Project no.:	H2020-MSCA-RISE-2017-778233
Project full title:	Behavioural Application Program Interfaces
Project Acronym:	BEHAPI
Deliverable no.:	D.4.2 (M24)
Title of Deliverable:	Assessing tool support within BehAPI
Work package:	WP4 (tasks 4.2, 4.3, and 4.4)
Type:	R
Lead Beneficiary:	**UNIVERSITY OF LEICESTER (BEN2, ULEIC)**
Dissemination Level:	PU
Number of pages:	12
Contract Delivery due date:	**29/2/2020 (M12, 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 assesses the use of top-down, bottom-up, and round-trip tool support for engineering b-APIs (behavioural application programming interfaces). The deliverable is related to the tasks "Top-down DevOps[1] support" (T4.2), "Bottom-up DevOps support" (T4.3), and "Tool interoperability" (T4.4) 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 techniques 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 describes our assessment of engineering techniques for b-APIs (O.4.2), level of integration of artefacts (O.4.2), and artefacts interoperability (O.4.3). The report also comments about continuous development (O.4.4). Of particular interest are the functionalities for the management, testing, verification, and monitoring of b-APIs. Activities on the tasks mentioned above have started toward the end of the first year; 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 in D.4.1 provided by the following partners:

- 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)

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

- University of Kent (BEN 4, UKENT)
- University of Glasgow (BEN 6, UGLA)
- 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	ER	27	17	1.03

Maldonado Medina				
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
BPMN: *B*usiness *P*rocess *M*odelling *N*otation
CTL: *c*omputation *t*ree *l*ogic
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 *E*xchange *L*ayer
SMT: *s*atisfiability *m*odulo *t*heories
TIE: *T*hreat *I*ntelligence *E*xchange
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 activities carried out in WP4 are driven by the aforementioned mission. In particular, information about the tools developed or used both in academic and industrial practices in API provision or consumption are being collated in D.4.1 and used in the assessment reported here. Our assessment concerns the following dimensions:

- top-down engineering (T.4.2)
- bottom-up engineering (T.4.3)
- interoperability and integration (T.4.4)

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

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 descriptions of those artefacts. Sections 2 and 3 address top-down and bottom-up engineering respectively. Section 4 comments about the level of interoperability and the possible integration of artefacts. Concluding remarks are in Section 5.

2. Top-down engineering

Top-down software engineering of b-APIs impacts on the quality of the software as well as on the cost and time-to-market of software development. As reported in D.4.1, the consortium features many artefacts supporting top-down development of b-APIs. The initial activities have extended existing functionalities based on model-driven development of academic tools to meet the needs of practitioners. This activities rely on the following general scheme recently advocated in the context of behavioural type systems:

1. a global model is devised capturing the main behavioural aspects of the system
2. local artefacts are (semi-)automatically obtained from the global model by *projection* algorithms.

This scheme is well supported by many of the artefacts in the project. Key features have been added to academic prototypes to support the above top-down approach or are in progress:

- ULEIC and UOM extended ChorGram with project algorithms to automatically generate executable Erlang components and monitors
- UBA and ULEIC are collaborating on model-based testing for b-APIs by combining functionalities of ChorGram, Evolutiz, and EvoSuite
- XIB, UBA, and ULEIC started to apply to an industrial case study automated and model-based testing using ChorGram, Evolutiz, and EvoSuite
- a collaboration among UGLA, BTL and UNIBO is integrating Mungo and JaDA for analysing correct usage of APIs in Java programs developed at BTL

A relevant role is played by the possibility of automatic code generation provided by projection mechanisms. In this respect we mention

- AIOJC, which generates executable distributed code free from deadlocks and races
- APIgator, which saves time coding with UI and generates standard project assets
- Autogen, for rapid development and automatic deployment with some guarantee for automatic backwards compatibility
- the Erlang-based projection of ChorGram

Other functionalities provided by the artefacts in BehAPI enrich the support for top-down engineering. More specifically, the conformance of local components to projected (or otherwise specified) behaviour is supported by

- CobaltBlue, for protocol conformance and deadlock analysis of Objective Join Calculus
- MCC, which checks protocol conformance and deadlock of actors
- T2, which works on C programs to verify temporal properties

The consortium has also devised a methodology for supporting software development in the Open Data Exchange Layer (OpenDXL) platform. OpenDXL is part of the McAfee Security Innovation Initiative, a consortium of about a hundred ICT companies including HP, IBM, and Panasonic. This 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.
MCF, UBA, and ULEIC devised a model-driven approach for the effective development and analysis of platform services. The methodology consists of the following steps:

1. Device a graphical model G representing the coordination among the components of the application; for this we use global graphs
2. Transform G into behavioural types formalising the protocol into a behavioural type K representing the global behaviour of the application; for this we use klaimographies

3. Transform K into specifications of each component of the application; for this we project K 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

Remarkably, 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.

The transformation from global graph to klaimographies is not prone to be used by laypersons and in fact it was developed by the academic partners. However, and remarkably, the transformation from local types to state machines was suggested and realised by MCF. This transformation yields a more streamlined way of documenting and sharing the specifications among practitioners (including those from third-party organisations).

It is worth noticing that the methodology has a wider application than the Threat Intelligence Exchange (TIE) service of MCF and it can in fact be integrated in most of the software development contexts of API based on message-passing. It is also worth mentioning that the use of visual models greatly mitigated the usual problems that formal methods create to non-academic users.

BTL, UGLA, and UKENT started to collaborate on APIgator. An analysis of application of StMungo/Mungo to model BTL has started. Also, the collaboration is identifying use-cases from BTL.To sum up, we conclude that there is a rather high level of support of top-down engineering both in academic and industrial artefacts. A limitation of the current state-of-the-art is the high heterogeneity of languages used by the artefacts. As commented in Section 4, This heterogeneity can be tackled by using models (and corresponding transformation tools) that, abstracting away from low-level details, can be applied in different contexts. More comments about this issue are in Section 5.

The realisation of appropriate levels of traceability is key to the success of this model-driven solution. A great leap ahead in this respect is the development of the Process Highlighter tool of DCR. The tool gives paramount support in the determination of models from requirements by distilling formal models (DCR graphs) from descriptions in English (and Danish) through natural language processing. This allows developers to significantly reduce the time for requirement analysis since DCR are amenable to algorithmic analysis. Moreover, model transformations are planned to integrate the Process Highlighter with ChorGram (through forthcoming secondments from DCR to ULEIC and from ULEIC to DCR).

3. Bottom-up engineering

The top-down scheme described in Section 2 is reversed in the bottom-up approach. More precisely, the bottom-up approach prescribes the following two steps:

1. determine local artefacts capturing components' behaviour

2. build (if any) a global model documenting the global behaviour of the system.

Intuitively, this scheme is the inverse of the projection operation advocated in the top-down scheme in Section 2. The main reasons for supporting bottom-up approaches are

(a) enabling behaviour-based composition of APIs
(b) validating behavioural properties (eg message orphanage and unspecified reception freedom or deadlock analysis) of b-APIs composition
(c) facilitating documentation of composition through the automatic devise of more abstract models of local artefacts of global models of their composition.

The consortium offers a range of artefacts to support bottom-up engineering of b-APIs. In the first instance, there are tools to extract abstract descriptions of local behaviours from code. In this respect, we mention

- ChorGram, which synthesises global graphs out of communicating automata abstracting local components
- FuSe, which infers several session types from OCaml programs
- UsInfer, which infers behavioural types.

The support provided by the artefacts listed above is paramount for the analysis and documentation of b-APIs. For instance, the inference of behavioural types may avoid the repetition of testing phases when modifications of components do not affect their communication behaviour. This saves time and production costs for the development of b-APIs. Also, the possibility of synthesising global models out of local artefacts (i) permits developers to identify and fix defects at early stages of the development and (ii) yields valuable documentation on overall behaviour of compositions that can be used when modifications are necessary. This enhances the so called 'program comprehension'. In this respect, it is significant the work initiated by ACT, UBA, and ULEIC on the modelling of the platform currently under development at ACT. The developers at ACT mainly develop and maintain the platform at the code level, for which they use the typescript language. During the collaboration a model of the local behaviour has been manually extracted from the implementation in order to analyse the properties of the eventual consistency algorithm designed by ACT. This 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. 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.
ULEIC has also started a new ambitious line of research advocating the use of choreographic models to develop model-driven testing of message passing applications. This line of research has just started thanks to a PhD studentship funded by the GSSI. The main goals of the research is to devise test-generation from global and local models of choreography to support white- and black-box testing of message-passing software. Initially we will consider Erlang and Go as target languages to develop

the main concepts. The validation of the methodology will be assessed with the industrial applications currently involved in the BehAPI project.
In conclusion, BehAPI provides good support for enabling bottom-up engineering. There is a high potential for tool interoperability (as witnessed e.g., by Contractor). The consortium should strive for the integration of the existing artefacts with a combination of solutions: (1) adoption of common formalisms for global and local artefacts and (2) development of model-transformation algorithms to map different models into each others. Both these solutions will facilitate interoperability and integration as discussed in the next section.

4.Tool Support for Round-trip Engineering of b-APIs

The stage we are at is too preliminary for a full assessment of the support to round-trip engineering. 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 work for tool integration has progressed during the second year of the project. We are approaching a stage where several tools can be chained up to support round-trip engineering of b-APIs. In fact, NOVA, UGLA, and GbW have started to develop proof-of-concept solutions that enable the extraction of models from code written in real programming languages (e.g., JAVA), analyse the model with different behavioural typing disciplines (including behavioural subtyping and typestate). This line of research will deliver theories and applications of behavioural type systems in object-oriented language. In particular StMungo and Mungo tools for Java can be applied in the context of sensor networks, one of the application domains of GbW. NOVA is also working on correct usage of linear objects to avoid null pointer exceptions in the Mungo language. UGLA is currently working on type-theoretic techniques for generalising access control beyond strict linearity. This work will be a foundation for better type systems for access control in object-oriented languages and in particular Mungo.
This work paves the way to use Mungo and StMungo in combination with ChorGram to extract global specifications of software, analyse it with the new features of ChorGram, evolve the global specifications and then generate code (in different target languages). This direction identified within the collaboration among GbW, NOVA, and ULEIC (triggered by a recent secondment from ULEIC to GbW) seems rather promising and will be explored further.
IXR, UOM, and ULEIC extended ChorGram with functionalities to generate executable Erlang programs and monitors for them from global graphs. The generated code is a set of skeleton programs realising the communication pattern specified by the global graph; developers need to complete the code by implementing the internal computations of programs. Although the automatically generated code is correct-by-construction, the manual intervention for realising internal computations is an error-prone activity which may introduce defects and generate unexpected patterns of communications. Integrations are planned in the near future with

- the monitoring features of DetectEr (to identify possible defects at run-time),
- testing and debugging tools of the consortium such as CauDEr, CodeDefenders, ETG, Evolutiz, or EvoSuite,
- and possibly adapting behavioural model-extraction tools such as FuSe, UsInfer, or EPAs.

This 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.

The future activities within the project will address those aspects and foster approaches to round-trip engineering to a greater extent. The next developments in this respect will be reported in the forthcoming revision of this deliverable.

5. Assessing Interoperability and Integration

As found out in D.2.1 (cf. section 3), the consortium is active in several application domains where partners consume and provide APIs both for internal and external usage. D.2.1 points out also that behavioural specifications are of practical relevance most of the time.

This variety is also reflected in the range of languages and formalisms adopted in the consortium. Such variety hinders interoperability of artefacts as well as their integration in software practice. This may have a negative effect in the adoption of BehAPI's artefacts. Crucially, interoperability of the artefacts within (and outside) the consortium has a rather high potential. Many artefacts are based on the same classes of models (automata, session types) or languages. Some artefacts support many languages (as Contractor) or can be promptly adapted to other languages (as MCC).

In this respect, GbW, NOVA and ULEIC plan to identify a common format of automata for the representation of communicating finite-state machines and the communicating automata used by NOVA. This endeavour will also consider the possibility to adopt formats used by other tools outside the consortium. For instance, we will consider the formats adopted by state-of-the-art model checkers. This common format can then be used to generate UML state machines (as done in the methodology developed by MCF, UBA, and ULEIC described in Section 2).

Another gap to fill in is the quality of the documentation related to b-APIs. Practitioners are reluctant to spend much time documenting their work for obvious reasons. Even when some documentation is available, it is not apt for automatic processing. The consortium started to tackle this issue. For example, an initial valuable attempt is the one explored in the Dynamic Condition Response artefact from DCR. The idea is to use highlighter tools to facilitate model creation and provide traceability between process models and textual descriptions of law and regulations. An first version of the process highlighter tool from DCR is now available (it has also been demonstrated in one of the bootcamps of the first shool of the project). DCR and ULEIC plan to consider extensions of ChorGram with highlighters.

The gap between informal descriptions in non-academic contexts needs to be addressed. The consortium is determined to develop model transformation approaches to integrate models and languages so to facilitate interoperability. This process is hindered by the large heterogeneity in the industrial sector. Non-academic partners rely on many different platforms and languages (see section 3.2.2 of D.2.1). Some of them are supported by some of the academic prototypes. Work is needed to

improve academic prototypes making them address technology used in practice. The collaborations with the industrial partners allowed us to identify some modelling languages used in the industrial practice that academic tools should support. In particular, UML state machines and the Business Protocol Modelling Notation (BPMN) seem to be appealing to the industrial partners.
Also, collaborations within the consortium should develop usability of prototype tools to meet the expectations of practitioners. The interactions among developers and young researchers is paramount in this respect. The schools has been an important venue to let these two groups meet, discuss and understand each other's needs. The presentations and the material of the courses have been conducive of the identification of concrete factors that limit the applicability of the theoretical models to the industrial context. For instance, the lack of modularity in choreographic approaches is a key problem for large systems. UNIBO and ULEIC have started to address this problem and devised a novel choreographic approach that enables the composition of global types. Another issue that limits the applicability of behavioural abstractions is their limited support for protocols where components dynamically join or leave the protocol or where properties different than the usual safety and liveness properties checked by behavioral types. UBA and ULEIC have defined a novel typing discipline (used in the case study of MCF described in Section 2) that focuses on the properties of the data flow of protocols, rather than on their control flow properties.
Another promising direction to explore is the integration of verification techniques and tools within industrial context. In D.2.1, non-academic partners of BehAPI reported about testing (backend test, black-box unit testing, and other testing approaches) and about monitoring of API's (backend through specific monitors based on custom code). However, there is a general feeling that quality of software must be improved towards several dimensions such as understandability, ease of integration of clients, backend, and components, reliability of implementation with precise specifications amenable for verification and validation, maintainability, and automatic code generation.
In this respect, the consortium is well-equipped with a suite of tools, techniques, and languages that can be applied. Hints for the suitability of such artefacts are already emerging within the aforementioned collaborations between academic and industrial partners. Another encouraging example is provided by DCR, which emerged from academic research and has already proved itself suitable in several case studies in Danish national projects.
Finally, this deliverable has found out that artefacts for the bottom-up support of b-APIs development must be fostered. Several artefacts do provide useful functionalities (FuSE, UsInfer, ChorGram, DetecEr) or the subtyping prototype under development in the collaboration between UNIBO and KENT. However, this needs to be applied to case studies, integrated among themselves, and used in connection with traceability mechanisms.

6. Conclusions

This deliverable has evaluated the artefacts developed or adopted in the project. The analysis highlights the following points:

- There is a substantial level of support for provision and consumption of b-APIs which has already triggered some interaction between academic and industrial partners.
- The artefacts of BehAPI support both top-down and bottom-up engineering approaches to b-APIs; this is crucial because, roughly speaking, top-down engineering is quite relevant for b-API provision while bottom-up engineering is more apt for the composition needed in b-API consumption.
- There is a wide range of applications covered in the project that determines an equally large range of technologies currently developed or used; this poses interesting challenges to interoperability.
- Currently, there are artefacts that offer some degree of automatic documentation based on executable or machine processable models, but improvement is necessary in this respect. The way b-APIs are documented is not ideal.

Addressing the challenges summarised above is paramount for evolvability, and maintainability of software based on b-APIs. The consortium has already developed some solutions that mitigate the problems of integration and interoperability; we believe that the solution can be attained with the work planned for next two year.

Some of the results reported in this document have been published in conferences and jorunals. We expect more publications to appear soon

This deliverable will be available at the website of the project https://www.um.edu.mt/projects/behapi/ in due course.