RESEARCH CENTRE

Inria Center
at Université Côte d'Azur

IN PARTNERSHIP WITH:

Université de Bologne (Italie)

2022
ACTIVITY REPORT

Team
FOCUS

Foundations of Component-based Ubiquitous Systems

Inria teams are typically groups of researchers working on the definition of a common project, and objectives, with the goal to arrive at the creation of a project-team. Such project-teams may include other partners (universities or research institutions)

IN COLLABORATION WITH: Dipartimento di Informatica - Scienza e Ingegneria (DISI), Universita' di Bologna

DOMAIN

Networks, Systems and Services, Distributed Computing

THEME

Distributed programming and Software engineering

Inria

Contents

Team FOCUS

Creation of the Team: 2022 June 22

Keywords

Computer sciences and digital sciences

A1. – Architectures, systems and networks

A1.3. – Distributed Systems

A1.4. – Ubiquitous Systems

A2.1.1. – Semantics of programming languages

A2.1.6. – Concurrent programming

A2.1.7. – Distributed programming

A2.4.3. – Proofs

Other research topics and application domains

B6.1. – Software industry

B6.3. – Network functions

B6.4. – Internet of things

B9.5.1. – Computer science

1 Team members, visitors, external collaborators

Research Scientists

- Martin Avanzini [INRIA, Researcher]
- Saverio Giallorenzo [UNIV BOLOGNE]

Faculty Members

- Davide Sangiorgi [Team leader, UNIV BOLOGNE, Professor]
- Mario Bravetti [UNIV BOLOGNE, Professor]
- Ugo Dal Lago [UNIV BOLOGNE, Professor]
- Maurizio Gabbrielli [UNIV BOLOGNE, Professor]
- Ivan Lanese [UNIV BOLOGNE, Associate Professor]
- Cosimo Laneve [UNIV BOLOGNE, Professor]
- Simone Martini [UNIV BOLOGNE, Professor, HDR]
- Gianluigi Zavattaro [UNIV BOLOGNE, Professor]

Post-Doctoral Fellows

- Francesco Gavazzo [UNIV BOLOGNE, until Jun 2022]
- Michael Lodi [UNIV BOLOGNE]
- Paolo Pistone [UNIV BOLOGNE, until Jul 2022]
- Ken Sakayori [UNIV BOLOGNE]
- Riccardo Treglia [UNIV BOLOGNE]
- Gabriele Vanoni [INRIA, from Sep 2022]
- Stefano Zingaro [UNIV BOLOGNE]

PhD Students

- Melissa Antonelli [UNIV BOLOGNE]
- Andrea Colledan [UNIV BOLOGNE]
- Enguerrand Prebet [ENS DE LYON]
- Adele Veschetti [UNIV BOLOGNE]

Administrative Assistant

- Christine Claux [INRIA]

External Collaborators

- Claudio Guidi [Italiana Software]
- Daniel Hirschkoff [ENS DE LYON]
- Fabrizio Montesi [University of Southern Denmark]

2 Overall objectives

Ubiquitous Computing refers to the situation in which computing facilities are embedded or integrated into everyday objects and activities. Networks are large-scale, including both hardware devices and software agents. The systems are highly mobile and dynamic: programs or devices may move and often execute in networks owned and operated by others; new devices or software pieces may be added; the operating environment or the software requirements may change. The systems are also heterogeneous and open: the pieces that form a system may be quite different from each other, built by different people or industries, even using different infrastructures or programming languages; the constituents of a system only have a partial knowledge of the overall system, and may only know, or be aware of, a subset of the entities that operate on the system.

A prominent recent phenomenon in Computer Science is the emerging of interaction and communication as key architectural and programming concepts. This is especially visible in ubiquitous systems. Complex distributed systems are being thought of and designed as structured composition of computational units, usually referred to as *components*. These components are supposed to interact with each other and such interactions are supposed to be orchestrated into conversations and dialogues. In the remainder, we will write *CBUS* for Component-Based Ubiquitous Systems.

In CBUS, the systems are complex. In the same way as for complex systems in other disciplines, such as physics, economics, biology, in CBUS theories are needed that allow us to understand the systems, to design or program them, and to analyze them.

Focus investigates the semantic foundations for CBUS. The foundations are intended as instrumental to formalizing and verifying important computational properties of the systems, as well as to proposing linguistic constructs for them. Prototypes are developed to test the implementability and usability of the models and the techniques. Throughout our work, 'interaction' and 'component' are central concepts.

The members of the project have a solid experience in algebraic and logical models of computation, and related techniques, and this is the basis for our study of ubiquitous systems. The use of foundational models inevitably leads to opportunities for developing the foundational models themselves, with particular interest for issues of expressiveness and for the transplant of concepts or techniques from a model to another one.

3 Research program

3.1 Foundations 1: Models

The objective of Focus is to develop concepts, techniques, and possibly also tools, that may contribute to the analysis and synthesis of CBUS. Fundamental to these activities is *modeling*. Therefore designing, developing and studying computational models appropriate for CBUS is a central activity of the project. The models are used to formalise and verify important computational properties of the systems, as well as to propose new linguistic constructs.

The models we study are in the process calculi (e.g., the π-calculus) and λ-calculus tradition. Such models, with their emphasis on algebra, address properly compositionality—a central property in our approach to problems. Accordingly, the techniques we employ are mainly operational techniques based on notions of behavioural equivalence, and techniques based on algebra, mathematical logics, and type theory.

3.2 Foundations 2: Foundational calculi and interaction

Modern distributed systems have witnessed a clear shift towards interaction and conversations as basic building blocks for software architects and programmers. The systems are made by components, that are supposed to interact and carry out dialogues in order to achieve some predefined goal; Web services are a good example of this. Process calculi are models that have been designed precisely with the goal of understanding interaction and composition. The theory and tools that have been developed on top of process calculi can set a basis with which CBUS challenges can be tackled. Indeed, industrial proposals of languages for Web services such as BPEL are strongly inspired by process calculi, notably the π-calculus.

3.3 Foundations 3: Type systems and logics

Type systems and logics for reasoning on computations are among the most successful outcomes in the history of the research in λ-calculus and (more recently) in process calculi. Type systems can also represent a powerful means of specifying dialogues among components of CBUS. For instance—again referring to Web services—current languages for specifying interactions only express basic connectivity, ignoring causality and timing aspects (e.g., an intended order on the messages), and the alternative is to use Turing Complete languages that are however undecidable. Types can come in handy here: they can express causality and order information on messages [48, 46, 49], while remaining decidable systems.

3.4 Foundations 4: Implicit computational complexity

A number of elegant and powerful results have been obtained in implicit computational complexity concerning the λ-calculus, where ideas from Linear Logics enable a fine-grained control over computations. This experience can be profitable when tackling issues of CBUS related to resource consumption, such as resource allocation, access to resources, certification of bounds on resource consumption (e.g., ensuring that a service will answer to a request in time polynomial with respect to the size of the input data).

4 Application domains

4.1 Ubiquitous Systems

The main application domain for Focus are ubiquitous systems, i.e. systems whose distinctive features are: mobility, high dynamicity, heterogeneity, variable availability (the availability of services offered by the constituent parts of a system may fluctuate, and similarly the guarantees offered by single components may not be the same all the time), open-endedness, complexity (the systems are made by a large number of components, with sophisticated architectural structures). In Focus we are particularly interested in the following aspects.

- *Linguistic primitives* for programming dialogues among components.
- *Contracts* expressing the functionalities offered by components.
- *Adaptability and evolvability* of the behaviour of components.
- *Verification* of properties of component systems.
- Bounds on component *resource consumption* (e.g., time and space consumed).

4.2 Service Oriented Computing and Cloud Computing

Today the component-based methodology often refers to Service Oriented Computing. This is a specialized form of component-based approach. According to W3C, a service-oriented architecture is “a set of components which can be invoked, and whose interface descriptions can be published and discovered”. In the early days of Service Oriented Computing, the term “services” was strictly related to that of Web Services. Nowadays, it has a much broader meaning as exemplified by the XaaS (everything as a service) paradigm: based on modern virtualization technologies, Cloud computing offers the possibility to build sophisticated service systems on virtualized infrastructures accessible from everywhere and from any kind of computing device. Such infrastructures are usually examples of sophisticated service oriented architectures that, differently from traditional service systems, should also be capable to elastically adapt on demand to the user requests.

5 Highlights of the year

5.1 Awards

- The paper "Session types revisited"[47], co-authored by D. Sangiorgi, has received the "10 Year Most Influential Paper Award", which is awarded each year to the paper from the PPDP proceedings 10 years earlier, and intended to "recognize the authors contribution to PPDP's influence in the area of declarative programming".
- I. Lanese received the ESOP'22 "distinguished reviewer award".
- The paper [10], co-authored by G. Vanoni and U. Dal Lago, received the distinguished paper award at ICFP'22.

6 New software and platforms

6.1 New software

6.1.1 JOLIE

Name: Java Orchestration Language Interpreter Engine

Keyword: Microservices

Scientific Description: Jolie enforces a strict separation of concerns between behaviour, describing the logic of the application, and deployment, describing the communication capabilities. The behaviour is defined using the typical constructs of structured sequential programming, communication primitives, and operators to deal with concurrency (parallel composition and input choice). Jolie communication primitives comprise two modalities of interaction typical of Service-Oriented Architectures (SOAs), namely one-way (sends an asynchronous message) and request-response (sends a message and waits for an answer). A main feature of the Jolie language is that it allows one to switch among many communication media and data protocols in a simple, uniform way. Since it targets the field of SOAs, Jolie supports the main communication media (TCP/IP sockets, Bluetooth L2CAP, Java RMI, and Unix local sockets) and data protocols (HTTP, JSON-RPC, XML-RPC, SOAP and their respective SSL versions) from this area.

Functional Description: Jolie is a language for programming service-oriented and microservice applications. It directly supports service-oriented abstractions such as service, port, and session. Jolie allows to program a service behaviour, possibly obtained by composing existing services, and supports the main communication protocols and data formats used in service-oriented architectures. Differently from other service-oriented programming languages such as WS-BPEL, Jolie is based on a user-friendly Java-like syntax (more readable than the verbose XML syntax of WS-BPEL). Moreover, the kernel of Jolie is equipped with a formal operational semantics. Jolie is used to provide proof of concepts around Focus activities.

Release Contributions: There are many fixes to the HTTP extension, improvements to the embedding engine for Javascript programs, and improvements to the support tools jolie2java and wsdl2jolie.

News of the Year: In 2022 Jolie saw 4 minor releases (1.10.7, 1.10.10, 1.10.12, 1.10.13) and the transition to the alpha and beta major release 1.11.0. The new major release includes many extensions, comprising both the language and its runtime. One main language extension regards the introduction of a new functional notation for operations. This is syntactic sugar for request-response operations so that they behave similarly to functions. For example, instead of writing 'contains@StringUtils(...)(resp), if (resp){ ... }' one can write 'if(contains@StringUtils(...)){ ... }', removing the need for the bypass variable 'resp' and resulting in a more concise and easier-to-follow logic. Runtime extensions include advanced, programmer-friendly error messages. This is both from the point of view of error visualisation (e.g., tracing of the flow of faults among services, modulation of the amount of detail the runtime provides on these) and of suggestions to fix errors (e.g., by reporting

possible spelling mistakes and their possible fixes). Other runtime updates regarded the transition of more pieces of the standard library from the old module system ('include') to the new one ('from ... import ...'). Minor releases mainly regarded optimisations (e.g., releasing unused cache memory) or fixing bugs in the runtime or in extensions. Further effort has been put also into the development of the Jolie Package Manager (jpm). Extensions regarded the inclusion of templates to quickly kickstart new Jolie projects and more integrated support for Docker development environments.

URL: http://www.jolie-lang.org/

Contact: Saverio Giallorenzo

Participants: Claudio Guidi, Fabrizio Montesi, Maurizio Gabbrielli, Saverio Giallorenzo, Ivan Lanese, Stefano Zingaro

6.1.2 NightSplitter

Keyword: Constraint-based programming

Functional Description: Nightsplitter deals with the group preference optimization problem. We propose to split users into subgroups trying to optimize members' satisfaction as much as possible. In a large city with a huge volume of activity information, designing subgroup activities and avoiding time conflict is a challenging task. Currently, the Demo is available only for restaurant and movie activities in the city of Paris.

URL: http://cs.unibo.it/t.liu/nightsplitter/

Contact: Maurizio Gabbrielli

6.1.3 CauDEr

Name: Causal-consistent Debugger for Erlang

Keywords: Debug, Reversible computing

Scientific Description: The CauDEr reversible debugger for Erlang is based on the theory of causal-consistent reversibility, which states that any action can be undone provided that its consequences, if any, are undone beforehand. This theory relies on a causal semantics for the target language, and can be used even if different processes have different notions of time. Replay is based on causal-consistent replay, which allows one to replay any future action, together with all and only its causes.

Functional Description: CauDEr is a debugger allowing one to explore the execution of concurrent and distributed Erlang programs both forward and backward. Notably, when going backward, any action can be undone provided that its consequences, if any, are undone beforehand. The debugger also provides commands to automatically find relevant past actions (e.g., send of a given message) and undo them, including their consequences. Forward computation can be driven by a log taken from a computation in the standard Erlang/OTP environment. An action in the log can be selected and replayed together with all and only its causes. The debugger enables one to find a bug by following the causality links from the visible misbehaviour to the bug.

News of the Year: In 2022 the development focused on the integration of imperative primitives to manage maps of atoms to pids. We plan to integrate it into CauDEr in early 2023. Apart from this we did basic maintenance.

URL: https://github.com/mistupv/cauder

Publications: hal-03005383v1, hal-01912894v1, hal-02313745v1

Contact: Ivan Lanese

Participant: Ivan Lanese

Partner: Universitat Politècnica de València

6.1.4 SUNNY-AS

Name: SUNNY FOR ALGORITHM SELECTION

Keywords: Optimisation, Machine learning

Functional Description: SUNNY-AS is a portfolio solver derived from SUNNY-CP for Algorithm Selection Problems (ASLIB). The goal of SUNNY-AS is to provide a flexible, configurable, and usable portfolio solver that can be set up and executed just like a regular individual solver.

URL: https://github.com/lteu/oasc

Contact: Maurizio Gabbrielli

6.1.5 eco-imp

Name: Expected Cost Analysis for Imperative Programs

Keywords: Software Verification, Automation, Runtime Complexity Analysis, Randomized algorithms

Functional Description: Eco-imp is a cost analyser for probabilistic and non-deterministic imperative programs. Particularly, it features dedicated support for sampling from distributions, and can thereby accurately reason about the average case complexity of randomized algorithms, in a fully automatic fashion. The tool is based on an adaptation of the ert-calculus of Kaminski et al., extended to the more general setting of cost analysis where the programmer is free to specify a (non-uniform) cost measure on programs. The main distinctive feature of eco-imp, though, is the combination of this calculus with an expected value analysis. This provides the glue to analyse program components in complete independence, that is, the analysis is modular and thus scalable. As a consequence, confirmed by our experiments, eco-imp runs on average orders of magnitude faster than comparable tools: execution times of several seconds become milliseconds.

News of the Year: In 2022, the tool has seen a major overhaul in its inference machinery so as to account for recursive procedures. It has also been extended from a reasoning tool about costs to general expectations.

URL: http://www-sop.inria.fr/members/Martin.Avanzini/software/eco-imp/

Publication: hal-03013544

Contact: Martin Avanzini

6.1.6 PRISM+

Keyword: Stochastic process

Functional Description: PRISM is a probabilistic model checker, a tool for formal modelling and analysis of systems that exhibit random or probabilistic behaviour. We extend the language in order to model the Bitcoin system. The tool now supports three dynamic data types: block, ledger and list. As a consequence, it is now possible to perform simulations and analyse transient probabilities, i.e. probabilities that are dependent on time, for the Bitcoin protocol. It has been used to understand how the system changes during the execution and to analyse the probabilities of reaching an inconsistent state in different settings.

URL: https://github.com/adeleveschetti/bitcoin-analysis

Contact: Adele Veschetti

6.1.7 Tquery

Keywords: Ephemeral Data, Microservices, Big data, Querying

Scientific Description: The adoption of edge/fog systems and the introduction of privacy-preserving regulations compel the usage of tools for expressing complex data queries in an ephemeral way—ensuring the queried data does not persist.

Database engines partially address this need, as they provide domain-specific languages for querying data. Unfortunately, using a database in an ephemeral setting has inessential issues related to throughput bottlenecks, scalability, dependency management, and security (e.g., query injection). Moreover, databases can impose specific data structures and data formats, which can hinder the development of microservice architectures that integrate heterogeneous systems and handle semi-structured data.

Tquery is the first query framework designed for ephemeral data handling in microservices. Tquery joins the benefits of a technology-agnostic, microservice-oriented programming language, Jolie, and of one of the most widely-used query languages for semi-structured data in microservices, the MongoDB aggregation framework. With Tquery, users express in a terse syntax how to collect data from heterogeneous sources and how to query it in local memory, defining pipelines of high-level operators. The development of Tquery follows a "cleanroom software engineering process", based on the definition of a theory for querying semi-structured data compatible with Jolie and inspired by a consistent variant of the key operators of the MongoDB aggregation framework.

Functional Description: Tquery is a query framework integrated into the Jolie language for the data handling/querying of Jolie trees.

Tquery is based on a tree-based instantiation (language and semantics) of MQuery, a formalisation of a sound fragment of the Aggregation Framework, the query language of the most popular document-oriented database: MongoDB.

Tree-shaped documents are the main format in which data flows within modern digital systems - e.g., eHealth, the Internet-of-Things, and Edge Computing. Tquery is particularly suited to develop real-time, ephemeral scenarios, where data shall not persist in the system.

Release Contributions: first release

News of the Year: In 2022, Tquery transitioned from the 0.x to the 1.x version. This marked the passage from the "research" to the "development" stage of the library. Indeed, the main effort dedicated to the transition has been the optimisation of the query engine, to maximise its performance and the implementation of a thorough test suite to compare Tquery with alternatives, first of all, MongoDB.

URL: https://github.com/jolie/tquery

Contact: Saverio Giallorenzo

Partner: University of Southern Denmark

6.1.8 APP

Name: Allocation Priority Policies

Keywords: Serverless, Scheduling, Cloud computing, Optimisation

Scientific Description: APP addresses the problem of function execution scheduling, i.e., how to schedule the execution of Serverless functions to optimise their performance against some user-defined goals, by specifying policies that inform the scheduling of function execution.

Functional Description: Serverless computing is a Cloud development paradigm where developers write and compose stateless functions, abstracting from their deployment and scaling.

APP is a declarative language of Allocation Priority Policies to specify policies that inform the scheduling of Serverless function execution to optimise their performance against some user-defined goals.

APP is currently implemented as a prototype extension of the Serverless Apache OpenWhisk platform.

Release Contributions: first release

News of the Year: In 2022, work on APP mainly regarded the formalisation of the language, both to formally reason on its semantics (e.g., on the complexity of schedulings) and to provide a standard, non-ambiguous definition of policy behaviours. This initiated a beneficial revision process that helped us consolidate APP into a more consistent language — which also lead to refinements of its runtime — and gave us useful insights for its evolution.

URL: https://github.com/giusdp/openwhisk

Contact: Saverio Giallorenzo

6.1.9 Choral

Keywords: Choreographic Programming, Compilation, Modularity, Distributed programming

Scientific Description: In essence, Choral developers program a choreography with the simplicity of a sequential program. Then, through the Choral compiler, they obtain a set of programs that implement the roles acting in the distributed system. The generated programs coordinate in a decentralised way and they faithfully follow the specification from their source choreography, avoiding possible incompatibilities arising from discordant manual implementations. Programmers can use or distribute the single implementations of each role to their customers with a higher level of confidence in their reliability. Moreover, they can reliably compose different Choral(-compiled) programs, to mix different protocols and build the topology that they need.

Choral currently interoperates with Java (and it is planned to support also other programming languages) at three levels: 1) its syntax is a direct extension of Java (if you know Java, Choral is just a step away), 2) Choral code can reuse Java libraries, 3) the libraries generated by Choral are in pure Java with APIs that the programmer controls, and that can be used inside of other Java projects directly.

Functional Description: Choral is a language for the programming of choreographies. A choreography is a multiparty protocol that defines how some roles (the proverbial Alice, Bob, etc.) should coordinate with each other to do something together.

Choral is designed to help developers program distributed authentication protocols, cryptographic protocols, business processes, parallel algorithms, or any other protocol for concurrent and distributed systems. At the press of a button, the Choral compiler translates a choreography into a library for each role. Developers can use the generated libraries to make sure that their programs (like a client, or a service) follow the choreography correctly. Choral makes sure that the generated libraries are compliant implementations of the source choreography.

Release Contributions: First release

News of the Year: In 2022, Choral moved to version 0.1.1. While this is a minor release, it includes many quality-of-life improvements for developers that use the language. These comprise: a revision of the class system of the Choral runtime library into a more consistent set of elements that users can combine (e.g., channels, data structures, etc), the extension of the Choral runtime, mainly importing frequently-used libraries from the Java standard library, better reporting of compilation errors, with additional information on how to fix bugs in Choral classes. The release also includes a consolidation of the codebase and minor bug fixes. 2022 saw also the implementation of a fairly big use-case distributed system in Choral (a clone of Twitter), which suggested some of the extensions and helped us spot the bugs mentioned above.

URL: https://www.choral-lang.org/

Contact: Saverio Giallorenzo

Participants: Saverio Giallorenzo, Fabrizio Montesi

Partner: University of Southern Denmark

6.1.10 Corinne

Keywords: Choreography automata, Communicating finite state machines

Scientific Description: Corinne relies on the theory of choreography automata, which is described in:

Franco Barbanera, Ivan Lanese, Emilio Tuosto: Choreography Automata. COORDINATION 2020: 86-106

Franco Barbanera, Ivan Lanese, Emilio Tuosto: Composition of choreography automata. CoRR abs/2107.06727 (2021)

Functional Description: Choreography automata (c-automata) are finite state automata whose transitions are labelled with interactions of the form A -> B : m, representing a communication in which participant A sends a message (of type) m to participant B, and participant B receives it. Corinne allows one to display c-automata represented in the dot format, and:

- project them on communicating finite state machines representing the behaviour of single participants
- compute a product c-automaton corresponding to the concurrent execution of two c-automata
- synchronize two participants of a c-automaton transforming them into couple gateways
- check well-formedness conditions ensuring that the system of participants obtained via projection behaves well

News of the Year: In 2022 Corinne was not further developed.

URL: https://github.com/lanese/corinne-3

Publication: hal-03468190

Contact: Ivan Lanese

Partner: Gran Sasso Science Institute

7 New results

7.1 Service-oriented and Cloud Computing

Participants: Mario Bravetti, Saverio Giallorenzo, Ivan Lanese, Gianluigi Zavattaro.

In Service-Oriented Computing (SOC), systems made of interacting and collaborating components are specified by describing the "local behaviour" of each of them (i.e., the way one component may interact with the others), and/or the "global behaviour" of the system (i.e., the expected interaction protocols that should take place within the system).

From the language perspective, the programming of "local" and "global" behaviours have been respectively addressed with orchestration and choreography languages. Regarding applications, where usually SOC meets Cloud Computing, we find two state-of-the-art architectural styles. Microservices are a revisitation of service-orientated architectures where fine-grained, loosely coupled, independent services help developers assemble reusable, flexible, and scalable architectures. Serverless is a programming style and deployment technique where users program Cloud applications in terms of stateless functions, which execute and scale in proportion to inbound requests.

Foundations of Orchestration and Choreography

Communication is an essential element of modern software, yet programming and analysing communicating systems are difficult tasks. In [25] we tackle the lack of compositional mechanisms that preserve relevant communication properties. This belongs in a strand of work hinged on the usage of sets of communicating components modelled as finite-state machines. We propose a composition mechanism based on gateways that, under some suitable compatibility conditions, guarantees deadlock freedom. Previous work focussed on (a)synchronous symmetric communications (where sender and receiver play the same part in determining which message to exchange). In [25] we provide the same guarantees for the synchronous asymmetric case (where senders decide independently which message should be exchanged).

We also worked on formalisations of choreographic languages. In [24] we introduce a metamodel dubbed formal choreographic languages. Our main motivation is to establish a framework for the comparison and generalisation of standard constructions and properties from the literature. Formal choreographic languages capture existing formalisms for message-passing systems; we detail the cases of multiparty session types and choreography automata. Unlike many other models, formal choreographic languages can naturally model systems exhibiting non-regular behaviour. We also worked on the modelling of real business processes taken from the official documentation of European customs business process models, using a formal choreographic approach [29].

In [35] we extend the theory of choreography automata by equipping communications with assertions on the values that can be communicated, enabling a design-by-contract approach. We provide a toolchain allowing the generation of APIs for TypeScript web programming. Programs communicating via the generated APIs follow, by construction, the prescribed communication pattern and are free from communication errors such as deadlocks.

Microservices, Serverless, and Cloud Deployment

The service-oriented programming language Jolie is a long-standing project within Focus. In [36] we study the usage of model-driven engineering languages to produce Jolie APIs. This allows us to implement a synthesiser that, given a data model (specified in the LEMMA modelling framework), can produce the interfaces of the corresponding data types and service interfaces (APIs) in Jolie. Another work related to Jolie is Tquery [18], the first query framework designed for ephemeral data handling (where queried data must not persist in the system) in microservices. We first formalise the theory for querying semi-structured data compatible with Jolie and we implement it into a Jolie library. We both present a use case of a medical algorithm to exemplify the usage of the library and report microbenchmarks that show that, for ephemeral data handling, Tquery outperforms the use of a database (e.g., MongoDB).

Looking more generally at Cloud deployment, we worked on both microservices and serverless.

In [14] we conducted a systematic review of the relationship between microservices and security, collecting 290 relevant publications (at the time, the largest curated dataset on the topic). We developed our analysis both by looking at publication metadata—charting publication outlets, communities, approaches, and tackled issues—and through 20 research questions, used to aggregate the literature and spot gaps left open. These results allowed us to summarise our conclusions in the form of a call for action to address the main open challenges of the field.

In previous work, Focus members developed “global scaling”, a microservices deployment technique that, given a functional specification of a microservice architecture, orchestrates the scaling of all its components, avoiding cascading slowdowns typical of uncoordinated, mainstream autoscaling. In [43] we propose a proactive version of global scaling, implemented using data analytics, able to anticipate future scaling actions. This eliminates inefficiencies of the original, reactive version of global scaling. We also present a hybrid variant that mixes reactive and proactive scaling. From our benchmarks, proactive global scaling consistently outperforms reactive, while the hybrid solution is the best-performing one.

Besides scaling techniques, we also worked on adaptability. In [12, 23] we focussed on workload orchestration, and we presented the SEAWALL platform, which enables heterogeneous data acquisition and low-latency processing for Industry 4.0. SEAWALL is developed within the homonymous project founded by the Italian BIREX industrial consortium. SEAWALL architecture features cutting-edge technologies (such as Kubernetes, ISTIO, KubeEdge, W3C WoT) to support the seamless orchestration of workloads

among the nodes of a cloud-edge continuum in QoS-aware scenarios where the latency requirement of anomaly detection must be continuously assessed. We present the industrial use case from the SEAWALL project and the cloud/edge microservice architecture. In [28] we discuss possible trade-offs and challenges of adopting autonomic adaptability, following a MAPE-K approach, to help software architects build adaptable microservice-based systems.

Finally, in [40] we worked on serverless function scheduling. Indeed, state-of-the-art serverless platforms use hardcoded scheduling policies that are unaware of the possible topological constraints of functions. Considering these constraints when scheduling functions leads to sensible performance improvements, e.g., minimising loading times or data-access latencies. To address this problem, we present and implement (as an extension of the OpenWhisk serverless platform) a declarative language for defining serverless scheduling policies to express constraints on topologies of schedulers and execution nodes.

7.2 Models for Reliability

Participants: Ivan Lanese.

Reversibility We have continued the study of reversibility started in the past years. As usual, our efforts target concurrent systems and follow the causal-consistent reversibility approach, where any action can be undone provided that its consequences, if any, are undone beforehand. We tackled both theoretical questions and practical applications in the area of debugging of concurrent Erlang programs.

From a theoretical point of view, we considered the impact of time on causal-consistent reversibility [27]. More precisely, we defined a causal-consistent reversible extension of the Temporal Process Language by Hennessy and Regan, a CCS with discrete time and a timeout operator. We proved that our extension satisfies the properties expected from a causal-consistent reversible calculus and we showed that it can also be interpreted as an extension of reversible CCS (in particular CCSK from Ulidowski and Phillips) with time.

Reversible debugging of concurrent systems can be tackled using causal-consistent debugging, which exploits causal-consistent reversibility to explore a concurrent execution backward following causality links from a visible misbehavior towards the bug causing it. In the past we exploited this idea to build CauDEr, a Causal-consistent Debugger for Erlang. We extended CauDEr to support the primitives used in Erlang to manage a shared map associating process identifiers to names [38]. Notably, these primitives are imperative, while the core of Erlang is functional. We also showcased the application of CauDEr to find a concurrency bug in a small case study in a form accessible to programmers with no previous knowledge of reversibility [45]. Finally, to bridge the gap between the theoretical foundations of causal-consistent reversibility and CauDEr, we developped a generator able to take a semantics in Maude (following a pre-defined structure) and automatically generating its causal-consistent extension [34]. We applied it to a semantics of Erlang we defined. The resulting executable causal-consistent semantics can be used as an oracle against which one can test CauDEr behavior, while being in a direct match with the theoretical foundations of Erlang and reversibility.

7.3 Quantitative Analysis

Participants: Melissa Antonelli, Martin Avanzini, Ugo Dal Lago, Gabriele Vanoni, Paolo Pistone, Francesco Gavazzo.

In Focus, we are interested in studying probabilistic higher-order programming languages and, more generally, the fundamental properties of probabilistic computation when placed in an interactive scenario, for instance the one of concurrent systems.

One of the most basic but nevertheless desirable properties of programs is of course termination. Termination can be seen as a minimal guarantee about the time complexity of the underlying program.

When probabilistic choice comes into play, termination can be defined by stipulating that a program is terminating if its probability of convergence is 1, this way giving rise to the notion of almost sure termination. Termination, already undecidable for deterministic (universal) programming languages, remains so in the presence of probabilistic choice, becoming provably harder. A stronger notion of termination is the one embodied in positive almost sure termination, which asks the average runtime of the underlying program to be finite. If the average computation time is not only finite, but also suitably limited (for example by a polynomial function), one moves towards a notion of bounded average runtime complexity. Over the recent years, the Focus team has established various formal systems for reasoning about (positive) almost sure termination and average runtime complexity, and has even established methodologies for deriving average runtime bounds in a fully automated manner. This trend continued in 2022.

In addition to the analysis of complexity, which can be seen as a property of individual programs, Focus has also been interested, for some years now, in the study of relational properties of programs. More specifically, we are interested in how to evaluate the differences between behaviours of distinct programs, going beyond the concept of program equivalence, but also beyond that of metrics. In this way, approximate correct program transformations can be justified, while it also becomes possible to give a measure of how close a program is to a certain specification.

Below we describe the results obtained by Focus this year, dividing them into several strands.

7.3.1 Randomized and Quantum Programs: Termination and Complexity

In FoCUS, we are interested in studying notions of termination and resource analysis for non-standard computing paradigms, like those induced by the presence of randomized and quantum effects. Noticeably, some of the contributions along these lines in 2022 have to do with the Curry-Howard correspondence.

In [31], a system of session types is introduced as induced by a Curry-Howard correspondence applied to bounded linear logic, suitably extended with probabilistic choice operators and ground types. The resulting system satisfies some expected properties, like subject reduction and progress, but also unexpected ones, like a polynomial bound on the time needed to reduce processes. This makes the system not only capable to characterize notions of efficiency in a randomized setting but also suitable for modeling experiments and proofs from the so-called computational model of cryptography.

We also showed that an intuitionistic version of counting propositional logic (itself introduced and studied within FoCUS in 2021) corresponds, in the sense of Curry and Howard, to an expressive type system for the probabilistic event λ-calculus, a vehicle calculus in which both call-by-name and call-by-value evaluation of discrete randomized functional programs can be simulated. In this context, proofs (respectively, types) do not *guarantee* that validity (respectively, termination) holds, but reveal *the underlying probability*. We also showed how to obtain a system precisely capturing the probabilistic behavior of λ-terms, by endowing the type system with an intersection operator [21].

We also looked at the nature of time in the context of security protocols. Timed cryptography refers to cryptographic primitives designed to meet their security goals only for a short (polynomial) amount of time, and has not been dealt with in the symbolic model, so far. We introduced a timed extension of the applied π-calculus, a common formalism to specify cryptographic protocols, and developed a logic for timed hyperproperties capturing many properties of interest, such as timeliness or time-limited indistinguishability. On the automation side, we mechanise proofs of timed safety properties by relying on the Tamarin tool as a backend [26].

Additionally, we achieved major improvements in the study of relational reasoning for higher-order continuous probabilistic programs, viz. higher-order programs sampling from continuous distributions. In [13], in fact, we have isolated a set of natural conditions on the expressive power of a programming language to guarantee full abstraction of bisimulation-based equivalences, viz. event and applicative bisimilarity, this way solving an open problem in the context of higher-order probabilistic reasoning.

Finally, we have also investigated to which extent techniques for reasoning about time (in expectation) and termination extends from classical, probabilistic programs to quantum programs. Specifically, in [22] we introduce a new kind of expectation transformer for a mixed *classical-quantum programming language*. This semantic approach relies on a new notion of a cost structure, which we introduce and which can be seen as a specialisation of the Kegelspitzen of Keimel and Plotkin. Along the analysis of several quantum algorithms within this new framework, we have shown that our calculus is both sound

and adequate with respect to the operational semantics endowed on our quantum language.

7.3.2 Differential Semantics of Programming Languages

Program semantics is traditionally concerned with program equivalence, in which all pairs of programs which are not equivalent are treated the same, and simply dubbed as incomparable. In recent years, various forms of program metrics have been introduced such that the distance between non-equivalent programs is measured as a not necessarily numerical quantity.

By letting the underlying quantale vary as the type of the compared programs become more complex, the recently introduced framework of differential logical relations allows for a new contextual form of reasoning. We showed [16] that all this can be generalised to effectful higher-order programs, in which not only the values, but also the effects computations produce can be appropriately distanced in a principled way. We show that the resulting framework is flexible, allowing various forms of effects to be handled, and that it provides compact and informative judgments about program differences. Moreover, we have introduced [30] a new general categorical construction that allows one to extract differential logical relations from traditional ones, and used such a construction to prove new correctness results for operationally-defined (pure and effectful) differential logical relations.

Mardare et al.'s quantitative algebras represent a novel and nice way of generalizing equational logic to a notion of distance. We explored the possibility of extending quantitative algebras to the structures which naturally emerge from Combinatory Logic and the λ-calculus [32]. First of all, we showed that the framework is indeed applicable to those structures, and give soundness and completeness results. Then, we proved some negative results clearly delineating to which extent categories of metric spaces can be models of such theories. Finally, we gave several examples of non-trivial higher-order quantitative algebras.

Sometimes, the quantitative aspects are in the language itself. This is the case for graded modal types systems and coeffects, which are becoming a standard formalism to deal with context-dependent, usage-sensitive computations, especially when combined with computational effects. We developed [15] a general theory and calculus of program relations for higher-order languages with combined effects and coeffects. The relational calculus builds upon the novel notion of a corelator (or comonadic lax extension) to handle coeffects relationally. Inside such a calculus, we define three notions of effectful and coeffectful program refinements: contextual approximation, logical preorder, and applicative similarity. We prove that all of them are precongruences, thus sound.

7.3.3 On the Space Consumption of Functional Programs

While the amount of *time* a functional program necessitates when evaluated can be taken as the number of reduction steps the underlying abstract machine or rewriting mechanism performs, the same cannot be said about space. What is a proper notion of *working space* for a functional program? We introduced [20] a new reasonable space cost model for the λ-calculus, based on a variant over the Krivine abstract machine. For the first time, this cost model is able to accommodate logarithmic space. Moreover, we study the time behavior of our machine and show how to transport our results to the call-by-value λ-calculus. We also designed [10] a new system of multi types (a variant of intersection types) and extract from multi type derivations the space used by the KAM, capturing into a type system the space complexity of the underlying abstract machine.

7.4 Qualitative semantics

Participants: Mario Bravetti, Daniel Hirschkoff, Ken Sakayori, Davide Sangiorgi.

In this area during the past year our efforts have gone in 3 main directions: unifying semantics; coinductive proof techniques; type-based techniques.

7.4.1 Unifying semantics

This direction has to with comparing, and possibly unifying two major approaches to the semantics of higher-order languages. One is the operational approach stemming from process calculi (such as the π-calculus), whereby an expressive algebraic model, rooted on the notion of process, is used to give semantics, by means of embedding, of the source language into the model. The other is the denotational approach stemming from game semantics, whereby the semantics of the language makes use of category theory. In [37], we establish a tight connection between existing process-calculus semantics and game semantics for both call-by-name and call-by-value λ-calculus. These connections allow us to transfer results and techniques between the process model and the game model.

In other works, we investigate the kind of models that pure process calculi such as π-calculus allow us to obtain in the case of functions, notably the λ-calculus. In [17] we consider the π-calculus encoding of call-by-value λ-calculus. We show that the equivalence on λ-terms induced by the encoding coincides with Lassen's eager normal-form bisimilarity, extended to handle η-equality. A crucial technical ingredient in the proofs is the recently-introduced technique of unique solutions of equations, further developed in the paper. We have continued work initiated in previous years where we try to tune existing proof techniques for pure process calculi so to be able to give sound, and possibly also complete, semantics to languages including references. In particular, [41] exhibit and proves a fully abstract encoding of a call-by-value λ-calculus with references, into the π-calculus. The encoding is then used to derive proof techniques for the source language stemming from the target language. These and other results are summarised in Prebet's PhD thesis [44].

7.4.2 Type-based techniques

In [11] we report on a tool that verifies Java source code with respect to typestates. A typestate defines the object's states, the methods that can be called in each state, and the states resulting from the calls. The tool statically verifies that when a Java program runs: sequences of method calls obey to object's protocols; objects' protocols are completed; null-pointer exceptions are not raised; subclasses' instances respect the protocol of their superclasses.

The paper [33] is a retrospection on our earlier work on session types, where we show that session types are encodable into standard π-types, relying on linear and variant types. Besides being an expressivity result, the encoding: (i) removes the redundancies in the syntax and semantics (when session types and session processes are added to the syntax of standard π-calculus, and (ii) the properties of session types are derived as straightforward corollaries, exploiting the corresponding properties of standard π-types.

7.4.3 Coinduction

There exist a rich and well-developed theory of enhancements of the coinduction proof method, widely used on behavioural relations such as bisimilarity. In [19], we study how to develop an analogous theory for inductive behaviour relations, i.e., relations defined from inductive observables. Similarly to the coinductive setting, our theory makes use of (semi)-progressions of the form R->F(R), where R is a relation on processes and F is a function on relations, meaning that there is an appropriate match on the transitions that the processes in R can perform in which the process derivatives are in F(R). For a given preorder, an enhancement corresponds to a sound function, i.e., one for which R->F(R) implies that R is contained in the preorder; and similarly for equivalences. We isolate a subclass of sound functions that contains non-trivial functions and enjoys closure properties with respect to desirable function constructors, so to be able to derive sophisticated sound functions (and hence sophisticated proof techniques) from simpler ones. We test our enhancements on a few non-trivial examples.

7.5 Computer Science Education

Participants: Maurizio Gabbrielli, Michael Lodi, Simone Martini.

7.5.1 Cryptography Education

We have designed and tested activities to teach the basic concepts of cryptography to high school students, creating both digital environments that simulate unplugged activities (for distance learning) and task-specific block-based programming languages in order to learn by manipulating computational objects [39]. The course focuses on the *big ideas of cryptography*, and the introduction of a cryptosystem is motivated by overcoming the limitations of the previous one.

We also designed and tested an interdisciplinary training module on cryptography [42] for prospective STEM teachers that leveraged some "boundary objects" between Math and CS (e.g., adjacency matrices, graphs, computational complexity, factoring) in an important social context (the debate on the benefits and risks of end-to-end cryptography). The module proved useful in making students mobilize concepts, methods, and practices of the two disciplines and making them move between semiotic representations of the interdisciplinary objects involved.

8 Bilateral contracts and grants with industry

8.1 Bilateral contracts with industry

SEAWALL (SEAmless loW latency cLoud pLatforms) is an industrial project financed by the Italian Ministry of Industry and Economical Development dedicated to the digital innovation of manufacturing companies. In particular, methods and techniques have been investigated in order to manage the development of cloud applications having low-latency constraints. The proposed solution is based on a platform, deployed on the edge-cloud continuum, in which latency-critical tasks are dynamically and automatically moved from cloud to edge and vice-versa. In this way, automatic workload mobility can be managed at run time following user desiderata constraints and preferences.

SEAWALL is coordinated by a company in Bologna (Poggipolini). M. Gabbrielli is coordinating the University of Bologna unit. The industrial partners are Aetna, Bonfiglioli Riduttori, IMA, Sacmi, Philip Morris, Siemens, CDM, and Digital River. The project started in July 2020, and ended in January 2022.

Ranflood Giallorenzo co-leads a three-year project collaboration, called "Ranflood", started in July 2021, between the "Regional Environmental Protection and Energy Agency" of Emilia-Romagna (ARPAE Emilia-Romagna) and the "Department of Computer Science and Engineering" (DISI) at the University of Bologna. The collaboration regards the development of techniques and software to combat the spread of malware by exploiting resource contention.

9 Partnerships and cooperations

9.1 International initiatives

9.1.1 Associate Teams in the framework of an Inria International Lab or in the framework of an Inria International Program

TCPro3

Title: Termination and Complexity Properties of Probabilistic Programs

Duration: 2019 →

Coordinator: Romain Péchoux (Inria project team Mocqua)

Partners: Inria project team Mocqua, Inria Nancy Grand-Est; University of Innsbruck (Austria)

Inria contact: Romain Péchoux

Summary: Probabilistic languages consist in higher-order functional, imperative languages, and reduction systems with sampling and conditioning primitive instructions. While deep theoretical results have been established on the semantic properties of such languages, applications of termination

and complexity analysis are restricted to academic examples so far. The associate team TCPro3 has the aim to contribute to the field by developing methods for reasoning on quantitative properties of probabilistic programs and models. Extensions of these methods on quantum programs will be studied.

9.2 International research visitors

9.2.1 Visits of international scientists

Guilhem Jaber

Status researcher (maitre de conference)

Institution of origin: Université de Nantes

Country: France

Dates: 7–9 November 2022

Context of the visit: collaboration on semantics of higher-order languages

Mobility program/type of mobility: research stay

Bruce Kapron

Status: Full Professor

Institution of origin: University of Victoria, British Columbia.

Country: Canada

Dates: 4 February 2022–1 April 2022

Context of the visit: ISA Visiting Fellows

Mobility program/type of mobility: Sabbatical

Georg Moser

Status researcher

Institution of origin: University of Innsbruck

Country: Austria

Dates: 28 August–9 September 2022

Context of the visit: collaboration on automation of average case complexity analysis

Mobility program/type of mobility: research stay

9.2.2 Visits to international teams

Martin Avanzini

Visited institution: University of Innsbruck

Country: Austria

Dates: 9 – 21. April 2022

Context of the visit: collaboration with G. Moser on automated average case complexity analysis

Mobility program/type of mobility: research stay

Martin Avanzini

Visited institution: University of Innsbruck

Country: Austria

Dates: 26 October – 6. November 2022

Context of the visit: collaboration with G. Moser on probabilistic complexity analysis and TCPro3 annual meeting

Mobility program/type of mobility: research stay

Simone Martini

Visited institution: Institut d'histoire et de philosophie des sciences et des techniques (IHPST), Paris

Country: France

Dates: November 6th–13th, 2022

Context of the visit: Bilateral research on a formal theory of algorithms, partially funded by INdAM (Istututo Nazionale di Alta Matematica)

Mobility program/type of mobility: research stay

Davide Sangiorgi

Visited institution: ENS Lyon

Country: France

Dates: 26–29 September 2022

Context of the visit: collaboration on proof techniques for process models with D. Hirschkoff

Mobility program/type of mobility: research stay

9.3 European initiatives

9.3.1 H2020 projects

BEHAPI (Behavioural Application Program Interfaces) is an European Project H2020-MSCA-RISE-2017, running in the period March 2018 — February 2024. The topic of the project is behavioural types, as a suite of technologies that formalise the intended usage of API interfaces. Indeed, currently APIs are typically flat structures, i.e. sets of service/method signatures specifying the expected service parameters and the kind of results one should expect in return. However, correct API usage also requires the individual services to be invoked in a specific order. Despite its importance, the latter information is either often omitted, or stated informally via textual descriptions. The expected benefits of behavioural types include guarantees such as service compliance, deadlock freedom, dynamic adaptation in the presence of failure, load balancing etc. The project aims to bring the existing prototype tools based on these technologies to mainstream programming languages and development frameworks used in industry.

Participants: Mario Bravetti, Maurizio Gabbrielli, Ivan Lanese, Cosimo Laneve, Stefano Zingaro, Davide Sangriorgi, Gianluigi Zavattaro.

9.4 National initiatives

9.4.1 DCore

DCore (Causal debugging for concurrent systems) is an ANR project that started on March 2019 and that will end in March 2024.

The overall objective of the project is to develop a semantically well-founded, novel form of concurrent debugging, which we call "causal debugging". Causal debugging will comprise and integrate two main engines: (i) a reversible execution engine that allows programmers to backtrack and replay a concurrent or distributed program execution and (ii) a causal analysis engine that allows programmers to analyze concurrent executions to understand why some desired program properties could be violated.

Participants: Ivan Lanese.

9.4.2 PROGRAMme

PROGRAMme (What is a program? Historical and philosophical perspectives) is an ANR project started on October 2017 and that will finish on October 2023 (extension of one year granted);

The aim of this project is to develop a coherent analysis and pluralistic understanding of "computer program" and its implications to theory and practice.

Participants: Simone Martini.

9.4.3 PPS

PPS (Probabilistic Programming Semantics) is an ANR PCR project that started on January 2020 and that will finish on July 2024.

Probabilities are essential in Computer Science. Many algorithms use probabilistic choices for efficiency or convenience and probabilistic algorithms are crucial in communicating systems. Recently, probabilistic programming, and more specifically, functional probabilistic programming, has shown crucial in various works in Bayesian inference and Machine Learning. Motivated by the rising impact of such probabilistic languages, the aim of this project is to develop formal methods for probabilistic computing (semantics, type systems, logical frameworks for program verification, abstract machines etc.) to systematize the analysis and certification of functional probabilistic programs.

Participants: Martin Avanzini, Ugo Dal Lago, Davide Sangiorgi, Gabriele Vanoni.

10 Dissemination

10.1 Promoting scientific activities

10.1.1 Scientific events: organisation

General chair, scientific chair

- IFIP Int. Conference on Formal Techniques for Distributed Objects, Components and Systems (I. Lanese, chair of steering committee)

Member of conference steering committees

- Conference on Reversible Computation (I. Lanese)
- Interaction and Concurrency Experience (I. Lanese)
- International Federated Conference on Distributed Computing Techniques (I. Lanese)
- Concurrency Theory (D. Sangiorgi)

Member of the conference program committees

- 12th International Workshop on Computing with Terms and Graphs (TERMGRAPH) 2022 (M. Avanzini)
- Workshop on Interaction and Concurrency Experience (ICE) 2022 (S. Giallorenzo)
- Agility with Microservices Programming 2022 workshop co-located with International Conference on Agile Software Development (XP) 2022 (S. Giallorenzo)
- European Conference and Service-Oriented and Cloud Computing (ESOCC) 2022 (S. Giallorenzo)
- International Conference on Fundamental Approaches to Software Engineering (FASE) 2022 Artefact Evaluation Committee (S. Giallorenzo)
- 50th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL) 2020 (U. Dal Lago)
- 26th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS) 2022 (U. Dal Lago)
- 27th ACM SIGPLAN International Conference on Functional Programming (ICFP) 2022 (U. Dal Lago)
- Agility with Microservices Programming workshop (S. Giallorenzo)
- 31st European Symposium on Programming (ESOP) 2022 (I. Lanese)
- 14th Conference on Reversible Computation (RC) 2022 (I. Lanese)
- 18th International Conference on Formal Aspects of Component Software (FACS) 2022 (I. Lanese)
- 24th International Conference on Coordination Models and Languages (COORDINATION) 2022 (I. Lanese)
- 15th Interaction and Concurrency Experience (ICE) 2022 (I. Lanese)
- 16th IFIP WG 1.3 Workshop on Coalgebraic Methods in Computer Science (CMCS) 2022 (D. Sangiorgi)
- 33rd International Conference on Concurrency Theory (Concur) 2022 (D. Sangiorgi)

10.1.2 Journal

Member of the editorial boards

- Journal of Universal Computer Science (M. Bravetti)
- Electronics Journal, section Computer Science and Engineering (M. Bravetti)
- Logical Methods in Computer Science (U. Dal Lago)
- Mathematical Structures in Computer Science (U. Dal Lago)

- Acta Informatica (U. Dal Lago)
- Acta Informatica (D. Sangiorgi)
- Distributed Computing (D. Sangiorgi)
- RAIRO – Theoretical Informatics and Applications (D. Sangiorgi)
- Foundations and Trends in Programming Languages (D. Sangiorgi)
- SN Computer Science (D. Sangiorgi)

10.1.3 Leadership within the scientific community

- The "Microservices Community" is a European-based, international, non-profit organisation purposed to promote he development of microservices by bridging research, education, and innovation within and between businesses, universities, organisations and individuals. Members of Focus have played active roles in the Community since its inception in 2019. The organisation includes members from the Innopolis University (Russia), the Dortmund University of Applied Sciences and Arts (Germany), SINTEF and the University of Oslo (Norway), the University of Pisa and the University of Sassari (Italy), WSO2 (U.S.A.), and the Zurich University of Applied Sciences (Swiss). Montesi is the president of the organisation, Guidi is a board member, Lanese and Giallorenzo are respectively part of the research and communication Community groups.
- I. Lanese is chair of the IFIP (International Federation for Information Processing) WG6.1 on Architectures and Protocols for Distributed Systems.
- U. Dal Lago is a member of the scientific councils of the Italian Chapter of the EATCS, and of the Italian Association on Logic and its Applications.
- U. Dal Lago is involved since September 1st in an Italian National initiative called CN HPC, namely a new research center about high-performance computing. U. Dal Lago is responsible for topics related to quantum computing inside the University of Bologna.
- S. Martini is a member of the Council of the Commission on History and Philosophy of Computing, an organism of the International Union for History and Philosophy of Science, 2017-2023.

10.2 Teaching - Supervision - Juries

10.2.1 Teaching

- Martin Avanzini
  - "Advanced Logic", 17 hours, Université Côte d'Azur, France.
  - "Probabilistic Rewriting", 6 hours, International Summer School on Rewriting, Tiblisi, Georgia
- Mario Bravetti
  - "Linguaggi, Compilatori e Modelli Computazionali", 120 hours, 1st year Master, University of Bologna, Italy.
  - "Programming, Algorithms and Data Structures" module of "Programming and Computer Architectures", 36 hours, 1st year Master, University of Bologna, Italy.
- Saverio Giallorenzo
  - "Programming Languages", 30 hours, 2nd year Bachelor, University of Bologna, Italy.
  - "Social Network Analysis", 30 hours, 2nd year Master, University of Bologna, Italy.
- Ugo Dal Lago

  - "Optimization", 36 hours, 2nd year, University of Bologna, Italy.
  - "Cryptography", 40 hours, 2nd year Master, University of Bologna, Italy.
  - "Languages and Algorithms for AI: Machine Learning Theory", 32 hours, 1st year Master, University of Bologna, Italy.

- Ivan Lanese
  - "Architettura degli Elaboratori", 34 hours, 1st year, University of Bologna, Italy.
  - "Computational Methods for Bioinformatics", 58 hours, 1st year Master, University of Bologna, Italy.

- Michael Lodi
  - "Computer Science Education", 26 hours, 2nd year Master, University of Bologna, Italy.

- Simone Martini
  - "Programmazione", 82 hours, 1st year, University of Bologna, Italy.
  - "Languages and Algorithms for Artificial Intelligence, module 1", 25 hours, 1st year Master, University of Bologna, Italy.

- Davide Sangiorgi
  - "Operating Systems", 110 hours, 2nd year undergraduate program, University of Bologna, Italy.
  - "Computer abilities", 16 hours, 2nd year Master in Medicine, University of Bologna, Italy.

- Gianluigi Zavattaro
  - "Algoritmi e Strutture di Dati", 70 hours, Bachelor in Computer Science, University of Bologna, Italy.
  - "Scalable and Cloud Programming", 50 hours, Master in Computer Science, University of Bologna, Italy.
  - "Architectures and Platforms for Artificial Intelligence", 24 hours, Master in Artificial Intelligence, University of Bologna, Italy.

10.2.2 Supervision

Below are the details on the PhD students in Focus: starting date, topic or provisional title of the thesis, supervisor(s).

- Melissa Antonelli, November 2019. "Probabilistic Arithmetic and Almost-sure Termination". Supervisor Ugo Dal Lago.

- Andrea Colledan, November 2021. "Complexity Analysis of Quantum Functional Programs". Supervisor: Ugo Dal Lago.

- Davide Davoli, October 2020, "Complexity Analysis of Higher-Order Randomized and Cryptographic Definitions". Supervisor: Tamara Rezk (INDES); Co-Supervisors: Martin Avanzini and Ugo Dal Lago.

- Enguerrand Prebet, September 2019, "The pi-calculus model of programming languages". Supervisors Daniel Hirschkoff and Davide Sangiorgi.

10.2.3 Juries

- S. Martini has been member of the PhD evaluation committee of Alice Martin, supervisors Stéphane Conversy and Mathieu Magnaudet, ISAE-SUPAERO, Institut Supérieur de l'Aéronautique et de l'Espace, Toulouse.
- M. Lodi has been both external reviewer for the thesis and member of the PhD evaluation committee of Emanuele Scapin, supervisor Alberto Policriti, co-supervisor Claudio Mirolo, Università degli Studi di Udine, Italy.

10.3 Popularization

10.3.1 Internal or external Inria responsibilities

- M. Avanzini is member of the "comite NICE"

10.3.2 Education

Michael Lodi and Simone Martini have carried out extended work of scientific popularization, including the following.

- They are members of the technical committee of Olimpiadi del Problem Solving (at Italian Ministry of Education); this involves preparation of material and supervision and jury during the finals.
- S. Martini gave the lecture "Perché scrivere codice è importante, per tutti", Software Heritage event–Bologna Big Code Lab, March 2022.

11 Scientific production

11.1 Major publications

[1] M. Bravetti and G. Zavattaro. 'A Foundational Theory of Contracts for Multi-party Service Composition'. In: *Fundam. Inform.* 89.4 (2008), pp. 451–478.

[2] N. Busi, M. Gabbrielli and G. Zavattaro. 'On the expressive power of recursion, replication and iteration in process calculi'. In: *Mathematical Structures in Computer Science* 19.6 (2009), pp. 1191–1222.

[3] P. Coppola and S. Martini. 'Optimizing optimal reduction: A type inference algorithm for elementary affine logic'. In: *ACM Trans. Comput. Log.* 7.2 (2006), pp. 219–260.

[4] M. Gabbrielli and S. Martini. *Programming Languages: Principles and Paradigms*. Springer, 2010.

[5] D. Hirschkoff, É. Lozes and D. Sangiorgi. 'On the Expressiveness of the Ambient Logic'. In: *Logical Methods in Computer Science* 2.2 (2006).

[6] U. D. Lago and M. Gaboardi. 'Linear Dependent Types and Relative Completeness'. In: *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011*. IEEE Computer Society, 2011, pp. 133–142.

[7] I. Lanese, C. A. Mezzina and J.-B. Stefani. 'Reversibility in the higher-order π-calculus'. In: *Theor. Comput. Sci.* 625 (2016), pp. 25–84. DOI: 10.1016/j.tcs.2016.02.019. URL: https://doi.org/10.1016/j.tcs.2016.02.019.

[8] F. Montesi, C. Guidi and G. Zavattaro. 'Composing Services with JOLIE'. In: *Fifth IEEE European Conference on Web Services (ECOWS 2007)*. 2007, pp. 13–22.

[9] D. Sangiorgi. *An introduction to Bisimulation and Coinduction*. Cambridge University Press, 2012.

11.2 Publications of the year

International journals

[10] B. Accattoli, U. Dal Lago and G. Vanoni. 'Multi types and reasonable space'. In: *Proceedings of the ACM on Programming Languages* 6.ICFP (29th Aug. 2022), pp. 799–825. DOI: 10.1145/3547650. URL: https://hal.inria.fr/hal-03912436.

[11] L. Bacchiani, M. Bravetti, M. Giunti, J. Mota and A. Ravara. 'A Java typestate checker supporting inheritance'. In: *Science of Computer Programming* 221 (Sept. 2022), p. 102844. DOI: 10.1016/j.scico.2022.102844. URL: https://hal.inria.fr/hal-03930280.

[12] L. Bacchiani, G. de Palma, L. Sciullo, M. Bravetti, M. Di Felice, M. Gabbrielli, G. Zavattaro and R. Della Penna. 'Low-Latency Anomaly Detection on the Edge-Cloud Continuum for Industry 4.0 Applications: the SEAWALL Case Study'. In: *IEEE Internet of Things Magazine* 5.3 (Sept. 2022), pp. 32–37. DOI: 10.1109/IOTM.001.2200120. URL: https://hal.inria.fr/hal-03916067.

[13] G. Barthe, R. Crubillé, U. Dal Lago and F. Gavazzo. 'On Feller continuity and full abstraction'. In: *Proceedings of the ACM on Programming Languages* 6.ICFP (29th Aug. 2022), pp. 826–854. DOI: 10.1145/3547651. URL: https://hal.inria.fr/hal-03923488.

[14] D. Berardi, S. Giallorenzo, J. Mauro, A. Melis, F. Montesi and M. Prandini. 'Microservice security: a systematic literature review'. In: *PeerJ Computer Science* 7 (2022), e779. DOI: 10.7717/peerj-cs.779. URL: https://hal.inria.fr/hal-03915125.

[15] U. Dal Lago and F. Gavazzo. 'A relational theory of effects and coeffects'. In: *Proceedings of the ACM on Programming Languages* 6.POPL (16th Jan. 2022), pp. 1–28. DOI: 10.1145/3498692. URL: https://hal.inria.fr/hal-03923470.

[16] U. Dal Lago and F. Gavazzo. 'Effectful program distancing'. In: *Proceedings of the ACM on Programming Languages* 6.POPL (16th Jan. 2022), pp. 1–30. DOI: 10.1145/3498680. URL: https://hal.inria.fr/hal-03923478.

[17] A. Durier, D. Hirschkoff and D. Sangiorgi. 'Eager Functions as Processes (long version)'. In: *Theoretical Computer Science* (2022). DOI: 10.1016/j.tcs.2022.01.043. URL: https://hal.archives-ouvertes.fr/hal-03466150.

[18] S. Giallorenzo, F. Montesi, L. Safina and S. P. Zingaro. 'Ephemeral data handling in microservices with Tquery'. In: *PeerJ Computer Science* 8 (2022), e1037. DOI: 10.7717/peerj-cs.1037. URL: https://hal.inria.fr/hal-03915136.

[19] D. Sangiorgi. 'From enhanced coinduction towards enhanced induction'. In: *Proceedings of the ACM on Programming Languages* 6.POPL (16th Jan. 2022), pp. 1–29. DOI: 10.1145/3498679. URL: https://hal.inria.fr/hal-03922092.

International peer-reviewed conferences

[20] B. Accattoli, U. Dal Lago and G. Vanoni. 'Reasonable Space for the λ-Calculus, Logarithmically'. In: LICS 2022 - 37th Annual ACM/IEEE Symposium on Logic in Computer Science. Haifa, Israel: ACM, 2nd Aug. 2022, pp. 1–13. DOI: 10.1145/3531130.3533362. URL: https://hal.inria.fr/hal-03912449.

[21] M. Antonelli, U. Dal Lago and P. Pistone. 'Curry and Howard Meet Borel'. In: LICS 2022 - 37th Annual ACM/IEEE Symposium on Logic in Computer Science. Haifa, Israel: IEEE, 2nd Aug. 2022. DOI: 10.1145/3531130.3533361. URL: https://hal.inria.fr/hal-03921650.

[22] M. Avanzini, G. Moser, R. Péchoux, S. Perdrix and V. Zamdzhiev. 'Quantum Expectation Transformers for Cost Analysis'. In: Symposium on Logic In Computer Science LICS '22. Haifa, Israel, 2nd Aug. 2022. URL: https://hal.inria.fr/hal-03540366.

[23] L. Bacchiani, G. de Palma, L. Sciullo, M. Bravetti, M. Di Felice, M. Gabbrielli, G. Zavattaro, R. Della Penna, C. Iorizzo, A. Livaldi, L. Magnotta and M. Orsini. 'SEAWALL: Seamless Low Latency Cloud Platforms for the Industry 4.0'. In: CIoT 2022 - 5th Conference on Cloud and Internet of Things. Marrakech, Morocco: IEEE, 28th Mar. 2022, pp. 90–91. DOI: 10.1109/CIoT53061.2022.9766643. URL: https://hal.inria.fr/hal-03916073.

[24] F. Barbanera, I. Lanese and E. Tuosto. 'Formal Choreographic Languages'. In: *Coordination Models and Languages : 24th IFIP WG 6.1 International Conference, COORDINATION 2022, Held as Part of the 17th International Federated Conference on Distributed Computing Techniques, DisCoTec 2022, Lucca, Italy, June 13-17, 2022, Proceedings*. COORDINATION 2022 - 24th International Conference on Coordination Models and Languages. Vol. LNCS - 13271. Lecture Notes in Computer Science. Lucca, Italy: Springer International Publishing, 14th June 2022, pp. 121–139. DOI: 10.1007/978-3-031-08143-9_8. URL: https://hal.inria.fr/hal-03917266.

[25] F. Barbanera, I. Lanese and E. Tuosto. 'On Composing Communicating Systems'. In: ICE 2022 - 15th Interaction and Concurrency Experience. Vol. 365. Lucca, Italy, 9th Aug. 2022, pp. 53–68. DOI: 10.4204/EPTCS.365.4. URL: https://hal.inria.fr/hal-03915946.

[26] G. Barthe, U. Dal Lago, G. Malavolta and I. Rakotonirina. 'Tidy: Symbolic Verification of Timed Cryptographic Protocols'. In: CCS 2022 - 2022 ACM SIGSAC Conference on Computer and Communications Security. Los Angeles, CA, United States: ACM, 7th Nov. 2022, pp. 263–276. DOI: 10.1145/3548606.3559343. URL: https://hal.inria.fr/hal-03921822.

[27] L. Bocchi, I. Lanese, C. A. Mezzina and S. Yuen. 'The Reversible Temporal Process Language'. In: *Formal Techniques for Distributed Objects, Components, and Systems :42nd IFIP WG 6.1 International Conference, FORTE 2022, Held as Part of the 17th International Federated Conference on Distributed Computing Techniques, DisCoTec 2022, Lucca, Italy, June 13–17, 2022, Proceedings*. FORTE 2022 - 42nd IFIP WG 6.1 International Conference on Formal Techniques for Distributed Objects, Components, and Systems. Vol. LNCS-13273. Lecture Notes in Computer Science. Lucca, Italy: Springer International Publishing, 12th June 2022, pp. 31–49. DOI: 10.1007/978-3-031-08679-3_3. URL: https://hal.inria.fr/hal-03917240.

[28] A. Bucchiarone, C. Guidi, I. Lanese, N. Bencomo and J. Spillner. 'A MAPE-K Approach to Autonomic Microservices'. In: ICSA-C 2022 - IEEE 19th International Conference on Software Architecture Companion. Honolulu, United States: IEEE, Mar. 2022, pp. 100–103. DOI: 10.1109/ICSA-C54293.2022.00025. URL: https://hal.inria.fr/hal-03916224.

[29] A. Coto, F. Barbanera, I. Lanese, D. Rossi and E. Tuosto. 'On Formal Choreographic Modelling: A Case Study in EU Business Processes'. In: *Leveraging Applications of Formal Methods, Verification and Validation. Adaptation and Learning : 11th International Symposium, ISoLA 2022, Rhodes, Greece, October 22–30, 2022, Proceedings, Part III*. Leveraging Applications of Formal Methods, Verification and Validation. Vol. LNCS13701. Lecture Notes in Computer Science. Rhodes (Grèce), Greece: Springer International Publishing, 17th Oct. 2022, pp. 205–219. DOI: 10.1007/978-3-031-19849-6_13. URL: https://hal.inria.fr/hal-03915950.

[30] F. Dagnino and F. Gavazzo. 'A Fibrational Tale of Operational Logical Relations'. In: FSCD 2022 - 7th International Conference on Formal Structures for Computation and Deduction. Haifa (Israël), Israel, 2nd Aug. 2022. DOI: 10.4230/LIPIcs.FSCD.2022.3. URL: https://hal.science/hal-03933446.

[31] U. Dal Lago and G. Giusti. 'On Session Typing, Probabilistic Polynomial Time, and Cryptographic Experiments'. In: CONCUR 2022 - 33rd International Conference on Concurrency Theory. Warsaw, Poland, 12th Sept. 2022. URL: https://hal.inria.fr/hal-03921809.

[32] U. Dal Lago, F. Honsell, M. Lenisa and P. Pistone. 'On Quantitative Algebraic Higher-Order Theories'. In: FSCD 2022 - 7th International Conference on Formal Structures for Computation and Deduction. Haifa, Israel, 2nd Aug. 2022. URL: https://hal.inria.fr/hal-03921800.

[33] O. Dardha, E. Giachino and D. Sangiorgi. 'Session Types Revisited: A Decade Later'. In: PPDP 2022 / 24th International Symposium on Principles and Practice of Declarative Programming. Tbilisi, Georgia: ACM, 20th Sept. 2022, pp. 1–4. DOI: 10.1145/3551357.3556676. URL: https://hal.inria.fr/hal-03922188.

[34] G. Fabbretti, I. Lanese and J.-B. Stefani. 'Generation of a Reversible Semantics for Erlang in Maude'. In: *Formal Methods and Software Engineering : 23rd International Conference on Formal Engineering Methods, ICFEM 2022, Madrid, Spain, October 24–27, 2022, Proceedings*. ICFEM 2022 - 23rd International Conference on Formal Engineering Methods. Vol. LNCS13478. Lecture Notes in Computer Science. Madrid, Spain: Springer International Publishing, 10th Oct. 2022, pp. 106–122. DOI: 10.1007/978-3-031-17244-1_7. URL: https://hal.inria.fr/hal-03916227.

[35] L. Gheri, I. Lanese, N. Sayers, E. Tuosto and N. Yoshida. 'Design-By-Contract for Flexible Multiparty Session Protocols'. In: ECOOP 2022 - European Conference on Object-Oriented Programming. Vol. 222. Berlin (DE), Germany, 6th June 2022. DOI: 10.4230/LIPIcs.ECOOP.2022.8. URL: https://hal.inria.fr/hal-03917259.

[36] S. Giallorenzo, F. Montesi, M. Peressotti and F. Rademacher. 'Model-Driven Generation of Microservice Interfaces: From LEMMA Domain Models to Jolie APIs'. In: 24th IFIP WG 6.1 International Conference, 17th International Federated Conference on Distributed Computing Techniques (DisCoTec 2022). Vol. 13271. Lecture Notes in Computer Science. Lucca, Italy: Springer International Publishing, 14th June 2022, pp. 223–240. DOI: 10.1007/978-3-031-08143-9_13. URL: https://hal.inria.fr/hal-03915132.

[37] G. Jaber and D. Sangiorgi. 'Games, mobile processes, Dfunctions'. In: CSL 2022 - 30th EACSL Annual Conference on Computer Science Logic. Göttingen, Germany, 2022, pp. 1–35. URL: https://hal.archives-ouvertes.fr/hal-03407123.

[38] P. Lami, I. Lanese, J.-B. Stefani, C. Sacerdoti Coen and G. Fabbretti. 'Reversibility in Erlang: Imperative Constructs'. In: *Reversible Computation : 14th International Conference, RC 2022, Urbino, Italy, July 5–6, 2022, Proceedings*. RC 2022 - 14th International Conference on Reversible Computation. Vol. LNCS-13354. Lecture Notes in Computer Science. Urbino, Italy: Springer International Publishing, 28th June 2022, pp. 187–203. DOI: 10.1007/978-3-031-09005-9_13. URL: https://hal.inria.fr/hal-03915947.

[39] M. Lodi, M. Sbaraglia and S. Martini. 'Cryptography in Grade 10: Core Ideas with Snap! and Unplugged'. In: *ITiCSE '22: Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*. ITiCSE 2022 - Innovation and Technology in Computer Science Education. Vol. 1. Dublin, Ireland: Association for Computing Machinery, 7th July 2022, pp. 456–462. DOI: 10.1145/3502718.3524767. URL: https://hal.inria.fr/hal-03916819.

[40] G. de Palma, S. Giallorenzo, J. Mauro, M. Trentin and G. Zavattaro. 'A Declarative Approach to Topology-Aware Serverless Function-Execution Scheduling'. In: 2022 IEEE (Institute of Electrical and Electronics Engineers) International Conference on Web Services (ICWS). Barcelona, Spain: IEEE, 11th July 2022, pp. 337–342. DOI: 10.1109/ICWS55610.2022.00056. URL: https://hal.inria.fr/hal-03915134.

[41] E. Prebet. 'Functions and References in the Pi-Calculus: Full Abstraction and Proof Techniques'. In: ICALP 2022 - 49th International Colloquium on Automata, Languages, and Programming. Paris, France, 4th July 2022. DOI: 10.4230/LIPIcs.ICALP.2022.114. URL: https://hal.science/hal-03920025.

Conferences without proceedings

[42] E.-I. Bartzia, S. Modeste, M. Lodi, M. Sbaraglia and V. Durand-Guerrier. 'Conception et organisation d'une situation didactique en cryptographie'. In: Didapro 9 – DidaSTIC – 9ème colloque francophone de didactique de l'informatique. Le Mans, France, 18th May 2022. URL: https://hal.inria.fr/hal-03916810.

Scientific book chapters

[43] L. Bacchiani, M. Bravetti, M. Gabbrielli, S. Giallorenzo, G. Zavattaro and S. P. Zingaro. 'Proactive-Reactive Global Scaling, with Analytics'. In: *Service-Oriented Computing*. Vol. 13740. Lecture Notes in Computer Science. Springer Nature Switzerland, 22nd Nov. 2022, pp. 237–254. DOI: 10.1007/978-3-031-20984-0_16. URL: https://hal.inria.fr/hal-03915139.

Doctoral dissertations and habilitation theses

[44] E. Prebet. 'Typed Behavioural Equivalences in the Pi-Calculus'. Ecole normale supérieure de lyon - ENS LYON; Università degli studi (Bologne, Italie), 27th Sept. 2022. URL: https://hal.archives-ouvertes.fr/tel-03920089.

11.3 Other

Scientific popularization

[45] I. Lanese, U. Schultz and I. Ulidowski. 'Reversible Computing in Debugging of Erlang Programs'. In: *IT Professional* 24.1 (1st Jan. 2022), pp. 74–80. DOI: 10.1109/MITP.2021.3117920. URL: https://hal.inria.fr/hal-03917301.

11.4 Cited publications

[46] M. Carbone, K. Honda and N. Yoshida. 'A Calculus of Global Interaction based on Session Types'. In: *Electr. Notes Theor. Comput. Sci.* 171.3 (2007), pp. 127–151.

[47] O. Dardha, E. Giachino and D. Sangiorgi. 'Session types revisited'. In: *Principles and Practice of Declarative Programming, PPDP'12, Leuven, Belgium - September 19 - 21, 2012*. Ed. by D. D. Schreye, G. Janssens and A. King. ACM, 2012, pp. 139–150. DOI: 10.1145/2370776.2370794. URL: https://doi.org/10.1145/2370776.2370794.

[48] A. Igarashi and N. Kobayashi. 'Resource usage analysis'. In: *POPL conference*. ACM Press, 2002, pp. 331–342.

[49] N. Kobayashi and D. Sangiorgi. 'A hybrid type system for lock-freedom of mobile processes'. In: *ACM Trans. Program. Lang. Syst.* 32.5 (2010).