Intelligent Verification/Validation for XR Based Systems

Research and Innovation Action

Grant agreement no.: 856716

D2.3 – Final version of iv4xr Framework

iv4XR – WP2 – D2.3

Version 2.1

December 2022

Project Reference	EU H2020-ICT-2018-3 - 856716
Due Date	31/12/2022
Actual Date	27/12/2022
Document Author/s	Wishnu Prasetya (UU)
Version	2.1
Dissemination level	Public
Status	Final
Type	OTHER

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)**
1.0	15/12/2022	Initial document structure and contents	WP
1.1	17/12/2022	Draft	WP
1.2	18/12/2022	Adding model support	WP
1.2.1	18/12/2022	Lifting Fw site and demos to subsec	WP
2.0	23/12/2022	Addressing QA comments	WP
2.1	27/12/2022	Final arrangements for submission	RP

Document Quality Control			
Version QA	**Date**	**Comments (and if appropriate reason for change)**	**Initials of QA Person**
1.2.1	20/12/2022	Comments and minor edits	FK

Document Authors and Quality Assurance Checks		
Author Initials	**Name of Author**	**Institution**
WP	Wishnu Prasetya	Utrecht University
FK	Fitsum Kifetew	FBK

RP	Rui Prada	INESC

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Deliverable D2.3 is the final version of the iv4XR Framework. Since the Deliverable-type D2.3 is OTHER, this document is not D2.3 itself, but rather a document that gives an overview of the Framework and its main components, and summarizes work done by WP2 on the Framework during the last project year. **The reader is also referred to Deliverable D2.4, which is a report describing the Framework and the Framework Core.**

Document structure

- In Section-1 we provide an overview of the main components/modules of the iv4xr Framework. WP2 itself focuses on the agent-based infrastructure of Framework --the so-called Framework Core component. For overview we mention other main components of the Framework but will refer to the Reports where they will be described in more detail.
- In Section-2 we will give an overview of improvement on the Framework that is done by WP2 during the final year of the project.

SECTION 1: THE IV4XR FRAMEWORK, AN OVERVIEW

As a typical EU project, iv4xr is organized by dividing the work into Work Packages and Tasks, whose execution each involves coordination over multiple partners. This project organization has, to some degree, influence on the architecture of the project's main deliverable: the iv4xr Framework. For example, we should not develop the Framework in a rigid centralized way. This would stifle the Work Packages' ability to move forward and adjust. Instead, the approach we took was to give autonomy to the work packages to develop their software modules in the way that they consider best. However, some principles were set to make sure that the produced components are able to interoperate. For example, the main programming languages of iv4xr were decided and fixed, namely Java and Python. To facilitate interoperability between the two languages, a bridge was implemented[1]. We also strive modules from different work packages can be built using the same build technology (Maven), and furthermore using a build technology that allows dependencies to be managed and automatically imported. This makes sure that the modules can be easily built, and when one module uses another, the dependency will be transparently imported, even if the modules were developed by two different teams on two different repositories. Using this approach, integrating the modules made by the Work Packages into a single Framework becomes easier, and moreover the modules can be kept and maintained in their own repositories.

[1] A socket-based bridge was implemented. This allows e.g. JSON objects to be exchanged, and moreover each side can keep its state. For a simpler scheme that does not require the callee to retain its state, nor exchanges of JSON objects, we can simply use system calls.

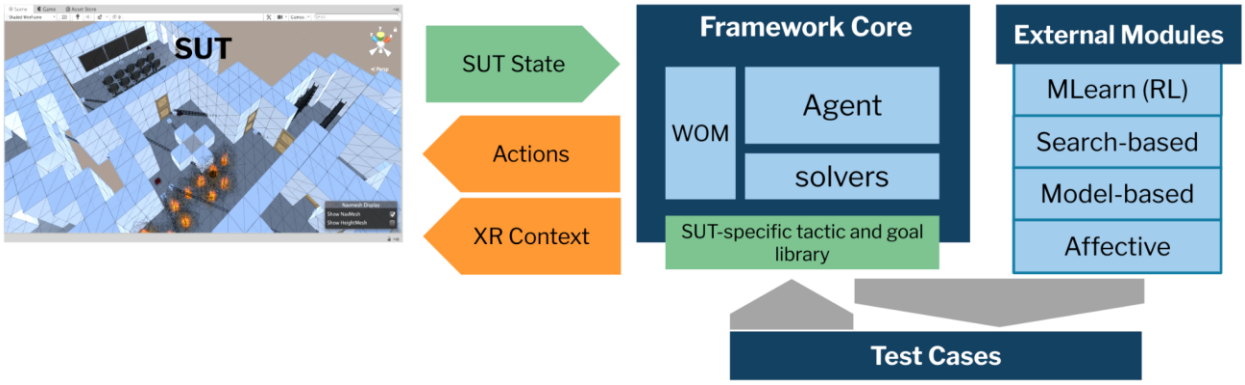

Fig. 1: *A top level architecture of the iv4xr Framework. The Framework Core refers to its agent-based programming and testing core. External modules refer to testing tools that are part of the Framework but outside the Core.*

Figure 1 above shows the top-level architecture of the iv4xr Framework. In this architecture the Framework Core refers to the agent-based core of the Framework. Among other things, the Core implements basic and functional test agents, and the underlying agents' execution system.
There are also modules that are indicated as external modules. These modules are part of the iv4xr Framework and are themselves tools for testing. However, these are modules that are at least conceptually outside the Core (though they may use functionalities from the Core). The Core is developed in WP2. Other modules are developed in other Work Packages, namely WP3 and WP4.

The Framework containing Core and all the external modules can be obtained from its GitHub site. Except for the Search-based module, each external module is also a project of its own. That is, it has its own Git repository and can be built separately from the Framework. This makes it possible for people to also use the module separately without having to deploy the entire Framework. This scheme also makes iv4xr more sustainable to maintain, as developers of each component, or its community, can continue maintaining the component after the project ends.

WP2 coordinated the integration of main modules from WP3 and WP4 into the Framework. This has resulted in the architecture as shown in Fig. 1.

Obtaining the Framework

It can be obtained from its GitHub page, along with instructions on how to deploy it, and other documentations:

https://github.com/iv4xr-project/iv4xr-framework

Demos

There are also several demos provided through the above site.

THE FRAMEWORK'S MAIN MODULES

The main modules/components of the Framework are summarized below:

- **Framework Core**. The Core provides the underlying agent-based system for iv4xr. It provides an implementation of basic and functional test agents and their runtime system. The Core features BDI (Belief-Desire-Intention) agents and adds a novel layer of tactical programming that provides an abstract way to exert control on agents' behavior. Declarative reasoning rules express when actions are allowed to execute. Although in theory just using reasoning is enough to find a solution (a plan that would solve the given goal state) if given infinite time, such an approach is not likely to be computationally efficient enough. For testing, this matters as no developers would want to wait for hours for their test to complete. The tactical layer allows developers to program an imperative control structure over the underlying reasoning-based behavior, allowing them to have greater control over the search process. So-called tactics can be defined to enable agents to choose and prioritize their short term actions and plans, whereas longer term strategies are expressed as so-called goal structures, specifying how a goal can be realized by choosing, prioritizing, sequencing, or repeating a set of subgoals.
  Several types of assertions to express correctness properties to check are supported by the Core, e.g. reachability assertions, invariants, and Linear Temporal Logic (LTL) formulas. Data collection, in particular in the form of traces, is supported. A trace records selected properties of the state of the System under Test (SUT) during a test run. Collected traces can be subjected to post-mortem verification or other kinds of analyses. For example, they typically contain the agents' positions, which can be aggregated to visualize (or calculate) the physical area coverage of the tests.

  More on the Core is elaborated in Report **D2.4**. The Core is also available as a separate component[2].

- **Search-based** modules**.** These modules are provided by WP3. They provide support for goal solving. In the context of agent-based testing a goal typically formulates a state the agent is interested in reaching (e.g. because it wants to do some checking on the state), or some intermediate state to reach as part of a larger test scenario. Solving such a goal amounts to **searching for** a sequence of actions that would drive the agent to reach the goal-state. Several goal solving approaches are supported, either offline or online. An offline approach searches for a solution without executing steps in the SUT. This is possible if a model of the SUT is available. On the other hand, an online approach performs the search directly on the SUT. An offline approach is fast, but it requires a model. If such a model is not available, we can use an online approach.

  Search-based goal solving is elaborated in Report **D3.5**. The goal solving modules are integrated into the Core.

[2] https://github.com/iv4xr-project/aplib

- **Model based testing** (MBT) module. This is provided by WP3. MBT allows good quality test suites to be rapidly generated using state of the art search-based testing algorithms, if a functional/behavioral model of the SUT is provided.

  MBT is elaborated in Report **D3.5**. The module is also available as a separate component[3].

- **Explorative testing** module (not shown in Fig. 1). This is provided by WP3. This makes use of a TESTAR agent that automatically executes non-sequential actions to navigate, interact and test that the System Under Test (SUT) and the functional aspects of the virtual entities are robust enough to respond to different user interactions. This agent automatically creates visual reports for end users and can infer a State Model that maps the information regarding the observed states and executed actions.

  Explorative testing is elaborated in Report **D3.5**. The module is also available as a separate component[4].

- **Affective testing** modules. These modules are developed by WP4. They allow models of users' emotion to be constructed and subsequently used to facilitate automated user experience (UX) assessment. Possible models to construct are PAD (Pleasure-Arousal-Dominance) models or OCC (Ortony-Clore-Collins) models. Emotion models are obtained either by learning them from data, or by crafting them. Such a model can either be attached to a test agent or applied to execution traces to produce emotion traces. UX requirements formulated, e.g., regular expression or LTL, which are then evaluated on emotion traces. Affective testing is elaborated in Report **D4.4**. These models are also available as a separate components[5].

- **Reinforcement learning**: this allows an agent to be trained using Reinforcement Learning to do a certain task. This task can be testing related, e.g., to verify the reachability of a certain state (so, RL is used to train a test agent to reach such a state), or to generate a test suite that maximizes coverage.
  Reinforcement learning support is elaborated in Report **D3.5**. Reinforcement Learning is also available as separate components[6].

SECTION 2: IMPROVEMENT ON THE FRAMEWORK CARRIED OUT BY WP2 IN YEAR-3

Much effort has been put on coordinating and implementing the integration of iv4xr components into the Framework, in addition to that, the following improvements to the Framework Core were implemented in WP2.

[3] https://github.com/iv4xr-project/iv4xr-mbt
[4] https://github.com/iv4xr-project/TESTAR_iv4xr
[5] https://github.com/iv4xr-project/jocc and https://github.com/iv4xr-project/PAD_emotion_game
[6] https://github.com/iv4xr-project/iv4xrl and https://github.com/iv4xr-project/iv4xr-rlbt

IMPROVED DYNAMIC GOALS

The Framework Core includes a Domain Specific Language (DSL) for writing tactics and goal structures for test agents. A dynamic goal is a goal whose specific target is decided at the runtime (rather than prescribed apriori by the tester). Another form of dynamic goal is a goal that can be deployed at runtime, depending on the SUT state at that time. Dynamic goals are essential for constructing smarter test agents. The feature was already present for some time in the Core, though it was not exposed to a more abstract programming level. This is now addressed, by extending the DSL with constructs for deploying dynamic goals. For example, the new constructs were exploited in implementing the goal-solving algorithm SA1 discussed in Report **D3.5**.

Further information:

- The new DSL constructs are discussed in Section 6 of Report **D2.4**.
- API documentation: https://iv4xr-project.github.io/apidocs/aplib/javadocs/nl/uu/cs/aplib/AplibEDSL.html

SUPPORT FOR ONLINE MODEL REASONING

In a recent study we showed that a test agent can be extended so it also builds a functional model of the virtual world that it is exploring/testing (linked below). Having such a model greatly improves the agent's ability to solve goals in the current and future runs. For future runs, it is then even possible to apply model-based offline test generation (which is very fast); this is discussed in Report **D3.5**. A re-implementation of the models used in the aforementioned study is now provided in the Framework Core, and is compatible with the LTL model checker provided in Core as well. This means that the model checker can be used to answer queries on a model. For example, if an agent becomes locked in an area, it could pose an LTL query to check if the model would know how to unlock the area.

Further information:

- API documentation: https://iv4xr-project.github.io/apidocs/aplib/javadocs/eu/iv4xr/framework/extensions/ltl/gameworldmodel/package-summary.html
- Related paper:
  Samira Shirzadehhajimahmood, Wishnu Prasetya, Frank Dignum, Mehdi Dastani, An Online Agent-based Search Approach in Automated Computer Game Testing with Model Construction. In Proceedings of the 13th International Workshop on Automating TEST Case Design, Selection, and Evaluation. 2022.
  https://zenodo.org/record/7230140#.Y3Edg3bMI2w

EXTENDED LTL

The Framework Core contains support for Linear Temporal Logic (LTL), which can be used to do bounded model checking, e.g., for the purpose of goal solving for WP3, or to do runtime

verification on traces produced by test runs. For the later use, we have extended LTL so that we can now also check membership in an area (in the sense of physical areas in a virtual world, e.g., rooms). For example, this allows us to express a requirement such as "*eventually room k must be visited*", or "*whenever the agent is in room k, a property x should never exceed C*". Additionally, it is also possible to express aggregation, for example: "*when the agent is in room k, the maximum value property x should be at least C*". This work is done in collaboration with the work on formalizing emotional requirements that is carried out within WP4, to be used for expressing user experience requirement (e.g., "*when the agent is in room k, eventually it would feel its hope increases*").

Extended LTL is integrated to the Framework (but outside Core) and is also available as a separate project called ltl-pxevaluation (linked below).

Further information:

- See the Github page of the above-mentioned ltl-pxevaluation: https://github.com/iv4xr-project/ltl-pxevaluation

SUPPORT FOR SPEEDING UP TESTS IN WP3 AND WP4

Many experiments in WP3 and WP4 use the Lab Recruits game as an experiment subject. However, running a large number of tests on Lab Recruits is relatively slow. To facilitate faster experimentation, we have implemented a 2D game called MiniDugeon. The game is highly configurable (we can configure the number of levels, the number of monsters, items, etc.) which makes it suitable as an experiment subject. It is implemented in Java, the same language as iv4xr. Tests on MiniDungeon can run much faster. This is useful, e.g., to test out new test heuristics, before we deploy it to the bigger, but slower, Lab Recruits. MiniDungeon, along with the accompanying library of basic tactics and goal structures and some sample test scenarios, is included in the Framework Core.

Further information:

- API documentation: https://iv4xr-project.github.io/apidocs/aplib/javadocs/nl/uu/cs/aplib/exampleUsages/miniDungeon/package-summary.html
- Related paper:
  Prasetya, I. S. W. B., Fernando Pastor Ricós, Fitsum Meshesha Kifetew, Davide Prandi, Samira Shirzadehhajimahmood, Tanja EJ Vos, Premysl Paska et al. "An agent-based approach to automated game testing: an experience report." In Proceedings of the 13th International Workshop on Automating Test Case Design, Selection and Evaluation, 2022. https://zenodo.org/record/7315182#.Y29rDexBwUo

CONCLUSION

We have implemented various improvements to the Framework Core. Main components of iv4xr have been integrated into the Framework. We refer to Report D2.4 for a more complete description of the Framework Core, and to D3.5 and D4.4 for the description of other main components. We also refer to Report D5.4 that describes how the Framework has been demonstrated to work on industrial setups, showing that it indeed has achieved a TRL6 state.