Appel à projet générique 2017

Instrument : **Projet de Recherche Collaborative**

Défi : **Société de l'information et de la communication**

Méthodes formelles pour la conception d'algorithmes distribués

FoRmal mEthods for the Design of Distributed Algorithms

FREDDA

Contents

Remark: The bibliography can be found in the Annex of this proposal.

Project summary

Distributed algorithms are nowadays omnipresent in most systems and applications. It is of utmost importance to develop algorithmic solutions that are both robust and flexible, to be used in large scale applications. Currently, distributed algorithms are developed under precise assumptions on their execution context: synchronicity, bounds on the number of failures, etc. The robustness of distributed algorithms is a challenging problem which has barely been considered until now, and there is no systematic way to guarantee or verify the behavior of an algorithm beyond the context for which it has been designed. We propose to develop automated formal method techniques to verify the robustness of distributed algorithms and to support the development of robust applications. Our methods are of two kinds: statically through classical verification, and dynamically, by synthesizing distributed monitors, that check either correctness or the validity of the context hypotheses at runtime.

Summary table of persons involved in the project

Partner	Name	First Name	Current Position	Involvement (person. month)	Role and responsibilities in the project
IRIF	SANGNIER	Arnaud	Associate Professor	33,6	**Global coordinator**
	BOUAJJANI	Ahmed	Professor	12	
	DELPORTE	Carole	Professor	20	
	FRAIGNIAUD	Pierre	Senior Researcher	20	
	FAUCONNIER	Hugues	Professor	20	
	LAROUSSINIE	François	Professor	20	
LaBRI	MILANI	Alessia	Associate Professor	12	
	MUSCHOLL	Anca	Professor	12	
	TRAVERS	Corentin	Associate Professor	24	**Local coordinator**
	WALUKIEWICZ	Igor	Senior Researcher	9,6	
LSV	BOLLIG	Benedikt	Researcher	12	**Local coordinator**
	FÜGGER	Matthias	Researcher	4	
	GASTIN	Paul	Professor	12	

Cezara Drăgoi (INRIA - ENS ULM) will as well participate to this project as an external member and her involvement will be of 4 person. month.

1 Context, positioning and objectives

1.1 Context and challenges

Distributed applications represent nowadays a significant part of our everyday life and their development is central in the Internet of Things. To mention just a few examples, our personal data are stored on remote distributed servers, health data management relies on remote applications reachable via smartphones or tablets, and data-intensive computations like MapReduce or Hadoop are performed on computer clusters. Such applications have to meet several, often conflicting requirements like, for instance, low latency for clients and high throughput on the application side. They furthermore have to be reliable and robust, and they have to satisfy stringent correctness criteria, while being subject to fluctuating network conditions. The basic building blocks of such applications are distributed algorithms, which address particular problems in specific contexts. For instance, consensus algorithms [FLP85, GR07] and their relaxations like set agreement and k-set agreement [Cha93] make systems with machine replications agree on a value or a set of values. Another example are renaming algorithms [CRR11], which are used to provide different entities in a network with a small set of names.

In the last decades, researchers from the *distributed computing* community have developed advanced techniques for the design and analysis of distributed algorithms. One of the key aspects in the development of these algorithms are the specific assumptions made on the considered *execution context* (usually referred to as *system model* in the realm of distributed algorithms). For example, communication can be synchronous or asynchronous, the entities in the network may or may not be equipped with a unique identifier, or a certain number of errors and failures can happen during an execution. These assumptions are crucial and may lead to different families of algorithms designed for a specific context. This is, for instance, the case for algorithms that solve the fundamental set-agreement task, in which processes collectively choose a small subset of values from a larger set of proposals (see, e.g., [AGGT12]). A problem with a simple algorithmic solution in some system model may become unsolvable after small changes in the model assumptions. For example, it is well known that it is impossible to solve consensus in an asynchronous environment when at most one process may fail [FLP85], though simple algorithms exists when there is no failure or the system is synchronous. And for the set-agreement problem, the precise requirements over the model remains unknown. While there are numerous sophisticated techniques for algorithm design and analysis, most of the existing distributed algorithms, together with their correctness proofs, remain tailored to one very specific system model and are surprisingly difficult to generalize to different, but very similar models.

Many execution frameworks have been proposed. Some are very intuitive, like the synchronous context, in which all entities behave synchronously, or the completely asynchronous context, in which arbitrary long communication delays may occur. In practice, systems are neither synchronous nor fully asynchronous. Many other models lying between the synchronous and the asynchronous model have thus been defined. Investigation of such *partially* synchronous models have also been motivated by the fact that tolerating at the same time asynchrony and failures is hard or even impossible depending on the problem being considered. Hence, one is interested in execution contexts that *degrades* (or *relaxes*) to some extend the assumptions of the synchronous model in order to capture real systems executions while leaving the problem under study solvable. For example, in the round-based *heard-of model* [CS09], a message in transit that is not received within a round is considered to be lost and processes then react according to the set of messages they have received in the previous round. In the Θ-*model* [WS09], a bound Θ represents the ratio between the shortest and longest time delay of messages in transit simultaneously. Contexts with partial synchrony have also been studied in the case of shared-memory systems, (for instance, [ADFT12]): one can restrict the way the different processes access the memory by, for example, bounding the number of times a process can perform read/write actions while other active processes do not access the memory.

Towards robust distributed algorithms. Even though most distributed algorithms to solve problems like leader election, consensus, set agreement, or renaming, are not very long and essentially consist of an iterated loop, their behavior is difficult to understand due to the numerous possible interleavings of an execution. As a result, correctness proofs of distributed algorithms are extremely intricate. On top of this comes the fact that most algorithms are designed for an unspecified, unbounded number of participants and may use resources (like registers or data transmitted in messages) that depend on that number. Finally, their correctness proofs strongly depend on the underlying execution context and, as a matter of fact, can be very sensitive to any deviation. In other words, distributed algorithms tend to be *not robust*.

> *The purpose of our project is to develop formal methods that provide* algorithmic *support for the design of* robust *distributed algorithms.*

Researchers from the *formal methods* and *verification* community have proposed techniques to analyze distributed systems and to show their correctness. When the algorithm at hand is designed for a fixed finite number of participants using only finite-domain variables, software tools like SPIN [Hol05] are able to show correctness properties automatically, as it has been done for some classical mutual exclusion protocols like Peterson's algorithm or the Bakery's algorithm. Some works aimed to circumvent these restrictions. In particular, parametrized and data-aware verification techniques enable the analysis of networks of arbitrary size or of algorithms working with unbounded data [ABG15]. However, the decidability frontier for verification problems is very thin. On the positive side, there exist a wide range of logical tools and formalisms that allow one to express properties of distributed systems, to model their executions, and to reason about them. Researchers from verification have, by now, acquired a strong knowledge on the feasibility and application of automatic analysis techniques.

One important approach to avoid undecidability consists in looking for bad behaviors within a *subset* of the global behavior. In other words, one imposes restrictions on the system so as to regain decidability. Such restrictions are sometimes quite severe and discard many realistic behaviors. On the other hand, they allow one to develop approximation frameworks which are very useful to detect 'bugs'. There are different ways to obtain such restrictions in distributed systems. For instance, one can impose a bound on the size of the message queues or on the number of phases, each allowing only one process to send messages [BE14]. These restrictions on the considered executions are similar to execution models proposed in the distributed-computing community, and it will be worthwhile to understand, and formalize, the common aspects behind these two approaches.

Checking the robustness of a system is not a new idea in the realm of verification. It has already been applied, for instance, in the context of timed automata [BMS15], which are models for real-time systems that were originally equipped with some real-valued clocks with infinite precision. Robustness in that case means that a given property is still satisfied when clock values are subject to small deviations. Closer to our context, a notion of robustness exists for concurrent programs with shared variables running on different models for memory consistency. In fact, most such programs are developed assuming that they are executed under the classical interleaving semantics, which is known as *sequential consistency*. However, for efficiency reasons, compilers and processors may use a different model of execution, aka *weak memory model*. The idea is that operations (read from or write to the memory) may follow a different order with respect to the sequential consistency model. In that case, a program that respects certain properties under sequential consistency is considered as robust if it still respects these properties under a weaker notion of consistency [BDM13]. It should be noted that there even exist methods to enforce robustness by inserting barriers at some key points of the program. The main issue is then to insert as few barriers as possible so as to remain efficient in terms of concurrency.

We hence observe that we have two areas with complementary skills. On one side, we have distributed computing with its know-how for establishing the correctness of a few particular distributed algorithms using ad-hoc techniques that strongly depend on the execution context. On the other side, the verification community provides formal models and automated analysis techniques.

In this project, we gather researchers from both communities in order to build a methodology and automated tools to ease and automatize the development of robust distributed algorithms.

1.2 Goals and novel aspect

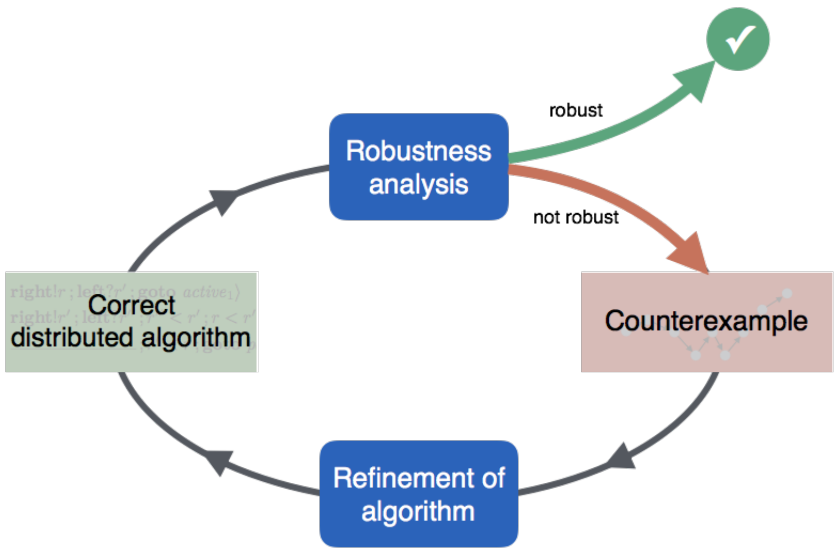

Figure 1: FREDDA development cycle

The goal of this project is to design automated techniques that can be used in the development of robust distributed algorithms. Figure 1 sums up the development cycle we envisage. Basically, the input is a distributed algorithm, designed for a specific execution context, that respects a given correctness criterion. Note that the algorithm may come with a witness for its correctness like, for instance, invariants. The FREDDA development cycle works in several stages:

1. *Robustness analysis.* The algorithm is analyzed with respect to a new execution context. The goal is to automatize this part as much as possible. There are two possible outcomes after this step. If the algorithm turns out to be robust, we are done. Otherwise, a counterexample is provided in terms of one or several example executions that violate the robustness condition.

2. *Refinement of the algorithm.* In this step, the counterexample obtained in Step 1 is used to build a more robust distributed algorithm that discards the non-robust behavior. However, as it will be difficult to generate a new algorithm from the witness automatically, this stage will be performed by experts from distributed computing.

In some cases, our development cycle will not terminate (otherwise, we would be able to refine a synchronous algorithm to make it work in an asynchronous environment, which is impossible for some problems like consensus). But an algorithm becomes more robust with each iteration of the development cycle. Some scenarios will require weakening the correctness criterion and a relaxation of the execution context in order to get out of the cycle. Even though we may detect, after some iterations, that the robust algorithm we are aiming at for a specific execution context is out of reach, our analysis ensures that the algorithm we have built so far is robust for another execution context. In any case, this procedure leads to an improved algorithm that is robust under certain conditions.

The robustness of distributed algorithms is a problem that has not received much attention until now but is relevant for the development of new algorithms and to improve the design methodology. Furthermore, developing automated techniques and tools for robustness analysis leads to challenging and interesting research tasks.

- First, there is a **need for formal definitions of different notions of robustness** and this also requires formal definitions of execution contexts. Note that degradations of some execution contexts (such as synchronism) have already been proposed, but most of the different options are not compared to each other. There is indeed no exhaustive study that relates the different relaxations. We may check, for example, whether one relaxation is included in another, or, conversely, whether it allows for strictly more behaviors. It is, hence, necessary to perform such a study to list and classify the different existing relaxation techniques and eventually propose a new form of relaxation. It is also important to understand the relation between the execution contexts developed in the distributed computing community and the restrictions used in the verification of distributed systems to under-approximate the behaviors of systems.

- Second, the **development of methods for checking robustness is challenging** for different reasons. In fact, even if they are often not very long (in terms of number of lines), the algorithms to be analyzed are complex due to different features. They are often designed for an unbounded number of participants and their correctness is guaranteed provided that enough resources (being registers, messages size, values of shared variables, etc.) are available. Note that often, the number of required resources is parametrized with respect to the number of participants (for example, in any shared memory k-set agreement algorithm the number of registers depends on the number of processes [DFKR15]). Reasoning about all possible scenarios that can arise during distributed algorithms executions may be computationally very hard and detailed models of such scenarios tend to be intractable: consequently, automated verification techniques such as model checking can only handle small instances of distributed systems. However, we believe that the ideas and techniques developed in verification, coming from automata theory and logic, can be of great use to reason systematically about cleverly abstracted models of distributed algorithms. Furthermore, one strong aspect in the development cycle represented in Figure 1 lies in the fact that the input distributed algorithm is assumed to be correct for a certain execution context, and the bottleneck for the analysis will hence be to understand how to use this information. We will also have to propose ways to transmit the results of the robustness analysis, when negative, to the algorithms designer, in such a way to be useful for the refinement phase.

- Finally, **another challenging question raised by our approach concerns the way we keep the system aware that the conditions under which the algorithm has been designed are not met anymore**. Note that it is of utmost importance to detect such situations at run-time. This does not only inform the participating processes that there is no guarantee anymore to get the desired result, but it also allows one to launch a recovery procedure, or to switch to another, less efficient, but more robust algorithm. Disposing of gracefully degrading, or adaptive algorithms would be of significant value for the development process of distributed applications. We believe that an automated approach to this problem can substantially help algorithm designers to provide flexible and reliable solutions. The challenge here is hence to design monitors to keep track of executions at run-time. The difficulties are to find the aspect to be monitored in order to detect changes in the behavior, and to design monitors for distributed systems which is an inherently hard task.

1.3 Expected results

We expect that the following results will be obtained during the FREDDA project:

1. **Propose formalisms to speak about robustness in distributed systems.** In fact, the project will begin by exploring, exhaustively, the techniques used in both distributed computing and verification to relax or restrict execution contexts of distributed systems. The goal will be to come up with families of interesting execution contexts and corresponding notions of robustness. As it has already been observed in the context of concurrent systems and weak memory models [BMM11, BDM13], it is important to consider different notions of robustness. Actually, asking for being robust with respect to a property stronger than the desired correctness property can facilitate the analysis considerably.

2. **Develop new automated tools for the analysis of distributed algorithms.** The heart of our project will be the development of tools and methods to perform the robustness analysis automatically. We also aim at a precise understanding of the decidability frontier. We will propose algorithms to check for robustness, as well as methods to build monitors that check, at run-time, whether an algorithm satisfies given constraints and is executed in the right execution context. We plan to implement some of our techniques in a prototype.

3. **Design new robust algorithms.** Our last goal will be to produce some robust algorithms obtained thanks to the FREDDA development cycle presented in Figure 1. In particular, we will study in detail algorithms for the renaming problem in message-passing systems. It is known that there exist solutions for both synchronous and asynchronous contexts, and we would like to deduce new algorithms for contexts that lie in between. We also plan to apply our methods to other families of distributed algorithms.

1.4 State of the art

Conception of distributed algorithms

In this section, we give an overview of various distributed system models relevant to FREDDA. One of the main paradigm in distributed computing when one is interested in fault-tolerance is consensus (in which processes have to eventually agree on one of their initial value). Consensus is an *universal* primitive, and more practically at the heart of *state machine replication*, which enable fault-tolerance by replicating a service on several servers and coordinating clients requests among them. The impossibility of consensus in asynchronous, crash-prone environments [FLP85] has motivated investigations of problems with less stringent requirements, various *partially synchronous* models, and development of best-effort algorithms that converge in well-behaved executions.

A classical relaxation of consensus is k-*set agreement*, which allows processes to decide up to k distinct values. Consensus is thus 1-set agreement. k-set agreement is solvable if and only no more that $k-1$ processes may fail. Interestingly, k-set agreement objects enable n processes to pick distinct new names in an interval of size $n-k+1$ [Gaf06].

In the seminal paper [DDS87], crude *partially synchronous* models are defined and investigated with respect to whether consensus can be solved or not. These models distinguish between process (a)synchrony and communications (a)synchrony, i.e., whether there exists global time bounds between any two consecutive steps of processes, or on message transfer delays. The paper [DLS88] shows that consensus can be solved in *eventually synchronous* models, in which executions are asynchronous for arbitrary long time, and then become synchronous. This model has later been refined at the channel level. If a (non-faulty) process is connected to every other processes via *eventually channels*, consensus is solvable [ADFT08]. Implicit references to real time may be replaced by bounds on the ratio between the slowest and fastest channels/processes as in the Θ-*model* [WS09]. In shared memory, timeliness properties of channels/processes may be replaced by constraining the scheduler, for example bounding the number of steps performed between two consecutive steps of some process. Finally, *set timeliness* [ADFT12] imposes constraints on the relative speed of *sets* of processes. This class of refined models gives separations results for t-resilient k-set agreement solvability.

Distributed systems models also address fault-tolerance by describing how part of the system (e.g., the processes and/or the communication medium) may fail. Several failure models have been defined, ranging from *crash* or *fail-stop* failures to more severe *byzantine* failures. A natural parameter for fault-tolerant distributed algorithms is the number of malfunctioning components they can tolerate. For example, when up to t processes may prematurely stop, n-processes fault-tolerant renaming can be solved with a set of new names of size $n + t$. Algorithms tolerating up to t processes failures are said to be t-resilient. More generally, one may specifies *set of processes* that may fail together in a single execution [DFGT11, HR13]. This notion of *adversary* accounts for the fact that failure are often correlated in real systems. Adversaries may be split into classes according to whether or not k-set agreement can be solved, which essentially correspond to 1-, 2-, . . . , $(n - 1)$-resiliency.

Finally, to bypass impossibility results, one may require algorithms to terminate only when the underlying execution satisfies some property. *Wait-freedom* requires each process to terminate regardless the behavior of other processes. Wait-free computing has been largely studied. In particular, characterization of what can be computed wait-free have been obtained using tools borrowed from algebraic and/or combinatorial topology [HKR13]. By contrast, *obstruction-freedom* only requires termination for a given process if it performs sufficiently many consecutive steps in isolation. Equivalently, it can be seen as a property of the scheduling that ensures that at least one process will run alone for an arbitrary number of steps. Assuming obstruction-free runs allows many problem to be solved including consensus. Many scheduling assumptions/termination requirement lying between obstruction-freedom and wait-freedom have been investigate, such as k-concurrency [GMT01] and asymmetric progress conditions [Tau10, IRT10]. In k-concurrent executions, no more than k processes may simultaneously *participate*, that is have started their local algorithm and not yet received an output. 1-concurrency corresponds to sequential computing while n-concurrency is wait-freedom. Interestingly, what can be computed k-concurrently is closely related to what can be computed using k-set agreement objects [GG11].

Among the large variety of models, one of the goal of the FREDDA project is to find relevant models in which algorithmic solutions, with the help of verification theory and techniques, may be extended to larger models.

Verification of distributed algorithms

Distributed algorithms are inherently hard to get right. Thus, one major challenge in the field of distributed computing, besides establishing complexity results and precise upper and lower bounds, is to come up with formal correctness proofs. As our project is targetting *automated* verification of robustness properties, we will focus on the state-of-the-art of *model checking* distributed algorithms.

Model checking is a fully automated method for determining whether a system model adheres to its correctness specification [BK08]. Classically, it covers finite-state systems, but it has since produced a rich theory in the realm of infinite-state systems. Of particular interest here is prameterized verification, which amounts to establishing correctness of an algorithm no matter how many processes actually interact. Unsurprisingly, the problem is undecidable in general, even when dealing with identical finite-state processes. However, a long line of research has established a variety of positive results. A recent textbook and survey article reflect the increasing interest in the area [BJK$^+$15, Esp16].

A widely used proof technique in parameterized verification is the cut-off principle [EN03, EK04, CTTV04], which builds on the following deduction rule: If the algorithm is correct when at most N processes intervene, then it is correct for any number of processes. The existence of a cut-off N allows one to reduce parameterized verification to model checking a finite-state system. Cut-offs can be applied to a couple of architectures and communication paradigms such as valueless token rings and certain rendez-vous systems and guarded protocols (see [AKR$^+$14] for a thorough analysis). Other well-established proof techniques include the theory of well-quasi orderings and well-structured transition systems [ACJT96, FS01] (which is suitable for broadcast systems [DSZ10]), the composition method (which has been applied to token-ring systems [AR16]), and network invariants [WL89, KM95]. It should be noted that, even though unified views of parameterized verification exist, most results actually depend on a particular architecture and execution context and are not readily generalizable.

Though model-checking research has considered the case of an unbounded number of processes, there is clearly a lack of techniques that are amenable to distributed algorithms. This is no wonder, since the latter combine parameterized verification with unbounded data types and a variety of possible execution contexts. Fault tolerance and timing issues complement the major challenges. However, progress has been made in combining at least two of these features. In [ABG15], an underapproximate model-checking approach is presented that can cope with an unbounded number of processes that exchange data in terms of totally ordered processs identifiers. In particular, this framework captures leader-election protocols and distributed sorting algorithms. Another research stream addresses model checking of fault-tolerant algorithms, which entails multiple parameters to describe the proportion of faulty processes. A promising approach here is a combination of abstraction and bounded model checking [JKS+13, KVW14]. In the FREDDA project, we will have to push the state-of-the-art forward to deal with distributed algorithms such as set agreement and renaming.

Monitoring in distributed systems

Runtime verification is an appealing, alternative method to traditional exhaustive exploration of a model, and is situated between testing and model checking. Its main distinguishing feature is that it is performed at runtime, by monitoring program executions against formal specifications. This opens the possibility to use it as support for error diagnosis, and also to react whenever an incorrect behavior of the program is detected.

Traditional runtime verification consists in constructing monitors from a given property. The monitor is used to check e.g. the current execution of the system. In other words, it reads a finite trace incrementally and is supposed to produce a verdict. While in model-checking all executions of the system are considered (often with emphasis on infinite executions) runtime verification deals with single (or a bounded number of finite) executions and does not require complete knowledge about the program or system. In other terms, runtime verification can be applied on black-box systems for which no model is available. It also represents a lightweight method regarding complexity, since monitoring single traces simply corresponds to the word problem. The main issue in runtime monitoring is the complexity of the monitor, i.e., its memory and computation time requirements, as a monitor runs in parallel with the system and should not slow it down too much.

Runtime verification is an ongoing and intense research topic, with a dedicated international workshop *Runtime verification* running since more than 15 years. A large body of work was devoted to monitoring LTL properties (see e.g. the survey [LS09]), and several tools (e.g. LOLA, EAGLE, RV-Monitor) are available. Extensions of monitors have been proposed in more complex settings such as for stochastic and timed automata, and for properties expressed in metric interval temporal logic, TLTL, and linear mu-calculus.

Designing distributed monitors from a given specification is far more challenging than for sequential programs, as the monitoring information has to be computed by a distributed algorithm. A straightforward, but impractical, way to monitor a distributed program is to synchronize the concerned components and to inquire about their states. A much better way to do this is to write a distributed monitor that deduces the required information by recording and exchanging suitable information using mainly the communication means provided by the execution of the monitored program. So the main point is to avoid adding synchronization in the program through the monitoring task, since this usually impacts negatively on the overall performance.

An additional challenge is dealing with failures. Distributed systems are indeed prone to failures, malfunction and unexpected communication delays. In such contexts, strong synchronization primitives such as consensus may not be implementable. Each monitor thus has to express a verdict about the state of the underlying system based only on its partial knowledge of the global state and perhaps after some asynchronous information exchanges with other monitors. In asynchronous systems, according to the lower bound [BFR+16], the number of local verdicts required for monitoring LTL properties grows linearly with the number of failures to be tolerated and the *alternation number* of the formula.

For the synthesis of distributed systems there is no general solution available. A first problematic issue is the lack of a canonical model for concurrency: reasonable frameworks can range between multi-threaded shared memory Java-like models and Scala-like programs with asynchronous function calls. A second, practical, problem is that monitors, when they exist, tend to be quite big.

A very successful example for the automatic generation of distributed monitors has its roots in the theory of Mazurkiewicz traces. This concurrency model was introduced in the late seventies by A. Mazurkiewicz [Maz77] as a simple model inspired by Petri nets. Within this theory, Zielonka's theorem [Zie87] is a prime example of a result on synthesis of distributed monitors. Many researchers contributed to simplify the construction and to improve its complexity. The most recent construction [GGMW10] produces deterministic distributed monitors of size that is exponential in the number of processes (and polynomial in the size of a DFA for the monitoring property). It is very challenging to try to adapt Zielonka's construction to models involving other types of synchronization. Generally speaking, designing synthesis algorithms on specific architectures, and with a reasonable computational overhead, is a significant endeavour.

2 Project organization and means implemented

2.1 Tasks description

2.1.1 Task 0: Project management

GLOBAL COORDINATOR Arnaud Sangnier (IRIF)
LOCAL COORDINATORS Benedikt Bollig (LSV), Corentin Travers (LaBRI)

OBJECTIVES

Ensure the achievement of goals - Manage the interactivity between the participants - Diffuse the obtained results

PROGRAM

Organization in tasks. To achieve our goals, the projet is organized in four tasks, each having its own coordinator (cf. Table 1). Each partner participates in each of the tasks. Each task is divided into subtasks. The description (and the risk analysis) of these tasks is provided hereafter. Their scheduling during the project is given in Subsection 2.2.

Num	Task	Coordinator
0	Project Management	Arnaud Sangnier (IRIF)
1	Formalization	Arnaud Sangnier (IRIF)
2	Robustness	Benedikt Bollig (LSV)
3	Monitoring	Anca Muscholl (LaBRI)
4	Tools and Case Study	Corentin Travers (LaBRI)
5	Organization of workshops	Pierre Fraigniaud (IRIF)

Table 1: Summary of the tasks and of the associated coordinator

Annual meetings. In order to achieve the different objectives of our project, it is important that there is both a strong local collaboration between the verification and distributed computing teams, but also a global collaboration between partners. We plan to organize two meetings each year. Their goal will be to present new results, to discuss the progress made with respect to the proposal, and to build working groups dedicated to some particular tasks. These meetings will be organized by each of the different partners and are open to other researchers and students.

Research stays. We will reserve part of the requested budget to finance research stays for the members of the project. These stays are, in fact, very important to allow the participants to collaborate. Please note that visits to other international experts are envisaged, too.

Website. It is very important that new results are made available to the community quickly. In fact, this will allow us to discuss them with other researchers and to estimate their impact. To this aim, we will maintain a website dedicated to the project, which provides our results, deliverables, summaries of annual meetings, and slides of the talks. The website will also be used to advertise any events connected to our project.

RISKS AND THEIR MANAGEMENT

There are two main risks in managing such collaborative project. The first is that each partner performs its research on its own leading to a lack of collaboration inside the project. However, the annual meetings and the research stays will minimize this risk. The other important issue is that we do not achieve our goals. To avoid this situation, this proposal contains a risk analysis for each task, which allows us to be conscious of those points where we have to be more attentive.

2.1.2 Task 1: Formalization

COORDINATOR Arnaud Sangnier

OBJECTIVES

Define the models for the considered algorithms - Propose some logical formalism for correctness properties - Formalize the notion of execution contexts

PROGRAM

The main role of this task will be to define a common mathematical language between researchers from the distributed computing community and the one from the verification world to represent distributed algorithms, the properties of the considered algorithms and to characterize the execution contexts which will be at the heart of our robustness analysis.

Subtask 1.1 - Models of distributed algorithms. In order to achieve the main goals of our project which consists in providing methods for the robustness analysis of distributed algorithms, we will focus on very specific families of distributed algorithms, namely those to solve problems like consensus, set-agreement, or renaming. We will consider algorithms using either message passing or shared memory systems. Our goal in this subtask will be to provide models for such algorithms (in other words, both syntax and semantics) to describe, faithfully, the considered distributed algorithms. The difficulty will be raised by the fact that researchers from the distributing community often describe their algorithms at a high level, which is not precise enough for a formal analysis. On the other hand, people from the verification community have developed over the years an expertise in developing models to represent algorithms and systems formally. We will use this agglomeration of expertise to build models that are suitable to represent behaviors of distributed algorithms. Among the features we will consider, we can, for instance, mention whether the algorithms are designed for an unbounded number of participants or not, whether they use unique identifiers or not, whether they use a number of register independent of the number of participants or not, whether they use some counting variables or not, whether they are dependent of the number of failure or not, etc. An exhaustive characterization will allow us to draw a cartography of the different aspects we will consider in this project.

Subtask 1.2 - Logical specification. Not only it is important to have a formal way to describe distributed algorithms but we need as well to develop formal languages to write the specification and as the well the invariants of our algorithms. There already exist many specification languages for computing systems like for instance the branching and temporal logic and their extension (see e.g. [BK08]) however these languages might not be expressive enough for the algorithms we will examine. First in many cases, the entities executing the algorithms will manipulate some data and the correctness criterion will need to be able to speak about those, for instance in the k-set agreement problem to specify that there are not more than k decided value. Then, since we plan to consider wait-free or obstruction-free algorithms, the specification language should be able to state properties like the following one : if a process executes itself alone, it will be able to take a decision. But one can imagine, especially if we want to use these logical formalisms to describe as well invariants, that we will need to describe even more stronger properties concerning the interleaving of the different executions. Some logics have been developed in the context of games for verification to describe both temporal properties and connection between the strategies of the player in the games (for instance the temporal logic Alernating-Time Temporal Logic ATL [AHK02]), and it could be interesting to inspire ourselves from such logic in order to develop specification languages for algorithms with unbounded number of participants (in ATL the number of player is fixed). We will have to be careful in the way we will develop our specification languages, because it is well known that as soon as one allow data in temporal logic, the undecidability frontier is very close. However using the fact that most of the time the precise value of the data do not really matter and as well restricting ourselves to very specific properties dedicated to the distributed algorithms under analysis, we have some chance to fight such negative result. The languages we will develop will be used to write the specification that have to respect the distributed algorithms, but they could as well serve to describe some invariants or properties to perform some rely-guarantee reasoning (which will be useful in the context of the robust analysis). For this last case, the specification expresses that if a property is verified, which could for instance restrict the set of considered execution,s then an other property is guaranteed to hold too.

Subtask 1.3 - Formal definitions of executive contexts. In the context of robustness analysis, our last formalization subtask will be to define some ways to describe executive contexts for the different models of distributed algorithms proposed in Subtask 1.1. There exist in the literature already different executive contexts as for instance the synchronous model, the asynchronous model, model with a certain number of failures, model with partial synchrony. As we have mentioned, in the context of verification of concurrent systems, in order to regain decidability or to be more efficient, researchers have propose too some ways to restrict the set of considered executions by imposing some restrictions on it (for instance by bounding the number of phase in message passing system, a phase being a period of time where only one process is allowed to send messages). Such restrictions can as well be seen as specific executive contexts. We believe that there are some connections between the executive contexts considered in the distributed computing community and the one used in verification, and we will examine them. Hence one aspect of this subtask will be as well to perform a comparison analysis between the already existing contexts. We will furthermore see if we can define new contexts based on the one that already exists but as well based on the analysis we will perform in Task 2. In fact, it is to be expected that the robustness analysis will lead us to see that we are able to refine an algorithm for some contexts that were maybe not the one proposed originally. Such a situation could happen for example, when we decide to stop the cycle of development represented in Figure 1 after some iterations without obtaining a full robust algorithm.

RISKS AND THEIR MANAGEMENT Tasks 1.1 and 1.2 are not very risky, however to achieve them it is necessary to have a strong collaboration between researchers from the formal model community and from the distributed community. Our consortium have in fact been conceived following this spirit. Even if these two tasks are not that challenging, they will allow to develop a common language which will be a building block for the rest of our project and hence they should not be neglected. The task 1.3 is the most challenging since it consists in developing formalism to describe executive contexts and try to provide some families of such contexts. The difficulty comes from the fact, that it is a new way of looking at distributed algorithm. However we are confident that we will be able to achieve this, in particular because there already exist definition of such specific contexts in the literature, that might be too informal for our purpose, but that we will use as a source of inspiration.

2.1.3 Task 2: Robustness

COORDINATOR Benedikt Bollig (LSV)

OBJECTIVES

Develop methods to verify statically the robustness of distributed algorithms

PROGRAM

In this task, we will develop different algorithmic techniques to solve the robustness problem. These techniques will be used in the robustness analysis of the development cycle presented on Figure 1. The following subtasks will lead our reasoning: we will first seek for methods in a very simple context (the one of finite-state systems). Then, we will perform a theoretical analysis in order to understand what can, or cannot, be achieved automatically and at which cost. Finally, we will propose methods based on approximation techniques, either to be more efficient or to regain decidability. For each of these subtasks, we will also study ways to transmit the counter-example obtained by robustness analysis.

Subtask 2.1 - Analysis of finite-state systems. Most of the distributed algorithms that we will analyze are designed to work with an unbounded number of participants and might also employ variable or messages ranging over infinite domains. However, even if our goal is to propose methods to perform a robustness analysis in such general context, the first step towards this goal is a thorough robustness analysis in the finite-state case. Finite-state models are obtained by fixing the number of participants and bounding the available resources. The main issue of this task is to see how robustness can be verified in this simpler context. In fact, there could be different notions of robustness. For instance, one could ask to be robust with respect to some specification, which means that a distributed algorithm which satisfies some specification in a specific execution context, will continue to verify it if the context is changed. But one could also require stronger properties like equivalence of the sets of reachable states or trace equivalence. We will develop algorithms to check the different notions of robustness we will consider. A precise understanding of this task is essential for the other subtasks.

Subtask 2.2 - Establish decidability frontier. In this subtask, we will consider more general models, closer to distributed algorithms (with an unbounded number of processes or unbounded data) and we will study the decidability and the complexity of the robustness checking for these new models. We expect that, for the general models we will define in Subtask 1.1., most simple verification problems will be undecidable due to the various sources of infinity that coe into play. However, it is important to understand what are the precise combinations of aspects that lead to undecidability. This may allows us to assess if our models could not be refined so that to fall in a decidable fragment. Furthermore, it might be the case that simple verification questions, like safety properties, are impossible to check on our models while some of the robustness problems are decidable. In fact, the input of the robustness analysis is an algorithm that has already been proven correct in a certain execution context, and it is possible that using this knowledge facilitates the verification of the robustness. To illustrate this point, note that there are some logics used in program verification, like separation logics, for which the satisfiability problem is in general undecidable but the entailment in some specific case becomes decidable [IRV14]. The reason is that, to check an implication, one restricts to the model satisfying the antecedent. Some similar reasoning could apply to robustness analysis.

Subtask 2.3 - Approximation techniques. In this subtask, we will develop some approximation schemes for the robustness analysis in order to tackle potential undecidability results or to be more efficient. The thing is that, in the robustness analysis, we can seek for some specific behaviors that will not be robust and that will consist in a subset of the general behaviors of the distributed algorithm at hand. If we can formally define subsets of behaviors for which robustness can be tested efficiently, then this yields efficients techniques for finding witnesses of non-robustness. However, such techniques are well suited to find counter-examples, but they cannot be used to prove robustness (since only an under-approximation of the behavior is observed). At that point, different options may be considered. Assume we found some restrictions on the set of executions that allows us to decide, efficiently, whether a given algorithm is robust. Now, suppose that we did not find a counter-example to the robustness in such a set. Then, we can either try to increase the set of observed executions and relaunch the analysis, or decide that we have obtained a robust algorithm for an execution context that is more restricted than the one for which robustness was originally tested. In this latter case, we obtain that the distributed algorithm is robust with respect to another execution context than the original one. Note that this is the reason why there is a strong connection between this subtask and Subtask 1.3. In fact, the restrictions that we will impose to define under-approximation of behaviors could be seen themselves as specific execution contexts.

RISKS AND THEIR MANAGEMENT. The subtask 2.1 is the less risky one because of the restrictions imposed on the model. For the subtask 2.2, there is not a real risk to not obtain results, but as mentioned it is possible that most of the robustness results will be undecidable in the general context. Thanks to the expertise of researchers in the consortium specialized in formal methods, there are high chance that we will be able to draw a nice cartography of the different aspects leading to undecidability and even obtain some interesting classes where the robustness analysis will be automatically feasible. The subtask 2.3 is the most challenging in this task and as well the more risky, because we have to provide useful approximation techniques, which allows in practice to detect non robustness. We are however confident that we will be able to obtain results as well here, because such a methodology has already been successfully applied for instance to search for bugs in concurrent programs.

2.1.4 Taks 3: Monitoring

COORDINATOR Anca Muscholl (LaBRI)

Objectives

Construct distributed monitors detecting deviations from the execution context at runtime, and design recovery procedures when a deviation is detected.

Program

The motivation behind this task is to construct monitors that detect changes in the execution context and initiate appropriate recovery actions. A distributed algorithm is designed for a certain execution context, like synchronous or asynchronous communication, uniqueness of process identifiers in the network, and certain types and frequency of failures that can occur, without affecting the outcome of the algorithm. In reality, such assumptions may become false during an execution, and a basic question is how to detect this at the runtime. After detecting such an anomaly a monitor can initiate a recovery action, or switch to another algorithm designed for a different execution context. The particularity of runtime monitoring needed in this context is that processes have to collect necessary information while running the distributed algorithm, without introducing additional synchronizations. Concretely it means that processes participating in the monitoring activity can use only means of communication provided by the algorithm they monitor. For example, they can pigyback monitoring information on messages being exchanged by the algorithm. This task will rely on Task 1, in particular on the formalization of families of distributed algorithms and the notions capturing various degrees of satisfaction of the execution context.

Distributed monitoring is a challenging subject. There are many models of concurrent systems but for none of them the problem is really well understood. In our case we are quite lucky since we can start from results on monitoring for Zielonka automata. These automata are a rather direct abstraction of distributed algorithms, and we have already some useful results on distributed monitoring in this setting. In view of our applications to analysis of distributed algorithms, the model of Zielonka automata would need to be extended to parametric automata where the number of participating processes is not fixed. We would also need to introduce data aspect, in order to handle process identifiers. Finally, we will use monitoring to implement control which is another challenging aspect of this task.

Subtask 3.1 This task will concentrate on a simplified case where the number of processes is fixed. This brings us to the model of Zielonka automata. Fundamental theorem for Zielonka automata [Zie87] can be understood as giving a construction of a distributed monitor. Building on our improvements of this theorem [GGMW10], we will provide efficient construction of distributed monitors. Another challenge in this task we will be to handle all properties linked to definitions of execution contexts identified in Task 1. Existing results on monitoring consider only regular properties, but these are not sufficient to describe execution contexts.

Subtask 3.2 This task will concentrate on monitoring of an extension of the Zielonka automata capable of modeling of distributed algorithms. We have in mind a conservative extension of Zielonka automata where unboundedly many processes communicate through shared registers storing process identities. Natural sequential specifications are provided by register automata [KF94] and their variants [BHLM14, SKMW17]. A central question is then whether Zielonka's theorem has a parametrized analog. Uniqueness of process identifiers may guarantee a smooth transfer from a fixed to an unbounded number of processes.

Another aspect is to consider more specific extensions of the model with data. A recent research stream aims at extending temporal specifications and monitors by parametrized events. They range from first-order extensions of LTL [BKM10] to register automata [GDPT13]. To some extent, the latter can handle data values such as object references and process identities. Current techniques, however, are restricted to centralized monitors and do not carry over to a distributed setting.

Subtask 3.3 The final step is to use distributed monitoring to control of distributed algorithms. We expect to be able to synthesize controllers for distributed algorithms of particular shape (like "wait-free" algorithms). Our approach is to use monitoring to detect changes and switch between different distributed algorithms. It seems to us that this is one of the most promising approaches to automatic construction of robust distributed algorithms.

Risks and their management We have already done some work that can serve as a basis of Subtask 3.1, so we do not expect surprises there. This task though will serve to find a common language, it will be done in parallel with Subtask 1.1, which also has this purpose. Subtask 3.2 is the main technical challenge of Task 3. At present it is hard to predict to what extent we will be able to extend Zielonka's theorem. We are quite confident though that we will be able to find an extension covering test cases of this project. Finally, Subtask 3.3 will very much depend on the progress in Task 1.

2.1.5 Task 4: Case study and protoypes

COORDINATOR Corentin Travers (LaBRI)

OBJECTIVES

Design new renaming algorithms demonstrating the methodology developed in Task 2 - Construct a collection of monitors for the execution contexts these new algorithms withstand -Implement prototypes to automatize robustness analysis and monitor synthesis.

PROGRAM

To guide and assess research in other tasks, we will focus on algorithms for the *renaming problem*. In this problem, the processes are initially provided with unique identifiers in a large space.The goal for each process is to pick a new name in a smaller interval such that no two processes share the same name.

Renaming Historically, renaming has been introduced as a problem that can be solved in unreliable asynchronous environments [ABD+90]. Indeed, provided that the size of the target name-space is large enough, the problem can always be solved, even in asynchronous, failure-prone environments. Surprisingly, despite its very simple specification, sophisticated lower bounds and algorithms for renaming have been discovered. Renaming has mainly been studied in the shared memory model. Several algorithms have been found, exhibiting a trade-off between performances and how loose the target name-space is (see [Ali15] for a survey of recent results). By contrast, little is known on renaming algorithms for message-passing systems, in which only the synchronous [AAGT12, CHT99] and the asynchronous (e.g., [ABD+90]) models have been investigated. Of course, by simulating shared registers, renaming algorithms designed for shared memory can be deployed in message-passing systems. This comes at a high price, induced on one hand by the cost of the simulation and, on the other hand, by the fact that shared-memory renaming algorithms usually assume the worst-case environment: fully asynchronous communication and unbounded number of crash failures.

Several variants of the problem have been defined. In *tight* renaming, the size of the target name-space is the total number of processes, while this constraint is relaxed in *loose* renaming. In *adaptive* renaming, the size of the target name-space is not a function of the total number of processes, but rather of the number of processes actually requesting new names. New names are *acquired* and then *released* in *long-lived* renaming. Hence, renaming might be seen as a *resource allocation* problem, by equating new names with resources. Exclusive access to a resource can be ensured with mutual exclusion. However, failure of one process in the critical section may impede progress. An alternative is to have several copies of the resource, and guaranteeing that each resource is accessed by at most one process at any time.

Adapting to the execution context In practice, although distributed executions exhibit asynchrony and failures do occur, timing anomalies and failures are somewhat rare. Moreover, many distributed problems of interest (e.g., consensus, k-set agreement, tight renaming, etc.) cannot be solved in asynchronous, failure-prone environments, but can be solved when the system satisfies some properties of synchrony. This has motivated the development of *indulgent* algorithms. Such algorithms never violate the safety part of their specification but may stall when the underlying execution is asynchronous and failures occur. Another approach consists in (gracefully) *degrading* the constraints on the algorithm outputs as the execution context becomes harsher. For example, for renaming, the size of the target name-space may depend on some notion of level of (a)synchrony of the underlying execution. Finally, it is often the case that tolerating failures and dealing with asynchrony requires inherently costly mechanisms, while rather simple solutions do exist for (partially) synchronous environments. Hence, efficient distributed algorithms often encompass various optimizations combined in an ad-hoc manner, each targeting a special case in which outputs can be obtained quickly (e.g., *fast-path* in mutual exclusion, various forms of *early decision* in agreement protocols, *optimistic speculations* techniques in software transactional memory, etc.). This leads to monolithic, complicated, non-modular, hard to extend and difficult to prove correct algorithms. A preeminent example is the classical Paxos consensus algorithm and its descendants (entire websites are dedicated to explain them[1]). Another major concern is to translate such algorithms into production code [CGR07].

Subtask 4.1 This tasks will concentrate on designing new message passing algorithms for renaming for ranges of executions contexts lying between full synchrony and full asynchrony. We will follow the methodology develop in Task 2, combining expertise in distributed algorithm design with (as much as possible) automated robustness analysis.

Subtask 4.2 Given a collection of renaming algorithms $A_1, \ldots, A_n$, each certified to withstand a range of variations in the execution context (for example, able to tolerate a certain number of failures below some threshold, or guaranteeing progress as long as the communication pattern has some level of synchrony), we would like to be able to select at run-time the best algorithm that fits the current execution context, and to switch from one algorithm to another as the executions context changes. To that end, this task will focus on designing distributed monitors and switching mechanisms in the framework of message passing renaming algorithms.

Subtask 4.3 To facilitate Tasks 4.1 and 4.2, and to demonstrate the benefit of the FREDDA approach, this task aims at producing prototype tools for automating robustness analysis and monitors synthesis.

RISKS AND THEIR MANAGEMENT This is a high-risk task. However, the members of the project from distributed computing have a strong expertise in designing (partially) asynchronous, fault-tolerant message-passing algorithms. We are confident that they can apply their knowledge to the renaming problem. Automatic robustness analysis and monitor synthesis may have prohibitive complexity. We may thus have to limit the number of processes, the range of execution context we consider and restrict ourselves to algorithm that follow some well-structured pattern (round-based, finite message types, etc.). Nevertheless, this has to be balanced with the potential large gains in terms of modularity, ease of extension and complexity of correctness proofs. Finally, progress in this task is very dependant to progress made in Task 2 and Task 3.

2.1.6 Task 5: Organization of workshops

COORDINATOR Pierre Fraigniaud (IRIF)

OBJECTIVES

Organize two workshops which involve researchers from distributed computing and from formal methods.

PROGRAM

In this project, we will bring together researchers working on the verification of concurrent programs and distributed systems, and researchers from distributed computing. We plan to organize two workshops to strengthen this alliance. Both communities have a deep understanding of distributed computation, but from two different perspectives. Historically, these communities have common roots. However, since more than two decades, they tend to evolve independently. These workshops will address several topics that can be viewed as bridges between the two aforementioned research fields, with potential fruitful co-developments. One of the workshops will be organized as a Dagstuhl seminar, the other one together with a conference or as an independent event.

2.2 Schedule and dependencies

The duration of this project will be four years. Figure 2 shows how the various tasks will be scheduled during the project. This scheduling is justified by the dependencies presented in Figure 3. We see that Task 4 which will guid our research thanks to the case study will feed the Subtasks 1.1 and 1.2 of Formalization which will themselves be given in input of the tasks where we will develop techniques for robustness and monitoring. The fact that Subtask 1.3 (*Formalization of execution contexts*) is interdependent of Task 3 and 4 is because the analysis techniques we will develop will also influence the way we define execution contexts. The results we will obtain in Tasks 2 and 3 will give a feedback to improve the distributed algorithms studied in Task 4. Finally, the two workshops that we will organize inside Task 5 will be scheduled at the end of the first year of the project and at the middle of the third year.

[1]E.g., https://understandingpaxos.wordpress.com/

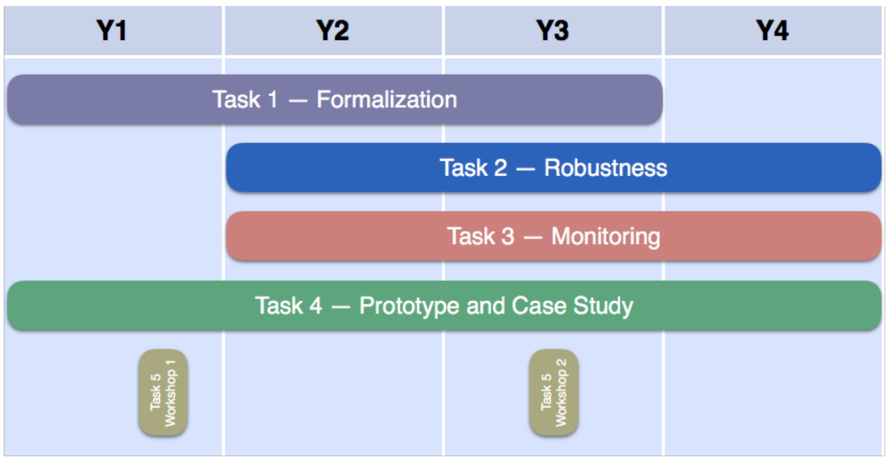

Figure 2: Schedule of the tasks for the project

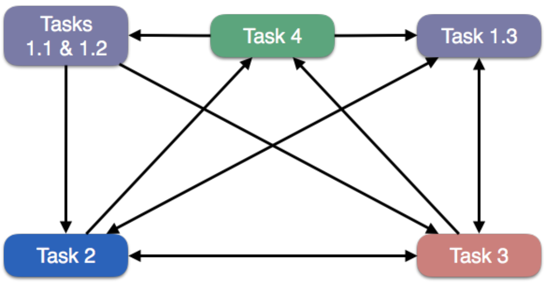

Figure 3: Dependencies between the tasks

2.3 Deliverables

For each task (except Tasks 0 and 5), we will provide two deliverables: the first one will be provided at the middle of the task to sum the progress of the task and one at the end of the task. If t_0 is the date at which will begin the project, this will lead to the following deliverables (time lengths are provided in months):

$t_0 + 18$	Report on progress for **Task 1**	$t_0 + 36$	Final report on progress for **Task 1**
$t_0 + 24$	Report on progress for **Task 4**	$t_0 + 48$	Final report on progress for **Task 2**
$t_0 + 30$	Report on progress for **Task 2**	$t_0 + 48$	Final report on progress for **Task 3**
$t_0 + 30$	Report on progress for **Task 3**	$t_0 + 48$	Final report on progress for **Task 4**

2.4 Consortium Description

We present here the consortium. CVs of the different participants are provided in the appendix.

2.4.1 Scientific coordinator

This project will be led by **Arnaud Sangnier**. Arnaud Sangnier is an assistant professor at University Paris Diderot since September 2010 and doing his research at laboratory IRIF in the team Modelization and Verification. He did his PhD jointly at the laboratory LSV of ENS Cachan and the french company for electricity (EDF) from 2005 until 2008 under the supervision of Alain Finkel and Étienne Lozes. His PhD focused on developing automatic methods for the verification of programs manipulating dynamic data structures and integer variables. After his PhD, Arnaud did a first year of postdoctoral study at the department of computer science of the University of Torino (Italy). There, he studied the verification of probabilistic systems with infinite data. His postdoctoral stay was financed by a scholarship from the french DGA. From January 2010 until September 2010, he did another postdoctoral stay at the department of computer science of the University of Genova (Italy) where he worked together with Giorgio Delzanno on the verification of parameterized systems to model the behavior of protocols running on mobile ad hoc networks. The main theme of Arnaud Sangnier's research is the verification of so-called infinite-state systems. He studies particularly the class of programs manipulating integer variables (also known as counter systems) and communication protocols developed for networks in which the number of active entities is a priori unknown and hence can be seen as a parameter. During the last years, he has established new fundamental results for this class of systems, published at some of the best conferences in formal methods (ICALP, CONCUR, FOSSACS). He has published 4 articles in international journals and 26 articles in international conferences. From 2012 to 2015, he has also co-supervised, together with Stéphane Demri (Senior Researcher at LSV-ENS Cachan), the PhD thesis of Amit Kumar Dhar, who is now an assistant professor at IIT-Allahabad.

2.4.2 Description of the consortium

The consortium gathers experts in formal methods and in distributed computing from three French labs which are international leader in these two domains: IRIF, LSV, and LaBRI. Furthermore the participants are divided up in such a way that, in each laboratory, a collaboration between experts from these two communities is locally possible. Hence, we hope that this project will give rise two new collaborations and will be able to bring techniques from distributed computing in the top international conferences in formal methods and vice-versa. In fact, even though, at some point, these two research communities were very close to each other (with, for instance, the works of Leslie Lamport or the automatic proof of concurrent algorithms like mutual exclusion protocol), the last decade has not seen much interaction between them.

IRIF. The Institut de Recherche en Informatique Fondamentale (IRIF) is a research laboratory co-founded by the CNRS and the University Paris-Diderot, resulting from the merging of the two research units LIAFA and PPS on January 1st, 2016. The scientific objectives of the laboratory are at the core of computer science, focusing on: the mathematical foundations of computer science; computational models and proofs; models, algorithms and system design. The IRIF participants to FREDDA are members either of the Distributed Algorithms and Graph team or of the Modeling and Verification team. The people from the distributed computing team are international experts in the design of distributed algorithms in specific contexts. For what concerns the researchers from verification, they are international specialists in logic for verification of systems as in developing automatic algorithms for the analysis of computing systems.

LaBRI. The Laboratoire Bordelais de Recherche en Informatique (LaBRI) is a research unit associated with the CNRS (UMR 5800), the University of Bordeaux and Bordeaux INP. It has significantly increased in staff numbers over recent years and now includes around 350 members (academics, researchers, PhDs, etc.). The members of the laboratory are grouped in six teams, each one combining basic research, applied research and technology transfer: Combinatorics and Algorithmics, Image and Sound, Languages and Systems and Networks, Formal Methods, Models and Algorithms for Bio-informatics and Data Visualisation and Supports and Algorithms for High Performance Numerical Applications. The LaBRI participants to FREDDA are members of either the Combinatorics and Algorithmics team or of the Formal Methods team. Both teams are recognized worldwide for their expertise, as testified by membership in programme committees, invited talks and tutorials, as well as the organization of conferences and workshops. The LaBRI team includes specialists on various models and topics in the scope of the project, in particular fault tolerance and synchronization issues in distributed computing, distributed monitoring and control, and parametrized verification.

LSV. The Laboratoire Spécification et Vérification (LSV) is a joint laboratory of ENS Paris-Saclay and the French Centre National de la Recherche Scientifique (CNRS). It counts currently 25 permanent members, 9 temporary members, and 22 PhD students. The LSV has 20 years of expertise on formal verification across a wide range of computer systems, such as distributed and database systems, time-critical systems, and security protocols. Benedikt Bollig and Paul Gastin are members of the VASCO team, while Matthias Függer is a member of the Inria team MExICo. VASCO's research is about the analysis of complex systems, in a broad sense. The focus is on automated verification and synthesis of safety-critical systems with quantitative constraints and complex interactions between their components. MExICo is engaged in studying concurrency and interaction, with the objective to increase reliability of distributed and asynchronous systems.

2.5 Scientific justification of requested resources

Since most of the requested resources are estimated in the same manner for each of the three partners, we present the following justification of the requested resources according to the type of resources, and for each of these, partner by partner, rather than in the opposite way. All calculations will be rounded to the closest multiple of 100 €.

2.5.1 Equipment

We will provide a work station (a laptop) to each staff funded by the ANR (PhD student and Postdoctoral fellow). The cost of a work station is estimated to 1500 €. Considering the duration of the project, some permanent member will also need to renew their work station. This leads to the following estimations:

For IRIF:	4×1500 €	=	6000 €
For LaBRI:	2×1500 €	=	3000 €
For LSV:	1×1500 €	=	1500 €

2.5.2 Staff

PhD student for IRIF and LSV We request the funding of a PhD student which will perform do his thesis between LSV and IRIF. His/her mission will be focus on Task 3. This student will be supervised by Arnaud Sangnier at IRIF and by Benedikt Bollig at LSV. However, he/she will also strongly collaborate with the experts in distributed computing at IRIF. His/her work will be mainly focus on first a theoretical analysis of what can be achieved for the robustness analysis (Subtask 1.2) and then he will have in charge to propose some efficient analysis techniques for the robustness analysis based on approximation (Subtask 1.3). He/she will start the PhD at the end of Year 1 in order to take benefit of the initial progress made in the project. The estimated cost incurred by this position is 95000 €.

Postdoctoral fellow for LaBRI We request the funding of a 12-month postdoc position. Depending on the expertise of the candidate (formal methods or distributed computing), her/his mission will be related to Task 3 or Task 4. In any case, she/he will be supervised by two members of the project, one from the team Algodist and the other from the team Méthodes Formelles. Specifically, in case of a candidate from formal methods, her/his work will be mainly dedicated to Substask 3.3, extending monitoring to control, especially for some well-structured class (e.g., round-based) of renaming algorithms. In a case of a candidate more specialized into distributed computing, we expect her/him to focus on the design of new renaming algorithms, following the methodology developed in Task 2. The postdoctoral fellow will start at the end of year 2, in order to benefit from the initial progress made in Tasks 1 and 2. The estimated cost incurred by this position is 50000 €.

Master internships They will be proposed in relation with the various tasks of the project. We expect to supervise two students at IRIF, three students at LaBRI, and two students at LSV, in the course of the project, for a four-months period each. Considering a cost of 600€/month, this gives an amount of 4800 € for IRIF and LSV and of 6700 € for LaBRI.
We obtain the following total for the staff part of the budget:

For IRIF:	95000 € + 4800 €	=	99800 €
For LaBRI:	50000 € + 6700 €	=	56700 €
For LSV:	4800 €	=	4800 €

2.5.3 Missions

Project plenary meetings The project gathers researchers from two almost disjoint scientific communities, namely Formal Methods and Distributed Algorithms. Intensive collaboration between consortium members from the different communities is crucial to the success of the project. We therefore plan to organize two plenary meetings per year, for the duration of the project. We plan each meeting to be 3-days long and expect that every member will participate in each meeting. However, to keep costs low, most of the meetings (6/8) will be hosted in Paris (at IRIF or LSV) and the other (2/8) in Bordeaux at LaBRI. For each meeting, and for each non-local member, the cost is estimated to 500 € (the cost is 0 when the meeting is hosted in the city of the lab of the member). We thus obtain (considering that the PhD student and the postdoctoral fellow will not attend every meeting):

For IRIF:	$(7 \times 2 + 1 \times 1) \times 500$ €	$=$	7500 €
For LaBRI:	$(4 \times 6 + 1 \times 2) \times 500$ €	$=$	13000 €
For LSV:	$(3 \times 2) \times 500$ €	$=$	3000 €

International exchanges To further foster collaborations and benefit from outside expertise, we request resources to fund short international visits (to or from abroad for one or two weeks) for each member of the project. The cost of a short visit is estimated to 1500 €.

For IRIF:	6×1500 €	$=$	9000€
For LaBRI:	4×1500 €	$=$	6000 €
For LSV:	3×1500 €	$=$	4500 €

Dissemination We expect the main results of the project to be communicated to top-level international conferences in distributed computing and formal methods and also to workshops, GdR, etc. We estimate the cost of attending a conference or a workshop to 1500 €. We request resources for each member to attend a conference per year. This might be seen as a large amount, but conferences remain the most important way to communicate new results in TCS. It should also be noted that the members of the project have been productive in the recent year.

For IRIF:	$6 \times 4 \times 1500$ €	$=$	36000 €
For LaBRI:	$4 \times 4 \times 1500$ €	$=$	24000 €
For LSV:	$3 \times 4 \times 1500$ €	$=$	18000 €

2.5.4 Total

	Total $(+00\%)$	Equipment	Staff	Mission
For IRIF:	158 300 €	6 000 €	99 800 €	52500 €
For LaBRI:	102 700 €	3000 €	56700 €	43000 €
For LSV:	31 800 €	1500 €	4800 €	25500 €
TOTAL(+00%)	**292 800 €**			

3 Impact and benefits of the project

3.1 A new framework for the development of distributed algorithm

The goal of the project is to propose a new framework to ease the development of distributed algorithms based on robustness analysis. In other words, it aims at bringing some automated methods in the process of development of these algorithms. At the moment, the way distributed algorithms are produced highly depends on the knowledge and the ability of the researchers in distributed computing. As a consequence, they can be false, they are difficult to adapt to other contexts, their description is sometimes ambiguous, and they are hard to prove correct. We believe that formal methods, and in particular techniques that have been developed in the last decades, can be adapted and used in order to improve the design process.

Our project will bring a new perspective to the design and analysis of distributed systems. Our working assumption is that we cannot expect to know the execution context precisely. In other words, every time an algorithm is deployed, its execution context will be different. To address this challenge, we will provide (i) means to describe context changes formally; (ii) methods to model and verify one algorithm under various execution contexts; (iii) monitors to detect, at runtime, if the initial assumptions on the execution context still hold. These goals require a paradigm shift from the two communities: From the distributed computing perspective, we need to be more general and think about changing execution contexts. From the verification perspective, we need to be more focused and think only about particular classes of systems that are actually variations on one algorithm. If successful, our project will open new horizons for the two fields and provide advanced automated techniques for the design of robust distributed applications.

Our case study will allow us to implement and test the methods we develop. We intend to construct a prototype that will serve as a proof-of-concept showing how a distributed algorithm can be adapted to an execution context at hand. This will address an issue that slowly but surely becomes a very important topic in software development.

Finally, we can expect as well to create new adaptative algorithms based on the robustness development of some already existing algorithms. It is to be expected that such algorithms are the future of distributed computing. In fact, there already exist algorithms that are based on this idea of adaptation to the execution contexts and that are succesfully used. An important example is the Paxos algorithm which is widely used in implementations of georeplicated data structures. In fact, the idea behind Paxos, whose role is to solve the consensus in a network with failures, is that a good execution context allows it to answer better, but in bad executive contexts where, for instance, a bounded number of entities do not answer, Paxos can however make progress.

3.2 Reconciliation of two research communities

This proposal aims to reconcile *distributed computing*, in its aspect of the construction and analysis of distributed algorithms, and *verification*, which is targeted on the automated analysis of distributed systems. It is in line with a more general trend of establishing links between the theory of algorithms (algorithm design and analysis, complexity, etc.) and the theory of software (verification, certification, semantics, programming, etc.).[2] More precisely, FREDDA gathers leading French experts from the two communities to develop new ways of analyzing and modifying distributed algorithms from a specifically chosen class of algorithms. At the international level, research topics related to our project are regularly presented at top conferences in both areas. However, leading conferences such as PODC (ACM Symposium on Principles of Distributed Computing) and CONCUR (International Conference on Concurrency Theory) have quasi disjoint audiences, as the two communities evolved independently during the past three decades. This situation slowly changes now, due to significant challenges raised by distributed systems. Members of our group were already involved in organizing a successful workshop gathering researchers from the two areas (`http://www.labri.fr/perso/travers/DRV2016`).

3.3 Dissemination

The scientific dissemination will consist in publications at top-level international conferences. As underlined above, such cross-community publications are only starting to appear (some of them have been co-authored by members of our project). We expect this trend to gain momentum. We will also aim at giving cross-community courses on the master level at our respective institutions, and present the results of our project at some summer schools. As we have seen, we will organize two workshops to promote our findings, to encourage new collaborations, and to build a cross-area community involving researchers from both verification and distributed computing. The first workshop is the Dagstuhl seminar 18211 "Formal Methods and Fault-Tolerant Distributed Computing: Forging an Alliance", which is scheduled in May 2018. The second workshop could be a satellite workshop to one of the top conferences either in verification or in distributed algorithms. These workshops will continue and extend a workshop organized in 2016 in Italy (Bertinoro Workshop on Distributed Runtime Verification) by participants of this project (P. Fraignaud and C. Travers).

We expect that the ideas developed in this project could be taught at some Master 2 level. They may also give birth to new courses where formal methods and the design of distributed algorithms are more interlaced than it is currently the case.

[2] Cf. Moshe Y. Vardi: Communications of the ACM 58(8), page 5, 2015.