Project no.:	H2020-MSCA-RISE-2017-778233
Project full title:	Behavioural Application Program Interfaces
Project Acronym:	BEHAPI
Deliverable no.:	D.4.1 (M24)
Title of Deliverable:	Portfolio of tools and platforms
Work package:	WP4 (tasks 4.1, 4.2, and 4.3)
Type:	R
Lead Beneficiary:	**UNIVERSITY OF LEICESTER (BEN2, ULEIC)**
Dissemination Level:	PU
Number of pages:	15
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 reports on the results of the task "Platforms survey and extensions" of work package (WP) 4 "Tool support". The goals of WP4 are to enhance and develop tools and technologies for the reliability of API-based (Application Programming Interface) systems. WP4 aims to realise the tool support for the round-trip engineering advocated by b-APIs (behavioural APIs). More specifically, WP4 concerns

- new model-driven approaches to testing and verifying behavioural properties of APIs,
- the development of reverse-engineering techniques (ranging from type inference to machine learning), and
- extensions of industrial frameworks.

This report describes our assessment of the state-of-the-art (O.4.1) achieved by reviewing the existing tools/platforms. The key criteria of the assessment are

- the possible integration among existing prototypes and tools (O.4.2)
- extendibility of prototypes, platforms, and tools in the light of the round-trip engineering methodologies advocated in BehAPI. Of particular interest are the functionalities for the management, testing, verification, and monitoring of b-APIs.

This first version of the deliverable concerns with the state-of-the-art (T.4.1) and does not touch upon T.4.2 and T.4.3; the progress on these tasks will be reported upon in the subsequent revisions of this deliverable (due in M24 and M36).

The information used in this report has been acquired during the first year. ULEIC and UBA have collaborated to identify the relevant data to gather. ULEIC then gathered the information from 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)
- University of Kent (BEN 4, UKENT)
- University of Glasgow (BEN 6, UGLA)
- Alma Mater Studiorum - Universita' di Bologna (BEN 8, UNIBO)
- Università 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 details activities that were planned in the application and have been carried out between UOM and ULEIC for the extension of ChorGram (a tool for synthesis of choreographies developed at ULEIC) with monitoring mechanisms for Erlang-based software.
The following table provides the essential information concerning the performed secondments.

Researcher	**Category**	**Declaration Number**	**Starting Month**	**Duration (PM)**
Hernan Melgratti	ERER	4, and 8, and 18, and 41	4, 7, and 11, 14 and 17	5.241.24, 1.83, and 1.17, and 1
Carlos Lopez Pombo	ER	5	4 and 15	1.3 and 110.3 (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)

Table of content

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 D*ata *E*xchange *L*ayer
RAML: *R*ESTful API *M*odeling *L*anguage
RESTful: *Re*presentational *S*tate *T*ransfer
SMT: *s*atisfiability *m*odulo *t*heories
TIE: *T*hreat *I*ntelligence *E*xchange
WP: *w*ork *p*ackage

1. Introduction

The mission of work package (WP) 4 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, the consortium made an initial effort to collect information about the tools developed or used both in academic and industrial practices in API provision or consumption. This effort has been sustained during the second year by extending and updating the collection of tools..

The analysis of the gathered information confirmed the expectation about the wide range of different tools available reflecting the spectrum of models and languages (including those envisaged in WP2 and WP3) required at various levels of abstraction. As reported in D2.1, languages and models adopted span from graphical and automata-based formalisms to textual DSLs (Domain Specific Languages) and programming languages. Hence, tools concerning WP4 reflect such variety and require the development of precise semantic relations to guarantee flexibility, compositionality, and interoperability of tool chains. Another challenge confirmed by the initial activities of WP4 are the amalgamation of top-down and bottom-up engineering of b-APIs.

Interestingly, this rich context provides an ideal framework for fruitful collaborations within the consortium. In fact, the school held at ULEIC offered courses based on some of the prototype tools described below as well as boot camps centred on platforms and tools from industrial partners (as reported in D1.3). Also, some of the activities in the 2nd year of the project have been driven by the possibility of applying techniques and tools based on behavioural types in industrial settings. In particular, researches at UBA and ULEIC have started a collaboration with ACT and XIB to apply academic prototype respectively for formal modelling and model-based testing of communication-centric applications. Also, MCF, UBA, and ULEIC have designed a methodology for the documentation and monitoring of 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.
The assessment of costs, scalability, and quality attributes such as verifiability, robustness, reliability, etc.) of tools is premature at this stage and will be scope for the refinements of this deliverable in the future.

The collected information is summarised in Section 2 and discussed in Section 3 together with the related activities. Section 4 gives details of the forthcoming collaborations mentioned above. Concluding remarks are in Section 5.

2. Artefacts relevant to BehAPI

Following the approach adopted in WP2 in harvesting the information about practices, it was decided to use a shared document that partners completed with the relevant information. In fact, given the geographical distribution of the consortium and the time constraints other approaches would have hardly been as effective as the one adopted.
Hereafter, we assume that the term ‘artefact’ refers to a tool, a platform, or a development or verification approach. In the rest of this section we first discuss the type of information we sought and then we report the main artefacts currently available or of interest in the consortium.

2.1 Tools, platforms, and techniques for b-APIs

Besides obvious information (name of the artefact, developers, contacts), the crucial type of information that we required is:

- nature of artefacts
- status, versions, and usability of the artefact
- type of documentation available
- main strengths and weaknesses
- availability

- potential integrations and interoperability.

We briefly comment about each of the item above.

2.1.1 Nature & Status, versions, and usability of artefacts

The consortium has (and uses) a rather large range of artefacts. Besides a number of academic prototypes and of commercial products, some partners (e.g., DCR) use and develop in-house artefacts which emerged from academic research. Among the objectives of WP4 is the fostering of cross-fertilisation among these types of artefacts. Hence, awareness of nature, status and usability of artefacts in the context of BehAPI is of great relevance.

A key aspect of BehAPI's approach is the study of the relation for top-down and bottom-up approaches to provision and consumption of b-APIs. The tool support we envisage has to reflect this aspect. For this it is paramount to distinguish those artefacts suitable for top-down development from those more apt to support bottom-up engineering. Also, it is crucial to identify which artefact can support provision and which ones can support consumption of b-APIs.

Another dimension of interest is the application domain the artefact is for.

Related to the above, it is also important to gather information about different versions. For instance, commercial products may feature different functionalities depending on the target user as e.g., commercial versus academic licenses. Also, academic prototypes.

2.1.2 Type of documentation

A typical drawback of prototype and state-of-the-art artefacts is the limited documentation available. The project needs to identify the artefacts that require attention on this point. Notably, this is relevant not only for the integration and the cross-fertilisation of artefacts, but for the dissemination and the adoption of artefacts as well. In this respect, an important action is the adoption of such artefacts in the courses and boot camps of the schools organised in BehAPI; this will be a good occasion to develop a solid body of documents and tutorials that will foster the adoption and interoperability of artefacts.

2.1.3 Main strengths and weaknesses, Availability and Potential integration & interoperability

This piece of information is the first specific attempt within the BehAPI project to single out the hooks enabling artefacts integration at a technical level. Partners have been invited to remark the strengths and weaknesses of current tools as well as practices. This information will be extremely useful to identify complementary features of artefacts which will lead to a starting point for future collaborations.

Related to strengths and weaknesses is the mechanisms adopted to access the artefacts.

Next section reports the set of artefacts together with the information related to the criteria commented above.

2.2 BehAPI's tools

We now summarise the artefacts involved in BehAPI.

AIOCJ (UNIBO, http://www.cs.unibo.it/projects/jolie/aiocj.html) is an academic prototype for top-down programming of distributed adaptive applications using a choreographic language. AIOCJ is at version 1.3.8 which features has an editor integrated into Eclipse. The too directly generates executable distributed code free from deadlocks and races; AIOCJ supports on-the-fly updating of running code.
The main limitations of the tool are the lack of integration software and that applications should be programmed in the AIOCJ DSL.
An interesting extension could be the compilation of more abstract models (eg. the global graphs used in ChorGram) into the AIOCJ language.

APIgator (BITLAND, https://www.apigator.io) is a truly complete toolchain to develop APIs following the OpenApi 2.0 and 3.0 standard. The beta design phase is completed and other phases are under development. Among the key features of APIgator are: flexible adaptation to various software development processes; support for fast coding; generation of standard project assets; quick creation of API mock (few minutes are enough to generate executables without manual coding).
The artefact does not feature software quality. It would be beneficial to explore connections with the Evolutiz, or EvoSuite.

Asynchronous session subtyping checker (UKENT; https://github.com/julien-lange/asynchronous-subtyping) implements the first algorithm to check asynchronous session subtyping. The algorithm is defined and proved sound in the paper "A Sound Algorithm for Asynchronous Session Subtyping" by Mario Bravetti, Marco Carbone, Julien Lange, Nobuko Yoshida, and Gianluigi Zavattaro (CONCUR'19). This tool/algorithm can be integrated in session types-checker. A limitation of the tool is that it does not scale to large session types.

Autogen (XIB; https://www.xibis.com) is a proprietary artefact released (version MVC 4.17) by Xibis and used in production. The tool is based on a Windows desktop application and it is designed to support rapid development, Key features are those for automatic deployment, upgrades and (some) backwards compatibility. A main limitation is that Autogen is available only on Windows (although other platforms can run the code once created). Potential extensions are the generation of partial domain model schemas/metadata to be used as inputs for tools.

CauDer (UNIBO; https://github.com/mistupv/cauder) is a causal-consistent reversible debugger for Erlang. It allows the analysis of concurrent Erlang programs by both back and forward exploration of executions in order to detect bugs. When going back, the programmer can undo any action provided that its consequences, if any, are undone beforehand. CauDer also allows programmers to undo a selected past action, including all and only its consequences. A version of CauDEr (available at https://github.com/mistupv/tracer/tree/master) also supports logging a real Erlang computation and replaying it inside the debugger. In this setting programmers can also redo a future action, including all and only its causes.

ChorGram (ULEIC, UKENT; https://bitbucket.org/emlio_tuosto/chorgram) is an academic prototype in beta phase. ChorGram supports the design and analysis for consumption & provision of b-APIs. The prototype features bottom-up and top-down choreographic design. Its main functionalities are the synthesis of global graphs from finite state machines representing communicating processes, projection from global graphs on communicating machines. Recently the tool has been extended with several new functionalities:

- to amend global views in order to support model-driven development and
- to generate Erlang executables and monitors for Erlang programs to support 'debugging'.

The main weaknesses are the limited support for traceability and a somehow patchy integration of top-down and bottom-up functionalities.
Plans to integrate ChorGram with Evolutiz and EvoSuite have started in order to support automatic testing for a case study at XIB and IXR.

CobaltBlue (UNITO; http://www.di.unito.it/~padovani/Software/CobaltBlue) is a stable academic prototype. The main functionalities offered by CobaltBlue are the checking of protocol conformance and deadlock analysis of concurrent objects implemented with typestate-oriented programming. The tool is based on behavioral type checking and deadlock analysis for Objective Join Calculus. CobaltBlue requires (some) user-provided type annotations and works with a tool-specific actor language. A type system similar to the one adopted in CobaltBlue has been implemented as plugin for analysing Scala-Akka actors. CobaltBlue could complement the funtionalities offered by DetectEr.

Code Defenders (ULEIC; https://github.com/CodeDefenders/CodeDefenders) is an open source academic prototype. It is now in a stable version. Code Defenders supports a gamification approach to software testing. Its main feature is the high quality crowdsourced unit tests for Java code. A main limitation is that Code Defenders is designed exclusively for JUnit/Java. The tools should evolve to handle large software. Code Defenders could be integrated into industrial development environments (e.g., continuous integration processes).

Contractor (UBA; http://lafhis.dc.uba.ar/dependex/contractor/Welcome.html) is an academic prototype to build enabledness-based program abstractions (EPAs) out of APIs written in C and pre/post contracts written in (kind of) FOL. Contractor generates a finite abstraction representing the contract of software and validates such abstraction with respect to user-defined properties. For programs in C, Contractor relies on Blast model checkers and for deciding contracts it uses tools based on SMT (Satisfiability Modulo Theories) (e.g, CVC/Z3).
The main strength of Contractor is the possibility of constructing sound and automatic EPA programs. A main drawback is that Contractor does not directly supports multi object protocols. Extensions of Contractor to .NET and Java/Scala are also available. The .NET extension supports all .NET languages (C#, VB, F#, etc). This extensions does not require manual intervention or access to the code, but has limitations in handling loops. The Java/Scala extension is amenable for interaction with other tools (e.g., Evosuite for test case generation).

Corinne (UNIBO; https://github.com/simoneorlando/Corinne) is a tool working on choreography automata (finite-state machines where transitions are labelled by interactions A -> B:m), performing visualization, product, synchronization and projection. Corinne supports choreographies in an automata-based setting and enables their composition. The tool is an academic prototype in alpha version. Corinne can take as input ChorGram global graphs. A limitation is that parallel composition of global graphs is not supported.

DCR graphs (DCR; https://dcrsolutions.net) is an industrial tool supporting a graphs process notation developed for the formalisation and mechanisation of adaptive case management processes. DCR (Dynamic Condition Response) graphs features (a) a (declarative) process language to define dynamic causal relations among activities; (b) a business process execution engine.
DCR graphs have been used in the formalisation of legally-compliant business processes in Danish Municipalities.
DCR graphs can be specified in a graphical language that users with no background in formal methods can use. It would be interesting to explore the connection between DCR graphs and global graphs.

DetectEr (UOM; https://bitbucket.org/casian/detecter2.0) is an academic prototype in beta phase. DetectEr synthesises Erlang monitors from mu-HML (Hennessy-Milner Logic) specifications and instruments them with the system under scrutiny. The tool separates specification from verification and treats the system under scrutiny as a black box relying only on its observable behaviour. An extension supporting degrees of synchronisation is planned.
The main weaknesses are that monitor instrumentation is not fully automatic and that different versions are not consolidated into one single tool.
Currently DetectEr is limited to Erlang, but extension for Elixir are planned. It would be interesting to intersect the features of DetectEr with the recent monitoring approach of ChorGram.

EasyJoin (UNITO; http://www.di.unito.it/~padovani/Software/EasyJoin) is a usable prototype enabling the specification of concurrent typestate-oriented programming for Java inspired by Objective Join Calculus. A key advantage of EasyJoin is that it does not need any language extensions, libraries, or external tools. EasyJoin checks linearity violations dynamically. The artefact is easily portable to other languages and works smoothly with user-provided Java code.

Error Messages for Gong (UBA; https://github.com/DamiFur/gong + https://github.com/DamiFur/migo) is an extension of Dingo-Hunter by Julien Lange, Nicholas Ng, Bernardo Toninho and Nobuko Yoshida (https://github.com/nickng/dingo-hunter) that generates error messages when a Go program cannot be shown live or safe. Hereafter we refer to this tool as GEM (after Gong with Error Messages). Error messages provided to the user are (summaries of) execution traces that lead to states that violate liveness or safety conditions in the original implementation of Dingo-hunter. This prototype tool is not integrated in any IDE (Integrated Development

Environment); error messages are textual with reduced usability features. The approach adopted in GEM could be extended to other tools (eg the analysis of ChorGram, FuSe, or UsInfer).

ETG (UBA; https://github.com/FlyingPumba/etg) automatically generates reproducible and human-readable test suites for Android applications in Espresso format, using as input sequences of interactions over widgets in the Android applications. The tool allows developers to generate executable tests (JUnit test-style) without having to write any test code.
Currently ETG is tied to the MATE tool which produces input sequences. The effectiveness of generated tests is currently under evaluation. Extensions and integrations to other tools to produce interaction sequences are being carried out as well as integration with IDEs.

Evolutiz (UBA; https://github.com/FlyingPumba/evolutiz) is an academic open source prototype for automated test generation for Android applications. The tool supports several evolutionary algorithms. Evolutiz also supports real devices. Some manual intervention may be needed to adapt related code for specific devices. The current version is ready to be integrated in Android emulators.
A limitation of the current prototype is that it is exclusive for certain versions of Android. It is planned to extend Evolutiz to other Android versions. In the next versions Evolutiz will also be extended with more search-based approaches.

EvoSuite (ULEIC; https://groups.google.com/forum/#!forum/evosuite) is an open source stable academic prototype for automated test generation in JUnit. A key advantage of EvoSuite is that it has high code coverage and decent fault detection rates. EvoSuites supports the latest versions of Java. A limitation of the tool is that it is designed exclusively for Java, but extensions to other languages are planned.
ULEIC started to explore possible uses of EvoSuite in the industrial context of XIB.

FABIoT (ULEIC; https://lra.le.ac.uk/handle/2381/42607) is an academic prototype. FABIoT is an agent-based model (ABM) that mimics the operation of different scale IoT systems over the time. FABIoT offers the toolkit for the definition of IoT environments and event-driven scenarios. The aim of FABIoT is to enable evaluation of distributed software systems (algorithms, services, middlewares or protocols) that are intended to be installed in IoT devices. FABIoT can simulate different and many scenarios. The main advantages of FABIoT are flexibility (models are designed to mimic common characteristics of several scenarios); scalability. FABIoT requires an adaptation stage that involves programming of the integration with the software to evaluate. The current version only incorporates interaction between Smart Objects. There are plans to support Cloud- and Fog- based environments.

FuSe (UNITO; http://www.di.unito.it/~padovani/Software/FuSe/FuSe.html) is a stable library implementing binary session types for OCaml. FuSe provides session type inference, equi-recursive session types, polymorphic session types, context-free session types, session subtyping. FuSe checks

endpoint linearity dynamically at runtime while many violations are detected statically nonetheless. The tool works out-of-the-box with OCaml and does not require external tools.

MCC (UNITO; http://www.di.unito.it/~padovani/Software/MCC/index.html) is a stable academic prototype for protocol conformance and deadlock analysis of actors with first-class mailboxes. The tool is based on MC, a language based on the asynchronous pi-calculus which features actors using selective-receive. MC is equipped with a behavioral typing discipline ensuring that well-typed processes are mailbox conformant (no unexpected message is ever received) and deadlock free (in a terminated process all mailboxes are empty and there are no unperformed actions). MCC checks whether an MC process is well typed. The tool requires (some) manual type annotations. There are interesting relations with plugins for analysing Scala-Akka actors.

StMungo and Mungo (UGLA; http://www.dcs.gla.ac.uk/research/mungo) is an academic prototypical toolchain composed of StMungo and Mungo tools.
StMungo (Scribble to Mungo; https://bitbucket.org/abcd-glasgow/stmungo/src/master) is a Java-based tool used to translate Scribble local protocol specifications into typestate specifications. It is implemented using the ANTLR v4.5 framework. After the translation, we use the Mungo tool to statically typecheck protocol implementation. We start from a multiparty protocol defined as Scribble global protocol and then use the Scribble tools to validate and project into local protocols describing the interactions from the view-point of each of the participants. For every Scribble local protocol, StMungo produces .mungo files containing:

- a typestate specification describing the local protocol as a sequence of method calls;
- an API for the participant implementing the methods in the typestate;
- a main() method skeleton calling the methods in the typestate.

Mungo (https://bitbucket.org/abcd-glasgow/mungo/src/master) is a Java front-end tool used to statically check the order of method calls. It is implemented using the JastAdd framework. A protocol definition is described as a sequence of method calls, the order of which determines the validity of the object protocol. A typestate file containing this protocol definition is defined and associated to a class. The Mungo tool checks that the object instantiating the class performs method calls by following its declared typestate. If the object respects the typestate, then .java files are produced for every .mungo file in the program. Finally, the Java tools can be used to compile and run the standard Java code. If the typestate is violated, Mungo reports the errors.
Using the StMungo and Mungo toolchain, UGLA developed a substantial real-world case study: the toolchain was used to typecheck a client for the standard Simple Mail Transfer Protocol (SMTP) able to communicate with widely-deployed servers; specifically for this case study, the gmail server (details can be found in the recording in http://www.dcs.gla.ac.uk/research/mungo and journal paper http://www.dcs.gla.ac.uk/~ornela/publications/KDPG17.pdf).

Open Case Manager (DCR; https://github.com/DCRGraphsNet/DCROpenCaseManager) is a simple web application that can instantiate DCR graphs as a task list, as well as execute robotic events. The application uses a Microsoft IIS and SQL infrastructure and can be hosted in Azure. DCR Open Case

Manager is available on Github under a AGPLv3 open source license. The tool can import and execute DCR graphs. An activity in a DCR graph is typically presented as a task in a task list and a DCR graph is instantiated as a process instance, or a case.
The tool provides an execution engine for a behavioural specification (DCR graphs), with a web interface where process related information (e.g. roles/data) can be executed/simulated/stored . A limitation is that only one input format (DCR graph) is supported.

OpenDXL (MCF; https://www.opendxl.com/) is an open API to enable devices to share intelligence and orchestrate security operations in real-time. This platform provides a unified framework for hundreds of products, services, and partners. Any organization can improve its security posture and minimize operational costs through the platform's capabilities. The platform leverages a real-time data exchange framework to build collective threat intelligence to make endpoint, network, and cloud countermeasures protect and detect as one.
The platform could offer a solid ground to enhance security of APIs developed with other artefacts such as APIgator, WebApi, or Autogen. The monitoring approaches supported by ChorGram and DetectEr may provide further enforcement of security policies.

Process Highlighter (DCR) is a tool that supports translations between natural language descriptions and declarative process models. The resulting models are given in terms of DCR Graphs. Traceability is at the core of the tool: Later changes in the process model due to, e.g., ambiguity resolution are traced back into the text. This allows users to either correct and complete their descriptions, or to derive models more refined than the text. A key feature is the possibility to extract behavioural specifications (DCR graphs) from natural language specifications, such as laws, guidelines, interviews, etc. Partial automation is provided via NLP modules. Traceability of changes in the formal specification are identifiable.
Limitations of the current version are (1) only modules for English and Danish languages have been implemented and (2) that only one output format is supported (DCR graphs), it could be interesting to apply it to other (e.g. choreographies).
SCSL (IXR, Currently proprietary to IXR and not publicly available) stands for "Service Contract Specification Language. SCSL is a human- and machine-readable language for the definition of a RESTful (Representational State Transfer) application programming interface (API) heavily inspired by RAML (RESTful API Modeling Language) and OpenAPI. SCSL is designed to improve the specification of the API by providing a format that the API provider and API consumers can use as a mutual contract. SCSL can, for example, facilitate providing user documentation and source code stubs for client and server implementations. Such provisions streamline and enhance the definition and development of interoperable applications that utilize RESTful APIs.

UsInfer (NOVA; http://usinfer.sourceforge.net) is an academic prototype in beta phase. Usinfer is a prototype for behaviour type inference approach that, from a program written in a variation of Mool

equipped with assertions, generates the usage types necessary for the program to have its behaviour statically checked by a type system. The tool starts by generating a permissive and nondeterministic state machine representing a typestate, based on the assertions on the code. The tool then translates the generated typestates into usages. In the end, it defines the usage state that each object of the class starts with by using the assertions and the usages obtained in the previous stage. Programs need to be manually annotated with assertions.

WebApi (XIB; https://github.com/RestCode/WebApiProxy) is an industrial tool released and in production at XIB. WebApi is integrated into Microsoft development environments via Nuget. WebAPI comprises of a server-side extension that provides a proxy end-point for serving service metadata & a client-side proxy in JavaScript and also a client-side task that generates a client proxy in C#. The tool is quick to use, easy to update, can be used by developers with no knowledge of the server side APIs. WebAPI requires the use of C# client side. A few concepts of WebAPI could be reused, in particular the nuget based integration and generation, but little else as it's heavily tied to the WSDL format

3. Activities within WP4

The tasks conducted so far within WP4 through secondments and remote interactions are progressing well and in line with the original plans. The information compiled in this deliverable allowed us to highlight the strengths of the consortium in terms of tool support for the activities of WP2, WP3, and WP5. Moreover, this information allowed us to identify areas of development and sharpen the activities of the planned collaborations. This information has been the basis to carry out a few activities in the project that developed new tool support and methodologies for the engineering of b-APIs as detailed in D4.3.

Overall, the analysis of the information collated above shows that there is a good degree of support for top-down development of b-APIs. The academic prototypes are very much concerned with the specification and verification of behavioural aspects of API applications. This complements pretty well industrial artefacts which focus on other software development practices (eg., quick coding). The information on the artefacts in this deliverable can help to overcome the obstacles mentioned in D.2.1 (cf. sections 3.2.2-3 and 4) about the lack of precise documentation of APIs. The models featured in many of the academic artefacts (DetectEr, UsInfer, FABIoT, to mention but a few) constitute a solid base to start interacting with academic partners. In this respect, UBA and ULEIC have initiated some collaborations with MCF, XIB, and ACT to enable modelling/analysis of industrial case studies and their automated testing. These activities were carried out during several secondments (declaration numbers 4, 8, 10, 12).

Mungo (developed at UGLA) ensures well-typed programs follow the declared class protocols. The demanding part for programmers is to specify those protocols as type terms, called typestates. Since it is actually more intuitive to define the protocols as automata, NOVA team developed a notion of object automaton and a tool to convert them to typestates (and back). We are accessing the tool with a

suite of Mungo exercises and accessing the increasing of productivity when using Mungo with use-cases that are real APIs. An ongoing collaboration with McAfee shall provide further elements on the usability and productivity gains of the integration of Mungo with the NOVA approach and tool.
Besides an important ground for collaborations within the consortium, artefacts are also paramount for dissemination (WP5). ULEIC presented the BehAPI project at ICE 2018 (http://2018.discotec.org/pdf/program_ICE.pdf) through a demonstration of ChorGram in the context of b-APIs. A similar presentation has been given in Rio Cuarto during a secondment from ULEiC to UBA (declaration number 14) in occasion of the International Symposium on the Mathematical Foundations of Software Engineering in honor of T.S.E. Maibaum (https://dc.exa.unrc.edu.ar/rio/es/InternationalSymposium). During this secondment UBA and ULEIC also organised tool-based dissemination activities of BehAPI to software companies in Argentina. UOM presented the underlying mechanisms underpinning a tool for the runtime enforcement of properties at a workshop discussing Open Problems in Concurrency Theory (https://popl19.sigplan.org/track/opct-2019-papers#program).
Relatedly, the first school that was held in Leicester from 8 to 12 July 2019 reflects the richness of the tool support within the consortium both within the courses and the associated boot camp.
We remark that activities towards the goals of WP4 happened also outside secondments. In particular, decisions for organising schools required a few meetings that were carried out with online conferences, emails, and preparation of shared documents. The school committee has actively sought ways to embed tool support within (and outside) the consortium in the design of the school.

4. Conclusions

This report surveyed the flora of artefacts of BehAPI. The information collated here has been analysed in D4.2 in the light of the support the consortium seeks to provide to WP2 and WP3 for the provision and consumption of b-APIs as well as to WP5 for the dissemination activities.
This report confirms the initial belief that artefacts are a suitable vehicle to enable collaborations among partners (especially academic and industrial partners) and for the dissemination of our results. Also, the activities carried out by UBA and ULEIC on ChorGram have generated some extra funding (the presentation held in Rio Cuarto (cf. Section 3) and the collaboration with MCF and companies in Buenos Aires outside the consortium has been fostered and supported by the KEEF scheme kindly awarded by the University of Leicester, besides the fundings of BehAPI.
The state-of-the-art summarised in this document has not been published elsewhere. However, we expect some of the conclusion of this activity to be communicated as part of a planned activities of WP4 and WP5. In fact, some of the artefacts described in this document will be the basis for courses and boot-camps in the schools (see https://www.um.edu.mt/projects/behapi/summer-school-in-leicester).

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

To sum up: T.4.1 has been fully achieved. The activities on T.4.2 and T.4.3 started in the year 2 and are ongoing.
For T.4.2 ULEIC, UBA, and MCF have designed a methodology for the specification of components in the OpenDXL framework. The methodology supports the rigorous development of components to avoid miscommunications. The artefacts used for such specifications can be turned into monitors to control the behaviour of non compliant components. Initial work is being conducted on model-driven testing based on choreographic models. There is no tool support for this yet as a body of theoretical results need to be achieved first. We expect to have some relevant tool support of choreography-based testing by M40. Also, UBA and ULEIC started to develop a framework to automatically generate executable tests for mobile applications. This framework is currently under development and implements static and dynamic analysis techniques for Android mobile applications. Using a search-based test data generation, the framework generates executable Espresso tests. A dataset of existing Android applications has been collected to test the developed prototype on them.
With respect to T.4.3 several lines of work have started. The integration of the actor model in ChorGram has begun with new features that allow the tool to generate Erlang executables and monitors for Erlang programs.
UBA and ULEIC defined a learning algorithm to create models from exploration of b-APIs available in the execution environment. This will foster interoperability in terms of the API compositions discovered with the learning algorithm.
DetectEr …
UNIBO and UKENT developed a asynchronous behavioural sub-typing discipline for message-passing systems and implemented a prototype tool for checking asynchronous sub-typing. This extends the applicability of BehAPI approaches as it allows for a more refined typing.

We will report on the progress of the activities on T.4.2 and T.4.3 in the subsequent revisions of this deliverable due in M36.