FOR-COALA: Formalization of Configuration Languages

Summary table of persons involved in the project

Partner	Name	Firstname	Position	Role & resp. in the project	Effort (p.m)
IMTA	Coullon	Hélène	Associate Professor	Project leader, leader for WP0, WP3, WP4	24.0
IMTA	Not Known, PhD Student to be hired in the project			Contributes to all work packages but mainly WP3 and WP4	36.0
IMTA	Not Known, Master Student to be hired in the project			Contributes to WP2 and WP3	6.0
LIFO	Loulergue	Frédéric	Professor	Partner leader, leader for WP1, WP2	19.2
LIFO	Not Known, Post-doctoral researcher to be hired in the project			Contributes to WP0, WP2, WP3	24.0
LIFO	Not Known, Intern then Engineer to be hired in the project			Contributes to WP0, WP1, WP2	11.5
LIFO	Not Known, Intern then Engineer to be hired in the project			Contributes to WP0, WP2, WP3, WP4	11.5

Figure 1: Summary table of persons involved in the project

Contents

1 Proposal's context, positioning and objective(s)

1.1 Objectives and research hypotheses

Context. Large distributed software systems (applications or infrastructures) are now ubiquitous, with component-based systems (e.g., service-oriented architectures or microservices) offering a convenient way to structure large systems, in particular distributed systems deployed in the Cloud, in the core, or at the edge of the network. Indeed, isolating functionalities in components and building systems through composition greatly enhances the adaptability and scalability of systems, two important requirements for many organizations.

However, the advantages of distributed architectures come at the price of increased complexity and technical challenges related to observability, coordination, maintenance, etc. A set of operations denoted as DevOps — i.e. operations handled somewhere in between the developer and the machine administrator – includes notably the system configurations and reconfigurations that are required to achieve adaptability. These operations include the initial and dynamic changes that may occur on a distributed service-oriented software architecture: adding or removing services, connecting or disconnecting services, and changing some parameters or the internal behavior of services [1]. They may be required to handle various kinds of dynamic scenarios such as fault tolerance, scalability, software updates, various optimizations, etc.

Such changes may lead to faults. For example, recent outages have been traced back to problems within maintenance codes[1][2]. A study of 597 unplanned outages that affected popular cloud services between 2009 and 2015 found that 16% of them were caused by a software or hardware upgrade [2]. The study concludes that "the complexity of cloud hardware and software ecosystem has outpaced existing testing, debugging, and verification tools". Indeed, testing and debugging methods are largely inadequate in the context of distributed systems, while the adoption of more suitable formal methods remains marginal in industry. The latter can be attributed to the difficulty of using formal methods and tools. Yet formal methods can lighten the burden of program developers, DevOps engineers, and system administrators instead of adding to it.

Motivation. On the one hand, many configuration tools and languages exist in the DevOps community, some of them being specific to the provisioning of resources in Cloud providers, packaging problems, containerized deployments, configuration management of applications or infrastructures, etc. Such languages are often referred to as *Infrastructure-as-Code* (IaC for short). IaC languages offer easier constructs than low-level scripts to facilitate software engineering properties such as composition and reuse. The main advantage of these tools is their full integration and adoption in the DevOps community. Their disadvantage is they lack both formal and textual specifications. Moreover, their contours are blurred. On the other hand, many initiatives have been studied in academia to contribute to the deployment, configuration, and reconfiguration of distributed software [1], bringing improvements such as expressivity, speed, safety, etc. Many come with precise and sometimes formal definitions. However, they lack the breadth of the mainstream DevOps tools.

In the domain of programming languages, the CompCert C formally verified compiler [3] can be used as a trustworthy compiler by users not versed in formal methods. CompCert shows that formal semantics of languages can have an impact beyond their initial target, including the verification of automatic analysis tools [4] that can be used to improve the quality of source code. But CompCert also shows that such a success is only possible when a large subset of a real-world language is considered. Although sometimes imprecise or ambiguous, there exists a standard textual specification of the C language. This is not the case for mainstream DevOps tools: each language is defined by its unique implementation. In this context, having an executable formal semantics allows one to easily compare it to the implementation, during the design of such a semantics. It also provides verified interpretation, via code extraction from formalizations.

[1] `https://blog.cloudflare.com/cloudflare-outage-on-july-17-2020/`
[2] `https://engineering.fb.com/2021/10/05/networking-traffic/outage-details/`

Research problem and project objectives.

The goal of FOR-COALA is twofold: (1) understand and bridge the gap between a popular tool from the DevOps community and a tool from academia; (2) improve the understanding of these languages based on mechanized formal semantics and develop verified semantic-preserving cross-language transformations. The outcomes of the project will be two open source formally certified configuration languages, their execution engine, and a verified translation tool.

Some projects have been initiated by academia to improve or change the way of configuring systems, applications, or infrastructures. However, most of them are outdated or not under active development or maintenance anymore [5, 6]. We have decided to restrict our study to the recent and active tool CONCERTO for the following reasons. First, CONCERTO is currently the tool that offers the highest parallelism level when executing configuration programs, hence offering lower configuration and reconfiguration durations [7]. For this reason, it is currently attracting attention from industrial partners, in particular OVH and Orange. Second, the execution semantics of CONCERTO has been manually formalized [8]. Third, a tool on top of CONCERTO adds declarative features to CONCERTO by automatically inferring (by using the SMT solver Z3) the configuration program from a set of higher-order DevOps goals or objectives [9]. In other words, CONCERTO offers recent and active contributions with a strong basis for mechanized formalization. From the academic tool perspective, the research challenge of FOR-COALA is to cope with complex additional language features compared to industrial tools, particularly the high level of parallelism (non-determinism) offered by CONCERTO.

From the industry tool perspective, we chose ANSIBLE[3]. First, ANSIBLE is currently one of the most widely used tools, probably because of its very wide spectrum (from low-level scripting to high-level modular and compositional features) and low requirements for installation (it is agentless unlike PUPPET, another popular tool). Second, an extensive set of ANSIBLE playbooks (i.e. ANSIBLE programs) and roles (i.e., a set of programs to configure a given application or system) are publicly available online and can be leveraged in FOR-COALA. Third, CONCERTO and ANSIBLE have already been compared and share common aspects that will facilitate a cross-language transformation. The research challenge is to identify and formalize the subset of ANSIBLE, so that it is sufficiently powerful to answer the needs of companies while keeping the formalization tasks doable within the FOR-COALA project's timeframe.

The project has three specific objectives and two objectives aimed at maximizing the project's impact:

- **Obj1**: offering a certain level of certification in the configuration management language ANSIBLE;
- **Obj2**: bridging the gap between industrial and academic DevOps tools;
- **Obj3**: alerting the community on ambiguities in the mainstream language ANSIBLE and offering certified tools with clear semantics that corresponds to industrial needs;
- **Obj-Progress**: progressing beyond the state of the art;
- **Obj-Visibility**: making visible and disseminating the project results.

Success criteria. To evaluate the project and if the above objectives are reached, we define six success criteria as follows:

- **SC1**: Ability of the resulting certified tools to offer guarantees on DevOps procedures;
- **SC2**: Ability of the resulting certified tools to answer industrial real use-cases;
- **SC3**: Ability to semi-automatically and formally switch from a popular DevOps tool to an academic tool that offers additional performance features;

[3]`https://www.ansible.com`

Additional success criteria will be used for **Obj-Progress** and **Obj-Visibility**:

- **SC4**: Publication of the results in top-level academic conferences and journals in software engineering and formal methods (ICSE, FM, ASE, ETAPS, DiscoTeC, TSE, FAC, JLAMP, JSS, etc.) and research reports;
- **SC5**: Participation and presentations at seminars, workshops, and national working group meetings interested in software engineering, formal methods, and distributed systems (e.g. GDR GPL, GDR RSD);
- **SC6**: Participation and presentations at major conferences of the DevOps open source community such as "Config Management Camp", "KubeCon", and "OpenInfra Summit", as well as maintaining a technical blog and code contributions.

1.2 Position of the project as it relates to the state of the art

Configuration management languages. As already mentioned IaC languages offer good programming support and software engineering properties when handling complex DevOps procedures such as (among others) initial deployments, blue/green deployments, packaging, provisioning in the Cloud, rolling updates, etc. Those procedures and the exact contour of each IaC language are difficult to classify, but in FOR-COALA we specifically focus on a given class of IaC languages called configuration management languages. Among these languages are the well-known PUPPET[4], CHEF, and ANSIBLE. What is typically called a configuration management task includes any configuration, deployment, or maintenance task to perform on machines, devices, virtual machines, etc. With these tools, programs are written by the DevOps on her machine and coordinated by this machine, and instructions are sent to distant nodes to execute tasks.

A few important concepts have to be introduced to understand these languages. First, some languages are *agentless* which means that no specific agents have to be deployed on distant machines. This is the case of ANSIBLE which exclusively relies on SSH and Python. Second, we can distinguish two types of IaC approaches, first, *imperative* or procedural languages where the user writes a program that explicitly prescribes the set of operations to perform, second, *declarative* languages where the user declares the desired configuration, and where the tool infers or compiles the associated program(s). ANSIBLE is closer to scripts and thus quite imperative (e.g., sequence of tasks to execute), while PUPPET and CHEF are more declarative. Finally, ANSIBLE offers some *idempotency* informal guarantees on operations, meaning that executing the same ANSIBLE operation multiple times results in the same state, thus eliminating the risk of unintended configuration changes and allowing informal safety and predictability when automating. One can note that idempotency is not required at the operation level by declarative languages such as PUPPET and CHEF (but may be required at the user level [10]), as the declarative approach does not directly execute a static program, but computes a program from the difference between the current and desired state of the system. In other words with declarative approaches, if nothing changes, nothing is executed.

Academic approaches on deployment and reconfiguration. In the academic literature, even if not called IaC languages, automatic deployment, configuration, and reconfiguration of distributed software systems has been extensively studied, very often under the spectrum of component-based software engineering (CBSE) [1]. When focusing on the DevOps approach, one important aspect is to make a separation between the functional code of the pieces of software, written by developers, and the fact of deploying and maintaining those pieces of software, performed by DevOps. For this reason, only a few contributions of the academic literature are closer to the DevOps tools: the ones focusing on the modeling of the lifecycle of the software components. In particular, AEOLUS [5], and CONCERTO [8] have been manually formally specified and have been successfully used on real use cases either for Cloud applications or complex distributed systems. CONCERTO is led by Hélène Coullon and has been compared to ANSIBLE on the deployment of real OpenStack and Kubernetes systems. CONCERTO was designed to ensure the coordination of multiple interdependent software entities when

[4] `https://www.puppet.com`

changing their states, their behaviors, and their relations (i.e., compositions) between each other. CONCERTO relies on parallel constructs either at the level of one software entity (i.e., component) or between entities to improve the efficiency compared to the state of the art. ANSIBLE also handles the coordination of some changes in-between interdependent software entities but with a sequential approach. ANSIBLE additionally comes up with a set of modules that are programming support on top of bash commands. Such modules are not offered and are not included in the scope of CONCERTO.

Software quality in DevOps. There are many contributions on tools to improve the quality of configuration programs (e.g. [11, 12]), but they are neither based on formal semantics nor verified correct using a proof assistant. Some do rely on a model of the considered DevOps tool but are not formalized in detail. ANSIBLE has been the subject of a few informal studies. Notably, in [11] are highlighted some ambiguities within the ANSIBLE language. In particular, the intricate variable precedence rules and the laziness evaluation of expressions create ambiguities in the value of variables. Three categories of "code smells" are presented: unsafe reuse of variables, unintentional override of variables, and too high precedence. The authors made an empirical evaluation of their framework on 19,661 roles extracted from ANSIBLE repositories [13]. This contribution on ANSIBLE ambiguities, while not being formalized, is very important to consider in WP2 as such ambiguities should be resolved in a formal semantics of ANSIBLE. The same authors have also presented an oral contribution relative to the Turing completeness of ANSIBLE at CONFLANG'23 [5]. ANSIBLE as a whole is already known to be Turing complete as embedding Python and the full Jinja2 templating framework. However, in this contribution, the authors show that ANSIBLE Turing completeness relying on a tiny subset of the core ANSIBLE language: `include_tasks`, `set_fact`, `when` task property, a subset of Jinja2. This Turing completeness of ANSIBLE will have to be taken into account when selecting the subset of ANSIBLE to formalize in WP1 and WP2.

To our knowledge, there are only two contributions offering formal semantics of configuration management languages: SMARTFROG [6] an academic tool (no longer maintained) and μPUPPET [14] a subset of PUPPET.

The semantics of SMARTFROG is a denotational semantics of a *compiler* for a core SMARTFROG fragment. From the high-level language SF which contains features such as inheritance, composition, references, etc., the compilation process produces a store which is basically a tree of attribute-value pairs. These trees are also the abstractions for system states in SMARTFROG's semantics. The authors wrote three implementations (in Scala, Haskell, and OCaml) of the compiler guided by the semantics but not proved correct w.r.t to the formal semantics (which is not mechanized). These implementations were randomly tested and a few implementation errors were found. They were also tested against the production compiler: it allowed them to find both a misunderstanding of the semantics of SMARTFROG and bugs in the production compiler.

The semantics of μPUPPET is operational. An implementation of a μPUPPET compiler exists, guided by the semantics but not proved correct w.r.t. the formal semantics which is also not mechanized. The output of the compiler is a catalog which is a structure close to the structure of stores, the output of SMARTFROG's compiler. A catalog is also an abstraction for a system state. The provided semantics is a small-step operational semantics. This design choice was made because the authors would not presume a priori of the semantics properties. They proved that μPUPPET is deterministic and is monotone. However, the semantics showed that μPUPPET (and PUPPET) programs are not always terminating and that the order in which variable definitions appear does have an effect on the result.

We chose to study ANSIBLE rather than building on the existing formalization of PUPPET because of the expanding notoriety of ANSIBLE, because of the existing comparisons between ANSIBLE and CONCERTO, but also because that is the tool deployed by the DevOps experts and industrial users of FOR-COALA's advisory board.

[5]https://2023.splashcon.org/details/conflang-2023-papers/3/Ansible-Is-Turing-Complete

Related projects. TARANIS (2023–2030) is one of the 7-year projects of the PEPR Cloud. It targets the modeling, deployment, orchestration, and optimization of Cloud applications and infrastructure. One of its objectives is to study the formal aspects of the above challenges. In particular, the two partners of the FOR-COALA project will co-advise a PhD thesis in TARANIS. The goal of this thesis is to study the formal aspects of CONCERTO-D, a decentralized version of CONCERTO. Notes that CONCERTO-D, unlike CONCERTO, cannot be compared to existing mainstream DevOps declarative tools as being fully decentralized, hence the work tackled in FOR-COALA cannot be applied in Taranis, and the addressed challenges are very different in practice.

The project leader also participates to the "Défi" between OVH and Inria on frugal clouds. Hélène Coullon is involved in the study of the energy consumption of reconfiguration procedures at OVH. In particular, CONCERTO and ANSIBLE are compared in terms of reconfiguration speed and reconfiguration energy consumption. This work is purely at an engineering level, not at a research level. Indeed, one of the deployment and reconfiguration procedures at OVH is studied by using the two existing tools. Furthermore, no formal aspects are studied in this project. Yet, it is clear that this work is useful in FOR-COALA as it compares ANSIBLE and CONCERTO from an engineering viewpoint. OVH will be part of the advisory board in FOR-COALA.

Two projects related to dynamic adaptation have been accepted to the ANR call AAPG 2023: ADAPT and SMARTCLOUD.

The goal of SMARTCLOUD (ANR-23-CE25-0012) is to build a Cloud-oriented framework for dynamic adaptations and reconfigurations of systems in the Cloud. Hélène Coullon is a member of the advisory board of this project[6]. The first difference is that SMARTCLOUD exclusively targets the Cloud, while FOR-COALA targets the well-known DevOps tools ANSIBLE that can be used either for the Cloud or for any configuration management task on on-premise servers, on small devices, or into a virtual machine, etc. The second difference is that SMARTCLOUD leverages formal methods to build correct-by-constructions reconfigurations by easing the way of expressing constraints on the application topology or behavior. In FOR-COALA the main focus is to provide a formal semantics of the IaC languages ANSIBLE and CONCERTO and verify their implementations.

Although the ADAPT project (ANR-23-CE25-0004) does relate to the general domain of configuration and reconfiguration, the kind of systems it studies is very different from FOR-COALA and SMARTCLOUD. Indeed, the context of this project is programmable matter: systems are assemblies of large numbers of small and similar modular robots. Two aspects important for ADAPT are not considered in FOR-COALA: the hierarchical nature of the modeling approach as well as the physical aspect of the considered cyber-physical systems.

In 2015, COLIS (ANR-15-CE25-0001) targeted the analysis and verification of shell scripts. One of the outcomes of this project is a verified interpreter for CoLiS, a shell-like programming language [15]. CoLiS has notably been used to analyze and check POSIX shell maintenance scripts associated with the package management in Debian [16]. The goal of the authors was not to replace POSIX with CoLiS but to highlight problems in those scripts. For instance, with CoLiS syntax errors, non-compliance with the requirements of the Debian policy, or failure of scripts can be detected. CoLiS could be a useful complementary contribution to WP2.

1.3 METHODOLOGY AND RISK MANAGEMENT

1.3.1 Methodology

The objectives of the project are ambitious. To achieve the expected results, the project is organized into five work-packages divided into tasks (see Section 1.3.3). Each work-package is led by one of the two permanent researchers (see Fig. 1) who will be responsible for the management of the work-package, for solving technical problems, and for achieving success criterias (**SC1** to **SC6**).

All work-packages (except WP0) require expertise in the DevOps/reconfiguration languages and mechanized semantics formalization. The first domain is the expertise of Hélène Coullon, while the second is the

[6] https://project.inria.fr/smartcloud/advisory-board/

WP	*Title*	*Leader*	*Effort, p.m*		
			IMTA	LIFO	*Total*
WP0	Management and dissemination	IMTA	8,0	5,0	*13,0*
WP1	Analysis of requirements	LIFO	6,0	8,0	*14,0*
WP2	Formalization of Ansible subsets	LIFO	12,0	33,0	*45,0*
WP3	Formalization of Concerto & tools	IMTA	28,0	8,7	*36,7*
WP4	Semantic preserving transformations	IMTA	12,0	11,5	*23,5*
		Total	*66*	*66,2*	*132,2*

Figure 2: Work-packages

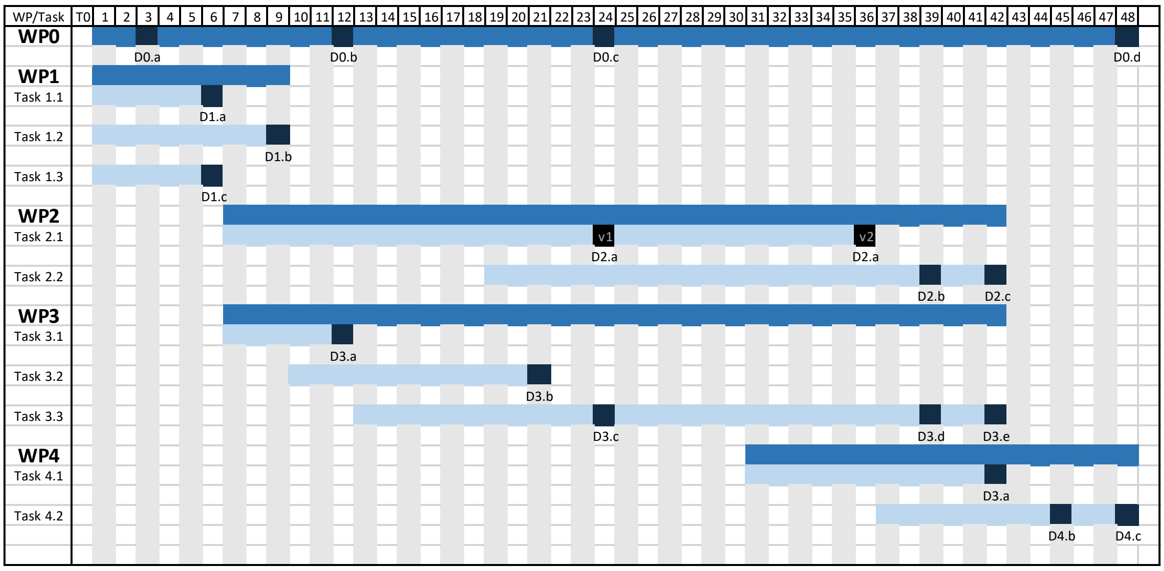

Figure 3: Gantt diagram: project work-packages and tasks

expertise of Frédéric Loulergue. This guarantees an active collaboration on all work-packages between the partners. We have decided to give the responsibility of work packages related to ANSIBLE to Frédéric Loulergue and the LIFO. The responsibility of WP3 and WP4 is given to Hélène Coullon as she leads the design and implementation of CONCERTO and has previously compared it to ANSIBLE.

All technical and organizational problems will be analyzed and discussed jointly during the project progress meetings. Figure 2 lists the work-packages and each of them indicates the efforts and the leading partner. The road map is depicted in Fig. 3. WP0 is the administrative work-package devoted to management and dissemination activities associated with **Obj-Visibility**. All the deliverables will be made publicly available, and the code will be released as open-source software. WP1 to WP4 are four technical work-packages and they all address **Obj-Progress**. WP1 establishes the requirements for the future solution. In particular, it identifies cases of real-life industrial code. It will guide the work of the technical work-packages and select a set of case studies and code patterns that will be the basis for the design and evaluation of our contributions. WP2 aims to design a mechanized formalization and a verified interpreter for (a subset of) ANSIBLE, while WP3 does the same for CONCERTO and associated tools. In WP4 the formally specified and verified transformation from ANSIBLE to CONCERTO will be produced. Work packages address the objectives of FOR-COALA as follows: WP2 and WP3 implement **Obj1**; WP2, WP3, and WP4 implement **Obj2**; WP0, WP1 and WP2 implement **Obj3**.

1.3.2 Risk Management

From an organizational point of view, the project team is comprised of two partners and six persons including two permanent researchers. The coordinator, Hélène Coullon and Frédéric Loulergue have previously worked together to write a survey on verified reconfiguration [1] successfully published to ACM Computing Surveys, a prestigious journal. The subject of the publication is directly related to FOR-COALA, hence a good level of expertise is guaranteed for the two permanent members. The project coordinator will organize both virtual and in-person meetings on a very regular basis, including bi-annual plenary meetings over a full day. The distance from IMT to LIFO is reasonable and the travel is especially convenient by train with a direct train which takes less than 3 hours. This makes additional in-person meetings easy to organize if needed. As a result, generic risks related to collaborative projects seem limited.

Scientific and technical risks exist, they are inherent to any ambitious innovative project. The project consortium and organization were chosen to reduce them. Indeed, the team gathers all the required expertise for the project's success (see Section 2.1). The order, duration, and required efforts of each work-package were carefully identified in order to reduce those risks. Nonetheless, there remain a certain number of risks related to each work-package that are described in Section 1.3.3 alongside discussions as to how their impact is reduced. Important technical and organizational problems will be reported to the project coordinator. Should some urgent issue arise, the problem will be discussed during dedicated online meetings quickly organized for that purpose.

A general risk for FOR-COALA is that the semantics and verified tool are not usable in practice, limiting the impact of the project. The two main causes may be that: the selected subset of ANSIBLE is not suited to industrial use and that the formal semantics and verified interpreters and tools provide different results. To limit the first cause, FOR-COALA includes an **Advisory Board** of DevOps experts and industrial users (see Section 2.1), page 15), which will be consulted on a regular basis during the project and especially several times during WP1. To limit the second cause, for each task where the outcome will be an executable verified tool, we will *test* the result of the verified tools with respect to their production counterparts. If test results are different, it may mean either that our semantics should be modified or that the production implementation contains a bug. In the former case, we will correct the semantics and interpreter and in the latter case, we will report the problem to the tool developers, as done in the past for the SmartFrog initiative [6]

1.3.3 Work-Packages

Work-Package 0: Management and dissemination (T0 – T0+48, led by IMTA)

Partner	***IMTA***	LIFO
Effort, person.month	8	5

WP0 is dedicated to project management and dissemination activities. It will be led by Hélène Coullon, FOR-COALA project coordinator, whereas all partners will be involved. Project management actions include preparing ANR reviews and monitoring the progress of scientific and technical work with respect to the project road-map (see Gantt diagram shown in Fig. 3), and organization of progress meetings (every 6 months). WP0 is also in charge of risk management (see Section 1.3).

Another important task for WP0 consists of establishing the FOR-COALA consortium agreement during the first year (before T0+12). WP0 also ensures the organization of technical means for the FOR-COALA technical and dissemination activities. They include in particular designing and deploying a project website containing usual news on publications and presentations, but also a technical "blog" to vulgarize and make easy the understanding of some problems when using ANSIBLE. WP0 will also set up collaborative solutions, including a dedicated GitHub/GitLab organization/group, to share the project documents and developments between the consortium members. The team will heavily rely on collaborative digital tools to make each work-package output continuously available to other work-packages, track issues, archive online discussions, *etc.* The FOR-COALA dissemination activities are described in Section 3.

Deliverables

ID	Name	Resp.	Kind	Date
D0.a	Public website & technical blog & collaborative tools	IMTA	Other	T0+3
D0.b	Consortium Agreement	IMTA	Other	T0+12
D0.c	Intermediate Report	LIFO	Report	T0+24
D0.d	Final Report	IMTA	Report	T0+48

The deliverables include a public website presenting the FOR-COALA project, its consortium, and deliverables (D0.a), the consortium agreement (D0.b), and the mid-term (D0.c) and final (D0.d) ANR reports providing an intermediate and final description of the project achievements. The website will also contain a technical blog to illustrate and inform the community of some limitations, ambiguities in the Ansible language, and some vulgarization articles on how we solve problems in FOR-COALA.

Risks and mitigation In general in collaborative projects, the risks in the administrative work-package include problems related to a large number of partners in the consortium, not used to work together, partners abandoning the project, and intellectual property issues. In FOR-COALA, we have only 2 partners, all of which are part of large institutions. The two involved permanent researchers have been successfully working together for several years, and all partners have preliminarily agreed that the verification tools developed during the project will be made available as open source (the exact type of license will be decided in the consortium agreement). Therefore, the risks related to this work-package are low. All technical and administrative issues or execution delays, if any, will be quickly reported to the coordinator, and discussed with all partners in order to find adequate solutions. To mitigate the potential risk of a delayed signature of the consortium agreement (planned by T0+12), the preparation of the agreement will start immediately after the acceptance of the proposal.

Work-Package 1: Analysis of requirements and case studies (T0 – T0+9, led by LIFO)

Partner	IMTA	LIFO
Effort, person.month	6	8

The goal of this work package is to analyze the requirements with respect to the real-world case studies provided and curated by all the partners. This will guide the prioritization of features to include in the formal semantics of WP2 and WP3 and also the features that will be handled by the verified transformation of WP4.

Task 1.1: Ansible case studies (T0 – T0+6) Ansible Galaxy is a repository of reusable Ansible items. It currently contains more than 30,000 roles and 2,500 collections. This offers a large variety of case studies. Most of the items in this repository refer to code managed in a version control system. Thus the history of the items is available. As a first step, we plan to mine Ansible Galaxy and related repositories to determine the most used Ansible features. The outcome of the study will be a list of prioritized features to formalize. This list will be discussed with our advisory board of industrial users. In a second step, we will use a frequent itemset mining algorithm to obtain a set of coherent subsets of features for which there exist playbooks in Ansible Galaxy using all the considered features. The first set of case studies which contain these coherent subsets of features is called LCS. Other criteria may be used to reduce the number of possible candidates such as the overall size of the case study and its popularity. The reduced list of candidate case studies will then be presented to the advisory board to select the final set of case studies (denoted by FCS). FCS will be used to guide our work in WP2 and WP4. LCS will be used for testing purposes.

Task 1.2: Concerto case studies (T0 – T0+9) We perfectly know and control the semantics of Concerto. Hence the difficulty will not be to understand how to cover its semantics, but rather to take the time to design and code new use cases from real-world scenarios. We already have a few such scenarios and we plan to add this set by manually translating a *subset* of the Ansible case studies to Concerto. This manual translation can also be considered a preliminary work for WP4. The full set of Ansible case studies (FCS) will be translated to Concerto using the translation tool of WP4.

Task 1.3: Formalization requirements (T0 – T0+6) This work package will also analyze more precisely the requirements for the tool we will use to formalize the semantics, define and prove the correctness of interpreters w.r.t. these semantics, and define and prove the correctness of transformations. Based on our experience, we plan to consider the Coq [17] proof assistant and the Why3 [18] proof and verification environment. Gallina is the specification language of Coq and WhyML the specification language of Why3. Gallina is a much richer specification language than WhyML but in most cases users are required to write proof scripts in Coq to prove properties. WhyML is mostly less expressive than Gallina (however, there are imperative features in the programming part of WhyML which Gallina does not offer) but proofs are mostly automatic. In WhyML, it may be necessary to write lemma functions which are a kind of annotations when proofs by induction are needed. Both tools provide a way to extract compilable code (in particular OCaml code) from specifications. Coq extraction mechanism only handles a pure functional subset of OCaml and Why3 handles a subset of OCaml which includes imperative aspects and exceptions. Why3 is able to translate WhyML code to Gallina. One use case is to prove lemmas not handled by automated provers in Why3, it is also possible to generate Gallina code from WhyML and prove the lemmas with Coq. In the context of FOR-COALA, we could specify the semantics and implement and proof the correctness of interpreters using imperative features using Why3 while using Coq for proving properties of these semantics and implementing and proving the correctness of program transformations. In this task, we will experiment several formalization and proof approaches with Coq and Why3 on a semantics and verified interpreter for a small, but representative, language. This will guide the choice of a formalization approach for WP2, WP3 and WP4.

Deliverables

ID	Name	Resp.	Kind	Date
D1.a	Dataset from Ansible Galaxy	LIFO	Dataset	T0+6
D1.b	Ansible and Concerto case studies	IMTA	Repository	T0+9
D1.c	Formalization requirements	LIFO	Report	T0+6

The main deliverable of this work-package is (D1.b) a repository of curated case studies for Ansible and Concerto. A byproduct is a dataset on the analysis of Ansible features found in Ansible Galaxy (D1.a). A report (D1.c) will explore design choices in formalizing a small semantics and an interpreter using Coq or Why3 or both as well as a few proofs of properties with Coq and Why3.

Risks and mitigation Possible risks of this work-package are:

- a difficulty or delay in identifying representative sets of industrial case studies,
- the small-scale experiments in formalization may not scale in WP2, WP3, and WP4.

Our methodology relies on both an automated analysis and expert feedback from the advisory board which limits the risk the sets will be non-representative. The automated analysis will leverage tools already used in the analysis of Ansible Galaxy which limits the risk of technical difficulties and delays.

We have experience in developing and maintaining Coq formalizations reaching 50,000 LoC and we expect the formalizations of FOR-COALA to be in the same order of magnitude thus limiting the risk related to the formalization scale in the case of Coq. There are quite large verification projects in Why3 but, to our knowledge, all of them are *program* verification projects. There are only a few verification efforts related to inductively defined predicates formalizing semantics and their sizes are in the few thousand LoC. However, as it is possible to translate WhyML to Gallina, even if choose WhyML as our default specification language and scaling problems arise with Why3, we could fall back to Coq.

Work-Package 2: Formalization of a subset of ANSIBLE (T0+6 – T0+42, led by LIFO)

Partner	IMTA	LIFO
Effort, person.month	12	33

Task 2.1: Mechanized ANSIBLE semantics (T0+6 – T0+36) Unlike CONCERTO, to our knowledge, there does not exist any formal semantics of a subset of ANSIBLE, mechanized, or on paper. As mentioned in Section 1.2, there are papers with contributions on the quality and defects of IaC programs that contain some informal model of ANSIBLE or that highlight aspects of ANSIBLE we need to formalize with care. Our goal is to obtain a semantics that will be not only enough for WP4 but that could be used as a foundation for further verification efforts in the ANSIBLE ecosystems.

Besides covering the largest possible subset of ANSIBLE, the main challenges we anticipate are:

- finding the right level of abstraction. In particular, as tools such as ANSIBLE do change the state of the system they manage, we will take particular care in designing the semantics in such a way that it can handle different granularity in systems abstractions. This would allow the semantics to be used in various verification efforts beyond FOR-COALA.
- handling both the commonality and the diversity of features. For example, there are several ANSIBLE modules of package management systems. We may not formalize all of them but our formalization should be flexible enough so that common aspects should not be duplicated while allowing us to express the specific aspects of each package manager system and be amenable to formalizing additional package management systems we might add after the project.
- designing the right modular architecture. We will formalize the modules as prioritized in WP1 but we will also make our best effort to make the formalization easily extensible in the future to take into account additional modules and features of ANSIBLE which are very different than the modules we have in the list of WP1.

It is important to put formal semantics to use to check it does express meaningful aspects. One use is to verify the properties of the semantics. In the context of infrastructure as code, one important property is *idempotency*, i.e. the execution of an IaC program yields a state of the system that remains unchanged if the program is executed again. We will prove the idempotency of the subset of ANSIBLE we formalize. Another use is to define an interpreter for the considered code and prove its correctness with respect to the semantics.

Task 2.2: Verified ANSIBLE interpreter (T0+18 – T0+42) The goal of this task is to implement an interpreter of ANSIBLE playbooks in the specification language of the chosen specification and verification tool and to verify its correctness with respect to the semantics defined in Task 2.1 as inductive predicates. As the execution of ANSIBLE playbooks is sequential, proving the correctness with respect to the semantics should not be very difficult.

All components of the interpreter may not be implemented in Coq or Why3. For example, parsing YAML files may be done with a Gallina program working on strings (using Menhir for Coq [19]) but reading from a file cannot directly be written in Gallina.

For a fully executable interpreter, the code extracted from the formal development of Task 2.2 should be completed, in particular for IO operations. Once this is done, the verified interpreter will be tested against the ANSIBLE production implementation on the large set of case studies LCS from WP1. Any mismatch will indicate either a faithfulness problem in the formalization, a bug in the non-verified part of our interpreter, or possibly a bug in the production implementation. In the first case, it provides feedback on Task 2.1, in the second case on Task 2.2, and in the last case, it has an impact on the quality of the production implementation of ANSIBLE.

Deliverables

ID	Name	Responsible	Kind	Date
D2.a	ANSIBLE semantics	LIFO	Repository	T0+24, T0+36
D2.b	Verified interpreter	LIFO	Repository	T0+39
D2.c	Evaluation of the verified interpreter	IMTA	Report	T0+42

The two main deliverables are the formal semantics of the subset of ANSIBLE we will consider (D2.a) and the verified interpreter for this subset (D2.b). There will be an intermediate version of the deliverable (D2.a) at T0+24 which will be subsumed by the final version at T0+36. The design and development of the semantics and interpreter will proceed incrementally which will allow to work on the two tasks T2.1 and T2.2 in parallel for a large part of the duration of these tasks. The evaluation effort will start as soon as the interpreter is rich enough to execute some of the selected case studies. The last deliverable is an evaluation of the verified interpreter with respect to the production implementation of ANSIBLE (D2.c).

Risks and mitigation

- Mechanized formalization is time-consuming. One difficulty is that there are often many ways to formalize the same concept. Usually, each way is well-suited to some classes of proofs which are not easy to determine before working on the proofs. In FOR-COALA, this aspect has a rather low risk, especially with the preliminary study of WP1. Another risk is related to the number of ANSIBLE features to formalize. As the list of features to formalize obtained in WP1 will be with priorities, we could adapt the number of considered features depending on our progress.
- A common risk of mechanized formalizations is that a mechanized artifact does not faithfully formalize a hand-written implementation, which decreases the trust users can have in the implementation. In FOR-COALA, this risk is mitigated by the fact we will implement verified interpreters that we can test with respect to ANSIBLE implementation, in particular on the set of case studies from WP1.

Work-Package 3: Formalization of CONCERTO and associated tools (T0+6 – T0+42, led by IMTA)

Partner	IMTA	LIFO
Effort, person.month	28	8,7

The goal of this work package is to mechanically formalize the semantics of CONCERTO and associated tools and to provide associated interpreter and tool implementations verified correct with respect to their semantics. This work package will produce a certified CONCERTO suite.

Task 3.1: Mechanized formal semantics of CONCERTO (T0+6 – T0+12) First, CONCERTO has already been manually formalized in [8]. This formalization has some limitations though. First, the formalization is manual and its executability has not been verified with an interpreter. Second, some important language constructions of CONCERTO have been omitted in this formalization: a subpart inherited from Python in the CONCERTO implementation. The goal is not to formalize Python or a large subset of Python but rather to identify a number of language features that are enough to express the semantics of actual CONCERTO case studies. These features include variables and loops for instance. In this first task, we will mechanically formalize CONCERTO. This formalization will include the set of constructions that are required within CONCERTO but not yet included in the manual formalization.

Task 3.2: Mechanized formal semantics of the synthesis tool (T0+9 – T0+21) Second, CONCERTO is a coordination execution model that offers a reconfiguration language to dynamically: transition from one state of a component lifecycle to another; and change the set of component instances and their connections (i.e., the assembly of the distributed software system). Hence, CONCERTO itself cannot be considered a declarative language as the set of instructions to reach the desired state has to be written. CONCERTO is imperative (i.e., procedural). For this reason, CONCERTO has been extended with a synthesis tool [9] that, from a simple and partial goal (i.e., the desired state), generates the correct reconfiguration program. This tool is based on an algorithm made of a small exhaustive search, a scheduling problem solved by the z3 SMT solver, and a set of correct rewriting rules to simplify the obtained program. Hence, the generated program is correct-by-construction. However, this tool (i.e., its input language and its algorithm) has not been formalized either manually or mechanically. This is the goal of this task.

Task 3.3: Verified implementation of the CONCERTO suite (T0+12 – T0+42) Finally, from the formalized semantics of the CONCERTO suite, a verified interpreter of CONCERTO programs and a verified implementation of the synthesis tool will be proved correct with respect to the semantics. As the execution of CONCERTO programs is parallel, proving the correctness of the interpreter and synthesis tool with respect to their semantics has additional challenges compared to task 2.2. The outcome of this task being verified executable code, we will, as we do in Task 2.2, test them with respect to the non-verified implementation of the CONCERTO suite.

Deliverables

ID	**Name**	**Resp.**	**Kind**	**Date**
D3.a	Mechanized semantics of CONCERTO	IMTA	Repository	T0+12
D3.b	Mechanized semantics of the synthesis tool	IMTA	Repository	T0+21
D3.c	Verified interpreter of CONCERTO	IMTA	Repository	T0+24
D3.d	Verified implementation of the synthesis	IMTA	Repository	T0+39
D3.e	Evaluation of the verified implementations	LIFO	Report	T0+42

The main deliverables are the formal semantics of the CONCERTO suite (D3.a and D3.b) and a verified implementation of this suite (D3.c and D3.d). The last deliverable (D3.e) will report on the evaluation of the verified suite implementation with respect to the Python implementation of the CONCERTO suite through tests.

Risks and mitigation This work-package shares the risks and mitigation of WP2. A formal semantics of the core of CONCERTO exists on paper which lowers the overall risk. There are additional risks related to the fact that CONCERTO is a language with parallel constructs and concurrency. These risks are however low because we have experience in formalizing and verifying concurrent and parallel programs [20, 21, 22, 23].

Work-Package 4: Semantics preserving transformations (T0+30 – T0+48, led by IMTA)

Partner	IMTA	LIFO
Effort, person.month	12	11,5

In different works, Concerto has been compared in terms of efficiency to ANSIBLE [24]. The goal of this comparison was to highlight that introducing more parallel constructs in a DevOps language could have an important impact on duration when deploying complex software systems. For example, when deploying OpenStack a gain of up to 70%, and when deploying a Kubernetes cluster a gain of up to 40% has been observed by using CONCERTO rather than ANSIBLE. However, this manual transformation has a high human cost. Either the person in charge of this transformation knows the initial ANSIBLE playbooks but has to increase her expertise in CONCERTO, or knows CONCERTO but does not know the details of the initial ANSIBLE playbooks.

In both cases, it is a long and complex task. Our goal in this work-package is to design a verified and automated semantics-preserving transformation from ANSIBLE to CONCERTO.

Task 4.1: Design and implementation of the transformation (T0+30 – T0+42) The first task is to design, formalize, and implement the transformation. Multiple challenges have to be addressed to accomplish this task. First, CONCERTO and ANSIBLE do not cover the same functionalities. On the one hand, ANSIBLE is a tool on top of SSH that offers low-level modules to replace bash commands, not offered by CONCERTO. On top of this, ANSIBLE adds a coordination level to apply operations in the right order (sequential order). On the other hand, CONCERTO focuses on the coordination level, it adds parallel and concurrency constructs, and also explicit dependencies that expose a partial order of tasks compared to ANSIBLE. Moreover, CONCERTO is enhanced by a synthesis tool to make it declarative which does not exist in ANSIBLE. The intersection between both is clearly at the coordination level. The first challenge is to exactly identify and extract this semantics intersection from WP2 and WP3.

Second, as it is, a transformation from ANSIBLE to CONCERTO can only be done without any parallelism and concurrency in the resulting CONCERTO program because of the lack of explicit dependencies and partial order expressiveness in ANSIBLE. We plan to have a three-level procedure for the transformation: (1) we strictly keep the sequential order specified within ANSIBLE playbooks and roles and prove the semantics preserving transformation; (2) we try to detect and make explicit some hidden dependencies within ANSIBLE playbooks and roles (e.g., Jinja2 templates and variables) to relax the sequential order while preserving the results; (3) we optionally build and formalize a set of annotations or a small DSL on top of ANSIBLE to explicitly express dependencies and partial orders, thus switching to an optimal CONCERTO code.

Task 4.2: Verification of the transformation (T0+36 – T0+48) Once formally and mechanically specified, the semantics preserving transformation from ANSIBLE to CONCERTO can be implemented and verified with respect to the semantics. This task should not be too difficult or long if having the semantics of WP2 (ANSIBLE), WP3 (CONCERTO), and task 4.1. However, this work may induce some changes or limitations in WP2 and WP3 that could require a few iterations. As the executable artifact of WP2 and WP3, the transformation will also be evaluated. Of course, there is not any non-verified transformation to test the verified one against. We will compare the execution of the transformed program with respect to the original ones. As mentioned in WP2, there are almost always unverified parts to add to obtain a compilable program from code extracted from a formalization. While the proof of correctness will ensure the core of the transformation does not contain any bugs, the non-verified parts (basically IO aspects) need to be tested.

Deliverables

ID	Name	Resp.	Kind	Date
D4.a	Executable ANSIBLE-CONCERTO transformation	IMTA	Repository	T0+42
D4.b	Verification of the transformation	LIFO	Repository	T0+45
D4.c	Evaluation of the transformation	IMTA	Report	T0+48

Risks and mitigation

- The risks are related to the usual difficulties of compiler verification with the added problem that the target language has a non-deterministic execution order. The literature on formally verified compilation is rich and we have experience in verified program transformation in various contexts (e.g. [22, 25]) which limits these risks.
- As WP2 and WP3, if we estimate, while working on the verified translation, that the effort to handle the full subset of ANSIBLE of WP2 or full Concerto of WP3 is not compatible with the duration of the project, we can consider a smaller subset of ANSIBLE and a subset of Concerto.

2 Organization and implementation of the project

2.1 Scientific coordinator and its consortium

The For-CoaLa scientific coordinator is Hélène Coullon, associate professor at IMT Atlantique and member of the STACK research team, a mixed team between Inria and Orange, and also a team of the LS2N UMR laboratory. She was also a 20% adjunct professor at the University of Tromsø (Norway) between 2020 and 2022. Her expertise is programming models in the context of distributed infrastructures and systems. In particular, she is interested in the configuration, deployment, reconfiguration, and self-adaptation of large and complex distributed software systems. She leads the design of Concerto [8, 7], a component-based reconfiguration model, and associated tools [9, 26]. She is a WP leader in the ANR PRCE project SeMaFoR (Self Management of Fog Resources, ending in 2025) [27], as well as a WP leader in the project TARANIS of the PEPR Cloud (2023-2030). She is also involved as a partner in the "Défi" project between OVH and Inria to compare the execution and energy consumption of reconfiguration procedures between Concerto and Ansible, and finally in the OTPaaS project (ending in 2025), a collaborative project including many industrial partners. She is the vice-president of the French ACM SIGOPS group and co-chairs the Yoda (trustworthY and Optimal Dynamic Adaptation) national working group of the GdR GPL. Finally, she is co-chair of the research group CyclOps (deployment and reconfiguration) of the Inria/Orange joint laboratory. She will devote 50% of her research time to For-CoaLa.

Consortium. In addition to IMT Atlantique, the consortium consists of Université d'Orléans, Laboratoire d'Informatique Fondamentale d'Orléans (LIFO). Also, it is worth noting that participants in the consortium have already successfully published together [1]. It offers the guarantee of fruitful cooperation between the For-CoaLa partners. The partners are also very complementary in terms of their respective areas of strong expertise: design of models for distributed systems deployment and reconfiguration (IMT Atlantique), and formalization using a proof assistant including languages semantics and program transformations (LIFO) [28, 25] and formal methods in general [23]. Both applied their expertise successfully in concrete use cases. Each partner is knowledgeable in the skills of the other partner which will guarantee a smooth collaboration.

IMT Atlantique participates in the project through the STACK Inria team. Hélène Coullon will lead WP0, WP3, WP4, and will contribute to all other work packages together with a PhD student and an intern to be hired for the project.

Université d'Orléans participates in the project through the Languages Modeling and Verification (LMV) team of LIFO. Frédéric Loulergue will be involved at 40% of this research time and will lead WP1 and WP2. He will also contribute to all other WPs together with part-time engineers, interns, and a postdoctoral researcher to be hired for the project.

Advisory Board. To collect external feedback and suggestions on a regular basis, we will create an Advisory Board of DevOps experts and industrial users including: Jérémie Monsinjon of OVH, Sébastien Bolle of Orange Labs, and Baptiste Jonglez of the *cfarm* volunteer-run project[7] supported by Tetaneutral and and SPI[8]. Between the acceptance of the proposal and the start of For-CoaLa, we plan to invite other experts and industrial users to reach a board of 5-6 persons.

Involvement of project coordinator and partner leaders in on-going projects. The involvement of the scientific coordinator and partner scientific leaders in regional, national, and international ongoing projects is illustrated in Fig. 4. In this figure, the project fundings are restricted to either IMTA or the LIFO, excluding the funding of their partners in these projects. The projects fundings associated to Hélène Coullon involve other researchers of IMTA. Assuming For-CoaLa will start the 1st of January 2025, the column "person.month"

[7] `https://portal.cfarm.net/`

[8] `https://tetaneutral.net`, `https://www.spi-inc.org/projects/cfarm/`

Researcher	**Pers. month**	**Call, funding agency, grant allocated**	**Projet's title**	**Coordinator**	**Start–End**
H. Coullon	15 (15)	PEPR 730k€	PEPR Cloud Taranis	A. Lebre	10/23–09/30
H. Coullon	12 (1)	ANR PRCE 230 k€	SeMaFoR	T. Ledoux	03/21–02/25
H. Coullon	3 (0)	BPI France 1,2M€	OTPaaS	CEA	10/21–04/25
H. Coullon	1.5 (0)	Défi OVH/Inria 55k€	Frugal Cloud	L. Lefevre	10/21–09/25
F. Loulergue	18 (6)	ANR PRCE, 107k€	CoMeMoV	F. Loulergue	01/23–12/25
F. Loulergue	3 (0)	Région CVL, 25k€	SIOMediC	P. Clemente	12/22–12/24
F. Loulergue	3 (1)	Région CVL, 51k€	AcceptAlgo	B. Boulu-Reshef	01/24–12/25

Figure 4: Involvement of the scientific coordinator and partner leaders in on-going projects. Assuming FOR-COALA would start the 1st of January 2025, the column “person.month” indicates the total effort planned in the given project as well as the remaining effort from this date (in parenthesis)

indicates the total effort planned in the given project as well as the remaining effort from this date (in parenthesis).

2.2 Implemented and requested resources to reach the objectives

Partner 1: IMT Atlantique

Staff expenses IMT researchers will devote 66 person.month to the project including 24 person.month of the scientific leader (50% of her research time), 36 person.month for a PhD student (138,000€), and 6 person.month for Master 2 internship (4,200€). The PhD student will be involved within all work-packages of FOR-COALA and will mainly contribute to WP3 and WP4. However, contributing to WP4 will require a good level of expertise in WP2 and more generally in Ansible. This means that the PhD student will also actively work on WP2 and collaborate with other members of the project. It is also likely that the PhD student will sporadically contribute to WP0 and WP1 during the three years. The intern will ideally be the same person as the PhD student. The goal of the internship will be to gain expertise in both Ansible and Concerto and to have a first small contribution to WP3 before starting the PhD.

Instruments and material costs IMT will acquire one laptop and one screen (3,000€) dedicated to the PhD.

Building and ground costs, outsourcing/subcontracting None

General and administrative costs & other operating expenses Travel costs (22,500€) include expenses for (1) travels from Nantes to Orléans, with one travel per year for two persons (4,500€); and (2) travels for participation in three to four top-tier international conferences (10,000€), two national conferences or workshops including GDR meetings (3,000€). We also plan to partly cover expenses for the organization of a FOR-COALA workshop and meetings with the advisory board (5,000€). General costs 24,316.50€ include administrative management and structure costs. This amount is within the structure costs allowed by the ANR.

Partner 2: LIFO – Université d'Orléans

Staff expenses LIFO will devote 56.2 person.month to the project including 19.2 person.month of the scientific leader (40% of his research time), 24 person.month of a post-doctoral researcher (99,317.05€). Two master's students will also be part of the project. One student during the two first years of the project, and one during the two last years. During the first year of their Master's program, they will be offered a 3 months summer internship, and during the second year (in the research track) they will be offered a 6 months internship

		Partner IMT	**Partner** LIFO
Staff expenses		224,853.36€	254,516.78€
Instruments and material costs (including the scientific consumables)		3,000.00€	3,000.00€
Building and ground costs		–	–
Outsourcing / subcontracting		–	–
General and administrative costs & other operating expenses	Travel costs	22,500.00€	16,200.00€
	Administrative management & structure costs	24,316.50€	21,082.07€
Sub-total		**274,669.86€**	**294,798.85€**
Requested ANR support		**192,016.50€**	**166,475.65€**
Total requested ANR support		**358,492.15€**	

Figure 5: Requested means by item of expenditure and by partner

(basically working on their Master's thesis). In our Master's program, the defense of the thesis occurs at the end of June. As most PhD funding starts in October, we will hire them as research engineers for 2.5 months each. The cost of the internships is 12,222€ and the cost of the 5 person.month of research engineer is 14,654.53€.

All members of the LIFO team will contribute to WP0, in particular to the blog of the project. The post-doctoral researcher will contribute mainly to WP2 but also participate in WP3. The first Master's student will work before T0+24 and will mainly contribute to WP2 but also get familiar with ANSIBLE by contributing to the study of ANSIBLE in WP1. The second Master's student will work after T0+24 and will mainly contribute to WP4. However, as WP4 relies on the semantics of ANSIBLE (WP2) and CONCERTO (WP3), this second Master's student will start by contributing to these work-packages.

Instruments and material costs LIFO will acquire a powerful laptop dedicated to development (3,000€).

Building and ground costs, outsourcing/subcontracting None

General and administrative costs & other operating expenses Travel and registration costs of 16,200€ include expenses of 13,000€ for participation in 4 leading international conferences (1 per year): 2 outside Europe and 2 in Europe, as well as 3 national conferences, 4 national scientific workshops (most probably of the GdR GPL), and expenses of 2,200€ for project meetings in Nantes (6 meetings but 11 two-day stays in Nantes as several members of the LIFO team will participate). During the project, two meetings will be organized in Orléans. General costs also include administrative management and structure costs of 21,082.07€ (14.5% of the other costs). This amount is within the structure costs allowed by the ANR for a public university.

Requested resources by item of expenditure and by partner: see Fig. 5 All in all, the complete cost of the project is 569,468.71€, and the FOR-COALA consortium kindly asks for **358,492.15€** of ANR funding.

3 IMPACT AND BENEFITS OF THE PROJECT

Scientific impact and dissemination. From a scientific standpoint, the FOR-COALA project results will provide three main contributions:

- a new mechanically formalized and certified DevOps configuration management tool that covers a subset of ANSIBLE, the currently most used DevOps tool;
- a mechanically formalized and certified reconfiguration tool that covers the CONCERTO academic model;

• a semantics preserving transformation to switch from ANSIBLE to CONCERTO.

The scientific impacts of these contributions are as follows. First, FOR-COALA is, as far as we know, the first project to contribute to formally verified DevOps tools which offers strong and mechanized guarantees at the whole language level, true for any given program. This is complementary to tools targeting the quality improvement of the user codes.

The formalized semantics of ANSIBLE will initially be a selected subset of ANSIBLE (in particular a subset of ANSIBLE modules) that could be easily enlarged after the project lifespan. The mechanized formal semantics of ANSIBLE will be designed in a modular way and in such a way that the level of abstraction at which the system being configured is considered could be adapted to other verification needs than the needs of FOR-COALA. The impact can therefore be important as these semantics could be used in further formalization and verification efforts, for example to verify the correctness of static analysis tools of ANSIBLE programs, refactoring of ANSIBLE programs, provenance tracking for debugging [29] or security [30], etc. We envision that the semantics of ANSIBLE modules we will propose in FOR-COALA will be flexible enough to play the role of *specifications* for a possible effort in the verification of the *implementation* of ANSIBLE modules.

Second, the academic tool CONCERTO offers native parallel and concurrent constructs for reconfiguration that is more difficult to formalize and certify but of interest to companies to reduce the execution time of deployments and maintenance procedures. FOR-COALA will solve this challenge.

Finally, the transformation mechanisms, which will also be mechanically formalized and certified, will offer a way for DevOps users to semi-automatically and safely transform their infrastructure codes towards an academic tool with better efficiency.

We plan to publish at the crossroads of two different domains. First, as contributing to DevOps and reconfiguration languages, we plan to publish in the Software Engineering (SE) and languages domains, as already done by partners [31]. Some high-ranking SE conferences with themes or tracks related to DevOps are, for instance, ICSE, SANER, ASE, ESEC/FSE. Second, as contributing to applied formal methods we also target conferences such as FM, iFM [26], SEFM [23], ITP [20], CPP, the federated conferences ETAPS [9] or DisCoTec, etc. Note that if developing our work to some specific use cases, the distributed system community could also be interested in our work, such as CCGrid where the project leader has already published about CONCERTO [7]. International journals such as FAC, JSS, TSE, TOSEM, JLAMP, etc. will also be targeted for extended contributions.

Valorisation strategy and industrial benefits. Our goal is to have an impact on the DevOps community. Our chosen strategy is to build our contributions according to the advice of multiple partner companies (advisory board) as well as from the open-source community. We also plan to build our initial dataset of ANSIBLE features from statistical mining from the public repository ANSIBLE Galaxy. We think that our contributions could be of interest to many companies using ANSIBLE that handle critical software systems evolution through Infrastructure-as-code and would like to have guarantees, but also many companies interested in speeding up their ANSIBLE codes.

To build an open-source community associated with our tools, we plan to participate in and give talks to DevOps major events such as “Config Management Camp”, “KubeCon”, and “OpenInfra Summit” for instance. We also plan to build a technical blog to explain the ambiguities of ANSIBLE and efficiency comparison with CONCERTO to illustrate the interest of our project. Finally, we plan to submit at least one vulgarization paper related to our contributions. It’s worth noting that it is difficult to sustain an open-source community with non-permanent resources such as those funded by the ANR. Partners will therefore study how to invest longer-term efforts in this community. The STACK team, for example, has a permanent research engineer dedicated to the transfer to open source.

Finally, the initial goal of our advisory board is to help us restrict our study to a subset of ANSIBLE features in WP1. However, it is also possible that some members of the advisory board become users of our certified

ANSIBLE and CONCERTO, or use our verified transformation tool to easily try CONCERTO, or even contribute to our open source projects.

Education. The two permanent researchers of FOR-COALA are also professors in their respective host institutions (IMT Atlantique and Université d'Orléans). In this context, they give several types of courses, as part of Master's degree programs or as speakers in summer schools dedicated to young researchers, on the main scientific and technical domains of the project: e.g., DevOps, Cloud, formal methods. FOR-COALA will contribute to a finer-grain understanding of ANSIBLE semantics. The ambiguities of these semantics can then be taught to students so that they can write better-quality playbooks. Furthermore, thanks to FOR-COALA, it will be possible to raise the awareness of students to the importance of formal semantics, mechanized semantics, and verification in DevOps and Cloud courses, which represents interesting knowledge for companies.

4 References related to the project

[1] H. Coullon, L. Henrio, F. Loulergue, and S. Robillard. "Component-Based Distributed Software Reconfiguration: A Verification-Oriented Survey". In: *ACM Comput. Surv.* (2023). DOI: 10.1145/3595376.

[2] H. S. Gunawi et al. "Why Does the Cloud Stop Computing? Lessons from Hundreds of Service Outages". In: *ACM Symposium on Cloud Computing*. 2016. DOI: 10.1145/2987550.2987583.

[3] X. Leroy. "Formal verification of a realistic compiler". In: *Commun. ACM* 52.7 (2009), pp. 107–115. DOI: 10.1145/1538788.1538814.

[4] J.-H. Jourdan, V. Laporte, S. Blazy, X. Leroy, and D. Pichardie. "A Formally-Verified C Static Analyzer". In: *SIGPLAN Not.* (2015). DOI: 10.1145/2775051.2676966.

[5] R. Di Cosmo, J. Mauro, S. Zacchiroli, and G. Zavattaro. "Aeolus: a Component Model for the Cloud". In: *Information and Computation* 239 (Jan. 2014), pp. 100–121. DOI: 10.1016/j.ic.2014.11.002.

[6] P. Anderson and H. Herry. "A Formal Semantics for the SmartFrog Configuration Language". In: *J. Netw. Syst. Manag.* 24.2 (2016), pp. 309–345. DOI: 10.1007/s10922-015-9351-y.

[7] M. Chardet, H. Coullon, and C. Pérez. "Predictable Efficiency for Reconfiguration of Service-Oriented Systems with Concerto". In: *CCGrid*. IEEE, 2020. DOI: 10.1109/CCGrid49817.2020.00-59.

[8] M. Chardet, H. Coullon, and S. Robillard. "Toward safe and efficient reconfiguration with Concerto". In: *Sci Comput Program* (2021). hal: hal-03103714. DOI: 10.1016/j.scico.2020.102582.

[9] S. Robillard and H. Coullon. "SMT-Based Planning Synthesis for Distributed System Reconfigurations". In: *Fundamental Approaches to Software Engineering (FASE)*. 2022. DOI: 10.1007/978-3-030-99429-7_15.

[10] K. Ikeshita, F. Ishikawa, and S. Honiden. "Test Suite Reduction in Idempotence Testing of Infrastructure as Code". In: *Tests and Proofs*. Ed. by S. Gabmeyer and E. B. Johnsen. Cham: Springer International Publishing, 2017, pp. 98–115. ISBN: 978-3-319-61467-0.

[11] R. Opdebeeck, A. Zerouali, and C. De Roover. "Smelly Variables in Ansible Infrastructure Code: Detection, Prevalence, and Lifetime". In: *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*. 2022. DOI: 10.1145/3524842.3527964.

[12] R. Shambaugh, A. Weiss, and A. Guha. "Rehearsal: A Configuration Verification Tool for Puppet". In: *SIGPLAN Not.* 51.6 (2016). DOI: 10.1145/2980983.2908083.

[13] R. Opdebeeck, A. Zerouali, and C. De Roover. "Andromeda: A Dataset of Ansible Galaxy Roles and Their Evolution". In: *IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. 2021. DOI: 10.1109/MSR52588.2021.00078.

[14] W. Fu, R. Perera, P. Anderson, and J. Cheney. "muPuppet: A Declarative Subset of the Puppet Configuration Language". In: *ECOOP*. LIPIcs. 2017. DOI: 10.4230/LIPIcs.ECOOP.2017.12.

[15] N. Jeannerod, C. Marché, and R. Treinen. "A Formally Verified Interpreter for a Shell-Like Programming Language". In: *VSTTE*. Vol. 10712. LNCS. Springer, 2017. DOI: 10.1007/978-3-319-72308-2_1.

[16] B. Becker, N. Jeannerod, C. Marché, Y. Régis-Gianas, M. Sighireanu, and R. Treinen. “Analysing installation scenarios of Debian packages”. In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer, 2020. DOI: `10.1007/978-3-030-45237-7_14`.

[17] The Coq Development Team. *The Coq Proof Assistant*. `http://coq.inria.fr`.

[18] J. Filliâtre and A. Paskevich. “Why3 - Where Programs Meet Provers”. In: *Programming Languages and Systems (ESOP)*. Springer, 2013. DOI: `10.1007/978-3-642-37036-6_8`.

[19] J.-H. Jourdan, F. Pottier, and X. Leroy. “Validating LR(1) Parsers”. In: *Programming Languages and Systems (ESOP)*. Springer, 2012. DOI: `10.1007/978-3-642-28869-2_20`.

[20] K. Emoto, F. Loulergue, and J. Tesson. “A Verified Generate-Test-Aggregate Coq Library for Parallel Programs Extraction”. In: *Interactive Theorem Proving (ITP)*. Springer, 2014. DOI: `10.1007/978-3-319-08970-6_17`.

[21] A. Blanchard, N. Kosmatov, M. Lemerre, and F. Loulergue. “conc2seq: A Frama-C Plugin for Verification of Parallel Compositions of C Programs”. In: *16th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*. 2016. DOI: `10.1109/SCAM.2016.18`.

[22] A. Blanchard, F. Loulergue, and N. Kosmatov. “From Concurrent Programs to Simulating Sequential Programs: Correctness of a Transformation”. In: *International Workshop on Verification and Program Transformation*. 2017. DOI: `10.4204/EPTCS.253.9`.

[23] O. Proust and F. Loulergue. “Verified Scalable Parallel Computing with Why3”. In: *Software Engineering and Formal Methods (SEFM)*. Springer, 2023. DOI: `10.1007/978-3-031-47115-5_14`.

[24] M. Chardet, H. Coullon, C. Pérez, D. Pertin, C. Servantie, and S. Robillard. “Enhancing Separation of Concerns, Parallelism, and Formalism in Distributed Software Deployment with Madeus”. working paper or preprint. June 2020. URL: `https://inria.hal.science/hal-02737859`.

[25] D. Ly, N. Kosmatov, F. Loulergue, and J. Signoles. “Sound Runtime Assertion Checking for Memory Properties via Program Transformation”. In: *Form. Asp. Comput.* (2023). DOI: `10.1145/3605951`.

[26] H. Coullon, C. Jard, and D. Lime. “Integrated Model-checking for the Design of Safe and Efficient Distributed Software Commissioning”. In: *Integrated Formal Methods (iFM)*. LNCS. Springer, 2019. DOI: `10.1007/978-3-030-34968-4_7`.

[27] A. Alidra, H. Bruneliere, H. Coullon, T. Ledoux, C. Prud’Homme, J. Lejeune, P. Sens, J. Sopena, and J. Rivalan. “SeMaFoR - Self-Management of Fog Resources with Collaborative Decentralized Controllers”. In: *SEAMS*. IEEE, 2023. DOI: `10.1109/SEAMS59076.2023.00014`.

[28] F. Dabrowski, F. Loulergue, and T. Pinsard. “A Formal Semantics of Nested Atomic Sections with Thread Escape”. In: *Comput Lang Syst Str* (2015). DOI: `10.1016/j.cl.2015.04.001`.

[29] W. Fu. “Semantics and provenance of configuration programming language μPuppet”. PhD thesis. University of Edinburgh, UK, 2019. URL: `https://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.798916`.

[30] P. Anderson and J. Cheney. “Toward Provenance-Based Security for Configuration Languages”. In: *4th Workshop on the Theory and Practice of Provenance (TaPP)*. USENIX Association, 2012. URL: `https://www.usenix.org/conference/tapp12/workshop-program/presentation/anderson`.

[31] J. Philippe, A. Omond, H. Coullon, C. Prud’Homme, and I. Raïs. “Fast Choreography of Cross-DevOps Reconfiguration with Ballet: A Multi-Site OpenStack Case Study”. In: *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2024. URL: `https://hal.science/hal-04457484`.