Project no.:	H2020-MSCA-RISE-2017-778233
Project full title:	Behavioural Application Program Interfaces
Project Acronym:	BEHAPI
Deliverable no.:	D.4.3 (M24)
Title of Deliverable:	Methodological approaches to b-APIs round-trip engineering
Work package:	WP4 (tasks 4.2, 4.3 and 4.5)
Type:	R
Lead Beneficiary:	**UNIVERSITY OF LEICESTER (BEN2, ULEIC)**
Dissemination Level:	PU
Number of pages:	9
Contract Delivery due date:	**29/2/2020 (M24, M36)**

Acknowledgement: This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 778233

Abstract:

This deliverable describes how the approaches followed in the project to support round-trip engineering of b-APIs (behavioural application programming interfaces) are supported by tools. The deliverable is related to the tasks "Top-down DevOps[1] support" (T4.2), "Bottom-up DevOps support" (T4.3), and "Tool chains in the SDLCs of b-APIs" (T4.5) of work package 4 (WP4) "Tool support". The goals of WP4 are to enhance and develop tools and methodologies for the reliability of API-based systems. A main goal is to support round-trip engineering by

- adopting new model-driven approaches to account for behavioural properties of APIs in software quality,
- developing reverse-engineering techniques (ranging from type inference to machine learning).

This report relates to engineering techniques for b-APIs (O.4.2), level of integration of artefacts (O.4.2), artefacts interoperability (O.4.3), and to continuous development (O.4.4). Management, testing, verification, and monitoring of b-APIs are crucial activities to entangle top-down and bottom-up features and devise round-trip engineering of b-APIs. The activities on T4.5 have been carried out during the second year and are still ongoing; hence all the related objectives have been partially achieved and will progress in the remaining period of the project.

The assessment conducted in this report uses the information provided by the following partners:

[1] After software development (*Dev*) and information technology operations (*Ops*)

- University of Malta (BEN 1, UOM)
- University of Leicester (BEN 2, ULEIC)
- NOVA ID FCT-Associacao para a Inovacao e Desenvolvimento da FCT (BEN 3, NOVA)
- University of Kent (BEN 4, UKENT)
- Alma Mater Studiorum - Universita' di Bologna (BEN 8, UNIBO)
- Universita' degli Studi di Torino (BEN 9, UNITO)
- Actyx AG (BEN 10, ACT)
- Bitland SRL (BEN 11, BTL)
- Xibs Ltd (BEN 15, XIB)
- DCR Solution (BEN 16, DCR)
- Universidad de Buenos Aires (TC/OPE 20, UBA)
- McAfee Argentina S.A (TC/OPE 22, MCF)

Finally, this report is based on the activities carried among the partners during the second year.

Researcher	**Category**	**Declaration Number**	**Starting Month**	**Duration (PM)**
Hernan Melgratti	ER	4,8, 18, and 41	4, 7, 11, 14 and 17	5.24
Carlos Lopez Pombo	ER	5	4 and 15	1.3 (partially on WP2 and WP3)
Agustín Martinez Suñé	ESR	6	4 and 15	2.37 (partially on WP2 and wP3)
Maurizio Grabrielli	ER	3	5	0.3
Iván Arcuschin	ESR	12	10 and 16	2 (partly on WP3)
Emilio Tuosto	ER	13, 38, and 43	11, 19, and 24 (partially on WP2)	2.14
Caroline Caruana	ESR	19	15	1
Diegeo Garbervetsky	ER	20	15	1.07 (partly on WP2 and WP3)
Adriana Laura Voinea	ESR	21	15	1
Leandro Nahabedian	ESR	22	15	1.03 (partly on WP2 and WP3)
João Costa Seco	ESR	25	16	.53

Facundo Nahuel Maldonado Medina	ER	27	17	1.03
Ivan Lanese	ER	28	17	1 (partly on WP3)
Adrian Francalanza	ER	29	17	.23
Cosimo Laneve	ER	30	17	.53
Ornela Dardha	ER	31	19	.5
Roland Kuhn	ER	37	19	.53

List of acronyms

API: *a*pplication *p*rogramming *i*nterface
b-APIs: *b*ehavioural APIs
DevOps: software *dev*evlopment and information technology *op*eration*s*
DSLs: *d*omain *s*pecific *l*anguages
EPAs: *e*nabledness-based *p*rogram *a*bstractions
HML: *H*ennessy-*M*ilner *l*ogic
IDE: *i*ntegrated *d*evelopment *e*nvironment
OpenDXL: Open Data Exchange Layer
SMT: *s*atisfiability *m*odulo *t*heories
TIE: Threat Intelligence Exchange
WP: *w*ork *p*ackage

Table of content

1. Introduction

The mission of work package 4 (WP4) is to support the activities of the other WPs of the project with the (i) development of new tools and methodologies, (ii) the integration and extension of tools, and (iii) the identification of new model-driven approaches to testing and verifying behavioural properties of APIs. The ultimate ambition of WP4 is to realise a significant improvement in the reliability of industrial API systems through the development of round-trip (re)engineering techniques advocated in BehAPI.
The main goal of this deliverable is to discuss how the tool support in the consortium fosters the round-trip engineering of b-APIs. Therefore, in line with the aforementioned mission, to describe the activities related to the round-trip engineering this deliverable relies on information about the tools collated in D.4.1 and discussed in D.4.2.
This deliverable describes methodologies for round-trip engineering and how they rely upon top-down engineering (T.4.2) and bottom-up engineering (T.4.3). Moreover, a description of how tool chains have been or can be embedded in the SDLCs of b-APIs (T.4.5) is given.

Terminology. Hereafter, we assume that the term ‘artefact’ refers to a tool, a platform, or a development or verification approach.

In the next sections, we refer to the artefacts available in the consortium and that have been described in D.4.1; the reader is referred to Section 2.2 of D.4.1 for details about those artefacts.
Below, Sections 2 and 3 describe methodologies for top-down and bottom-up engineering respectively. Section 4 comments about round-trip methodologies.

2. Methodologies based on top-down or bottom-up engineering

In the following we describe the case studies that academic and industrial partners used to start sketching the methodologies that have been identified in the activities of the second year. These case studies are in preliminary phases where partners are investigating promising directions of work. It is foreseeable that in the future new methodologies and case studies will be included in this collection.

The ACT case study
In the manufacturing industry downtime is very expensive, therefore most small and midsize factories are still managed using paper-based processes. The problem space is perfectly suited for a microservices approach: well-defined and locally encapsulated responsibilities, collaboration and loose coupling, rapid evolution of individual pieces. But how can we operate microservices such that

they can deliver the resilience of paper? How can we leverage the locality of process data and benefit from high bandwidth and low latency communication in the Internet of Things?
ACT's radical approach is to deploy autonomous actor-like agents in a peer-to-peer network on the factory shop-floor, using event sourcing as the only means of communication and observation. A key problem is to maintain a coordination-free totally ordered eventually consistent event log and its consequences on the programming model inspired by the time warp algorithm.
ACT, UBA, and ULEIC are collaborating to devise a model of ACT's platform. The current approach is to represent ACT actors as finite-state machines that operate on a shared set of data according to two semantics. An abstract semantics characterises the ideal execution where changes to the information is atomic and globally observable. A lower level semantics captures the actual asynchonous execution of actors. The methodology for the analyses of actors will be based on a comparison between the abstract execution and the asynchronous one.
This preliminary model allowed developers to have a more abstract view of the software that highlights and clarifies several aspects of the implementation and helps to maintain and evolve it. On the one hand, this methodology is inspired by choreographic approaches since the abstract semantics can be seen as a specification of the global protocol. On the other hand, the abstract semantics is still too level and more suitable global specifications should be found.
This line of work is still ongoing and not supported by tools; however advances are expected via forthcoming secondments to ACT from UBA and ULEIC. The partners will also consider how to automatically extract finite-state machines directly from the code implementing ACT actors. Also, we envisage the possibility of using algorithms from the synthesis of local actors into global specifications.
In the third year the work on this case study will progress with two further planned secondments from UBA and ULEIC to ACT.

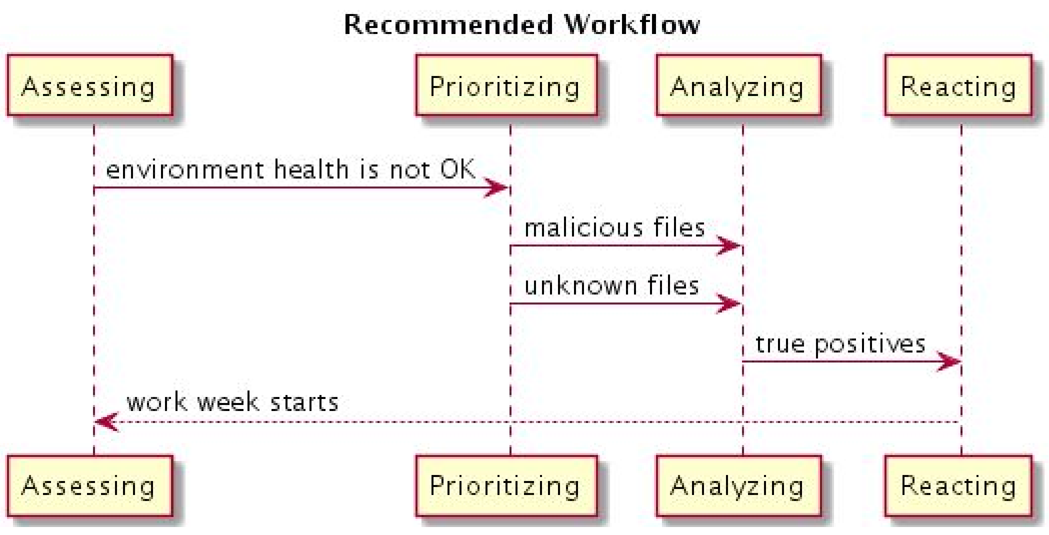
The MCF case study

The Open Data Exchange Layer (OpenDXL, https://www.opendxl.com) is an open-source initiative aiming to support exchange of timely and accurate cyber-security information in order to foster the dynamic adaptation of interconnected services to security threats. OpenDXL is part of the McAfee Security Innovation Initiative involving a hundred ICT companies (including HP, IBM, and Panasonic). A main goal of OpenDXL is to provide a shared platform to enable the distributed coordination of security-related operations. This goal is supported by the threat intelligence exchange (TIE) reputation APIs designed to enable the coordination of activities involving

1. the assessment of the security threats of an environment (configuration files, certificates, unsigned or unknown files, etc.);
2. the prioritisation of analysis steps (focusing on malicious or unknown files);
3. the customisation of security queries based on reputation-based data (such as product or company names);
4. the reaction to suspicious indicators.

A key aspect of OpenDXL lays in its service-oriented nature. Providers use the APIs to offer various services such as reporting services, firewalls, security analytics, etc. Consumers of these APIs

(typically companies or large institutions) can either use existing services, or combine them to develop their own functionalities.
The basic communication infrastructure features an event-notification architecture whereby participants subscribe to topics of interests to generate events or query services. Such topics are also used to broadcast security information of general interest. The main components of are clients, servers, and brokers. The latter mediate interactions among clients and servers in order to guarantee service availability. Brokers interact with each other to dynamically assign servers to clients when default servers are unavailable.
The high-level workflow of the TIE API is specified with semi-formal documentation; for instace sequence diagrams like the following one



describes the main workflow of TIE and, together with other informal documentation, guides the implementation of new components or the composition of services available on the platform. For instance, the documentation describing how clients can set the reputation of a file specifies that a client must have permission to send messages to the `/mcafee/service/tie/reputation/set` topic.
Basically, this is the documentation that third party developers follow to implement the client side of the application that must consume the TIE API.
This state of affairs forces MCF to adopt a very defensive approach in the provision of the server side in order to make TIE cope with anomalous behaviour exhibited by third-party clients (reported to third-parties after a post-mortem analysis of execution logs).
To tackle this issue, MCF, UBA, and ULEIC devised a top-down methodology following principles of model-driven engineering.
The proposed methodology provides a systematic approach
i. to turn informal documentation of APIs in precise models and
ii. to enable the application of formal methods to develop and analyse services.
The methodology consists of the following steps:

1. Device a graphical model representing the coordination among the components of the application; for this we use global graphs.
2. Transform into behavioural types formalising the protocol into a behavioural type representing the global behaviour of the application; for this we use klaimographies.
3. Transform into specifications of each component of the application; for this we project on local types.
4. Transform the local types into state machines from which to derive monitors to check possible deviations from expected behaviour and verify implementations of components.

Crucially, in the chain of model transformation sketched above, steps 1 and 4 are paramount for practitioners to apply this methodology: the use of visual, intuitive, yet formal models enables a fruitful collaboration among stakeholders on the one hand and to apply tools and techniques for developing and analysing applications, on the other hand.

This activity suggests that this is a promising line of research that could be beneficial to both academics and practitioners. In fact, behavioural types (as many formal methods) may not be easy for practitioners to handle. To address this issue we opted for models offering a visual and intuitive presentations of the formal models used in the specifications. This has been fundamental to have an effective communication among academic and industrial partners. Once the global graph expressing the intended behaviour has been identified, the session type formalising the expected behaviour has been defined mainly by the academic partners. Remarkably, the transformation from local types to state machines was suggested by MCF who saw it as a more streamlined way of sharing the specifications among practitioners (including those from third-party organisations).

It is worth noticing that the methodology above is not bound to the cyber-security domain and indeed it can find applications in other contexts or application domains.

It is in the scope of future work to use the formal framework of global graphs. In fact, global graphs have been key to facilitate the collaboration between academics and industrial partners. For the former can use global graphs precisely (since they come with a precise semantics) and the latter can use the visual and intuitive presentation of global graphs. However, none of the formal methods based on global graphs have been exploited so far in the methodology.

In the future, a possible use of global graphs in this methodology is the automatic generation of code from the projections to clients. This is already possible for Erlang clients, but extensions of ChorGram to other target languages could be developed. This would be very useful for instance when a service provider needs to develop several versions of the components for different execution environments. Having tool support to generate template code for implementing the communication protocol of each component would speed up the development process and reduce the time of testing (which would not need to focus on communications which would be correct-by-costruction). In order to attain this it could be useful to dress up global graphs with existing industrial standards that practitioners may find more familiar (and may be more appealing). An interesting candidate for this endeavour is BPMN bpmn since its coordination mechanisms are very close to those of global graphs. In fact, BPMN is becoming popular in industry and it has recently gained the attention of the scientific community which is proposing formal semantics of its constructs. For instance, the formal semantics of BPMN could be conducive of a formal mapping from BPMN to global graphs or global types. In this way

practitioners may specify global views within a context without spoiling the rigour of our methodology.
Another benefit of the methodology is that it enables the analysis to further properties expected of TIE components and that can be checked from the logs. For instance, TIE clients are supposed to guarantee a so-called time-window property which requires that

> "a request for the analysis of the same file from a client must not happen before a given amount of time elapsed from the previous request from the client for the same file."

This property (as well as others) can be checked by monitor derived from the local types. Also, it is possible to use DetectEr for the run-time monitoring of TIE.
To showcase the use of behavioural types, and at the same time, to test the Mungo tool, NOVA will implement a simplified version of TIE in Java with Mungo usages. This exercise is useful not only to assess the expressiveness of Mungo but also to provide MCF with a concrete replicable usae-case.

4. Methodologies for round-trip engineering

The cornerstone of round-trip engineering is the entanglement of top-down and bottom up techniques. Intuitively, any round-trip engineering methodology can be thought of as being constituted by

I. top-down techniques that from abstract models produces code for b-APIs
II. a phase where automatically generated code is completed (eg with internal computations), or evolved (eg by using new third-party components), or else extended with new functionalities
III. bottom-up techniques that validates code and possibly builds up abstract models of b-APIs
IV. traceability mechanisms that allows developers to keep to relate requirements, code, and abstract models with each other.

The need for bottom-up techniques is typically required since alterations may break correctness and introduce defects. When this happens it is crucial to identify the defects, correct them, and reflect them back in the new abstract models in order to maintain aligned b-APIs documentation. This is where traceability mechanisms play an important role.
Albeit the stage we are at is preliminary for devising full-fledged methodologies for round-trip engineering of b-APIs, some important steps have been already achieved towards this direction. As noted in D.4.2, tool integration is progressing. During the third year several tools will be chained up to support round-trip engineering of b-APIs. The work done by NOVA, UGLA, and GbW on extracting models from Java code is an example of key work that needs to be realised in order to apply analysis techniques such as behavioural (sub)typing disciplines. Session types are recently being integrated into mainstream distributed programming languages. In practice, a very important notion for dealing with such types is that of subtyping, since it allows for typing larger classes of system, where a program has not precisely the expected behaviour but a similar one. Unfortunately, recent work has shown that subtyping for session types in an asynchronous setting is undecidable. To cope with this negative result, the only approaches we are aware of either restrict the syntax of session types or limit communication (by considering forms of bounded asynchrony). Both approaches are too restrictive in practice, hence UNIBO and UKENT proceeded differently by introducing an algorithm for checking

subtyping which is sound, but not complete (in some cases it terminates without returning a decisive verdict). The algorithm is based on a tree representation of the coinductive definition of asynchronous subtyping; this tree could be infinite, and the algorithm checks for the presence of finite witnesses of infinite successful subtrees. The algorithm has been implemented by UNIBO and UKENT and tested on many examples that cannot be managed with the previous approaches. This provides an empirical evaluation of the time and space cost of the algorithm.

In the same spirit, although based on different techniques, is the work done by ACT and UGLA to model state machines as dependently typed linear functions in Idris 2. This is a rather interesting activity because, but it is in an inception phase and it has not yet yielded concrete (usable) results. It is worth noticing that an extension of session types to dependent session types may be necessary in order to capture some of the interesting protocols between microservices used by industrial partners (e.g., ACT). One example is http://www.reactive-streams.org/, which requires dependent typing since the value transmitted from the Subscriber to the Publisher when requesting items describes how many calls the Publisher may subsequently make to the Subscriber's onNext function.

The advances of tool integration activities will enable the combination of tools such as ChorGram, Mungo/StMungo to automatically infer global specifications of software, and the prototype for behavioural subtyping developed by UNIBO and UKENT. These models are not only amenable of rigorous analysis but, crucially, to foster program comprehension and b-API precise documentation. This type of integration will provide a solid support for round-trip engineering by using the complementary features of top-down and bottom-up functionalities offered by the various tools.

Such support is currently enabled by the existence of artefacts for the top-down and bottom-up development, but requires a closer integration and interoperability of those artefacts. The future activities within the project will address those aspects and be reported in the forthcoming revision of this deliverable.

There are examples of methodologies that may be independent of tool integration since they rely on one tool only. For instance, the recent extension of ChorGram with top-down features (eg the generation of Erlang executable programs) can be chained-up with ChorGram's constitutive feature of synthesis of global graphs from communicating-finite state machines. ULEIC is currently exploring this possibility; a main obstacle to overcome is the improvement of the rudimentary traceability mechanisms currently supported by the tool.

Also relevant to this round-trip methodology is the Typestate visualizer developed at NOVA, that converts Mungo usages into a form of finite-state machine, and the latter in the former (https://jdmota.github.io/mungo-typestate-parser). In a top-down approach, when developing a component, one starts by scribbling its behaviour as an automaton, The typestate visualizer could then be used to obtain a usage to typecheck a Java implementation of that component. If and/or when such component needed to be changed, the conversion to automaton could again help reasoning about how to introduce or revise functionalities and get usages whenever the resulting automaton should be tested in an implementation.