Project no.:	H2020-MSCA-RISE-2017-778233
Project full title:	Behavioural Application Program Interfaces
Project Acronym:	BEHAPI
Deliverable no.:	D3.2 (M24)
Title of Deliverable:	Report on advances on model-driven verification of composite b-APIs and on static and dynamic verification (runtime monitoring and testing) techniques for b-APIs consumers
Work package:	WP3 (T3.2 T3.3 T3.4 T3.5)
Type:	R
Lead Beneficiary:	**UNIVERSITY OF KENT (BEN4, UKENT)**
Dissemination Level:	PU
Number of pages:	

Contract Delivery due date: 29/02/2020 (M24)
Actual delivery date of version n.1: 28/02/2020 (M24)

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 the advances of the BehAPI consortium on some of the themes of Work Package 3 (WP3 - API Consumption), that is techniques for static and dynamic verification of b-APIs consumers especially in the context of a model-driven approach. The deliverable has been submitted at the end of year 2 of the BehAPI project and describes activities, secondments, outcomes, and plans at the time in which was it written. We expect all activities discussed in this deliverable to continue after the end of year 2 and to be reported in a subsequent deliverable due in year 3.

The table below reports on the BehAPI secondments that were planned on WP3 and that have been (totally or partially) performed at the time of writing this deliverable. In the table: the Research Declaration **number** in Sygma is the declaration number for each secondment as declared in Sygma (the Project Continuous Report system); the **plan number** is the secondment number according to the plan in the Latest Legal Data (on Sygma) of the BehAPI project; the **start month** is the actual starting month of each secondment according to the declaration on Sygma.

Researcher	**Fellow ID**	**Category**	**Sending partner**	**Receiving partner**	**RD number in Sygma**	**Plan number**	**Start Month**	**Duration (PMs) actual / plan**	**Tasks**
Lange	7	ER	UKENT	GBW	7	107	4	1 / 1	T3.6 (1PM)
Attard	24	ECR	UMA	XIB	26	16	4	0.53 / 1	T3.5 (1PM)
Francalanza	9	ER	UMA	CMU	11	9	5	3.83 / 4	T3.5 (1PM)
Melgratti	3	ER	UBA	ULEIC	8	81	7	1/1	T3.2(0.5PM), T.3.4(0.5PM)
Arcuschin	10	ECR	UBA	ULEIC	12	80	10	1/1	T3.3 (1PM)
de'Liguoro	13	ER	UNITO	UBA	15	108	11	1.07/1	T3.2 (1PM)
Gabervetsky	17	ER	UBA	NOVA	20	75	14	1.07/1	T3.4 (1PM)
Nahabedian	19	ECR	UBA	ITU	22	74	15	1.03/1	T3.7 (1PM)
Bocchi	32	ER	UKENT	BTL	33	101	16	1.13 / 1	T3.3 (0.8PM), T3.5 (0.2PM)
Laneve	29	ER	UNIBO	IXR	30	121	17	0.53/1	T3.4 (1PM)
Rojas Siles	33	ER	ULEIC	UBA	34	53	17	1.1/1	T3.3 (1PM)
Hernandez-Castro	37	ER	UKENT	IXR	39	102	21	1 / 1	T3.5 (1PM)
Tuosto	11	ER	ULEIC	IXR	38	51	21	0.4/1	T3.2 (0.5PM), T3.4 (0.5PM)
Costa Seco	23	ER	NOVA	DCR	25	31	15	0.53/1	T3.4 (1PM)

In the rest of this document, we will discuss the link between each of the above secondments and the tasks reported on in this deliverable.

The following table gives information on supporting secondments on other Work Packages

that also contributed to WP3. No time in supporting secondment was used exclusively to work on WP3. By supporitng secondment we intend secondments where work was done on the planned WP, but for which the work done also benetifs WP3 tasks. The column **Work Package** states in which other BehAPI work package each secondment was planned and reported.

Researcher	**Sending partner**	**Receiving partner**	**Sygma number**	**Work Package**
M. Carbone	ITU	CMU	9	WP2
H. Melgratti	UBA	ACT	18	WP4
A. L. Laura Voinea	UGLA	GBW	21	WP4
O. Dardha	UGLA	BTL	31	WP4
S. Gay	UGLA	GBW	35	WP2
H. Huttel	AAU	IXR	41	WP2

Table of contents

1. Introduction

This deliverable describes the advances of the BehAPI consortium on techniques for static and dynamic verification of b-APIs consumers, especially in the context of a model-driven approach. Models are powerful tools for rigorous reasoning on system's behavior and properties, as well as for supporting system's engineering. Model-driven development is a methodology centred on exploiting models. The engineering of *composite* distributed systems, which is the focal matter of WP3, requires suitable models for representing the interrelationships between different processes. In the BehAPI context, as we are concerned on structured interactions between distributed processes (producers and consumers), these models are *behavioural* APIs (b-API).

Formal models for binary interaction. One of the most notable approaches to model interaction behaviour is based on *session types* [Honda, Vasconcelos, Kubo, 1998], [Takeuchi, Honda, Kubo, 1994]. Session types model the communication behaviour between a client and a server in "units of interaction" called sessions. When sessions involve pairs of participants (e.g., client and server) we call session types *binary*. A session type can be seen as the type of a communication channel, describing which types of messages can be exchanged on that channel (e.g., integers, strings, other channels), and their causal relationship.

Multiparty interactions: the top-down approach. Binary session types have been extended to model sessions of more than two participants (aka multiparty sessions) [Honda, Yoshida, Carbone, 2016]. Multiparty session types are central for WP3 as they support coordination and well-disciplined interactions among a set of API producers and consumers. Multiparty session types advocate a *top-down* approach to system engineering: (1) the designer specifies the overall communication behaviour of a multiparty session as a *global model*, (2) the global model guarantees (by construction or well-formedness conditions) global properties such as safety and deadlock-freedom, (3) a mechanism of projection from global models to local models provides a blue-print to implement and verify each participant in the session in a modular way. Static and dynamic verification of the vendor's code against local models ensures, for example, that the overall system conforms to the global protocol, and enjoys other properties such as communication safety and global progress.

The bottom-up approach. The top-down approach requires developers to build their application starting from pre-defined global models. This fact may limit the flexibility of the organization building producer or consumer applications. Moreover, the top-down approach is not easy to apply when organizations want to reuse already-existing code (e.g., legacy applications, well-established e-commerce services). In the last few years, approaches mixing top-down and bottom-up composition have been studied in session types [Deniélou, Yoshida, 2013][Montesi, Yoshida, 2013][Lange, Tuosto, 2012][Bartoletti et al., 2015] and related automata-based formalisms [Lange, Tuosto, Yoshida, 2015]. For a more comprehensive overview of modelling languages for b-APIs, the interested reader may refer to BehAPI deliverable D2.2[1].

1.1 Links between D3.2 and project objectives

The activities reported in this deliverable are core to meet Objective O3.2 "Enhancing software development lifecycle for API consumption", which is is recalled below:

"Extend current development practices of b-APIs so to ensure their smooth integration, and support their evolution. This will involve enriching top-down (from models to code) and bottom-up (from code, including legacy systems, to models) approaches to software construction and exploiting their synergy."

The model-driven approach, on which this deliverable centres, serves us as an overall framework to integrate different techniques for supporting the development lifecycle of API consumption: test generation (T3.3), static verification (T3.4) and runtime monitoring (T3.5). The fact that all activities in T3.3, T3.4, T3.5 are based on models, and specifically b-APIs, provides cohesion and facilitates the integration and transition between the phases within the lifecycle. These techniques are naturally exploitable in the top-down approach. The bottom-up approach is instead central to T3.6. T3.6 was not due to be reported in this deliverable. However, as its activities have no dependencies with activities of the other tasks and have already started, and since the topic is relevant to this deliverable as it serves to "complete

[1] BehAPI Deliverable D.2.2. Available at https://www.um.edu.mt/projects/behapi/wp-content/uploads/2019/02/Behapi-Deliverable-2.2.pdf

the loop" in a top-down and bottom-up development approach, we have included a discussion on T3.6 activities to this deliverable.

The other objectives involved in the activities reported here are O3.3 "Providing correctness and quality assurance in API consumption", and O3.4 "Inferring composite b-APIs from APIs". Objective 3.3 is recalled below:

"Providing correctness and quality assurance in API consumption. Provide techniques to reason about, validate and carry out integration testing of applications built up from b-APIs, with focus on correctness and quality (e.g., reliability, scalability, etc.)."

One of the advantages of relying on formal models within a model-driven framework, is that it supports reasoning and verification, targeted at ensuring correctness and quality. O3.3 is addressed by Tasks 3.3 (testing), T3.4 (static verification) and T3.5 (dynamic verification). As expected, correctness has been addressed earlier than quality in the project. However, some activities have already started considering quality, e.g., security aspects and data-related properties.

Objective 3.4 is recalled below:

"Provide techniques to extract composite b-APIs from code (source, executable, or execution logs) that implements multiple APIs. This objective will extract rigorous behavioural information from code implementing simple APIs providing only syntactic information (e.g., specifying only the set of valid actions, but not their causal relationship)."

This objective is central to facilitate the bottom-up approach to API development. Techniques to facilitate the bottom-up approach are developed in T3.6. The objective is also facilitated by activities in T3.2 which focusing on synthesis in a model-driven context may also support bottom-up transformations.

1.2 Synopsys

This deliverable is then structured according to the tasks it reports on: Section 2 will discuss activities on Task 3.2, Section 3 on Task 3.3, Section 4 on Task 3.4, Section 5 on Task 3.5, and Section 6 on Task 3.6. Activities on Task 3.7 are will be discussed in BehAPI deliverable D3.3 due in M48. In the report of the planned

tasks, we have omitted the partners that have left the consortium and for which there are no activities to report.

2. Model-Driven b-API development: the top-down approach

In this section we will report on the activities linked to Task 3.2: "Model-driven verification of composite b-API". The activities planned for this task mainly focus on the top-down approach, providing techniques to transform abstract b-API models into more concrete b-API models (i.e., closer to a final implementation), and synthesising b-API *compound* implementations from b-API models. In Section 2.1 we report on theoretical foundations for identifying subtyping relations between b-API models expressed as session types. When a session type A is a subtype of a session type B, an implementation of A can be used in a system that "behaves well" with an implementation of B. On of the advantages of subtyping is that it gives b-API consumers more flexibility in setting the behaviour of their implementations (i.e., they can implement a subtype), with the guarantee that the system will still behave as expected. In Section 2.2 we report on an infrastructure for (top-down) model-driven development of composite b-APIs, modelled as data-driven choreographies expressed with a graphic notation. The infrastructure is relevant for T3.2 as it shows how the top-down approach, with its notion of projection, supports development of systems that follow a planned global behaviour (i.e., a choreography) while being implemented by multiple b-API producers and consumers. In Section 2.3 we give a summary report of ongoing activities in the consortium linked to Task 3.2. The evaluation of the status and progress of T3.2 will be discussed in Section 8.

2.1 Towards synthesis via subtyping

Before being synthesized into code, models may be transformed into less abstract models, that are closer to final implementations. Relations between models at different levels of abstraction are usually expressed as refinements. These can be used to concretize abstract models into implementations. In the case of asynchronous interactions, which is most common for Web APIs, refinement has been investigated under the name of subtyping between session types [Bartoletti et al., 2014][Chen et al., 2014][Demangeon, Honda, 2011][Mostrous, Yoshida, 2009],

[Mostrous et al., 2009]. Unfortunately, recent work has shown that subtyping for session types in an asynchronous setting is undecidable [Bravetti et al., 2017][Lange, Yoshida, 2017]. This negative result can be mitigated by restricting the syntax of session types or limit communication (by considering forms of bounded asynchrony).

UKENT, **UNIBO**, and **ITU** (in team with Imperial College London, which is external to the consortium) have recently presented [Bravetti et al., 2019] (BehAPI **publication no. 28** in Sygma) an alternative, less restrictive approach to checking subtyping for asynchronous multiparty session types. Unlike previous work, the approach they propose requires no additional restriction on the syntax of session types or on the channel size. Instead, they provided an algorithm for checking subtyping which is sound, but not complete (i.e., 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 team also provided a tool that implements the algorithm, and applied it to many examples that cannot be managed with other approaches that require syntax or channel size restrictions. This result is a shared outcome with WP2, and it pertains to WP3 (in particular Task 3.2) as it facilitates modular subtyping (hence implementation) of composite APIs still preserving the original overall behaviour.

2.2 Model-Driven Development of Data-Driven Choreographies

UBA, **ULEIC** and **MCF** (in team with the University of Pisa, which is external to the consortium) developed a model-driven approach based on a formal data-driven choreographies [Bruni et al., 2019] (BehAPI **publication no. 21** in Sygma). Their target scenario was provided by **MCF**: the choreographies have been used to model the Threat Intelligence Exchange (TIE) services that McAfee provides through the Open source Data Export Layer (ODXL) industrial platform. The team advocates a methodology consisting of a chain of model transformations that (i) devises a visual presentation of communication protocols, (ii) derives a global model from the visual presentation that rigorously describe the interactions required by services, (iii) enables the automatic derivation of models of single components, and (iv) enables the analysis of software implementations. Global models (steps (i) and (ii) of the

chain) are described through *global graphs and data-driven choreographies*. Data-driven choreographies are key to identify the main aspects of the protocol as well as to ease the collaboration between practitioners and academics. However, global graphs can hardly capture the execution model of ODXL. The team has, therefore, formalised TIE with *klaimographies*, a data-driven model of choreographies. Klaimographies allow one to automatically derive local specifications (step (iii)) and use them as precise blue-prints of components as well as to automatically derive monitors (step (iv)). The automated transformation of global models to local ones provides a powerful mechanism to ensure that the composition of the implementations by b-API producers and consumers behaves as expected. One remarkable outcome of this collaboration is the development of a suitable abstraction to present rich a graphical b-API modelling formalism that is usable by both practitioners and academics. Moreover, the local specifications are used not only to produce blue-prints of b-API implementations, but also to analyze implementations and to derive monitors.

2.3 Ongoing activities on b-API synthesis

The BehAPI plan includes two sub-tasks for T3.2. The first sub-task is:

"UKENT, UNITO, and ITU will address the problem of synthesising global descriptions from infinite state distributed implementations."

This task has been partially achieved, and the progress is in line with the plan. We expect to complete this task by M36. **UKENT**, **UNIBO** and **ITU** provided an algorithm for subtyping, described in Section 2.1. (BehAPI **publication no. 28** in Sygma). This is a shared output with WP2. **UKENT** is also working on a tool for producing implementations in the Go programming language given as Timed Automata.

UNITO has collaborated with **UBA** (secondment **declaration 15**) and **ULEIC** on a notion of refinement in the context of choreographies. The team has defined a choreographic formalism that includes *underspecified actions*. The purpose is to support the incremental development of choreographies, through a precise modeling of the conditions that must be satisfied for guaranteeing safety and deadlock freeness properties of compliant processes. Choreographies with underspecified actions have then been interpreted into event structures, extending

previous work based on pomsets (and allowing leverage on the tools and techniques known for pomsets). There is no paper published yet, although one is in the works. The research is also related to Task 3.6 and Task 3.7 (if adaptation is interpreted as refinement of protocols to cope with particular contexts).

The second sub-task is:

"UBA, ULEIC, and AAU will adapt algorithms for synthesising choreographies that take into account rely-guarantee contracts and use synthesis techniques for refinement of specifications from composite models."

This task has been partially achieved, and is due to be completed in M36. The progress is in line with the plan. **UBA** and **ULEIC** have proposed a top-down approach for model-driven development for data-driven choreographies, described in Section 2.2. (BehAPI **publication no. 21** in Sygma). The collaboration was partially carried on in two secondments (secondment **declarations 8** and **38**). One of the steps supported by the approach is the production of blue-prints of implementations. A secondment from **ULEIC** to **GBW** planned in February 2020, where the partners will consider possible projections from global views of choreographies to the RUST programming language. **ULEIC** is also developing a projection from global graphs to executable Erlang components base on the work in [Francalanza et al., 2018].

AAU worked on this task during a secondment to **IXR** (supporting secondment **declaration 41** - WP2). In this secondment, two main topics were addressed, that are related to b-API usage/consumption: (1) the overhead of working with behavioural types in practice, and (2) a presentation of previous theoretical work as a way to lessen this overhead. Working with behavioural types requires the programmer to annotate programs with much larger and/or more explicit types. The team discussed a language where classes in an object oriented programming language are annotated with the usage-protocol. They then proposed a way to lessen this overhead, by presenting an inference system that could automatically infer some of these usage-protocols, and discussed whether that could make the specification of behavioural properties of the programs practical, for real-world software development projects. This work pertains Task 3.2 as it relates to specifications refinement and also to Task 3.6 as it allows inference of b-API models. **AAU** is currently working on usage inference for Object-Oriented implementation.

3. b-API Test Generation

In this section we will report on the activities linked to Task 3.3: "Automatic test generation from b-APIs compositions". In the testing of composite applications, the `Big Bang' approach, which consists of testing all the single components in isolation and then do integration testing of the whole system in once, is a notoriously high risk approach. Moreover, in a composite scenario involving b-API producers and consumers, the testing of the complete composite system may be unpractical (e.g., components may be developed by different organizations) or not possible (consumer components are developed at a later stage). On the other hand, running tests on partially integrated components requires the development of suitable test harnesses. Manual development of test harnesses is expensive. The goal of Task 3.3 is to exploit the information contained in b-API models to automatically generate tests and test harnesses. In principle, this allows producers to provide their consumers with support on how to test their consumer applications. In Section 3.1 we report on a testing framework for Android User Interface (UI). In Section 3.2 we will report on all ongoing activities and progress of the consortium on Task 3.3.

3.1 A Testng Framework for Android UI

Before the beginning of BehAPI, **UBA** has been working on a testing framework for writing Android UI tests, called Espresso. In a nutshell, Espresso[2] provides an API which lets developers emulate user interactions with the app under test programmatically. The tool was originally released in October 2013, and since its 2.0 release it is part of the Android Support Repository and officially supported by the Android ecosystem. This, coupled with the fact that it enables developers to write concise and reliable UI tests, has made the framework very popular among developers. Thanks to two **secondments** from **UBA** to **ULEIC** (secondment **declarations 34** and **12**), the team developed a tool named Espresso Test Generator (ETG). ETG automatically generates reproducible and human-readable Espresso test suites from sequences of interactions over widgets. The technique devised is generalizable to different widget-based testing tools. In particular, the participants showed how ETG can be combined with an extension of the MATE testing tool to automatically generate an Espresso test suite. To the best of the our

[2] The Espresso Testing Framework. Available at https://developer.android.com/training/testing/ui-testing/espresso-testing

knowledge, this is the first tool aimed at automatically generating fully executable Espresso test cases. A tool paper was submitted to the 42nd International Conference on Software Engineering (ICSE 2020) with the following title: "ETG: Creating Espresso tests from sequences of widget actions". The submission is under review, and the partners are awaiting a decision.

3.2 Ongoing activities on test generation

In this section we report on the advances made at exploiting the information contained in b-API compositions to automatically generate test harnesses and tests. The concrete task in the plan was the following:

"GBW will collaborate with UKENT, XIB, ULEIC, UBA and MCF on the mechanical generation of mocks and stubs from suitable API descriptions."

This task has been partially achieved, and will be fully finished by M36. The progress is in line with the plan.

ULEIC and **UBA** made remarkable progress on this task in two secondments. Thanks to two **secondments** from **UBA** to **ULEIC** (secondment **declarations 34** and **12**), as discussed in Section 3.1. They are awaiting for a decision on their article submission to a peer-reviewed conference and may continue working on this tasks in year 3.

During a secondment from **UKENT** to **BTL** (secondment **declaration 33**), also involving collaboration with **UNIBO**, the partners started working on a theory for reversible computation to support debugging in time-sensitive scenarios. This activity is not strictly on test generation but will nevertheless contribute to support testing and debugging of composite real-time APIs. More work is planned in year 3.

A different team from **UKENT** is currently working on mechanical generation of mocks and stubs for suitable API descriptions. The aim is to develope new techniques for automatically generating programs based on linear types in the context of the research language Granule[3]. Granule novelly combines dependent types and linear types with the new notion of "graded modal types" which provide a way to capture fine-grained quantitative information about data use. Graded

[3] The Granule Project. Available at https://granule-project.github.io/granule.html

modalities complement the guarantees provided by linear and dependent types. Granule supports session types via the aforementioned idea of embedding session types into linear types, but with added abilities to reason quantitatively about communication protocols via the combination of dependent and graded modal types. As part of this project, the team is looking at automatically generating (synthesising) program stubs from these quantitative session type protocols. This leverages a body of work on program synthesis and automated proof search for linear types, extending it to the new notions of graded modal types and novelly applying it to API descriptions. A supporting secondment (plan number 97 - WP2) is planned from a member of this team at **UKENT** to **DCR** in August 2020. In this occasion the project partners will explore mechanical generation of mocks and stubs from suitable API descriptions. The generated mocks and stubs are usable by and would benefit both b-API producers and b-API consumers.

AAU is also working on usage inference for OO programs and testbed implementation (a forthcoming secondment is expected in year 3 by an Early Stage Researcher).

All the partners not explicitly mentioned plan to work on this task in Year 3.

4. Static verification of b-API consumers

Task 3.4 has the purpose of supporting engineering of code that consumes existing b-API with guarantees that the code: (1) conforms to composite b-API specifications, (2) is communication sound, in the sense that each process engaging in an interaction will only receive expected messages, and (3) satisfies given reliability and quality requirements. The main approach followed by the b-API partners is that of developing techniques based on behavioural typing, global graphs and/or klaimographies (reported in Section 2.2), or other automata-based formalisms. In Sections 4.1 we will report on BehAPI progress on theories for static verification of b-API aimed at increasing the flexibility of verification. In Section 4.2 we will report on progress on b-API static verification in a concrete usage scenario: Blockchains. In Section 4.3 we will provide a summary of the ongoing activities and progress of the consortium on Task 3.4.

4.1 Flexible composition verification with capabilities

Enforcing the compliance of implementations to API usage protocols is notoriously hard due to possible *aliasing* of objects through multiple references [Bierhoff, Aldrich, 2008]. This is particularly hard when API usage protocols are *behavioural*.

Example 1 (Aliasing with two API consumers). Consider an b-API producer P interacting simultaneously with two b-API consumers A and B. If A and B both run the client side of a protocol with the same produced P, using the same endpoint (aliasing), then a message sent by A may advances the session state of S without B's knowledge, and interferes with B's attempt to run the protocol and/or causing race conditions.

Aliasing is a problem in distributed b-API implementations, which session types address in two ways: linear and affine typing. In order to guarantee unique ownership of channel endpoints and eliminate aliasing. Most session type systems use strict *linear typing* requiring that each channel is used once. For more flexibility, some others use *affine typing*, which allow values to be used once or not at all (but not more than once). Both linear and affine typing forbid aliasing, hence disallowing *any* possible instance of Example 1, even in cases in which e.g., A and B actually take turns in the use of the endpoint and do not cause race-conditions.

The **UGLA** team addressed the fundamental question on whether session types are intrinsically related to linearity or affinity. In [Voinea et al., 2019] (BehAPI first occurring **publication no. 27** in Sygma) the authors have shown that session types can be extended to allow aliasing in some safe scenarios, still providing the usual guarantees of session fidelity and session safety. Their proposal is based on the use of *capabilities*, which allow to separate resources from their usage. Session types with capabilities allow more flexible typing as well as more flexible composition between b-APIs, with respect to linear and affine typing, and still ensure the same type safety properties. In Example 1, the use of capabilities may allow aliasing if the usage of the shared endpoint is well-disciplined. Concretely, one would associate the shared endpoint with a tracked type (instead of a session type) and the provider P with a capability. The capability can be passed between A and B, so that they can alternate in their use of the end-point without causing race-conditions. The paper also features a producer-consumer case study that shows the link between b-API production and consumption, and resource sharing with the use of capabilities.

4.2 b-API Consumption in a Blockchain

A Blockchain is an open, distributed infrastructure that can record transactions between two parties in a verifiable and permanent way. Smart contracts are programs stored on blockchains that control the transfer of assets between users under certain conditions. Smart contracts are a hot topics, as they implement *decentralized applications*, which can handle/transfer assets of considerable value in the form of cryptocurrency like Bitcoin. Smart contracts are *behavioural* entities*:* they are programs, written in programming languages that are targeted to different blockchains. Two such languages are Solidity[4] for Ethereum (which is imperative) and Liquidity[5] for Tezos (which is functional). Solidity is the most common language used to develop smart contracts, and there is a growing literature on its formalization and analysis. For instance, recent work proposing a minimal calcului for Solidity contracts [Bartoletti at al. 2019] aimed at allowing simple rigorous reasoning on smart contracts.

UNIBO has been working on Solidity smart-contracts, towards supporting techniques to analyze the interaction between smart-contract providers and consumers. In their published work [Laneve et al., 2019] (BehAPI **publication no. 29** in Sygma) [Laneve, Sacerdoti, 2020] (to appear) the authors analyze the assets movements of Solidity smart contracts functions by means of *behavioural types*. To this aim they define a subset of the Solidity language. In [Laneve et al., 2019] the authors analyze the behaviour of smart contracts and the interaction with external actors in order to maximize objective functions (which allow to maximize the users revenue in a transaction). The author therefore express the system behaviour as a first logic formula in Presburger arithmetics and study the maximum profit for each actor by solving arithmetic constraints. In [Laneve, Sacerdoti, 2020], the authors associate cost equations to functions to smart contracts. This association is defined by means of a type system. Cryptocurrency movements are computed by feeding the equations to an off-the-shelf cost analyzer.

[4] Ethereum Foundation. Solidity 0.4.24 documentation (2019). https://solidity. readthedocs.io/en/develop/

[5] OCamlPro SAS. Welcome to Liquidity's documentation! (2019). http://www. liquidity-lang.org/doc/

4.3 Ongoing activities on static verification

In this section we report on the advances on static verification of b-API compositions. The plan includes four sub-tasks. The first sub-tasks of T3.4 is the following:

"UNIBO will investigate with UGLA such techniques for the development of applications that build on one or more APIs."

The progress of **UNIBO** towards the completion of this task have been reported in Section 4.3. The work was partially carried on in a secondment (secondment **declaration 30**) and produced one outcome paper (BehAPI **publication no. 29** in Sygma) (another one is to appear). The outcome is in the context of smart-contracts.

Another partner is working on smart-contracts. **UBA** worked on static analysis of smart-contracts based on behavioural descriptions. During a secondment to **NOVA** (secondment **declaration 20**) the partners studies the state of the art and the practice of the area, including works on BitCoin, Tezos, Zilliqa networks. They focused on: i) mechanisms to formalize low level languages and virtual machines executing in the blockchain environment, ii) formalizations of consensus protocols on the blockchain ecosystems, iii) high level languages to write smart contracts and they formal underpinnings, iv) static analysis technique to enforce security properties. The partners analyzed the advances in those areas and the current limitations and challenges. The next steps are the introduction of session types to a core language and formalize its semantics in a machine provable language like Coq.

The progress of the **UGLA** team has been reported in Section 4.1. **UGLA** did not have secondments planned for WP3. However, during secondments on other Work Packages, the Glasgow team has produced results that are relevant to T3.4 (BehAPI first occurring **publication no. 27** in Sygma). The activity has been facilitated by a number of supporting secondments: two to **GBW** (supporting secondment **declarations 21-WP4** and **35** WP2) which also involved collaboration with **NOVA**; and one to **BTL** (supporting secondment **declaration 31** - WP4) which also involved collaboration with **UNIBO**. The work relates to Task 3.4 as it considers Multiparty Session Types, which abstract the interactions of two or more b-API as a composite system. The current progress on this sub-task is satisfactory. **UNIBO** and **UGLA** may continue working on this in year 3.

NOVA contributed to this sub-task thanks to one secondment (secondment **declaration 25** – WP4) to **DCR**. The secondment activities concerned the development of a new connection between databases and business processes declarative languages. Notably, it focused on the static properties of a generic mechanism to orthogonally define the structure of data (or schema) and control flow of a system. This model is being used as the base for the **DCR** product that now incorporates persistent data fields in the definition of business processes. The work is still ongoing as the secondment is still to be completed.

The second sub-tasks of T3.4 is the following:

"ULEIC and UIUC will study model checking techniques for directly relating code to composition models."

The activities on these sub-tasks have not started yet. The involved partners plan to perform these in year 3.

The third sub-tasks of T3.4 is the following:

"GBW will collaborate with UBA on the static verification of client-side code against API descriptions."

The model-driven approach developed by **UBA** and other partners (reported in Section 2.2) involves static verification, which is pertinent to this sub-task. Recall, this work was partially carried on in two secondments (secondment **declarations 8** and **38**) and produced one outcome (BehAPI **publication no. 21** in Sygma). Step (iv) of the methodology described in Section 2.2 is, indeed, aimd at enabling the analysis of software implementations. The partners plan to continue working on this in year 3.

The fourth sub-task of T3.4 is the following:

"CMU and ITU will collaborate on the introduction of refinement types to session types and will explore their generalization to multiparty settings."

The work started in November 2018 during a secondment from **ITU** to **CMU** (supporting secondment **declaration 9** - WP2 - declared for WP2 and WP3). No

published outcomes have been reported yet but, during the aforementioned secondment, the partners have established preliminary work that will lead to it. The work will be continued in future visits: **ITU** is planning to visit **CMU** in 2020 (3 secondments of one month) and **CMU** plans a short visit to **ITU** in summer 2020.

5. Dynamic verification of b-API compositions

In this section we will report on the activities linked to Task 3.5: "Dynamic verification of composite b-APIs implementations". Static verification (Task 3.4) in some cases unpractical or not possible, for example when components are untrusted or not statically verified, or when one wants to observe non functional properties such as, e.g., round-trip delay. The aim of Task 3.5 is to develop techniques for dynamic (run-time) verification of composite b-APIs. In Section 5.1 we will report on the ongoing activities and progress of the consortium. The state of progress of this task in Person Months (PMs) is discussed in Section 8.

5.1 Ongoing activities on dynamic verification

In this section we report on the advances on dynamic verification of b-API compositions. Work is already ongoing for most of the activities, but none is completed and we expect more secondments and outcomes to be reported for year 3. The plan includes two sub-tasks.

The first sub-tasks of T3.5 is the following:

"UNITO, UOM, UKENT, ULEIC, IXR, and GBW will study techniques based on gradual/hybrid typing and runtime monitoring for compound, heterogeneous systems where not all components provide statically verified guarantees."

UOM has a forthcoming secondment planned that will be used by a student working on an implementation of lchannels that allows for one side of a (binary) session type to be statically checked and another side to be dynamically checked. It is broadly based on [Bocchi et al., 2017] [Scalas, Yoshida, 2016]. The secondment (going to **XIB**) is planned for March/April2020 but work is already underway. The other partners plans this activity in years 3 and 4.

The second sub-tasks of T3.5 is the following:

"UKENT, UOM, GBW and UIUC, ACT will develop techniques that generate monitors from composite b-APIs."

During a secondment (**declaration 33**) from **UKENT** to **BTL** (involving collaboration with **UNIBO**) the partners discussed the feasibility of an automatic generator of monitors for REST APIs from choreographic specifications. **BTL** developed a tool (before the start of BehAPI) called APIgator[7] that supports the development of API documentations in the standard OpenAPI format. A behavioural extension of a tool as APIgator may include (i) extension of OpenAPI specifications with behavioural information, and (ii) generation of orchestrators (in fact, execution monitors) that check the correct execution of a b-API. A proof-of concept prototype is being developed at **UKENT** for simple choreographic specifications (specified on the basis of "happens-before" and "excludes" constraints). This work will continue in year 3.

During another secondment (**declaration 39**) from **UKENT** to **IXR** (involving collaboration with **UMA**), the partners discussed potential collaboration on b-API and security. The partners focused on protocol design, implementation and testing to hardware random number generation, embedded security, and software fuzzing. The partners specifically focused on b-API fuzzing. In general, fuzzing consists of monitoring the behaviour of a software program in response to a fuzz, that is a massive amounts of random data including invalid or unexpected inputs. b-API can be exploited to refine fuzzing. b-API-driven fuzzing is mostly linked to Task 3.5 due to its monitoring aspect. Potentially, this activity will contribute to Task 3.3 (test generation), and Task 3.7 (adaptive APIs). More work is expected in years 3 and 4.

UOM and **CMU** (secondment **declarations 9**) have been working on the expressivity of (intuitionistic) session types to express monitoring algorithms for deadlock detections. The partners have worked on an implementation over the language Concurrent C0 developed at **CMU** that supports concurrency and session types. The partners produced a preliminary implementation, and the work is still ongoing. During a secondment (secondment **declaration 26** - partially completed, total completion is expected by the end of March 2020) from **UOM** to **Xibis** and involving collaboration with **UKENT**. The secondee from **UOM** worked on a monitoring framework in Erlang to monitor massively parallel systems. These

[7] Freelands. APIgator. Available at https://www.apigator.io/

monitors can be synthesised from session types (in principle) even though they are currently synthesised from Hennessy Milner Logic (HML) formulae with maximal and minimal fixpoints (i.e., recHML). The work is still ongoing.

ACT has been collaborating with **UBA** and **ULEIC** (supporting secondment **declaration 18** - WP4) on a formal description (currently unpublished) of a system from **ACT**, which is a decentralised eventually consistent event log with state machines executed based on the local view of the log at different sites. This activity is related to T3.5 as event logs can be used for off-line monitoring. The collaboration will continue. A forthcoming secondment is planned in year 3 from **ULEIC** to **ACT** on WP3 (secondment plan number 136).

6. b-API inference from composite systems

Software evolution often requires the integration of new components with legacy code, for instance code that was not developed with b-APIs in mind. As explained in Section 1.1, T3.6 was not due to be reported in this deliverable. However, b-APi inference, the key concern of T3.6, is crucial to enable the bottom-up development approach and therefore provides a complement to the other reported activities that focused mostly on the top-down approach. We have therefore included a discussion on T3.6 activities to this deliverable.

In Task 3.6 we are focusing on providing techniques to reverse-engineering systems, so that they can be adapted and integrated using b-API. In Section 6.1 we will report on the ongoing activities and progress of the consortium.

6.1 Ongoing activities on b-API inference

We have already discussed, earlier in this deliverable, existing work that supports the bottom-up approach to system's engineering, by extracting choreographies from composite BehAPI [Deniélou, Yoshida, 2013][Montesi, Yoshida, 2013][Lange, Tuosto, 2012][Bartoletti et al., 2015][Lange, Tuosto, Yoshida, 2015], just to mention some. These approaches check if the interaction among sets of component can be described as a global protocol (or choreography, or global type) and hence ensure the desirable properties that implementation of global protocols ensure,

such as communication safety and progress. One of the limitation is that these approaches offer a sound but not complete characterization of correct.

Task 3.6 has one sub-task:

"GBW, UKENT, and ULEIC will develop and apply (extended) finite-state machine inference to reverse-engineer the sequential behaviour of legacy components."

One of the open problems is to synthesize global protocols when the number of participants is not fixed a priori, but may change during the execution of the protocol. This has been shown to be a common and relevant pattern in practice [Dilley, Lange, 2019]. The secondment from **UKENT** to **GBW** (secondment **declaration 7**), also involving collaboration with **NOVA**, was targeted at this. The involved parties are working on addressing the problem of statically verifying systems of concurrent processes where some processes have an unknown number of replicas, e.g., to verify the compatibility of systems consisting of two "manager" nodes and 'n' worker nodes. The activity is ongoing and not completed, and will continue in years 3 and 4. **ULEIC** has not reported activities on this task and may work on this in year 3. Although **AAU** did not have planned activities on this task, the work on inference of usage types already reported in Task 3.2 (supporting secondment **declaration 41** - WP2) also concerns Task 3.6.

8. Conclusions

Activities have started and are on the way on all tasks concerned in this deliverable (Tasks 3.2 - 3.6). As expected, none of the tasks is completed, and more work is expected in year 3. So far, the key achievements overall have been:

- Two techniques for linking models at different levels of abstractions via subtyping/refinement, one based on session types (BehApi **publication no. 28** in Sygma) and one exploring a more user-friendly notation (kleimographies) (BehApi **publication no. 21** in Sygma) that may facilitate use by practitioners. These achievements may be at the basis for outcomes in WP4 (in particular Task 4.2 "Top-down DevOps support"). A tool has been produced on the theory in (BehApi **publication no. 28** in Sygma) and will be included in D4.3. This achievement contributes to O3.2 and O3.4.

- A tool to automatically generate fully executable test cases for Espresso, a framework for developing Android UI (reported in Section 3.1). The tool is available on a git repository and an article is awaiting review. This outcome may be possibly used in WP4 to integrate it with other tools (e.g., Task 4.4 on tool interoperability and Task 4.5 tool-chains). This achievement contributes to O3.2 and O3.3.
- A fine-grained typing system based on session types that allow to verify API composition that use aliasing (BehApi first occurring **publication no. 27** in Sygma). Theoretically, this is a relevant result as it shows that linearity and affinity are not necessary to attain a well behaved system via session typing. Allowing aliasing has a practical relevance as it allows vendors for more flexibility in the composition of their APIs. This typing system may, in the future, be implemented as a language type checker, hence may benefit WP4 and in particular Task 4.2 "Top-down DevOps support". This achievement contributes to O3.4.
- The application of formal methods to Blockchain is an emerging (and hot) topic. The application of behavioural types to the smart-contract language Solidity (BehApi **publication no. 29** in Sygma) may benefit WP2 (Task 3.2 on static verification) as verification of smart-contracts producers and consumers are inter-linked. This achievement contributes to O3.4.

As far as outcomes are concerned: progress towards O3.2 is partial (we expect more work on code generation), progress towards O3.3 is partial (a good number of techniques have been provided for correctness, we expect more outcomes on quality and security), and progress towards O3.4 is preliminary (some work has been done in Task 3.2 that contributes (BehApi **publication no. 21** in Sygma) with a notation and theory for model-driven development but most of the work towards this objective is expected as part of Task 3.6, which is not reported here and has no outcomes yet).

Task 3.2. The activities have been oriented to the more theoretical aspects of model-driven consumption of b-APIs, in particular transformation between models via subtyping or refinement. There have been **3 secondments** (declarations 8, 15, 38 -- 38 is still to complete), **1 supporting secondment** (declaration 41 -- still to complete), and **2 published outcomes** (declaration 21, 28) on this task. The published outcomes (Sections 2.1 and 2.2) result from collaborative work, each from two or more BehAPI partners. The work on synthesis techniques to produce code is ongoing. The total Person-Months (PMs) spent so far on this task is about 3.54PMs over the 6PM planned in the project overall: 1.77PMs for the sending

partners in the secondments, that are matched to 1.77PMs for the receiving partners.

Task 3.3. On this task there have been **2 secondments** (declarations 12, 33, 34) and **1 published outcome**. The outcome reported for Task 3.3 (Section 3.1) addresses the problem of test generation from b-APIs from a practical perspective: extending an existing framework to support the generation of test doubles for the specific, widespread, and relevant case of Android UI. More work is expected in year 3 and specific activities are planned (at least 2 secondments on this task). One more outcome is expected (submitted work) and more work is expected in year 3. The total Person-Months (PMs) spent so far on this task is 6.28PMs over the 10PMs planned in the project overall: 3.14PMs for the sending partners in the secondments, that are matched to 3.14PMs for the receiving partners.

Task 3.4. There has been a considerable activity on this task: **5 secondments** (declarations 8, 20, 25, 30, 38 -- the last three still to complete), **4 supporting secondments** (declarations 9, 21, 31, 35), and **3 published outcomes** (declarations 21, 27, 29). The contributions to this task are theoretical (which is expected, considering the nature of the topic) but with practical relevance. Practical relevance is ensured by providing theories targeted at specific application scenarios (smart-contracts), analyzing programs in specific real languages (e.g., Solidity), and targeting aspects as flexibility of verification (e.g., allowing verification of b-API implementation that use aliasing). We have good progress on this task but we expect more activity in year 3. The total Person-Months (PMs) spent so far on this task is 5.66PM over the 10PM planned in the project overall: 2.83PMs for the sending partners in the secondments, that are matched to 2.83PMs for the receiving partners.

Task 3.5. There have been **4 secondments** (declarations 9, 26, 33, 39 -- 26 is still to complete) and **1 supporting secondment** (declaration 18) on this task. Activities are ongoing. Published outcomes are expected in year 3. The total Person-Months (PMs) spent so far on this task is 10.12PMs over the 9PMs planned in the project overall: 5.06PMs for the sending partners in the secondments, that are matched to 5.06PMs for the receiving partners.

Task 3.6. There has been **1 secondment** (declaration 7) and **1 supporting secondment** (declaration 41). The partners have established ongoing

collaborations and outcomes are expected for year 3. As mentioned in Section 1.1, T3.6 was not due to be reported in this deliverable. However, as its activities have no dependencies with activities of the other tasks and have already started, and since the topic is relevant to this deliverable as it serves to "complete the loop" in a top-down and bottom-up development approach, we have included a discussion on T3.6 activities to this deliverable.

Activities on Task 3.7 are due to be reported on in Deliverable D3.3 due in Month 48. Some partners have already started on this task, in particular **UBA** had a secondment (secondment **declaration 22**) to **ITU** to work on dynamic reconfiguration of business processes based on DCR Graphs developed by **DCR**. **UMA**, **ULEIC**, **UNIBO** and **UKENT** are working on code reversibility.

Secondments. The BehAPI partners had planned to perform 18 secondments in M0-M24. At the time of writing this deliverable (M23): 10 have been completed (1 of them has been anticipated from year 3), 5 have been partially completed, 4 have not started. Considering that the non-started secondments are scheduled one in M18, two in M23, and one in M24 the misalignment with the plan is not concerning. The fact that partners reported activities on WP3 also in secondments that were not planned for WP3 (6 supporting secondments from other WPs reported here) is positive, as it demonstrated integration of the project's themes which is essential for integration of the proposed solutions. Indeed, the fact that WP2 and WP4 secondments contributed to WP3 is positive towards theories and tools for seamless engineering of producers and consumers b-API.

Outcomes. We have reported on 4 peer-reviewed published articles on the WP3 themes. Other 2 articles have been submitted (but still unpublished). The quantity of outcomes, at present, is satisfactory. Due to the intrinsic inertia of starting new collaborations on novel topics and the time taken to produce results we expect more outcomes to be reported in year 3. The relevance of the outcomes has been discussed through the deliverable and summarized in the next paragraph. The progress is overall satisfactory.

Bibliographic references

- M. Bartoletti, A. Scalas, R. Zunino. A semantic deconstruction of session types. Proc. CONCUR (2014), pp. 402–418.

- M. Bartoletti, J. Lange, A. Scalas, R. Zunino. Choreographies in the wild. JLAMP(109), pp. 36-60 (2015).
- M. Bartoletti, L. Galletta, M. Murgia. A Minimal Core Calculus for Solidity Contracts. In Data Privacy Management, Cryptocurrencies and Blockchain Technology (2019), pp. 233-243.
- K. Bierhoff, J. Aldrich. PLURAL: checking protocol compliance under aliasing. Proc. ICSE (2008), pp. 971-972.
- L. Bocchi, T. Chen, R. Demangeon, K. Honda, N. Yoshida: Monitoring networks through multiparty session types. TCS 669 (2017), pp. 33-58.
- M. Bravetti, M. Carbone, G. Zavattaro. Undecidability of asynchronous session subtyping. IC(256), pp. 300–320 (2017).
- M. Bravetti, M. Carbone, J. Lange, N. Yoshida, G.i Zavattaro: A Sound Algorithm for Asynchronous Session Subtyping. Proc. CONCUR (2019), pp. 1-16. BehAPI publication. Available at: https://cris.unibo.it/handle/11585/690920#.XlkUD5P7TOQ
- R. Bruni, A. Corradini, F. Gadducci, H. Melgratti, U. Montanari, E. Tuosto: Data-Driven Choreographies à la Klaim. In Models, Languages, and Tools for Concurrent and Distributed Programming - Essays Dedicated to Rocco De Nicola on the Occasion of His 65th Birthday (2019), pp. 170-190. BehAPI publication. Available at: https://zenodo.org/record/3260239#.XlkUA5P7TOR
- R. Demangeon, K. Honda. Full abstraction in a subtyped pi-calculus with linear types. Proc. CONCUR (2011), pp. 280–296.
- P.-M. Deniélou, N. Yoshida. Multiparty compatibility in communicating automata: characterisation and synthesis of global session types. Proc. ICALP (2013), pp. 174-186.
- N. Dilley, J. Lange. An Empirical Study of Messaging Passing Concurrency in Go Projects. SANER (2019): pp 377-387.
- A. Francalanza, C. Mezzina, E. Tuosto. Reversible Choreographies via Monitoring in Erlang**.** DAIS (2018), pp. 75-92.
- K. Honda, V.T. Vasconcelos, M. Kubo. Language primitives and type discipline for structured communication-based programming. Proc. ESOP (1998), pp 122-138.
- K. Honda, N. Yoshida, M. Carbone. Multiparty Asynchronous Session Types. J. ACM 63(1): 9:1-9:67 (2016).
- C. Laneve, C. Sacerdoti, A. Veschetti. On the Prediction of Smart Contracts' Behaviours. In "From Software Engineering to Formal Methods and Tools, and Back - Essays Dedicated to Stefania Gnesi on the Occasion of Her 65th

Birthday" (2019), LNCS 11865, pp. 397-415. BehAPI publication. Available at: https://cris.unibo.it/handle/11585/701791#.XlkU55P7TOQ

- C. Laneve, C. Sacerdoti. Ether Analysis of Solidity Functions. To appear as ``Oral Communication'' in Distributed Ledger Technology Workshop 2020.
- J. Lange, E. Tuosto. Synthesising choreographies from local session types. Proc. CONCUR (2012), pp. 225-239.
- J. Lange, E. Tuosto, N. Yoshida. From Communicating Machines to Graphical Choreographies. Proc. POPL (2015), pp. 221-232.
- J. Lange, N. Yoshida. On the Undecidability of Asynchronous Session Subtyping. Proc. FoSSaCS (2017), pp. 441–457.
- F. Montesi, N. Yoshida. Compositional choreographies Proc. CONCUR (2013), pp. 425-439.
- D. Mostrous, N. Yoshida. Session-based communication optimisation for higher-order mobile processes. Proc. TLCA (2009), pp. 203–218.
- D. Mostrous, N. Yoshida, K. Honda. Global principal typing in partially commutative asynchronous sessions. Proc. ESOP (2009), pp. 316–332.
- A. Scalas, N. Yoshida: Lightweight Session Programming in Scala. ECOOP (2016), pp.: 21:1-21:28.
- K. Takeuchi, K. Honda, M. Kubo. An interaction-based language and its typing system. Proc. PARLE (1994), pp 398-413.
- A.L.Voinea, O.Dardha and S.J.Gay. Resource Sharing via Capability-Based Multiparty Session Types. Proc. FM (2019), pp. 437-455. BehAPI publication. Available at: https://zenodo.org/record/3661263?page=1&size=20#.XlkUbJP7TOQ