project (and not only for a demo). Learning such a model is very time consuming and often needs strong AI expertise and, for some algorithms, a large amount of data.

The problem is, thus, much larger than the simple use of AI tools to develop Test Agents but implies the development of a next generation of simulation architecture that will help the use of Artificial Intelligence tools for testing, such as the iv4XR framework, or other simulation needs.

In parallel with the iv4XR project, Thales AVS has made a study on the contribution of simulation for intelligent autonomous systems [4] which results have conducted to several modifications in MAEV and to our decision to use a simpler simulation (SE-STAR) to perform the early phases of Machine Learning.

The SIMSIA study has identified 5 major improvements of the simulation means that will help, in the future, the development of AI functionalities, for instance for XR application testing.

The first improvement concerns the interfaces of the simulation. Most legacy simulations propose limited but standardised interfaces (HLA, DIS, DDS, etc.) that prevent the use of such simulation for AI applications.

During the iv4XR project, we developed for MAEV a more open interface which relies on the object model defined in iv4XR and which allows the use of the AI tools integrated in the framework and, especially, the Reinforcement Learning algorithms.

Note: The development of these new interfaces has also benefited from internal fundings (ALEXIA) that directly contributed to the project since this budget was directly used for the development of the new interfaces of MAEV.

The second improvement concerns the execution of the simulation and, in particular, the capacity to accelerate this execution. Most legacy simulations are real time applications and have difficulties to run faster than the real time. The problem does not only come from the complexity of their models but also to the fact that these models were designed to be real-time. During the iv4XR project, we have modified the models used for the "Nuclear Plant Intrusion Simulation" in order to be able to perform them faster than the real time. Together with the evolution of the interface described above, we are now able to run MAEV 10 times faster than the real time (for an application such as our case study). As mentioned above, It is not enough to run AI algorithms such as Reinforcement Learning, but this acceleration factor seems sufficient for the testing of a large number of scenarios very quickly.

The third improvement concerns the simulation models. We have already pointed out the fact that legacy simulation's models are very complex because our application usually involves very realistic representations of the environment and of the behaviour of the simulated objects and actors. For Machine Learning in general, and Reinforcement Learning in particular, this high level of details of the models is not absolutely necessary during the first phases of learning that imply a great amount of trials to develop the preliminary version of the RL policy. It would then be useful to use a simulation with multiple levels of details of their models to adapt the simulation to the AI algorithms' current needs.

Because this kind of modification was too costly to be implemented in a project such as iv4XR, we used another simulation (SE-STAR), with simpler models, to prove this concept. We have also performed, with success, the transfer learning between SE-STAR and MAEV with, however, a reduction of the performances in MAEV that will compensate in future works by the continuation of the Machine Learning process in MAEV.

The fourth improvement concerns the simulation architecture. Future simulations, suitable for the development of AI models, will need a more distributed architecture than will allow the practicable implementation of multi-resolutions models and the parallelisation of the execution of these models and other simulation components. This evolution of simulation architecture may be implemented thanks to the concept known as Modelisation & Simulation as a Service (MSaaS). MSaaS architectures implement distributed infrastructures (e.g. Cloud, Edge, etc.) that will allow the parametrization of the level of details of models and the parallelization of the calculations of models and components. This main advantage of using such a kind of architecture will be to offer a greater acceleration factor of the simulation without having to reduce the quality of its models.

We have begun the modification of our simulation components and models, and especially of MAEV, to prepare for their evolution towards a MSaaS architecture.

Note: Additional internal fundings has been used to prepare this evolution of the “Architecture & Components” of MAEV based on the lessons learnt from the iv4XR project.

Finally, the fifth improvement concerns the representativeness of data and situations generated by the use of AI agents in comparison with the real operational data and the situations observed in real environments.

For the “Nuclear Plant Intrusion Simulation” use-case, this improvement concerns the representativeness of the tests performed in the simulation in order to better evaluate the quality of the defence strategy designed by the security officer and reflect the performance of this strategy in the real nuclear plant. We demonstrated that the use of iv4XR test agents allows us to significantly increase the coverage of the tests in comparison with the limited set of tests that may be performed by human testers in reasonable time. This also validates the use of automatic testing in such applications, and the use of a testing framework, such as iv4XR, to perform these tests.

5 LIVESITE

5.1 INTRODUCTION

Livesite is a real-time instrumentation and monitoring system for the building industry.

Sensors are installed on building and construction sites to monitor ground movements, building movements, temperature variations, rainfall, water levels, vibrations, dust and other environmental factors to ensure safety of both the site being worked on, and the neighbouring area and structures.

The various sensors connect to network hubs and the internet directly and upload readings to the Livesite servers. Readings can be uploaded hundreds of times per second in some instances, and a very large volume of data is produced.

The Livesite servers analyse and process the data, providing real-time graphs, numerical tables, and 3D visualisations of the sensor readings, allowing engineers to monitor the site activity and behaviours.

Ensuring the integrity and accuracy of the readings is critical to provide a safe building environment and prevent works or ground issues from causing problems with the structures involved.

5.2 READING VERIFICATION AND ANALYSIS

For ivXR4 we have set up an additional server (the API Server) which can access readings from certain Livesite projects and provides a web-API to access and parse the readings and their associated meta-information.

The API server controls and provides an API to the JAVA tool, which runs tests and queries on Livesite project data and provides results which list errors detected in the project configuration, sensor configurations, or the sensor readings themselves.

The meta-information accompanying a project includes a wide variety of parameters that can be used to assist in verification of the validity of the readings. For instance, the meta for a sensor will include what type of sensor it is, and where it is installed. It can then be deduced that a temperature sensor on the outside of a building for example, should never give a temperature reading significantly higher than the ambient temperature of the area on that particular day.

5.3 INTEGRATION OVERVIEW

Using the API server, a script for a Livesite project is provided. This script contains items to test and validate in the given project.

The API Server can provide the readings from sensors with transformations and filtering applied, allowing the iv4XR framework to detect inconsistencies or problems when tested with the JAVA tool command scripts.

The API server can also be used to retrieve tables of readings with meta-information from Livesite projects which can then be analysed by the iv4XR framework.

The meta and readings can be downloaded in JSON, XML and raw formats.

For basic integration, the iv4XR framework detected some obvious errors automatically, for example values out of range or missing readings.

Monitoring projects typically run for months or years. The sensor monitoring server executes maintenance tasks over defined periods during the lifecycle of the project, including backups, updating reports, generating alarms etc.

The top layer has been designed to be an additional task which can run alongside the other maintenance tasks, providing a near real-time verification system.

The device error shown below indicates a problem is occurring on a particular device, more detail will be available after further tests have been run in subsequent tasks, but the engineers know that readings from the device may be invalid and should be ignored until this is verified.

Status	Started	Date	Type	Duration	Description
Not set	1 day	29/09/2021 08:21:59	Device	Ongoing	UNIT 1, Device failure
Not set	7 mins	30/09/2021 13:16:31	Red	Ongoing	UNIT 2.LAeq (10 Minute) [dB] (16) 92.18dB ≥ 80.0dB red threshold
Not set	7 mins	30/09/2021 13:16:31	Amber	Ongoing	UNIT 2.LAeq (10 Minute) [dB] (16) 92.18dB ≥ 78.0dB amber threshold
Not set	30 wks	26/02/2021 08:47:50	Reading	Ongoing	UNIT 3.SYSTEM.VOLTAGE (0) Reading Fail

Figure 18. Livesite Diagnostic error list example

The hierarchical task and goal approach also allows testing to be adjusted in terms of accuracy, as well as grouping of errors.

Multiple flat-lines over a period of hours or weeks, as shown below, indicate some kind of general problem with communication with the sensor, and it's clearer to see an overview like this, than a long list of individual errors.

Not set	2 days	28/09/2021 10:42:29	Flatline	Ongoing	SITE01.Location8_Height (21) 0.0mm ≤ 0.01mm flatline level
Not set	1 day	29/09/2021 10:42:29	Flatline	Ongoing	SITE01.Location1_Height (12) 0.0mm ≤ 0.01mm flatline level
Not set	7 hrs	30/09/2021 05:43:12	Flatline	Ongoing	SITE02.Location8_Height (21) 0.0mm ≤ 0.01mm flatline level
Not set	2 wks	10/09/2021 14:42:46	Flatline ²	Ongoing	2 similar events. Click to show these events.
Not set	6 hrs	30/09/2021 07:33:59	Flatline ³	Ongoing	3 similar events. Click to show these events.
Not set	2 hrs	30/09/2021 11:33:59	Flatline ¹²	Ongoing	12 similar events. Click to show these events.
Not set	1 hr	30/09/2021 12:33:59	Flatline ²¹	Ongoing	21 similar events. Click to show these events.
Not set	1 yr	02/04/2020 08:39:25	Reading	Ongoing	SITE05. (23) Reading Fail
Not set	21 hrs	29/09/2021 16:07:19	Amber	Ongoing	SITE07.Tilt.03.Y (17) -2.01301deg ≥ 2.0deg amber threshold
Not set	12 mins	30/09/2021 13:23:25	Device	Ongoing	SITE07. Device failure
Not set	1 day	29/09/2021 11:19:22	Amber	Ongoing	SITE08.Tilt.01.Y (13) 2.2814deg ≥ 2.0deg amber threshold
Not set	20 hrs	29/09/2021 17:25:31	Device	Ongoing	SITE09. Device failure
Not set	1 hr	30/09/2021 11:55:05	Flatline ⁵	Ongoing	5 similar events. Click to show these events.
Not set	41 mins	30/09/2021 12:55:05	Flatline ²⁵	Ongoing	25 similar events. Click to show these events.

Figure 19. Livesite diagnostics error message overview

Testing against years of historical data, with adjusting granularity in time, allows simple overviews of a sensor's state to be determined also, without having to go through all the readings for a particular sensor (of which there could be millions).

Monitoring projects typically save snapshots of readings at various time intervals, such as every minute, as well as every second, or fraction thereof, to minimise database reads when looking back over the data. One may want to see the data for a week overall, in which case looking at readings every 1/100 second is not necessary.

5.4 TRAIN MOVING OVER STRUCTURE

The hierarchical approach for test case type 1 is equally valid for trains moving over structures.

For this type of project, the frequency of readings of sensor data is typically increased when the train is detected moving over the bridge, so vibrations and behaviour can be monitored more closely until the train exits the bridge.

For this case our test script is triggered with normal mode, or detail mode, depending on whether the train is on the bridge. In detail mode, the vibrations are checked as a priority, whereby in normal mode, the scripts trigger agents similar to the fixed structure, to verify the overall state of the system and look for errors in sensors and sensor relationships.

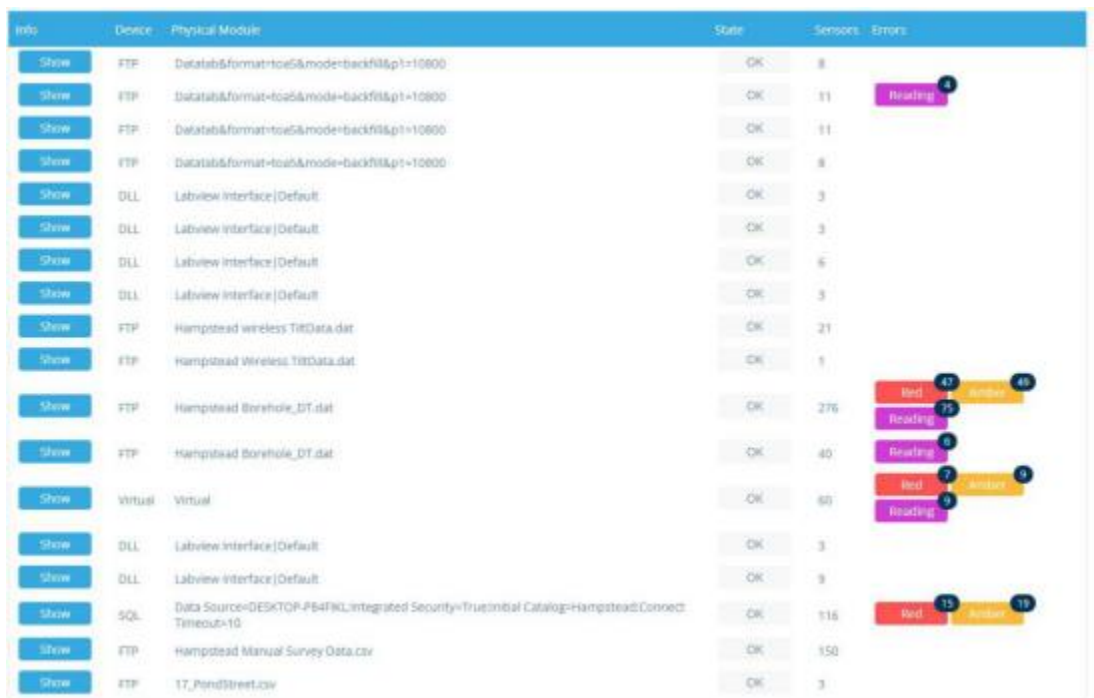
Detecting excessive displacements along sections of a bridge is shown below.



5.5 MULTI-SITES

Many engineering monitoring projects are so complex the site is broken down into sub-sites, each of which have their own hosting server which connects to the sensors for that area.

We used the same hierarchical approach for these types of complex sites, to trigger initial agents for each sub-site, each of which can then further analyse the sensors at that sub-site. Simple summaries of errors are then made available, highlighting problems at each sub-site.



Info	Device	Physical Module	State	Sensors	Errors
Show	FTP	Datab&format=to&mode=backfill&p=10800	OK	8	
Show	FTP	Datab&format=to&mode=backfill&p=10800	OK	11	Reading 4
Show	FTP	Datab&format=to&mode=backfill&p=10800	OK	11	
Show	FTP	Datab&format=to&mode=backfill&p=10800	OK	8	
Show	DLL	Labview Interface (Default)	OK	3	
Show	DLL	Labview Interface (Default)	OK	3	
Show	DLL	Labview Interface (Default)	OK	6	
Show	DLL	Labview Interface (Default)	OK	3	
Show	FTP	Hampstead wireless T10Data.dat	OK	21	
Show	FTP	Hampstead Wireless T10Data.dat	OK	1	
Show	FTP	Hampstead Borehole_DT.dat	OK	276	Reading 41, Reading 75, Reading 45
Show	FTP	Hampstead Borehole_DT.dat	OK	40	Reading 6, Reading 7, Reading 9
Show	Virtual	Virtual	OK	60	Reading 15, Reading 19
Show	DLL	Labview Interface (Default)	OK	3	
Show	DLL	Labview Interface (Default)	OK	9	
Show	SQL	Data Source=DESKTOP-P64FKL;Integrated Security=True;Initial Catalog=Hampstead;Connect Timeout=10	OK	116	
Show	FTP	Hampstead Manual Survey Data.csv	OK	150	
Show	FTP	T7_PondStreet.csv	OK	3	

Figure 20. Overview of errors on sites in a large multi-site project

5.6 INTEGRATION WITH LIVESITE

At the start of the project, and until the final integration phase, testing was performed on mirrors of monitoring servers, to ensure no disruption to actual monitoring systems happened whilst testing.

We have since integrated the JAVA tool we developed for iv4XR, which uses multiple agents triggered by scripts, completely into Livesite as a Diagnostics and warning module. This allows the detection of simple sensor errors such as flatlines, jitter, missing values, calibration errors, as well as more complex errors such as formula-based readings being invalid due to one or more errors in any sensor which is used by the formulae.

The JAVA tool is run with multiple instances, with each being given assigned tasks within the overall testing of a project. This allows the tool to run using multiple processors on the server, minimising any latency which may occur due to multiple read/write requests of the data.

Although tasks are given to the JAVA tool in a hierarchical manner, sometimes a sub-task is a more intensive/time-consuming task than the parent task which requested it (such as checking readings for a sensor every second, instead of every minute).

For this reason, we have been optimising how tasks are broken down, terminated after partial completion, and then restarted, and cancelled. If a sensor is deemed to be invalid for some

reason (such as no power going to the hub to which the sensor is connected), checking the sensor in more detail is a waste of bandwidth.

Each major test is thus now broken down into a series of steps, and only successful completion of a test step leads to the next step. Later tests in the pipeline are typically more complex as they require results from other sensors or calculation breakdowns and analysis, whereas initial tests are more of a valid/invalid Boolean test.

5.7 VALIDATION

For Livesite validation, we compare the number of errors found by the existing Livesite testing functions, with extra errors found with the iv4XR testing module.

The iv4XR testing module retrieves five types of errors. (1) Black errors are alerts the system should have sent out but did not. (2) Device errors represent entire regions of sensors failing. (3) Jitter errors indicate a problem with a sensor due to the environment. They indicate positioning problems, i.e., something in the environment likely affects the readings, and the sensor should be moved. (4) Reading errors are errors or omissions that frequently indicate sensor or communication failures. (5) Finally, sanity errors occur due to configuration setup problems and should never occur.

Livesite tested the iv4xr module on 60 projects. Of those 60 projects, 14 had additional errors reported. On 11 of those 14 projects, the module retrieved errors. The existing Livesite testing would not have found these errors. In three different projects, the module found additional sanity errors.

Notably, in 8 projects, the iv4xr module identified errors when the existing Livesite testing issued no alerts.

The iv4xr module retrieved over 80,000 additional errors. The average number of errors found without the iv4xr module in the 14 projects was 215.4286, and the iv4xr module produced an average of 5858.5000 errors. The number of extra errors found with the iv4xr module is significantly higher than those found with the previous method ($Z = -1.99$; $p = .046$).

Status	Started	Date	Type	Duration	Description
Not set	5 yrs	08/07/2016 12:57:47	Amber ⁶	Ongoing	6 similar events. Click to show these events.
Not set	5 yrs	08/07/2016 13:27:47	Amber ³	Ongoing	3 similar events. Click to show these events.
Not set	5 yrs	08/07/2016 14:27:47	Amber	Ongoing	BOK2_12.1_Long (231) 3.5319mm ≥ 3.0mm amber threshold
Not set	5 yrs	08/07/2016 14:27:47	Red ³	Ongoing	3 similar events. Click to show these events.
Not set	5 yrs	08/07/2016 14:57:47	Red ²	Ongoing	2 similar events. Click to show these events.
Not set	5 yrs	08/07/2016 14:57:47	Amber	Ongoing	BOK2_11.3_Long (225) 3.43799mm ≥ 3.0mm amber threshold
Not set	5 yrs	08/07/2016 14:57:47	Red ²	Ongoing	2 similar events. Click to show these events.
Not set	5 yrs	08/07/2016 14:57:47	Amber ³	Ongoing	3 similar events. Click to show these events.
Not set	5 yrs	08/07/2016 15:27:47	Amber ⁴	Ongoing	4 similar events. Click to show these events.

Figure 21. Errors and events summarised in a Livesite project after detection by iv4XR java tool

The hierarchical task testing strategy also allows visualisation of the key issues at any point. At a glance, it is obvious where the problems are on the project as shown below, as the sensor has been marked amber due to excessive vibrations.



The hierarchical testing enables the exact start/endpoint of problems to be identified without requiring full linear search of all the readings in a project, as a tree-based approach is used, with each iteration down reading data with finer granularity.

The hierarchical task testing strategy also allows visualisation of the key issues at any point. At a glance, it is obvious where the problems are on the project as shown below, as the sensor has been marked amber due to excessive vibrations.

STRUCTURAL					
Sensor	Alert	Description	Start Time	Repeats	Duration
L[Aeq (1 Hour)](dB]	Amber	85.0dB ≥ 80.0dB amber threshold	17/09/2021 07:26:47:000	2963/1	Ongoing

THRESHOLD GRAPH	
No readings available.	

THRESHOLD TABLE										
Time	Latest	-30 mins	-1 hour	-1.5 hours	-2 hours	-2.5 hours	-3 hours	-3.5 hours	-4 hours	-4.5 hours
L[Aeq (1 Hour)](dB]	No readings	No readings	No readings	No readings	No readings	No readings	No readings	No readings	No readings	No readings

FUNCTIONALITY					
Sensor	Alert	Description	Start Time	Repeats	Duration
	Device	Device failure	17/09/2021 14:30:56:448	2536/0	Ongoing

SYSTEM HEALTH	
No health events.	

LATEST READINGS AND MIN/MAX/AVG				
Name	Latest	Min	Max	Average

Figure 22. Summary of hierarchical testing provided in email alerts

6 CONCLUSION

This deliverable has been concerned with the evaluation of the iv4XR project from the perspective of the pilots provided by the industrial partners. The objective of iv4XR from the outset was to improve the hitherto manual testing processes through the addition of automation.

For each of the pilots in the project, some improvements have been observed. For Space Engineers, significant reductions in manual testing time and increases in code coverage have been obtained. For MAEV, the number of human operators required to operate a simulation has been reduced to only one. For LiveSite, a statistically significant number of previously undetected errors have been brought to light by the automated agents.

The pilot studies have shown that there is value to be added by the integration of the iv4XR framework into the various testing workflows, and that the diversity of the pilots has shown the framework to be quite flexible.

The success of the pilot studies for these partners has encouraged some of them to make more extensive use of the framework in both the pilot in question, and potentially in other projects in the business.

7 BIBLIOGRAPHY

- [1] 25010, ISO/IEC ISO/IEC 25010:2011, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. (2011).
- [2] Marat Akhin, Mikhail Belyaev et al. “Kotlin language specification: Kotlin/Core”, JetBrains / JetBrains Research, 2020
- [3] L. Garcés, F. Oquendo and E. Y. Nakagawa, “A Quality Model for AAL Software Systems”, *2016 IEEE 29th International Symposium on Computer-Based Medical Systems (CBMS)*, 2016, pp. 175-180, doi: 10.1109/CBMS.2016.46.
- [4] J.-Y. Donnart, “SIMSIA: Study of the contribution of simulation for intelligent autonomous systems”, NMSG Annual Symposium, MSG-197 on Emerging and Disruptive Modelling and Simulation Technologies to Transform Future Defence Capabilities, 20-21 October 2022 in Bath (UK).
- [5] Thorn Jansen, Fernando Pastor Ricós, Yaping Luo, Kevin van der Vlist, Robbert van Dalen, Pekka Aho, and Tanja E. J. Vos, “Scriptless GUI testing on mobile applications”, QRS 2022, 22nd IEEE International Conference on Software Quality, Reliability, and Security.
- [6] Axel Bons, Beatriz Marín, Pekka Aho, Tanja Vos, “Evaluating the Complementarity of Scriptless and Scripted Testing in Industry: A web case study”, *Information and Software Technology Journal*. (Submitted in August 2022- In review process).