<table>
<thead>
<tr>
<th>Project-Team</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ACES Project-Team</td>
<td>5</td>
</tr>
<tr>
<td>2. ADAM Project-Team</td>
<td>9</td>
</tr>
<tr>
<td>3. AMAZONES Team</td>
<td>12</td>
</tr>
<tr>
<td>4. ARLES Project-Team</td>
<td>14</td>
</tr>
<tr>
<td>5. ASAP Project-Team</td>
<td>20</td>
</tr>
<tr>
<td>6. ASCOLA Project-Team</td>
<td>22</td>
</tr>
<tr>
<td>7. ATLANMOD Team</td>
<td>26</td>
</tr>
<tr>
<td>8. CIDRE Project-Team</td>
<td>28</td>
</tr>
<tr>
<td>9. FOCUS Project-Team</td>
<td>32</td>
</tr>
<tr>
<td>10. INDES Project-Team</td>
<td>33</td>
</tr>
<tr>
<td>11. MYRIADS Team</td>
<td>34</td>
</tr>
<tr>
<td>12. OASIS Project-Team</td>
<td>37</td>
</tr>
<tr>
<td>13. PHOENIX Project-Team</td>
<td>39</td>
</tr>
<tr>
<td>14. POPS Project-Team</td>
<td>42</td>
</tr>
<tr>
<td>15. REGAL Project-Team</td>
<td>44</td>
</tr>
<tr>
<td>16. RMOD Project-Team</td>
<td>46</td>
</tr>
<tr>
<td>17. SARDES Project-Team</td>
<td>51</td>
</tr>
<tr>
<td>18. SCORE Team</td>
<td>53</td>
</tr>
<tr>
<td>19. TRISKELL Project-Team</td>
<td>55</td>
</tr>
<tr>
<td>20. ALGORILLE Project-Team</td>
<td>59</td>
</tr>
<tr>
<td>21. CEPAGE Project-Team</td>
<td>64</td>
</tr>
<tr>
<td>22. GRAAL Project-Team</td>
<td>68</td>
</tr>
<tr>
<td>23. GRAND-LARGE Project-Team</td>
<td>71</td>
</tr>
<tr>
<td>24. HIEPACS Project-Team</td>
<td>80</td>
</tr>
<tr>
<td>25. KERDATA Team</td>
<td>89</td>
</tr>
<tr>
<td>26. MESCAL Project-Team</td>
<td>93</td>
</tr>
<tr>
<td>27. MOAIS Project-Team</td>
<td>97</td>
</tr>
<tr>
<td>28. RUNTIME Project-Team</td>
<td>103</td>
</tr>
<tr>
<td><strong>DISTRIBUTED AND HIGH PERFORMANCE COMPUTING</strong></td>
<td></td>
</tr>
<tr>
<td>29. DIONYSOS Project-Team</td>
<td>106</td>
</tr>
<tr>
<td>30. DISTRIBCOM Project-Team</td>
<td>108</td>
</tr>
<tr>
<td>31. DNET Team</td>
<td>112</td>
</tr>
<tr>
<td>32. GANG Project-Team (section vide)</td>
<td>115</td>
</tr>
<tr>
<td>33. HIPERCOM Project-Team</td>
<td>116</td>
</tr>
<tr>
<td>34. MADYNES Project-Team</td>
<td>119</td>
</tr>
<tr>
<td>35. MAESTRO Project-Team</td>
<td>123</td>
</tr>
<tr>
<td>36. MASCOTTE Project-Team</td>
<td>124</td>
</tr>
<tr>
<td>37. PLANETE Project-Team</td>
<td>125</td>
</tr>
<tr>
<td>38. RAP Project-Team</td>
<td>126</td>
</tr>
<tr>
<td><strong>NETWORKS AND TELECOMMUNICATIONS</strong></td>
<td></td>
</tr>
</tbody>
</table>


3. Scientific Foundations

3.1. Programming Context

The goal of ambient computing is to seamlessly merge virtual and real environments. A real environment is composed of objects from the physical world, e.g., people, places, machines. A virtual environment is any information system, e.g., the Web. The integration of these environments must permit people and their information systems to implicitly interact with their surrounding environment.

Ambient computing applications are able to evaluate the state of the real world through sensing technologies. This information can include the position of a person (caught with a localization system like GPS), the weather (captured using specialized sensors), etc. Sensing technologies enable applications to automatically update digital information about events or entities in the physical world. Further, interfaces can be used to act on the physical world based on information processed in the digital environment. For example, the windows of a car can be automatically closed when it is raining.

This real-world and virtual-world integration must permit people to implicitly interact with their surrounding environment. This means that manual device manipulation must be minimal since this constrains person mobility. In any case, the relative small size of personal devices can make them awkward to manipulate. In the near future, interaction must be possible without people being aware of the presence of neighbouring processors.

Information systems require tools to capture data in its physical environment, and then to interpret, or process, this data. A context denotes all information that is pertinent to a person-centric application. There are three classes of context information:

- The digital context defines all parameters related to the hardware and software configuration of the device. Examples include the presence (or absence) of a network, the available bandwidth, the connected peripherals (printer, screen), storage capacity, CPU power, available executables, etc.
- The personal context defines all parameters related to the identity, preferences and location of the person who owns the device. This context is important for deciding the type of information that a personal device needs to acquire at any given moment.
- The physical context relates to the person’s environment; this includes climatic condition, noise level, luminosity, as well as date and time.

All three forms of context are fundamental to person-centric computing. Consider for instance a virtual museum guide service that is offered via a PDA. Each visitor has his own PDA that permits him to receive and visualise information about surrounding artworks. In this application, the pertinent context of the person is made up of the artworks situated near the person, the artworks that interest him as well as the degree of specialisation of the information, i.e., if the person is an art expert, he will desire more detail than the occasional museum visitor.

There are two approaches to organising data in a real to virtual world mapping: a so-called logical approach and a physical approach. The logical approach is the traditional way, and involves storing all data relevant to the physical world on a service platform such as a centralised database. Context information is sent to a person in response to a request containing the person’s location co-ordinates and preferences. In the example of the virtual museum guide, a person’s device transmits its location to the server, which replies with descriptions of neighbouring artworks.
The main drawbacks of this approach are scalability and complexity. Scalability is a problem since we are evolving towards a world with billions of embedded devices; complexity is a problem since the majority of physical objects are unrelated, and no management body can cater for the integration of their data into a service platform. Further, the model of the physical world must be up to date, so the more dynamic a system, the more updates are needed. The services platform quickly becomes a potential bottleneck if it must deliver services to all people.

The physical approach does not rely on a digital model of the physical world. The service is computed wherever the person is located. This is done by spreading data onto the devices in the physical environment; there are a sufficient number of embedded systems with wireless transceivers around to support this approach. Each device manages and stores the data of its associated object. In this way, data are physically linked to objects, and there is no need to update a positional database when physical objects move since the data physically moves with them.

With the physical approach, computations are done on the personal and available embedded devices. Devices interact when they are within communication range. The interactions constitute delivery of service to the person. Returning to the museum example, data is directly embedded in a painting’s frame. When the visitor’s guide meets (connects) to a painting’s devices, it receives the information about the painting and displays it.

3.2. Spatial Information Systems

One of the major research efforts in ACES over the last few years has been the definition of the Spread programming model to cater for spatial context. The model is derived from the Linda [11] tuple-space model. Each information item is a tuple, which is a sequence of typed data items. For example, <10, 'Peter', -3.14> is a tuple where the first element is the integer 10, the second is the string "Peter" and the third is the real value -3.14. Information is addressed using patterns that match one or a set of tuples present in the tuple-space. An example pattern that matches the previous tuple is <int, 'Peter', float>. The tuple-space model has the advantage of allowing devices that meet for the first time to exchange data since there is no notion of names or addresses.

Data items are not only addressed by their type, but also by the physical space in which they reside. The size of the space is determined by the strength of the radio signal of the device. The important difference between Spread and other tuple-space systems (e.g., Sun’s JavaSpaces [10], IBM’s T-Space [14]) is that when a program issues a matching request, only the tuples filling the physical space of the requesting program are tested for matching. Thus, though SIS (Spatial Information Systems) applications are highly distributed by nature, they only rely on localised communications; they do not require access to a global communication infrastructure. Figure 1 shows an example of a physical tuple space, made of tuples arranged in the space and occupying different spaces.

As an example of the power of this model, consider two of the applications that we have developed using it.

- **Ubi-bus** is a spatial information application whose role is to help blind and partially blind people use public transport. When taking a bus, a blind person uses his PDA to signal his intention to a device embedded in the bus stop; this device then contacts the bus on the person’s behalf. This application illustrates how data is distributed over the objects of the physical world, and generally, how devices complement human means of communication.

- **Ubi-board** is a spatial information application designed for public electronic billboards. Travel hotspots like airports and major train stations have an international customer base, so bill-board announcements need to be made in several languages. In Ubi-bus, a billboard has an embedded device. When a person comes within communication range of the billboard, his device sends a request to the billboard asking it to print the message in the language of the person. In the case where several travellers are in proximity of the billboard, the board sends a translation of its information message to each person. The Ubi-board application illustrates personal context in use, i.e., the choice of natural language, and also how actions can be provoked in the physical world without explicit intervention by the person.
3.3. Coupled objects

Integrity checking is an important concern in many activities, both in the real world and in the information society. The basic purpose is to verify that a set of objects, parts, components, people remains the same along some activity or process, or remains consistent against a given property (such as a part count).

In the real world, it is a common step in logistic: objects to be transported are usually checked by the sender (for their conformance to the recipient expectation), and at arrival by the recipient. When a school get a group of children to a museum, people responsible for the children will regularly check that no one is missing. Yet another common example is to check for our personal belongings when leaving a place, to avoid lost. While important, these verification are tedious, vulnerable to human errors, and often forgotten.

Because of these vulnerabilities, problems arise: E-commerce clients sometimes receive incomplete packages, valuable and important objects (notebook computers, passports etc.) get lost in airports, planes, trains, hotels, etc. with sometimes dramatic consequences.

While there are very few automatic solutions to improve the situation in the real world, integrity checking in the computing world is a basic and widely used mechanism: magnetic and optical storage devices, network communications are all using checksums and error checking code to detect information corruption, to name a few.

The emergence of ubiquitous computing and the rapid penetration of RFID devices enable similar integrity checking solutions to work for physical objects. We introduced the concept of coupled object, which offers simple yet powerful mechanisms to check and ensure integrity properties for set of physical objects.

Essentially, coupled objects are a set of physical objects which defines a logical group. An important feature is that the group information is self contained on the objects which allow to verify group properties, such as completeness, only with the objects. Said it another way, the physical objects can be seen as fragments of
a composite object. A trivial example could be a group made of a person, his jacket, his mobile phone, his passport and his cardholder.

The important feature of the concept are its distributed, autonomous and anonymous nature: it allows the design and implementation of pervasive security applications without any database tracking or centralized information system support. This is a significant advantage of this approach given the strong privacy issues that affect pervasive computing.
3. Scientific Foundations

3.1. Introduction

In order to cope with our objective, we will consider software paradigms that will help us in our approach at the various levels of our life-cycle of adaptive systems, but also in the tools themselves for their composition. We will also study these paradigms in the middleware and application design in order to extend them and to have a better understanding. These extensions will be formalized as much as possible.

3.1.1. Aspect-Oriented Software Development (AOSD)

In modern software engineering, language constructs are classified according to how they recombine partial solutions for subproblems of a problem decomposition. Some constructs (e.g., methods and classes) recombine partial solutions using classic hierarchical composition. Others recombine the partial solution using what is known as crosscutting (a.k.a. aspectual) composition. With crosscutting composition, two partial solutions (called aspects) are woven into each other in a way that is dictated by so-called pointcut languages. The necessity of crosscutting composition is the main motivation for the AOSD [49], [64] paradigm. The challenge will be first to study new expressive pointcut languages in order to have a better description of composition locations in adaptable software. The second objective will be to extend and to integrate new techniques of weaving at design time, but also at run time in order to compose software safely. The third objective will be to go beyond simple aspects as persistence and logging services. We plan to study complex aspects such as transactions or replication and to control their weaving in order to master the evolution of complex software.

3.1.2. Component-Based Software Engineering (CBSE)

In a post-object world [62], software components [66] are, with other artifacts such as aspects, one of the approaches that aims at overcoming the limitations of objects and providing more flexibility and dynamicity to complex applications. For that, software components present many interesting properties, such as modularity, encapsulation, and composability. Yet, many different component models and frameworks exist. A survey of the literature references more than 20 different models (including the most well-known, such as EJB [48] and CCM [47]), but the exact number is certainly closer to 30. Indeed, each new author proposes a model to address her/his own need related to a particular execution environment (from grid computing to embedded systems) or the technical services (from advanced transactions to real-time properties), which must be provided to the application components. These different component models seldom interoperate and their design and implementation are never founded on a common ground. The research challenge that we identify is to define and implement solutions for adaptive software components. These components will be adaptive in the sense that they will be able to accommodate execution environments of various granularities (from grid computing, to Internet-based applications, to mobile applications, to embedded systems) and incorporate on-demand different technical services. This challenge will be conducted by designing a micro-kernel for software components. This micro-kernel will contain a well-defined set of core concepts, which are at the root of all component models. Several concrete software component models will then be derived from this micro-kernel.

3.1.3. Context-Aware Computing (CAC)

In adaptive systems, the notion of “context” becomes increasingly important. For example, mobile devices sense the environment they are in and react accordingly. This is usually enabled by a set of rules that infer how to react given a certain situation. In the Ambient/Ubiquitous/Pervasive domain\(^1\), CAC is commonly referred to as the new paradigm that employs this idea of context in order to enmesh computing in our daily lives [68]. Many efforts that exist today focus on human-computer interaction based on context. On the one hand,\(^1\)These terms are more or less equivalent.
computational models, middleware, and programming languages are being developed to take the inherent characteristics of multi-scale environments into account, such as connection volatility, ambient resources, etc. An important challenge is to bridge the gap between the domain level and the computational level. The former is concerned with the expected behavior of the system from a user’s viewpoint, such as how and when a system responds to changes in the context, when information can be made public, etc. On the other hand, the computational level deals with the inherent and very stringent hardware phenomena of multi-scale environments. Nevertheless, both levels have to coexist: the computational level needs to be steered by the concepts, behavior and rules which exist at the domain level, whereas the domain needs to adapt to the specificities of the ever changing environment that is monitored and managed by the computational level. In order to address this challenge, we first intend to investigate representations at the domain level of concepts such as user profile, local positioning information and execution context [78]. Furthermore, a mapping has to be devised between these concepts and generic concepts at the computational level, the latter being as independent as possible from concrete platforms or languages. This mapping has to be bidirectional: the computational level needs to be steered by the concepts, behavior and rules that exist at the domain level, whereas the domain needs to adapt to the particulars of the ever-changing environment that is monitored and managed at the computational level. Furthermore, the mapping has to be dynamic since the changes have to be propagated between the levels at run time. An explicit domain level is not only useful for bridging the aforementioned gap, but also for designing and developing open task-specific languages at the domain level, which allow users to dynamically adapt the behavior of the applications in multi-scale environments in well-defined ways.

We will base the design approach of the future implementation prototype on MDE. The goal of MDE [74] consists of developing, maintaining and evolving complex software systems by raising the level of abstraction from source code to models. The latter is in our case the domain level, which will be connected to the computational level by means of MDE techniques. One added benefit of MDE is that it provides means for managing model inconsistencies.

3.2. Two Research Directions

We propose to follow two research directions to foster software reuse and adaptation. The first direction, that could be coined as the spatial dimension of adaptation, will provide middleware platforms to let applications be adapted to changing execution contexts. The second direction, the so-called temporal dimension of adaptation, will provide concepts and artifacts to let designers specify evolvable applications.

3.2.1. Adaptable Component Frameworks for Middleware

As a cornerstone of next generation software, adaptation is a property which must be present throughout the entire life cycle, from design to execution. We develop then a vision where adaptation is not only a property that is desirable for end-user applications, but also for the middleware platform that executes these applications. Until now, middleware is a rather specialized activity where each new environment forces the development of a corresponding platform, which is specific to the given environment. This has led to a large number of platforms (from Web Services, to EJB, to CORBA, to ad hoc middleware for embedded systems). Although at a high level, solutions for communication interoperability often exist between these platforms, they stay loosely coupled and separated. Furthermore, the concepts which are at the core of these platforms and their architectures are too different to allow, for example, sharing technical services.

The research challenge that we propose here is to define and develop middleware and associated services which could be adapted to a broad range of environments from grid computing, to Internet-based applications, to local networks, to mobile applications on PDA’s and smart phones, to embedded systems. The benefits of that are twofold. First, it enables the easier deployment of mobile applications in different environments by taking advantage of the common ground provided by adaptable middleware. Second, middleware is a rapidly changing domain where new technologies appear frequently. Yet, up to now, each new technological shift has imposed a complete re-development of the middleware. Having a common ground on which middleware is built would help in such transitions by fostering reuse. In terms of industrial output, the impact of these
results will also be helpful for software editors and companies to adapt their products more rapidly to new and emerging middleware technologies.

This research challenge has close links with MDE and product line families. We believe that the added value of our proposal is to cover a more integrated solution: we are not only interested in middleware design with MDE technologies, but we also wish to integrate them with software component technologies and advanced programming techniques, such as AOP. We will then cover a broad spectrum of middleware construction, from design (MDE) to implementation (CBSE) to application development (AOP).

3.2.2. Distributed Application Design for Adaptive Platforms

Considering adaptation in the first design steps of an application allows for its preparation and follow-up during the entire life-cycle. As mentioned previously, some software paradigms help already in the design and the development of adaptable applications. AOSD proposes separation of concerns and weaving of models in order to increase the mastering and the evolution of software. MDE consists of evolving complex software systems by raising the level of abstraction from source code to models. Several programming approaches, such as AOP or reflective approaches, have gained in popularity to implement flexibility. Other approaches, such as CBSE, propose compositional way for reuse and compose sub-systems in the application building. Finally, context-aware programming for mobile environment proposes solutions in order to consider context evolution. Overall, the objective of these approaches is to assist the development of applications that are generic and that can be adapted with respect to the properties of the domain or the context.

The research challenge that we propose to address here is similar to static points of variation in product line families. We plan to study dynamic points of variation in order to take into account adaptation in the first design steps and to match this variation. The first research challenge is the introduction of elements in the modeling phase that allow the specification of evolution related properties. These properties must make it possible to build safe and dynamic software architectures. We wish to express and validate properties in the entire software life cycle. These properties are functional, non-functional, static, behavioral, or even qualitative properties. We also want to be able to check that all the properties are present, that the obtained behavior is the expected one, and that the quality of service is not degraded after the addition or the withdrawal of functionalities. We will base our approach on the definition of contracts expressed in various formalisms (e.g., first order logic, temporal logic, state automata) and we will propose a composition of these contracts.

The second challenge will be to implement design processes that maintain coherence between the various stages of modeling in a MDE approach of the applications, as well as maintaining coherence between the phases of modeling and implementation. To do so, we will design and implement tools that will enable traceability and coherence checking between models, as well as between models and the application at execution time.

Finally, we will introduce context information in the development process. At the modeling level, we will represent concepts, behavior and rules of adaptive systems to express adaptation abstraction. These models will be dynamic and connected to implementation levels at the computational level and they will consider context knowledge. The goal is to bridge the gap between the computational level and the domain level in adaptive systems by synchronization of models and implementations, but also by representation of such common knowledge.
3. Scientific Foundations

3.1. An Ambiant Software Stack

Amazones focuses on three goals: designing new software architectures for ambient environments, designing new approaches to the problem of managing these architectures, in particular using formal methods techniques, and finally designing new toolboxes and techniques for observation and control of the system at runtime.

We use a reference operating system stack as a confluence for our research directions. It will allow us to identify and study transversal topics so as to improve operating systems that cope with the specific constraints of both hardware and software for ambient applications.

We design and build this stack ourselves, and it is divided into three layers:

1. a hardware management layer,
2. a portable code layer,
3. a dynamic services layer.

A substantial set of existing approaches, frameworks and tools can be combined to fulfill each level requirements. Without loss of generality, we chose to leverage L4 [43] and the Genode [37] infrastructure at the microkernel management layer, JamVM [42] at the portable code layer, and Apache Felix / ROCS [25] at the dynamic services layer.

The stack is initially used to demonstrate testbeds and transversal demonstrators, but will also serve as a reference model for works. We plan to optimize the stack to obtain at least an adequate operating system for Amazones studies. Nevertheless, our ambition goes beyond “just” assembling those pieces together. Our aim, is indeed to blur the limits between each of the three aforementioned layers, so as to see it as a coherent dynamic operating system rather than yet another proof-of-concept software stack tailored for our very own needs. Merging the middle-layers enables us to reuse standard services such as the registries and the resource allocators and minimizing their number. Merging layers will also need to rely on the same programming paradigms, such as component, service, binding and registries and will lead to a more homogeneous, easier to manipulate operating system. It will be comparable to an OSGi framework since we maintain the same high-level API, and we will be targeted to ARM-like based equipments.
3.2. Programming abstraction and verification

For the local level of abstraction, recursive rule languages (variants of Datalog) have been used to describe communication protocols [44] [45], thus reviving the recursive languages developed in the 80’s for deductive databases [26] [27] [51], well-suited to define routes in networks. Query languages allow the expression of protocols, one or two orders of magnitude simpler than classical imperative programming languages. We continue this trend to demonstrate the potential of declarative rule languages for the local abstraction level, clarifying their semantics in asynchronous distributed computation, investigating further their expressive power and the complexity of their distributed evaluation. The definition of a rule language which admits efficient distributed execution while offering enough expressive power is still an active topic of research [40].

Having correct and robust protocols is fundamental for critical distributed systems. Ensuring the desirable properties of protocols is a very difficult problem. Neither simulations nor testbed implementations can ensure the quality required for network protocols. As an alternative to these methods, some researchers have successfully investigated the use of formal verification as a mean to guarantee the quality of protocols [30] [36] [46] [48].

Formal verification is a technique that assures whether a system enjoys a given property, based on a formal model of the system under evaluation. There are roughly two approaches to formal verification. The first approach is model checking [35], which consists of an exhaustive exploration of all states and transitions in the formal model of the system. A lot of model checking tools have been developed, such as SPIN [41], UPPAAL [24], PRISM [23], etc. The second approach is logical inference. It relies on a formal mathematical model for reasoning about the system, usually using theorem proving software such as the HOL [22], or the Coq [21]. This is usually only partially automated and is driven by the user’s understanding of the system to validate.

We have pursued the second approach, by using the Coq proof assistant for protocols expressed in rule languages. This requires the modeling in Coq of the distributed systems, the machine evaluating the rule programs, as well as the theory of interest for the class of protocols considered.
3. Scientific Foundations

3.1. Introduction

Research undertaken within the ARLES project-team aims to offer comprehensive solutions to support the development of pervasive computing systems that are dynamically composed according to networked resources in the environment. This leads us to investigate methods and tools supporting the engineering of pervasive software systems, with a special emphasis on associated middleware solutions.

3.2. Engineering Pervasive Software Systems

Since its emergence, middleware has proved successful in assisting distributed software development, making development faster and easier, and significantly promoting software reuse while overcoming the heterogeneity of the distributed infrastructure. As a result, middleware-based software engineering is central to the principled development of pervasive computing systems. In this section, we (i) discuss challenges that middleware brings to software engineering, and (ii) outline a revolutionary approach to middleware-based software engineering aiming at the dynamic runtime synthesis of emergent middleware.

3.2.1. Middleware-based Software Engineering

Middleware establishes a new software layer that homogenizes the infrastructure’s diversities by means of a well-defined and structured distributed programming model, relieving software developers from low-level implementation details, by: (i) at least abstracting transport layer network programming via high-level network abstractions matching the application computational model, and (ii) possibly managing networked resources to offer quality of service guarantees and/or domain specific functionalities, through reusable middleware-level services. More specifically, middleware defines:

- A resource definition language that is used for specifying data types and interfaces of networked software resources;
- A high-level addressing scheme based on the underlying network addressing scheme for locating resources;
- Interaction paradigms and semantics for achieving coordination;
- A transport/session protocol for achieving communication; and
- A naming/discovery protocol with related registry structure and matching relation for publishing and discovering the resources available in the given network.

Attractive features of middleware have made it a powerful tool in the software system development practice. Hence, middleware is a key factor that has been and needs to be further taken into account in the Software Engineering (SE) discipline. The advent of middleware standards have further contributed to the systematic adoption of this paradigm for distributed software development.

---

In spite of the above, mature engineering methodologies to comprehensively assist the development of middleware-based software systems, from requirements analysis to deployment and maintenance, are lagging behind. Indeed, systematic software development accounting for middleware support is rather the exception than the norm\(^6\), and methods and related tools are dearly required for middleware-based software engineering. This need becomes even more demanding if we consider the diversity and scale of today’s networking environments and application domains, which makes middleware and its association with applications highly complex\(^5\), raising new, challenging requirements for middleware. Among those, access to computational resources should be open across network boundaries and dynamic due to the potential mobility of host- and user-nodes. This urges middleware to support methods and mechanisms for description, dynamic discovery and association, late binding, and loose coordination of resources. In such variable and unpredictable environments, operating not only according to explicit system inputs but also according to the context of system operation becomes of major importance, which should be enabled by the middleware. Additionally, the networking infrastructure is continuing to evolve at a fast pace, and suggesting new development paradigms for distributed systems, calling for next-generation middleware platforms and novel software engineering processes integrating middleware features in all phases of the software development.

### 3.2.2. Beyond Middleware-based Architectures for Interoperability

As discussed above, middleware stands as the conceptual paradigm to effectively network together heterogeneous systems, specifically providing upper layer interoperability. That said, middleware is yet another technological block, which creates islands of networked systems.

**Interoperable middleware** has been introduced to overcome middleware heterogeneity. However, solutions remain rather static, requiring either use of a proprietary interface or a priori implementation of protocol translators. In general, interoperability solutions solve protocol mismatch among middleware at syntactic level, which is too restrictive. This is even truer when one considers the many dimensions of heterogeneity, including software, hardware and networks, which are currently present in ubiquitous networking environments, and that require fine tuning of the middleware according to the specific capacities embedded within the interacting parties. Thus, interoperable middleware can at best solve protocol mismatches arising among middleware aimed at a specific domain. Indeed, it is not possible to a priori design a universal middleware solution that will enable effective networking of digital systems, while spanning the many dimensions of heterogeneity currently present in networked environments and further expected to increase dramatically in the future.

A revolutionary approach to the seamless networking of digital systems is to synthesize connectors on the fly, via which networked systems communicate. The resulting emergent connectors then compose and further adapt the interaction protocols run by the connected systems, from the application layer down to the middleware layer. Hence, thanks to results in this new area, networked digital systems will survive the obsolescence of interaction protocols and further emergence of new ones.

We have specifically undertaken cooperative research on the dynamic synthesis of emergent connectors which shall rely on a formal foundation for connectors that allows learning, reasoning about, and adapting the interaction behavior of networked systems\(^8\). Further, compared to the state of the art foundations for connectors, it should operate a drastic shift by learning, reasoning about, and synthesizing connector behavior at run-time. Indeed, the use of connector specifications pioneered by the software architecture research field has mainly been considered as a design-time concern, for which automated reasoning is now getting practical even if limitations remain. On the other hand, recent effort in the semantic Web domain brings ontology-based semantic knowledge and reasoning at run-time; however, networked system solutions based thereupon are currently mainly focused on the functional behavior of networked systems, with few attempts to capture their interaction behavior as well as non-functional properties. In this new approach, the interaction protocols (both

---


application- and middleware-layer) behavior will be learnt by observing the interactions of the networked systems, where ontology-based specification and other semantic knowledge will be exploited for generating connectors on the fly.

3.3. Middleware Architectures for Pervasive Computing

Today’s wireless networks enable dynamically setting up temporary networks among mobile nodes for the realization of some distributed function. However, this requires adequate development support and, in particular, supporting middleware platforms for alleviating the complexity associated with the management of dynamic networks composed of highly heterogeneous nodes. In this section, we present an overview of: (i) service oriented middleware, a prominent paradigm in large distributed systems today, and (ii) middleware for wireless sensor networks, which have recently emerged as a promising platform.

3.3.1. Service Oriented Middleware

The Service Oriented Computing (SOC) paradigm advocates that networked resources should be abstracted as services, thus allowing their open and dynamic discovery, access and composition, and hence reuse. Due to this flexibility, SOC has proven to be a key enabler for pervasive computing. Moreover, SOC enables integrating pervasive environments into broader service oriented settings: the current and especially the Future Internet is the ultimate case of such integration. We, more particularly, envision the Future Internet as a ubiquitous setting where services representing resources, people and things can be freely and dynamically composed in a decentralized fashion, which is designated by the notion of service choreography in the SOC idiom. In the following, we discuss the role that service oriented middleware is aimed to have within our above sketched vision of the Future Internet, of which pervasive computing forms an integral part.

From service oriented computing to service oriented middleware: In the last few years, there is a growing interest in choreography as a key concept in forming complex service-oriented systems. Choreography is put forward as a generic abstraction of any possible collaboration among multiple services, and integrates previously established views on service composition, among which service orchestration. Several different approaches to choreography modeling can be found in the literature: Interaction-oriented models describe choreography as a set of interactions between participants; while process-oriented models describe choreography as a parallel composition of the participants’ business processes. Activity-based models focus on the interactions between the parties and their ordering, whereas the state of the interaction is not explicitly modeled or only partly modeled using variables; while state-based models model the states of the choreography as first-class entities, and the interactions as transitions between states.

The above modeling categorizations are applied in the ways in which: service choreographies are specified (e.g., by employing languages such as BPMN, WS-CDL, BPEL); services are discovered, selected and composed into choreographies (e.g., based on their features concerning interfaces, behavior, and non-functional properties such as QoS and context); heterogeneity between choreographed services is resolved via adaptation (e.g., in terms of service features and also underlying communication protocols); choreographies are deployed and enacted (e.g., in terms of deployment styles and execution engines); and choreographies are maintained/adapted given the independent evolution of choreographed services (e.g., in terms of availability and QoS). These are demanding functionalities that service oriented middleware should provide for supporting service choreographies. In providing these functionalities in the context of the Future Internet, service oriented middleware is further challenged by two key Future Internet properties: its ultra large scale as in number of users and services, and the high degree of heterogeneity of services, whose hosting platforms may range from that of resource-rich, fixed hosts to wireless, resource-constrained devices. These two properties call for considerable advances to the state of the art of the SOC paradigm.

Our work in the last years has focused on providing solutions to the above identified challenges, more particularly in the domain of pervasive computing. Given the prevalence of mobile networking environments and powerful hand-held consumer devices, we consider resource constrained devices (and things, although we focus on smart, i.e., computation-enabled, things) as first-class entities of the Future Internet. Concerning middleware that enables networking mobile and/or resource constrained devices in pervasive computing
environments, several promising solutions have been proposed, such as mobile Gaia, TOTA, AlfredO, or work at UCL, Carnegie Mellon University, and the University of Texas at Arlington. They address issues such as resource discovery, resource access, adaptation, context awareness as in location sensitivity, and pro-activeness in a seamless manner. Other solutions specialize in sensor networks; we, more specifically, discuss middleware for wireless sensor networks in the next section. In this very active domain of service-oriented middleware for pervasive computing environments, we have extensive expertise that ranges from lower-level cross-layer networking to higher-level semantics of services, as well as transversal concerns such as context and privacy. We have in particular worked on aspects including semantic discovery and composition of services based on their functional properties, heterogeneity of service discovery protocols, and heterogeneity of network interfaces. Based on our accumulated experience, we are currently focusing on some of the still unsolved challenges identified above.

**QoS-aware service composition:** With regard to service composition in pervasive environments, taking into account QoS besides functional properties ensures a satisfactory experience to the end user. We focus here on the orchestration-driven case, where service composition is performed to fulfill a task requested by the user along with certain QoS constraints. Assuming the availability of multiple resources in service environments, a large number of services can be found for realizing every sub-task part of a complex task. A specific issue emerges in this regard, which is about selecting the best set of services (i.e., in terms of QoS) to participate in the composition, meeting user’s global QoS requirements. QoS-aware composition becomes even more challenging when it is considered in the context of dynamic service environments characterized by changing conditions. As dynamic environments call for fulfilling user requests on the fly (i.e., at run-time) and as services’ availability cannot be known a priori, service selection and composition must be performed at runtime. Hence, the execution time of service selection algorithms is heavily constrained, whereas the computational complexity of this problem is NP-hard.

**Coordination of heterogeneous distributed systems:** Another aspect that we consider important in service composition is enabling integration of services that employ different interaction paradigms. Diversity and ultra large scale of the Future Internet have a direct impact on coordination among interacting entities. Our choice of choreography as global coordination style among services should further be underpinned by support for and interoperability between heterogeneous interaction paradigms, such as message-driven, event-driven and data-driven ones. Different interaction paradigms apply to different needs: for instance, asynchronous, event-based publish/subscribe is more appropriate for highly dynamic environments with frequent disconnections of involved entities. Enabling interoperability between such paradigms is imperative in the extremely heterogeneous Future Internet integrating services, people and things. Interoperability efforts are traditionally based on, e.g., bridging communication protocols, where the dominant position is held by ESBs, wrapping systems behind standard technology interfaces, and/or providing common API abstractions. However, such efforts mostly concern a single interaction paradigm and thus do not or only poorly address cross-paradigm interoperability. Efforts combining diverse interaction paradigms include: implementing the LIME tuple space middleware on top of a publish/subscribe substrate; enabling Web services/SOAP-based interactions over a tuple space binding; and providing ESB implementations based on the tuple space paradigm.

**Evolution of service oriented applications:** A third issue we are interested in concerns the maintenance of service-oriented applications despite the evolution of employed services. Services are autonomous systems that have been developed independently from each other. Moreover, dynamics of pervasive environments and the Future Internet result in services evolving independently; a service may be deployed, or un-deployed at anytime; its implementation, along with its interface may change without prior notification. In addition, there are many evolving services that offer the same functionality via different interfaces and with varying quality characteristics (e.g., performance, availability, reliability). The overall maintenance process amounts to replacing a service that no longer satisfies the requirements of the employing application with a substitute service that offers the same or a similar functionality. The goal of seamless service substitution is to relate the substitute service with the original service via concrete mappings between their operations, their inputs and outputs. Based on such mappings, it is possible to develop/generate an adapter that allows the employing application to access the substitute service without any modification in its implementation. The
service substitution should be dynamic and efficient, supported by a high level of automation. The state of the art in service substitution comprises various approaches. There exist efforts, which assume that the mappings between the original and the substitute service are given, specified by the application or the service providers. The human effort required makes these approaches impractical, especially in the case of pervasive environments. On the other hand, there exist automated solutions, proposing mechanisms for the derivation of mappings. The complexity of these approaches scales up with the cardinality of available services and therefore efficiency is compromised. Again, this is an important disadvantage, especially considering the case of pervasive environments.

3.3.2. Middleware for Wireless Sensor Networks

Wireless sensor networks (WSNs) enable low cost, dense monitoring of the physical environment through collaborative computation and communication in a network of autonomous sensor nodes, and are an area of active research. Owing to the work done on system-level functionalities such as energy-efficient medium access and data-propagation techniques, sensor networks are being deployed in the real world, with an accompanied increase in network sizes, amount of data handled, and the variety of applications. The early networked sensor systems were programmed by the scientists who designed their hardware, much like the early computers. However, the intended developer of sensor network applications is not the computer scientist, but the designer of the system using the sensor networks, which might be deployed in a building or a highway. We use the term domain expert to mean the class of individuals most likely to use WSNs – people who may have basic programming skills but lack the training required to program distributed systems. Examples of domain experts include architects, civil and environmental engineers, traffic system engineers, medical system designers etc. We believe that the wide acceptance of networked sensing is dependent on the ease-of-use experienced by the domain expert in developing applications on such systems.

The obvious solution to enable this ease-of-use in application development is sensor network middleware, along with related programming abstractions. Recent efforts in standardizing network-layer protocols for embedded devices provide a sound foundation for research and development of middleware that assist the sensor network developers in various aspects that are of interest to us, including the following.

Data-oriented operations: A large number of WSN applications are concerned with sampling and collection of data, and this has led to a large body of work to provide middleware support to the programmer of WSNs for easy access to the data generated and needed by the constituent nodes. Initial work included Hood, and TeenyLIME, which allowed data-sharing over a limited spatial range. Further work proposed the use of the DART runtime environment, which exposes the sensor network as a distributed data-store, addressable by using logical addresses such as “all nodes with temperature sensors in Room 503”, or “all fire sprinklers in the fifth and sixth floors”, which are more intuitive than, say, IP addresses. Taking a different approach toward handling the data in the sensor network, some middleware solutions propose to manipulate them using semantic techniques, such as in the Triple Space Computing approach, which models the data shared by the nodes in the system as RDF triples (subject-predicate-object groups), a standard method for semantic data representation. They propose to make these triples available to the participating nodes using a tuple space, thus giving it the “triple space” moniker. S-APL or Semantic-Agent Programming Language uses semantic technologies to integrate the semantic descriptions of the domain resources with the semantic prescription of agent behavior.

Integration with non-WSN nodes: Most of the work above focuses on designing applications that exhibit only intra-network interactions, where the interaction with the outside world is only in the form of sensing it, or controlling it by actuation. The act of connecting this data to other systems outside the sensor network is mostly done using an external gateway. This is then supported by middlewares that expose the sensor network as a database (e.g., TinyDB and Cougar), allowing the operator to access the data using a SQL-like syntax, augmented with keywords that can be used to specify the rate of sampling, for example. Another direction of integrating WSNs in general with larger systems such as Web servers has been toward using REST

---

(REpresentational State Transfer) technologies, which are already used for accessing services on the Web as a lightweight alternative to SOAP. There has also been work proposing a system that will enable heterogeneous sensor and actuators to expose their sensing and actuation capabilities in a plug and play fashion. It proposes a middleware that defines a set of constraints, support services and interaction patterns that follow the REST architectural style principles, using the ATOM Web publishing protocol for service description, and a two-step discovery process. Additionally, there has been work in implementation of a REST-oriented middleware that runs on embedded devices such as Sun SPOT nodes, and the Plogg wireless energy monitors. This involves a two-fold approach — embedding tiny Web servers in devices that can host them, and employing a proxy server in situations where that is not the case. However, it has been noticed that the abstractions provided by REST might be too simplistic to compose complex applications over the services provided by WSN nodes. Some of the most recent work in this area also proposes to convert existing (network-layer) gateways into smart gateways, by running application code on them.

In addition to supporting the above interactions, sensor network middleware has also been proposed to address the challenges arising from the fact that a particular sensor or actuator may not be always available. This leads to the need for transparent reconfiguration, where the application developer should not have to care about reliability issues. The PIRATES event-based middleware for resource-rich nodes (hosting sensors/actuators, or just processing data) includes a third-party-remapping facility that can be used to remap a component’s endpoints without affecting the business logic. In that sense, it is similar to the RUNES middleware targeted at embedded systems.

Finally, we also note the recent initial WSN middleware research focused on the new nascent classes of systems. Most recently, the field of participatory sensing has emerged, where the role of sensing is increasingly being performed by the mobile phones carried by the users of the system, providing data captured using the sound, GPS, accelerometer and other sensors attached to them. This has led to the emergence of middleware such as JigSaw. The core additional challenges in this domain come from the inherent mobility of the nodes, as well as their extremely large scale.

---

3. Scientific Foundations

3.1. Distributed Computing

Distributed computing was born in the late seventies when people started taking into account the intrinsic characteristics of physically distributed systems. The field then emerged as a specialized research area distinct from networks, operating systems and parallelism. Its birth certificate is usually considered as the publication in 1978 of Lamport’s most celebrated paper “Time, clocks and the ordering of events in a distributed system” [60] (that paper was awarded the Dijkstra Prize in 2000). Since then, several high-level journals and (mainly ACM and IEEE) conferences have been devoted to distributed computing. The distributed systems area has continuously been evolving, following the progresses of all the above-mentioned areas such as networks, computing architecture, operating systems.

The last decade has witnessed significant changes in the area of distributed computing. This has been acknowledged by the creation of several conferences such as NSDI and IEEE P2P. The NSDI conference is an attempt to reassemble the networking and system communities while the IEEE P2P conference was created to be a forum specialized in peer-to-peer systems. At the same time, the EuroSys conference originated as an initiative of the European Chapter of the ACM SIGOPS to gather the system community in Europe.

3.2. Theory of distributed systems

Finding models for distributed computations prone to asynchrony and failures has received a lot of attention. A lot of research in this domain focuses on what can be computed in such models, and, when a problem can be solved, what are its best solutions in terms of relevant cost criteria. An important part of that research is focused on distributed computability: what can be computed when failure detectors are combined with conditions on process input values for example. Another part is devoted to model equivalence. What can be computed with a given class of failure detectors? Which synchronization primitives is a given failure class equivalent to?). Those are among the main topics addressed in the leading distributed computing community.

A second fundamental issue related to distributed models, is the definition of appropriate models suited to dynamic systems. Up to now, the researchers in that area consider that nodes can enter and leave the system, but do not provide a simple characterization, based on properties of computation instead of description of possible behaviors [61], [55], [56]. This shows that finding dynamics distributed computing models is today a "Holy Grail", whose discovery would allow a better understanding of the essential nature of dynamics systems.

3.3. Peer-to-peer overlay networks

A standard distributed system today is related to thousand or even millions of computing entities scattered all over the world and dealing with a huge amount of data. This major shift in scalability requirements has lead to the emergence of novel computing paradigms. In particular, the peer-to-peer communication paradigm imposed itself as the prevalent model to cope with the requirements of large scale distributed systems. Peer-to-peer systems rely on a symmetric communication model where peers are potentially both client and servers. They are fully decentralized, thus avoiding the bottleneck imposed by the presence of servers in traditional systems. They are highly resilient to peers arrivals and departures. Finally, individual peer behavior is based on a local knowledge of the system and yet the system converges toward global properties.
A peer-to-peer overlay network logically connect peers on top of IP. Two main classes of such overlays dominate, structured and unstructured. The differences relate to the choice of the neighbors in the overlay, and the presence of an underlying naming structure. Overlay networks represent the main approach to build large-scale distributed systems that we retained. An overlay network forms a logical structure connecting participating entities on top of the physical network, be it IP or a wireless network. Such an overlay might form a structured overlay network [62], [63], [64] following a specific topology or an unstructured network [59], [65] where participating entities are connected in a random or pseudo-random fashion. In between, lie weakly structured peer-to-peer overlays where nodes are linked depending on a proximity measure providing more flexibility than structured overlays and better performance than fully unstructured ones. Proximity-aware overlays connect participating entities so that they are connected to close neighbors according to a given proximity metric reflecting some degree of affinity (computation, interest, etc.) between peers. We extensively use this approach to provide algorithmic foundations of large-scale dynamic systems.

3.4. Epidemic protocols

Epidemic algorithms, also called gossip-based algorithms [58], [57], are consistently used in our research. In the context of distributed systems, epidemic protocols are mainly used to create overlay networks and to ensure a reliable information dissemination in a large-scale distributed system. The principle underlying the technique, in analogy with the spread of a rumor among humans via gossiping, is that participating entities continuously exchange information about the system in order to spread it gradually and reliably. Epidemic algorithms have proved efficient to build and maintain large-scale distributed systems in the context of many applications such as broadcasting [57], monitoring, resource management, search, and more generally in building unstructured peer-to-peer networks.

3.5. Malicious process behaviors

When assuming that processes fail by simply crashing, bounds on resiliency (maximum number of processes that may crash), number of exchanged messages, number of communication steps, etc. either in synchronous and augmented asynchronous systems (recall that in purely asynchronous systems some problems are impossible to solve) are known. If processes can exhibit malicious behaviors, these bounds are seldom the same. Sometimes, it is even necessary to change the specification of the problem. For example, the consensus problem for correct processes does not make sense if some processes can exhibit a Byzantine behavior and thus propose arbitrary value. In this case, the validity property of consensus, which is normally "a decided value is a proposed value", must be changed to "if all correct processes propose the same value then only this value can be decided". Moreover, the resilience bound of less than half of faulty processes is at least lowered to "less than a third of Byzantine processes". These are some of the aspects that underlie our studies in the context of the classical model of distributed systems, in peer-to-peer systems and in sensor networks.

3.6. Online Social Networks

Social Networks have rapidly become a fundamental component of today’s distributed applications. Web 2.0 applications have dramatically changed the way users interact with the Internet and with each other. The number of users of websites like Flickr, Delicious, Facebook, or MySpace is constantly growing, leading to significant technical challenges. On the one hand, these websites are called to handle enormous amounts of data. On the other hand, news continue to report the emergence of privacy threats to the personal data of social-network users. Our research aims to exploit our expertise in distributed systems to lead to a new generation of scalable, privacy-preserving, social applications.
3. Scientific Foundations

3.1. Overview

Since we mainly work on new software structuring concepts and programming language design, we first briefly introduce some basic notions and problems of software components (understood in a broad sense, i.e., including modules, objects, architecture description languages and services), aspects, and domain-specific languages. We conclude by presenting the main issues related to distribution and concurrency that are relevant to our work.

3.2. Software Components

Modules and services. The idea that building software components, i.e., composable prefabricated and parametrized software parts, was key to create an effective software industry was realized very early [78]. At that time, the scope of a component was limited to a single procedure. In the seventies, the growing complexity of software made it necessary to consider a new level of structuring and programming and led to the notions of information hiding, modules, and module interconnection languages [87], [60]. Information hiding promotes a black-box model of program development whereby a module implementation, basically a collection of procedures, is strongly encapsulated behind an interface. This makes it possible to guarantee logical invariant properties of the data managed by the procedures and, more generally, makes modular reasoning possible. In a first step, it is possible to reason locally, about the consistency between the module implementation and the module interface. In a second step, it is possible to reason about composing modules by only considering their interfaces. Modern module systems also consider types as module elements and consider, typically static, modules as a unit of separate compilation, with the most sophisticated ones also supporting modules parametrized by modules [77].

Object-Oriented Programming. Classes and objects provide another kind of software component, which makes it necessary to distinguish between component types (classes) and component instances (objects). Indeed, unlike modules, objects can be created dynamically. Although it is also possible to talk about classes in terms of interfaces and implementations, the encapsulation provided by classes is not as strong as the one provided by modules. This is because, through the use of inheritance, object-oriented languages put the emphasis on incremental programming to the detriment of modular programming. This introduces a white-box model of software development and more flexibility is traded for safety as demonstrated by the fragile base class issue [82].

Architecture Description Languages. The advent of distributed applications made it necessary to consider more sophisticated connections between the various building blocks of a system. The software architecture [90] of a software system describes the system as a composition of components and connectors, where the connectors capture the interaction protocols between the components [52]. It also describes the rationale behind such a given architecture, linking the properties required from the system to its implementation. Architecture Descriptions Languages (ADLs) are languages that support architecture-based development [79]. A number of these languages make it possible to generate executable systems from architectural descriptions, provided implementations for the primitive components are available. However, guaranteeing that the implementation conforms to the architecture is an issue.
3.3. Aspect-Oriented Programming

The main driving force for the structuring means, such as components and modules, is the quest for clean *separation of concerns* [62] on the architectural and programming levels. It has, however, early been noted that concern separation in the presence of crosscutting functionalities requires specific language and implementation level support. Techniques of so-called *computational reflection*, for instance, Smith’s 3-Lisp or Kiczales’s CLOS meta-object protocol [91], [74] as well as metaprogramming techniques have been developed to cope with this problem but proven unwieldy to use and not amenable to formalization and property analysis due to their generality.

*Aspect-Oriented Software Development* [73], [50] has emerged over the previous decade as the domain of systematic exploration of crosscutting concerns and corresponding support throughout the software development process. The corresponding research efforts have resulted, in particular, in the recognition of *crosscutting* as a fundamental problem of virtually any large-scale application, and the definition and implementation of a large number of aspect-oriented models and languages.

However, most current aspect-oriented models, notably AspectJ [72], rely on pointcuts and advice defined in terms of individual execution events. These models are subject to serious limitations concerning the modularization of crosscutting functionalities in distributed applications, the integration of aspects with other modularization mechanisms such as components, and the provision of correctness guarantees of the resulting AO applications. They do, in particular, only permit the manipulation of distributed applications on a per-host basis, that is, without direct expression of coordination properties relating different distributed entities [92]. Similarly, current approaches for the integration of aspects and (distributed) components do not directly express interaction properties between sets of components but rather seemingly unrelated modifications to individual components [59]. Finally, current formalizations of such aspect models are formulated in terms of low-level semantic abstractions (see, e.g., Wand’s et al semantics for AspectJ [94]) and provide only limited support for the analysis of fundamental aspect properties.

Recently, first approaches have been put forward to tackle these problems, in particular, in the context of so-called *stateful* or *history-based aspect languages* [63], [64], which provide pointcut and advice languages that directly express rich relationships between execution events. Such languages have been proposed to directly express coordination and synchronization issues of distributed and concurrent applications [86], [54], [66], provide more concise formal semantics for aspects and enable analysis of their properties [53], [65], [63], [51]. Due to the novelty of these approaches, they represent, however, only first results and many important questions concerning these fundamental issues remain open.

3.4. Protocols

Today, protocols constitute a frequently used means to precisely define, implement, and analyze contracts between two or more hardware or software entities. They have been used to define interactions between communication layers, security properties of distributed communications, interactions between objects and components, and business processes.

Object interactions [85], component interactions [95], [88] and service orchestrations [61] are most frequently expressed in terms of *regular interaction protocols* that enable basic properties, such as compatibility, substitutability, and deadlocks between components to be defined in terms of basic operations and closure properties of finite-state automata. Furthermore, such properties may be analyzed automatically using, e.g., model checking techniques [57], [68].

However, the limited expressive power of regular languages has led to a number of approaches using more expressive *non-regular* interaction protocols that often provide distribution-specific abstractions, e.g., session types [71], or context-free or turing-complete expressiveness [89], [56]. While these protocol types allow conformance between components to be defined (e.g., using unbounded counters), property verification can only be performed manually or semi-automatically.
Furthermore, first approaches for the definition of aspects over protocols have been proposed, as well as over regular structures [63] and non-regular ones [93], [84]. The modification of interaction protocols by aspects seems highly promising for the integration of aspects and components.

3.5. Patterns

Patterns provide a kind of abstraction that is complementary to the modularization mechanisms discussed above. They have been used, in particular, to define general architectural styles either by defining entire computation and communication topologies [83], connectors between (complex) software artifacts [80], or (based on, possibly concretizations of, design patterns [70]) as building blocks for object-oriented software architectures. The resulting pattern-based architectures are similar to common component-based architectures and are frequently used to implement the latter, see, for instance, Sun’s J2EE patterns.

Patterns have also been used to implement architectural abstractions. This is the case, for instance, for the numerous variants of the publish/subscribe pattern [67] as well as the large set of so-called skeletons [58], that is, patterns for the implementation of distributed and concurrent systems. While these patterns are essentially similar to architecture-level patterns, their fine-grained application to multiple code entities often results in crosscutting code structures. An important open issue consists in the lack of pattern-based representations for the implementation of general distributed applications — in sharp contrast to their use for the derivation of massively parallel programs.

3.6. Domain-Specific Languages

Domain-specific languages (DSLs) represent domain knowledge in terms of suitable basic language constructs and their compositions at the language level. By trading generality for abstraction, they enable complex relationships among domain concepts to be expressed concisely and their properties to be expressed and formally analyzed. DSLs have been applied to a large number of domains; they have been particularly popular in the domain of software generation and maintenance [81], [96].

Many modularization techniques and tasks can be naturally expressed by DSLs that are either specialized with respect to the type of modularization constructs, such as a specific brand of software component, or to the compositions that are admissible in the context of an application domain that is targeted by a modular implementation. Moreover, software development and evolution processes can frequently be expressed by transformations between applications implemented using different DSLs that represent an implementation at different abstraction levels or different parts of one application.

Functionalities that crosscut a component-based application, however, complicate such a DSL-based transformational software development process. Since such functionalities belong to another domain than that captured by the components, different DSLs should be composed. Such compositions (including their syntactic expression, semantics and property analysis) have only very partially been explored until now. Furthermore, restricted composition languages and many aspect languages that only match execution events of a specific domain (e.g., specific file accesses in the case of security functionality) and trigger only domain-specific actions clearly are quite similar to DSLs but remain to be explored.

3.7. Distribution and Concurrency

While ASCOLA does not investigate distribution and concurrency as research domains per se (but rather from a software engineering and modularization viewpoint), there are several specific problems and corresponding approaches in these domains that are directly related to its core interests that include the structuring and modularization of large-scale distributed infrastructures and applications. These problems include crosscutting functionalities of distributed and concurrent systems, support for the evolution of distributed software systems, and correctness guarantees for the resulting software systems.
Underlying our interest in these domains is the well-known observation that large-scale distributed applications are subject to numerous crosscutting functionalities (such as the transactional behavior in enterprise information systems, the implementation of security policies, and fault recovery strategies). These functionalities are typically partially encapsulated in distributed infrastructures and partially handled in an ad hoc manner by using infrastructure services at the application level. Support for a more principled approach to the development and evolution of distributed software systems in the presence of crosscutting functionalities has been investigated in the field of open adaptable middleware [55], [76]. Open middleware design exploits the concept of reflection to provide the desired level of configurability and openness. However, these approaches are subject to several fundamental problems. One important problem is their insufficient, framework-based support that only allows partial modularization of crosscutting functionalities.

There has been some criticism on the use of AspectJ-like aspect models (which middleware aspect models like that of JBoss AOP are an instance of) for the modularization of distribution and concurrency related concerns, in particular, for transaction concerns [75] and the modularization of the distribution concern itself [92]. Both criticisms are essentially grounded in AspectJ’s inability to explicitly represent sophisticated relationships between execution events in a distributed system: such aspects therefore cannot capture the semantic relationships that are essential for the corresponding concerns. History-based aspects, as those proposed by the ASCOLA project-team provide a starting point that is not subject to this problem.

From a point of view of language design and implementation, aspect languages, as well as domain specific languages for distributed and concurrent environments share many characteristics with existing distributed languages: for instance, event monitoring is fundamental for pointcut matching, different synchronization strategies and strategies for code mobility [69] may be used in actions triggered by pointcuts. However, these relationships have only been explored to a small degree. Similarly, the formal semantics and formal properties of aspect languages have not been studied yet for the distributed case and only rudimentarily for the concurrent one [53], [66].
3. Scientific Foundations

3.1. MDE Foundations

MDE can be seen as a generalization and abstraction of object technology allowing to map more abstract organizations on class-based implementations. In MDE, (software) models are considered as the unifying concept [48].

Traditionally, models were often used as initial design sketches mainly aimed for communicating ideas among developers. On the contrary, MDE promotes models as the primary artifacts that drive all software engineering activities. Therefore, rigorous techniques for model definition and manipulation are the basis of any MDE framework.

The MDE community distinguishes three levels of models: (terminal) model, metamodel, and metametamodel. A terminal model is a (partial) representation of a system/domain that captures some of its characteristics (different models can provide different knowledge views on the domain and be combined later on to provide a global view). In MDE we are interested in terminal models expressed in precise modeling languages. The abstract syntax of a language, when expressed itself as a model, is called a metamodel. A complete language definition is given by an abstract syntax (a metamodel), one or more concrete syntaxes (the graphical or textual syntaxes that designers use to express models in that language) plus one or more definition of its semantics. The relation between a model expressed in a language and the metamodel of that language is called conformsTo. Metamodels are in turn expressed in a modeling language called metamodeling language. Similar to the model/metamodel relationship, the abstract syntax of a metamodeling language is called a metametamodel and metamodels defined using a given metamodeling language must conform to its metametamodel. Terminal models, metamodels, and metametamodel form a three-level architecture with levels respectively named M1, M2, and M3. A formal definition of these concepts is provided in [56] and [49]. MDE promotes unification by models, like object technology proposed in the eighties unification by objects [47]. These MDE principles may be implemented in several standards. For example, OMG proposes a standard metametamodel called Meta Object Facility (MOF) while the most popular example of metamodel in the context of OMG standards is the UML metamodel.

In our view the main way to automate MDE is by providing model manipulation facilities in the form of model transformation operations that taking one or more models as input generate one or more models as output (where input and output models are not necessarily conforming to the same metamodel). More specifically, a model transformation $Mt$ defines the production of a model $Mb$ from a model $Ma$. When the source and target metamodels are identical ($MMa = MMMb$), we say that the transformation is endogenous. When this is not the case ($MMa \neq MMMb$) we say the transformation is exogenous. An example of an endogenous transformation is a UML refactoring that transforms public class attributes into private attributes while adding accessor methods for each transformed attribute. Many other operations may be considered as transformations as well. For example verifications or measurements on a model can be expressed as transformations [51]. One can see then why large libraries of reusable modeling artifacts (mainly metamodels and transformations) will be needed.

Another important idea is the fact that a model transformation is itself a model [2]. This means that the transformation program $Mt$ can be expressed as a model and as such conforms to a metamodel $MMt$. This allows an homogeneous treatment of all kinds of terminal models, including transformations. $Mt$ can be manipulated using the same existing MDE techniques already developed for other kinds of models. For instance, it is possible to apply a model transformation $Mt'$ to manipulate $Mt$ models. In that case, we say that $Mt'$ is a higher order transformation (HOT), i.e. a transformation taking other transformations (expressed as transformation models) as input or/and producing other transformations as output.
As MDE developed, it became apparent that this was a branch of language engineering [50]. In particular, MDE offers an improved way to develop DSLs (Domain-Specific Languages). DSLs are programming or modeling languages that are tailored to solve specific kinds of problems in contrast with General Purpose Languages (GPLs) that aim to handle any kind of problem. Java is an example of a programming GPL and UML an example of a modeling GPL. DSLs are already widely used for certain kinds of programming; probably the best-known example is SQL, a language specifically designed for the manipulation of relational data in databases. The main benefit of DSLs is that they allow everybody to write programs/models using the concepts that actually make sense to their domain or to the problem they are trying to solve (for instance Matlab has matrices and lets the user express operations on them, Excel has cells, relations between cells, and formulas and allows the expression of simple computations in a visual declarative style, etc.). As well as making domain code programmers more productive, DSLs also tend to offer greater optimization opportunities. Programs written with these DSLs may be independent of the specific hardware they will eventually run on. Similar benefits are obtained when using modeling DSLs. In MDE, new DSLs can be easily specified by using the metamodel concept to define their abstract syntax. Models specified with those DSLs can then be manipulated by means of model transformations (with ATL for example [7]).

When following the previously described principles, one may take advantage of the uniformity of the MDE organization. Considering similarly models of the static architecture and models of the dynamic behavior of a system allows at the same time economy of concepts and economy of implementation. Considering models of products (e.g., software artifacts like UML) and models of processes (e.g., software processes like SPEM) may lead to a dual process/product organization. Considering transformation models, weaving models, and traceability models as special cases of correspondence models may also lead to simplicity and efficiency of implementations. These are some of the use cases that are being explored in the team.
3. Scientific Foundations

3.1. Introduction

For many aspects of our everyday life, we rely heavily on information systems, many of which are based on massively networked devices that support a population of interacting and cooperating entities. While these information systems become increasingly open and complex, accidental and intentional failures get considerably more frequent and severe.

Two research communities traditionally address the concern of accidental and intentional failures: the distributed computing community and the security community. While both these communities are interested in the construction of systems that are correct and secure, an ideological gap and a lack of communication exist between them that is often explained by the incompatibility of the assumptions each of them traditionally makes. Furthermore, in terms of objectives, the distributed computing community has favored systems availability while the security community has focused on integrity and confidentiality, and more recently on privacy.

By contrast with this traditional conception, we are convinced that by looking at information systems as a combination of possibly revisited basic protocols, each one specified by a set of properties such as synchronization and agreement, security properties should emerge. This vision is shared by others and in particular by Myers et al [50], whose objectives are to explore new methods for constructing distributed systems that are trustworthy in the aggregate even when some nodes in the system have been compromised by malicious attackers. In accordance with this vision, the first main characteristic of the CIDRE group is to gather researchers from the two aforementioned communities in order to address in a complementary manner both the concerns of accidental and intentional failures.

The second main characteristic of the CIDRE group lies in the scope of the systems it considers. Indeed, during our research, we will consider three complementary levels of study: the Node Level, the Group Level, and the Open Network Level:

- **Node Level**: The term node either refers to a device that hosts a network client or service or to the process that runs this client or service. Node security management must be the focus of a particular attention, since from the user point of view, security of his own devices is crucial. Sensitive information and services must therefore be locally protected against various forms of attacks. This protection may take a dual form, namely prevention and detection.

- **Group Level**: Distributed applications often rely on the identification of sets of interacting entities. These subsets are either called groups, clusters, collections, neighborhoods, spheres, or communities according to the criteria that define the membership. Among others, the adopted criteria may reflect the fact that its members are administrated by a unique person, or that they share the same security policy. It can also be related to the localization of the physical entities, or the fact that they need to be strongly synchronized, or even that they share mutual interests. Due to the vast number of possible contexts and terminologies, we refer to a single type of set of entities, that we call set of nodes. We assume that a node can locally and independently identify a set of nodes and modify the composition of this set at any time. The node that manages one set has to know the identity of each of its members and should be able to communicate directly with them without relying on a third party. Despite these two restrictions, this definition remains general enough to include as particular cases most of the examples mentioned above. Of course, more restrictive behaviors can be specified by adding other constraints. We are convinced that security can benefit from the existence and the identification of sets of nodes of limited size as they can help in improving the efficiency of the detection and prevention mechanisms.
• Open Network Level: In the context of large-scale distributed and dynamic systems, interaction with unknown entities becomes an unavoidable habit despite the induced risk. For instance, consider a mobile user that connects his laptop to a public Wifi access point to interact with his company. At this point, data (regardless it is valuable or not) is updated and managed through non trusted undedicated entities (i.e., communication infrastructure and nodes) that provide multiple services to multiple parties during that user connection. In the same way, the same device (e.g., laptop, PDA, USB key) is often used for both professional and private activities, each activity accessing and manipulating decisive data.

The third characteristic of the CIDRE group is to focus on three different aspects of security, i.e., trust, intrusion detection, and privacy, and on the different bridges that exist between these aspects. Indeed, we believe that to study new security solutions for nodes, set of nodes and open network levels, one must take into account that it is now a necessity to interact with devices whose owners are unknown. To reduce the risk to rely on dishonest entities, a trust mechanism is an essential prevention tool that aims at measuring the capacity of a remote node to provide a service compliant with its specification. Such a mechanism should allow to overcome ill-founded suspicions and to be aware of established misbehaviors. To identify such misbehaviors, intrusion detection systems are necessary. Such systems aimed at detecting, by analyzing data flows, whether violations of the security policies have occurred. Finally, Privacy Protection which is now recognized as a basic user right, should be respected despite the presence of tools that continuously observe or even control users actions or behaviors.

3.2. Intrusion Detection

By exploiting vulnerabilities in operating systems, applications, or network services, an attacker can defeat the preventive security mechanisms and violate the security policy of the whole system. The goal of intrusion detection systems (IDS) is to be able to detect, by analyzing some data generated on a monitored system, violations of the security policy. From our point of view, while useful in practice, misuse detection is intrinsically limited. Indeed, it requires to update the signatures database in real-time similarly to what has to be done for antivirus tools. Given that there are thousands of machines that are every day victims of malware, such an approach may appear as insufficient especially due to the incredible expansion of malware, drastically limiting the capabilities of human intervention and response. The CIDRE group takes the alternative approach, i.e. the anomaly approach, which consists in detecting a deviation from a referenced behavior. Specifically, we propose to study two complementary methods:

• Illegal Flow Detection: This first method intends to detect information flows that violate the security policy [53], [46]. Our goal is here to detect information flows in the monitored system that are allowed by the access control mechanism, but are illegal from the security policy point of view.

• Data Corruption Detection: This second method aims at detecting intrusions that target specific applications, and make them execute illegal actions by using these applications incorrectly [45], [52]. This approach complements the previous one in the sense that the incorrect use of the application can possibly be legal from the point of view of the information flows and access control mechanisms, but is incorrect considering the security policy.

In both approaches, the access control mechanisms or the monitored applications can be either configured and executed on a single node, or distributed on a set of nodes. Thus, our approach must be studied at least at these first two levels. Moreover, we plan to work on intrusion detection system evaluation methods. For that research, we set a priori aside no particular IDS approach or technique. Here are some concrete examples of our research goals (both short term and long term objectives) in the intrusion detection field:

• at node level, we are going to apply the defensive programming approach (coming from the dependability field) to data corruption detection. The challenge is to determine which invariant/properties must be and can be verified either at runtime or statically. Regarding illegal flow detection, we plan to extend this method to build anti-viruses and DBMS tools by determining viruses signatures.
at the set of nodes level, we are going to revisit the distributed problems such as clock synchronization, logical clocks, consensus, properties detection, to extend the solutions proposed at node levels to cope with distributed flow control checking mechanisms. Regarding illegal flow detection, one of the challenges is to enforce the collaboration and consistency at nodes and set of nodes levels to obtain a global intrusion detection mechanism. Regarding the data corruption detection approach, the challenge is to identify local predicates/properties/invariants so that global predicates/properties/invariants would emerge at the system level.

3.3. Privacy
In our world of ubiquitous technologies, each individual constantly leaves digital traces related to his activities and interests which can be linked to his identity. In forthcoming years, the protection of privacy is one of the greatest challenge that lies ahead and also an important condition for the development of the Information Society. Moreover, due to legality and confidentiality issues, problematics linked to privacy emerge naturally for applications working on sensitive data, such as medical records of patients or proprietary datasets of enterprises. Privacy Enhancing Technologies (PETs) are generally designed to respect both the principles of data minimization and data sovereignty. The data minimization principle states that only the information necessary to complete a particular application should be disclosed (and no more). This principle is a direct application of the legitimacy criteria defined by the European data protection directive (Article 7). The data sovereignty principle states that data related to an individual belong to him and that he should stay in control of how this data is used and for which purpose. This principle can be seen as an extension of many national legislations on medical data that consider that a patient record belongs to the patient, and not to the doctors that create or update it, nor to the hospital that stores it. In the CIDRE project, we will investigate PETs that operate at the three different levels (node, set of nodes or open distributed system) and are generally based on a mix of different foundations such as cryptographic techniques, security policies and access control mechanisms just to name a few. Examples of domains where privacy and utility aspects collide and that will be studied within the context of CIDRE include: identity and privacy, geo-privacy, distributed computing and privacy, privacy-preserving data mining and privacy issues in social networks. Here are some concrete examples of our research goals in the privacy field:

- at the node level, we aim at designing privacy preserving identification scheme, automated reasoning on privacy policies [51], and policy-based adaptive PETs.
- at the set of nodes level, we plan to augment distributed algorithms (i.e., consensus) with privacy properties such as anonymity, unlinkability, and unobservability.
- at the open distributed system level, we plan to target both geo-privacy concerns (that typically occur in geolocalized systems) and privacy issues in social networks. In the former case, we will adopt a sanitization approach while in the latter one we plan to define privacy policies at user level, and their enforcement by all the intervening actors (e.g, at the social network sites providers).

3.4. Trust Management
While the distributed computing community relies on the trustworthiness of its algorithms to ensure systems availability, the security community historically makes the hypothesis of a Trusted Computing Base (TCB) that contains the security mechanisms (such as access controls, and cryptography) that implement the security policy. Unfortunately, as information systems get increasingly complex and open, the TCB management may itself get very complex, dynamic and error-prone. From our point of view, an appealing approach is to distribute and manage the TCB on each node and to leverage the trustworthiness of the distributed algorithms in order to strengthen each node’s TCB. Accordingly, the CIDRE group proposes to study automated trust management systems at all the three identified levels:

- at the node level, such a system should allow each node to evaluate by itself the trustworthiness of its neighborhood and to self-configure the security mechanisms it implements;
• at the group level, such a system might rely on existing trust relations with other nodes of the group to enhance the significance and the reliability of the gathered information;
• at the open network level, such a system should rely on reputation mechanisms to estimate the trustworthiness of the peers the node interacts with. The system might also benefit from the information provided by a priori trusted peers that, for instance, would belong to the same group (see previous item).

For the last two items, the automated trust management system will de facto follow the distributed computing approach. As such, emphasis will be put on the trustworthiness of the designed distributed algorithms. Thus, the proposed approach will provide both the adequate security mechanisms and a trustworthy distributed way of managing them. By way of examples of our research goals regarding the trust management field, we briefly list some of our short and long term objectives at node, group and open networks levels:

1. at node level, we are going to investigate how implicit trust relationships, identified and deduced by a node during its interactions with its neighborhood, could be explicitly used by the node (for instance by means of a series of rules) to locally evaluate the trustworthiness of its neighborhood. The impact of trust on the local security policy, and on its enforcement will be studied accordingly.

2. at the set of nodes level, we plan to take advantage of the pre-existing trust relationship among the set of nodes to design composition mechanisms that would guarantee that automatically configured security policies are consistent with each group member security policy.

3. at the open distributed system level, we are going to design reputation mechanisms to both defend the system against specific attacks (whitewashing, bad mouthing, ballot stuffing, isolation) by relying on the properties guaranteed at nodes and set of nodes levels, and guaranteeing persistent and safe feedback, and for specific cases in guaranteeing the right to oblivion (i.e., the right to data erasure).
3. Scientific Foundations

3.1. 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 formalize 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 \(\pi\)-calculus) and \(\lambda\)-calculus tradition. Such models, with their emphasis on algebra, well address compositionality—a central property in our approach to problems. Accordingly, the techniques we employ are mainly operational techniques based on notions of behavioral equivalence, and techniques based on algebra, mathematical logics, and type theory.

The sections below provide some more details on why process calculi, \(\lambda\)-calculi, and related techniques, should be useful for CBUS.

3.2. 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 \(\pi\)-calculus.

3.3. 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 \(\lambda\)-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 at hand here: they can express causality and order information on messages [58], [56], [59], while remaining decidable systems.

3.4. Implicit computational complexity

A number of elegant and powerful results have been recently obtained in implicit computational complexity in the \(\lambda\)-calculus in which 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 resources 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).
3. Scientific Foundations

3.1. Parallelism, concurrency, and distribution

Concurrency management is at the heart of diffuse programming. Since the execution platforms are highly heterogeneous, many different concurrency principles and models may be involved. Asynchronous concurrency is the basis of shared-memory process handling within multiprocessor or multicore computers, of direct or fifo-based message passing in distributed networks, and of fifo- or interrupt-based event handling in web-based human-machine interaction or sensor handling. Synchronous or quasi-synchronous concurrency is the basis of signal processing, of real-time control, and of safety-critical information acquisition and display. Interfacing existing devices based on these different concurrency principles within HOP or other diffuse programming languages will require better understanding of the underlying concurrency models and of the way they can nicely cooperate, a currently ill-resolved problem.

3.2. Web and functional programming

We are studying new paradigms for programming Web applications that rely on multi-tier functional programming [4]. We have created a Web programming environment named HOP. It relies on a single formalism for programming the server-side and the client-side of the applications as well as for configuring the execution engine.

HOP is a functional language based on the SCHEME programming language. That is, it is a strict functional language, fully polymorphic, supporting side effects, and dynamically type-checked. HOP is implemented as an extension of the BIGLOO compiler that we develop [5]. In the past, we have extensively studied static analyses (type systems and inference, abstract interpretations, as well as classical compiler optimizations) to improve the efficiency of compilation in both space and time.

3.3. Security of diffuse programs

The main goal of our security research is to provide scalable and rigorous language-based techniques that can be integrated into multi-tier compilers to enforce the security of diffuse programs. Research on language-based security has been carried on before in former Inria teams [2], [1]. In particular previous research has focused on controlling information flow to ensure confidentiality.

Typical language-based solutions to these problems are founded on static analysis, logics, provable cryptography, and compilers that generate correct code by construction [3]. Relying on the multi-tier programming language HOP that tames the complexity of writing and analysing secure diffuse applications, we are studying language-based solutions to prominent web security problems such as code injection and cross-site scripting, to name a few.
3. Scientific Foundations

3.1. Introduction

Research activity within the MYRIADS team encompasses several areas: distributed systems, middleware and programming models. We have chosen to provide a brief presentation of some of the scientific foundations associated with them: autonomic computing, future internet and SOA, distributed operating systems, and unconventional/nature-inspired programming.

3.2. Autonomic Computing

During the past years the development of raw computing power coupled with the proliferation of computer devices has grown at exponential rates. This phenomenal growth along with the advent of the Internet have led to a new age of accessibility - to other people, other applications and others systems. It is not just a matter of numbers. This boom has also led to unprecedented levels of complexity for the design and the implementation of these applications and systems, and of the way they work together. The increasing system scale is reaching a level beyond human ability to master its complexity.

This points towards an inevitable need to automate many of the functions associated with computing today. Indeed we want to interact with applications and systems intuitively, and we want to be far less involved in running them. Ideally, we would like computing systems to entirely manage themselves.

IBM [60] has named its vision for the future of computing "autonomic computing." According to IBM this new computer paradigm means the design and implementation of computer systems, software, storage and support that must exhibit the following basic fundamentals:

- Flexibility. An autonomic computing system must configure and reconfigure itself under varying, even unpredictable, conditions.
- Accessibility. The nature of the autonomic system is that it is always on.
- Transparency. The system will perform its tasks and adapt to a user’s needs without dragging the user into the intricacies of its workings.

In the Myriads team we will act to satisfy these fundamentals.

3.3. Future Internet and SOA

Traditional information systems were built by integrating applications into a communication framework, such as CORBA or with an Enterprise Application Integration system (EAI). Today, companies need to be able to reconfigure themselves; they need to be able to include other companies’ business, split or externalize some of their works very quickly. In order to do this, the information systems should react and adapt very efficiently. EAI's approaches did not provide the necessary agility because they were too tightly coupled and a large part of business processes were "hard wired" into company applications.

Web services and Service Oriented Architectures (SOA) partly provide agility because in SOA business processes are completely separated from applications which can only be viewed as providing services through an interface. With SOA technologies it is easily possible to modify business processes, change, add or remove services.

However, SOA and Web services technologies are mainly market-driven and sometimes far from the state-of-the-art of distributed systems. Achieving dependability or being able to guarantee Service Level Agreement (SLA) needs much more agility of software elements. Dynamic adaptability features are necessary at many different levels (business processes, service composition, service discovery and execution) and should be coordinated. When addressing very large scale systems, autonomic behaviour of services and other parts of service oriented architectures is necessary.
SOAs will be part of the "Future Internet". The "Future Internet" will encompass traditional Web servers and browsers to support companies and people interactions (Internet of services), media interactions, search systems, etc. It will include many appliances (Internet of things). The key research domains in this area are network research, cloud computing, Internet of services and advanced software engineering.

The Myriads team will address adaptability and autonomy of SOAs in the context of Grids, Clouds and at large scale.

### 3.4. Distributed Operating Systems

An operating system provides abstractions such as files, processes, sockets to applications so that programmers can design their applications independently of the computer hardware. At execution time, the operating system is in charge of finding and managing the hardware resources necessary to implement these abstractions in a secure way. It also manages hardware and abstract resource sharing between different users and programs.

A distributed operating system makes a network of computer appear as a single machine. The structure of the network and the heterogeneity of the computation nodes are hidden to users. Members of the Myriads team members have a long experience in the design and implementation of distributed operating systems, for instance in Kerrighed, Vigne and XtreemOS projects.

Clouds can be defined as platforms for on-demand resource provisioning over the Internet. These platforms rely on networked computers. Three flavours of cloud platforms have emerged corresponding to different kinds of service delivery:

- **IaaS** (Infrastructure as a Service) refers to clouds for on-demand provisioning of elastic and customizable execution platforms (from physical to virtualized hardware).
- **PaaS** (Platform as a Service) refers to clouds providing an integrated environment to develop, build, deploy, host and maintain scalable and adaptable applications.
- **SaaS** (Software as a Service) refers to clouds providing customers access to ready-to-use applications.

The cloud computing model [49], [55] introduces new challenges in the organization of the information infrastructure: security, identity management, adaptation to the environment (costs). The organization of large organization IT infrastructures is also impacted as their internal data-centers, sometimes called private clouds, need to cooperate with resources and services provisioned from the cloud in order to cope with workload variations. The advent of cloud and green computing introduces new challenges in the domain of distributed operating systems: resources can be provisioned and released dynamically, the distribution of the computations on the resources must be reevaluated periodically in order to reduce power consumption and resource usage costs. Distributed cloud operating system must adapt to these new challenges in order to reduce cost and energy, for instance, through the redistribution of the applications and services on a smaller set of resources.

The Myriads team will work on the design and implementation of system services to autonomously manage cloud and cloud federations resources and support collaboration between cloud users.

### 3.5. Unconventional/Nature-inspired Programming

Facing the complexity of the emerging ICT landscape in which highly heterogeneous digital services evolve and interact in numerous different ways in an autonomous fashion, there is a strong need for rethinking programming models. The question is "what programming paradigm can efficiently and naturally express this great number of interactions arising concurrently on the platform?"

It has been suggested [47] that observing nature could be of great interest to tackle the problem of modeling and programming complex computing platforms, and overcome the limits of traditional programming models. Innovating unconventional programming paradigms are requested to provide a high-level view of these interactions, then allowing to clearly separate what is a matter of expression from what is a question of implementation. Towards this, nature is of high inspiration, providing examples of self-organising, fully decentralized coordination of complex and large scale systems.
As an example, chemical computing [50] has been proposed more than twenty years ago for a natural way to program parallelism. Even after significant spread of this approach, it appears today that chemical computing exposes a lot of good properties (implicit autonomy, decentralization, and parallelism) to be leveraged for programming service infrastructures.

The Myriads team will investigate nature-inspired programming such as chemical computing for autonomous service computing.
3. Scientific Foundations

3.1. Programming with distributed objects and components

The paradigm of object-oriented programming, although not very recent, is clearly still not properly defined and implemented; for example notions like inheritance, sub-typing or overloading have as many definitions as there are different object languages. The introduction of concurrency and distribution into objects also increases the complexity. It appeared that standard Java constituents such as RMI (Remote Method Invocation) do not help building, in a transparent way, sequential, multi-threaded, or distributed applications. Indeed allowing, as RMI does, the execution of the same application to proceed on a shared-memory multiprocessors architecture as well as on a network of workstations (intranet, Internet), or on any hierarchical combination of both, is not sufficient for providing a convenient and reliable programming environment.

The question is thus: how to ease the construction (i.e. programming), deployment and evolution of distributed applications?

One of the answers we suggest relies on the concept of active object, that act as a single entity, abstraction of a thread, a set of objects and a location. Active objects communicate by asynchronous method calls thanks to the use of futures. ProActive is a Java library that implements this notion of active objects. ProActive can also be seen as a middleware supporting deployment, runtime support, and efficient communication for large scale distributed applications.

Another answer we provide relies on component-oriented programming. In particular, we have defined parallel and hierarchical distributed components starting from the Fractal component model developed by INRIA and France-Telecom [43]. We have been involved in the design of the Grid Component Model (GCM) [4][20], which is one of the major results produced by the CoreGrid European Network of Excellence. The GCM is now officially intended to become a standard for Grid components, and most of our research on component models are related to it. GCM is now fully a standard for Grid components (48 for the last published standard), and most of our research on component models are related to it. On the practical side, ProActive/GCM is a prototype implementation of the GCM above the ProActive library. ProActive/GCM is intended to become the reference implementation of the GCM, that was the goal of the European project GridCOMP.

We have developed overtime competencies in both theoretical and applicative side fields, such as distribution, fault-tolerance, verification, etc., to provide a better programming and runtime environment for object and component applications.

3.2. Formal models for distributed objects

A few years ago, we designed the ASP calculus [7] for modelling distributed objects. It remains to this date one of our major scientific foundations. ASP is a calculus for distributed objects interacting using asynchronous method calls with generalized futures. Those futures naturally come with a transparent and automatic synchronisation called wait-by-necessity. In large-scale systems, our approach provides both a good structure and a strong decoupling between threads, and thus scalability. Our work on ASP provides very generic results on expressiveness and determinism, and the potential of this approach has been further demonstrated by its capacity to cope with advanced issues, such as mobility, group communications, and components [6].

ASP provides confluence and determinism properties for distributed objects. Such results should allow one to program parallel and distributed applications that behave in a deterministic manner, even if they are distributed over local or wide area networks.
The ASP calculus is a model for the ProActive library. An extension of ASP has been built to model distributed asynchronous components. A functional fragment of ASP has been modelled in the Isabelle theorem prover [8].

3.3. Verification, static analysis, and model-checking

Even with the help of high-level libraries, distributed systems are more difficult to program than classical applications. The complexity of interactions and synchronisations between remote parts of a system increases the difficulty of analysing their behaviours. Consequently, safety, security, or liveness properties are particularly difficult to ensure for these applications. Formal verification of software systems has been active for a long time, but its impact on the development methodology and tools has been slower than in the domain of hardware and circuits. This is true both at a theoretical and at a practical level, from the definition of adequate models representing programs, the mastering of state complexity through abstraction techniques or through new algorithmic approaches, to the design of software tools that hide to the final user the complexity of the underlying theory.

We concentrate on the area of distributed component systems, where we get better descriptions of the structure of the system, making the analysis more tractable, but we also find out new interesting problems. For instance, we contributed to a better analysis of the interplay between the functional definition of a component and its possible runtime transformations, expressed by the various management controllers of the component system. Our approach is bi-directional: from models to program, or back. We use techniques of static analysis and abstract interpretation to extract models from the code of distributed applications, or from dedicated specification formalisms [3]. On the other hand, we generate “safe by construction” code skeletons, from high level specifications; this guarantees the behavioural properties of the components. We then use generic tools from the verification community to check properties of these models. We concentrate on behavioural properties, expressed in terms of temporal logics (safety, liveness), of adequacy of an implementation to its specification and of correct composition of software components.
3. Scientific Foundations

3.1. Introduction

Our proposed project builds upon results previously obtained by the Compose research group whose aim was to study new approaches to developing adaptable software components in the domain of systems and networking. In this section, we review the accomplishments of Compose, only considering the ones achieved by the current project members, to demonstrate our expertise in the key areas underlying our project, namely:

- Programming language technology: language design and implementation, domain-specific languages, program analysis and program transformation.
- Operating Systems and Networking: design, implementation and optimization.
- Software engineering: software architecture, methodologies, techniques and tools.

By combining expertise in these areas, the research work of the Compose group contributed to demonstrating the usefulness of adaptation methodologies, such as domain-specific languages, and the effectiveness of adaptation tools, such as program specializers. Our work aimed to show how adaptation methodologies and tools could be integrated into the development process of real-size software components. This contribution relied on advances in methodologies to develop adaptable programs, and techniques and tools to adapt these programs to specific usage contexts.

3.2. Adaptation Methodologies

Although industry has long recognized the need to develop adaptable programs, methodologies to develop them are still at the research stage. We have presented preliminary results in this area with a detailed study of the applicability of program specialization to various software architectures [31]. Our latest contributions in this area span from a revolutionary approach based on the definition of programming languages, dedicated to a specific problem family, to a direct exploitation of specialization opportunities generated by a conventional programming methodology.

3.2.1. Domain-Specific languages

DSLs represent a promising approach to modeling a problem family. Yet, this approach currently suffers from the lack of methodology to design and implement DSLs. To address this basic need, we have introduced the Sprint methodology for DSL development [23]. This methodology bridges the gap between semantics-based approaches to developing general-purpose languages and software engineering. Sprint is a complete software development process starting from the identification of the need for a DSL to its efficient implementation. It uses the denotational framework to formalize the basic components of a DSL. The semantic definition is structured so as to stage design decisions and to smoothly integrate implementation concerns.

3.2.2. Declaring adaptation

A less drastic strategy to developing efficient adaptable programs consists of making specific issues of adaptation explicit via a declarative approach. To do so, we enrich Java classes with declarations, named adaptation classes, aimed to express adaptive behaviors [20]. As such, this approach allows the programmer to separate the concerns between the basic features of the application and its adaptation aspects. A dedicated compiler automatically generates Java code that implements the adaptive features.
3.2.3. Declaring specialization

When developing components, programmers often hesitate to make them highly generic and configurable. Indeed, genericity and configurability systematically introduce overheads in the resulting component. However, the causes of these overheads are usually well-known by the programmers and their removal could often be automated, if only they could be declared to guide an optimizing tool. The Compose group has worked towards solving this problem.

We introduced a declaration language which enables a component developer to express the configurability of a component. The declarations consist of a collection of specialization scenarios that precisely identify what program constructs are of interest for specialization. The scenarios of a component do not clutter the component code; they are defined aside in a specialization module [26], [27], [25], [28].

This work was done in the context of C and declarations were intended to drive our C specializer.

3.2.4. Specializing design patterns

A natural approach to systematically applying program specialization is to exploit opportunities offered by a programming methodology. We have studied a development methodology for object-oriented languages, called design patterns. Design patterns encapsulate knowledge about the design and implementation of highly adaptable software. However, adaptability is obtained at the expense of overheads introduced in the finished program. These overheads can be identified for each design pattern. Our work consisted in using knowledge derived from design patterns to eliminate these overheads in a systematic way. To do so, we analyzed the specialization opportunities provided by specific uses of design patterns, and determined how to eliminate these overheads using program specialization. These opportunities were documented in declarations, called specialization patterns, and were associated with specific design patterns [39]. The specialization of a program composed of design patterns was then driven by the corresponding declarations. This work was presented in the context of Java and uses our Java specializer [38].

3.2.5. Specializing software architectures

The sources of inefficiency in software architectures can be identified in the data and control integration of components, because flexibility is present not only at the design level but also in the implementation. We proposed the use of program specialization in software engineering as a systematic way to improve performance and, in some cases, to reduce program size. We studied several representative, flexible mechanisms found in software architectures: selective broadcast, pattern matching, interpreters, layers and generic libraries. We showed how program specialization can be applied systematically to optimize these mechanisms [30], [31].

3.3. Adaptation in Systems Software

3.3.1. DSLs in Operating Systems

Integrating our adaptation methodologies and tools into the development process of real-size software systems was achieved by proposing a new development process. Specifically, we proposed a new approach to designing and structuring operating systems (OSes) [34]. This approach was based on DSLs and enables rapid development of robust OSes. Such an approach is critically needed in application domains, like appliances, where new products appear at a rapid pace and needs are unpredictable.

3.3.2. Devil - a DSL for device drivers

Our approach to developing systems software applied to the domain of device drivers. Indeed, peripheral devices come out at a frantic pace, and the development of drivers is very intricate and error prone. The Compose group developed a DSL, named Devil (DEvice Interface Language), to solve these problems; it was dedicated to the basic communication with a device. Devil allowed the programmer to easily map device documentation into a formal device description that can be verified and compiled into executable code.
From a software engineering viewpoint, Devil captures domain expertise and systematizes re-use because it offers suitable built-in abstractions [36]. A Devil description formally specifies the access mechanisms, the data type and layout of data, as well as behavioral properties involved in operating the device. Once compiled, a Devil description implements an interface to an idealized device and abstracts the hardware intricacies.

From an operating systems viewpoint, Devil can be seen as an interface definition language for hardware functionalities. To validate the approach, Devil was put to practice [35]: its expressiveness was demonstrated by the wide variety of devices that have been specified in Devil. No loss in performance was found for the compiled Devil description compared to an equivalent C code.

From a dependable system viewpoint, Devil improves safety by enabling descriptions to be statically checked for consistency and generating stubs including additional run-time checks [37]. Mutation analysis was used to evaluate the improvement in driver robustness offered by Devil. Based on our experiments, Devil specifications were found up to 6 times less prone to errors than writing C code.

Devil was the continuation of a study of graphic display adaptors for a X11 server. We developed a DSL, called GAL (Graphics Adaptor Language), aimed to specify device drivers in this context [42]. Although covering a very restricted domain, this language was a very successful proof of concept.

### 3.4. Adaptation Tools and Techniques

To further the applicability of our approach, we have strengthened and extended adaptation tools and techniques. We have produced a detailed description of the key program analysis for imperative specialization, namely binding-time analysis [22]. This analysis is at the heart of our program specializer for C, named Tempo [22]. We have examined the importance of the accuracy of these analyses to successfully specialize existing programs. This study was conducted in the context of systems software [32].

Tempo is the only specializer which enables programs to be specialized both at compile time and run time. Yet, specialization is always performed in one stage. As a consequence, this process cannot be factorized even if specialization values become available at multiple stages. We present a realistic and flexible approach to achieving efficient incremental run-time specialization [29]. Rather than developing new techniques, our strategy for incremental run-time specialization reuses existing technology by iterating a specialization process. Our approach has been implemented in Tempo.

While program specialization encodes the result of early computations into a new program, data specialization encodes the result of early computations into data structures. Although aiming at the same goal, namely processing early computations, these two forms of specialization have always been studied separately. The Compose group has proposed an extension of Tempo to perform both program and data specialization [21]. We showed how these two strategies can be integrated in a single specializer. Most notably, having both strategies enabled us to assess their benefits, limitations and their combination on a variety of programs.

Interpreters and run-time compilers are increasingly used to cope with heterogeneous architectures, evolving programming languages, and dynamically loaded code. Although solving the same problem, these two strategies are very different. Interpreters are simple to implement but yield poor performance. Run-time compilation yields better performance, but is costly to implement. One approach to reconciling these two strategies is to develop interpreters for simplicity but to use specialization to achieve efficiency. Additionally, a specializer like Tempo can remove the interpretation overhead at compile time as well as at run time. We have conducted experiments to assess the benefits of applying specialization to interpreters [41]. These experiments have involved Bytecode and structured-language interpreters. Our experimental data showed that specialization of structured-language interpreters can yield performance comparable to that of the compiled code of an optimizing compiler.

Besides targeting C, we developed the first program specializer for an object-oriented language. This specializer, named JSpec, processes Java programs [38]. JSpec is constructed from existing tools. Java programs are translated into C using our Java compiler, named Harissa. Then, the resulting C programs are specialized using Tempo. The specialized C program is executed in the Harissa environment. JSpec has been used for various applications and has shown to produce significant speedups [40].
POPS Project-Team

3. Scientific Foundations

3.1. Embedded Operating Systems

We focus our activities on “adaptability” and on “connectivity” of embedded platforms dedicated to POPS. From then on, our researches have evolved around the smart card. In fact, in the nineties (birth date of POPS research group) smart card was the only valuable and industrially deployed POPS. Smart card integration in database management systems, smart card integration in Corba (using the Card Object Adapter), open platform for smart card (the first smart card virtual machine), have been milestones of the POPS research. More recently, we have focused our attention (according to our industrial inputs) on embedded operating system techniques, enabling “on-card” type checking and bytecode compression. Today, smart card manufacturers and other emerging POPS manufacturers have to deal with new technological ‘lock-in’ inside and outside the mobile object. Dedicated operating systems are now powerful enough to run dynamically downloaded applications in a safe way. Typically, Java Card loads and runs a Java-like bytecode. Nevertheless, “Java-like” means “non-Java”. Embedded virtual machines do not support standard abstractions. And so, Java applications cannot be deployed in a limited embedded system. On the other hand, embedded applications do not limit their needs to the Java APIs. To overcome these limitations, we will focus on two complementary studies:

1. Firstly we study a new architectural way to embed a Java virtual machine. Conventional virtual machines are not operating systems but they overlap the abstractions proposed by the system. We plan to define a Java virtual machine designed to be the operating system (the virtual machine will manage the hardware itself).

2. Java is one of the possible hardware abstractions. However different applications require different abstractions: file-system, database systems, and so on. Camille OS is a smart card Exo-kernel enabling the download of different hardware abstractions in a safe way. In this way Camille ensures POPS “adaptability” to the applications requirements. Nevertheless some critical system extensions (enhanced IO protocols for example) need additional guaranties: real-time properties and hardware resources control.

3.2. Mobile Networking

POPS also have a non-conventional communication interface. Due to their mobility, they have transient and unpredictable communications with other entities. This fact motivates our focusing on the ad hoc network communication model which is the most flexible model.

Indeed wireless ad hoc networks [51], [46], [47], [44] encompass a wide range of self-organized network types, including sensor, mobile ad hoc, personal area, and rooftop/mesh networks. The design of data communication techniques in multi-hop ad hoc networks comprises challenges at all layers of communication: physical, medium access control (MAC), network, transport and application layers. This research project concentrates on the network layer. The network layer problems can be divided into three groups: data communication, service access, and topology control problems. Data communication problems include routing, quality-of-service routing, geocasting, multicasting, and broadcasting. The protocols need to minimize the communication overhead (since bandwidth in wireless communication is typically limited) and the power consumption of battery operated POPS. In service access problems, such as multi-hop wireless Internet (hybrid network, see Fig. 4), the goal is to provide or receive services from a fixed infrastructure with other hosts serving as relays if necessary. Topology control problems include neighbor discovery problems (detecting neighboring nodes located within transmission radius) and network organization problems (deciding what communication links to establish with neighboring nodes, operating sleeping period and adjusting transmission radii). Secure routing faces the following challenges: node selfishness, threats using modification
of routing information, misrepresenting identity, fabrication of routing messages by one node, or between two malicious nodes (wormhole attack), and self-organized public-key management and authentication services. The main paradigm shift is to apply localized (or greedy) schemes as opposed to existing protocols requiring global information. Localized algorithms are distributed algorithms where simple local node behavior achieves a desired global objective. Localized protocols provide scalable solutions, that is, solutions for wireless networks with an arbitrary number of nodes, which is one of the main goals of this research project.

Figure 4. From wireless network to hybrid networks.
3. Scientific Foundations

3.1. Foundation 1

Scaling to large configurations is one of the major challenges addressed by the distributed system community lately. The basic idea is how to efficiently and transparently use and manage resources of millions of hosts spread over a large network. The problem is complex compared to classical distributed systems where the number of hosts is low (less than a thousand) and the inter-host links are fast and relatively reliable. In such "classical" distributed architectures, it is possible and reasonable to build a single image of the system so as to "easily" control resource allocation.

In large configurations, there is no possibility to establish a global view of the system. The underlying operating system has to make decisions (on resource allocation, scheduling ...) based only on partial and possibly wrongs view of the resources usage.

Scaling introduces the following problems:

- **Failure**: as the number of hosts increases, the probability of a host failure converges to one.\(^1\) Compared to classical distributed systems, failures are more common and have to be efficiently processed.
- **Asynchronous networks**: on the Internet, message delays vary considerably and are unbounded.
- **Impossibility of consensus**: In such an asynchronous network with failures, consensus cannot be solved deterministically (the famous Fischer-Lynch-Patterson impossibility result of 1985). The system can only approximate, suspecting hosts that are not failed, or failing to suspect hosts that have failed. As a result, no host can form a consistent view of system state.
- **Failure models**: the classical view of distributed systems considers only crash and omission failures. In the context of large-scale, open networks, the failure model must be generalised to include stronger attacks. For instance, a host can be taken over ("zombie") and become malicious. Arbitrary faults, so-called Byzantine behaviours, are to be expected and must be tolerated.
- **Managing distributed state**: In contrast to a local-area network, establishing a global view of a large distributed system is unfeasible. The operating system must make its decisions, regarding resource allocation or scheduling, based on partial and incomplete views of system state.

Three architectures in relation with the scaling problem have emerged during the last years:

- **Grid computing**: Grid computing offers a model for solving massive computational problems using large numbers of computers arranged as clusters interconnected by a telecommunications infrastructure as internet or renater.
  
  If the number of involved hosts can be high (several thousands), the global environment is relatively controlled and users of such systems are usually considered safe and only submitted to host crash failures (typically, Byzantine failures are not considered).

- **Peer-to-peer overlay network**: Generally, a peer-to-peer (or P2P) computer network is any network that does not rely on dedicated servers for communication but, instead, mostly uses direct connections between clients (peers). A pure peer-to-peer network does not have the notion of clients or servers, but only equal peer nodes that simultaneously function as both “clients” and “servers” with respect to the other nodes on the network.

\(^1\) For instance if we consider a classical host MTBF (Mean Time Between Failure) equals to 13 days, in a middle scale system composed of only 10000 hosts, a failure will occur every 4 minutes.
This model of network arrangement differs from the client-server model where communication is usually relayed by the server. In a peer-to-peer network, any node is able to initiate or complete any supported transaction with any other node. Peer nodes may differ in local configuration, processing speed, network bandwidth, and storage capacity.

Different peer-to-peer networks have varying P2P overlays. In such systems, no assumption can be made on the behavior of the host and Byzantine behavior has to be considered.

Cloud computing Cloud computing offers a conceptually infinite amount of computing, storage and network resources for rent. From a user’s perspective, cloud computing has many advantages, including low upfront investment and outsourcing of system administration. From the provider’s perspective, cloud computing shares many characteristics with both grid computing (e.g., very large geographical and numeric scale, or service-oriented interfaces) and with P2P computing (e.g., self-administration). It also has some unique characteristics, such as systematic virtualisation of all resources, highly variable load, fast elastic adaptation, and quality-of-service objectives that are negotiated with clients via SLAs.

Regal is interested in how to adapt distributed middleware to these large scale configurations. We target Grid and Peer-to-peer configurations. This objective is ambitious and covers a large spectrum. To reduce its spectrum, Regal focuses on fault tolerance, replication management, and dynamic adaptation.

We concentrate on the following research themes:

Data management: the goal is to be able to deploy and locate effectively data while maintaining the required level of consistency between data replicas.

System monitoring and failure detection: we envisage a service providing the follow-up of distributed information. Here, the first difficulty is the management of a potentially enormous flow of information which leads to the design of dynamic filtering techniques. The second difficulty is the asynchronous aspect of the underlying network which introduces a strong uncertainty on the collected information.

Adaptive replication: we design parameterizable techniques of replication aiming to tolerate the faults and to reduce information access times. We focus on the runtime adaptation of the replication scheme by (1) automatically adjusting the internal parameters of the strategies and (2) by choosing the replication protocol more adapted to the current context.

The dynamic adaptation of application execution support: the adaptation is declined here to the level of the execution support (in either of the high level strategies). We thus study the problem of dynamic configuration at runtime of the low support layers.
3. Scientific Foundations

3.1. Software Reengineering

Strong coupling among the parts of an application severely hampers its evolution. Therefore, it is crucial to answer the following questions: How to support the substitution of certain parts while limiting the impact on others? How to identify reusable parts? How to modularize an object-oriented application?

Having good classes does not imply a good application layering, absence of cycles between packages and reuse of well-identified parts. Which notion of cohesion makes sense in presence of late-binding and frameworks? Indeed, frameworks define a context that can be extended by subclassing or composition: in this case, packages can have a low cohesion without being a problem for evolution. How to obtain algorithms that can be used on real cases? Which criteria should we select for a given remodularization?

We plan to enrich Moose, our reengineering environment, with a new set of analyses [51], [49]. We decompose our approach in three main and potentially overlapping steps:

1. Tools for understanding applications at large: packages/modules,
2. Remodularization analyses, and
3. Software Quality and Open DashBoard.

3.1.1. Tools for understanding applications at large: packages/modules

Context and Problems. As we are going to design and evaluate several algorithms and analyses to remodularize applications, we need a way to understand and assess the results we will obtain. Our experience on real application analyses taught us that analyses tend to produce a huge amount of data that we should understand and correlate to the original source code situation [50]. The problem is that understanding large systems is already difficult [102], [74], [76], [69], but in our case we need to understand an existing system and the results of our analysis. Parallelism between software programs and cities is commonly used to reason about evolution [69], [103]. While interesting, this metaphor does not scale because location of houses does not have any semantics information related to the connection between classes. A notion of radar has also been proposed [46], but this mechanism suffers from the same problem.

Therefore, there is a definitive need to have ways to support the understanding of large applications at the level of their structure.

Research Agenda. We are going to study the problems raised by the understanding of applications at the larger level of granularity such as packages/modules. We will develop a set of conceptual tools to support this understanding. These tools will certainly be visual such as the Distribution Map Software visualization [50] or based on the definition of new metrics taking into account the complexity of packages. Such a step is really crucial as a support for the remodularization analyses that we want to perform. The following tasks are currently ongoing:

MoQam. The Qualixo model has been originally implemented on top of the Java platform. An implementation of this model, named MoQam (Moose Quality Assessment Model), is under development in the Moose open-source and free reengineering environment. A first experiment has been conducted [70]. Exporters from Moose to the Squale software are under development.

Cohesion Metric Assessment. We are assessing the metrics and practices used originally in the Qualixo model. We are also compiling a number of metrics for cohesion and coupling assessment. We want to assess for each of these metrics their relevance in a software quality setting.
DSM. Dependency Structure Matrix (DSM), an approach developed in the context of process optimization, has been successfully applied to identify software dependencies among packages and subsystems. A number of algorithms helps organizing the matrix in a form that reflects the architecture and highlights patterns and problematic dependencies between subsystems. However, the existing DSM implementations often miss important information in their visualization to fully support a reengineering effort. We plan to enrich them to improve their usefulness to assess system general structure.

3.1.2. Remodularization analyses

Context and Problems. It is a well-known practice to layer applications with bottom layers being more stable than top layers [80]. Until now, few works have attempted to identify layers in practice: Mudpie [101] is a first cut at identifying cycles between packages as well as package groups potentially representing layers. DSM (dependency structure matrix) [100], [95] seems to be adapted for such a task but there is no serious empirical experience that validates this claim. From the side of remodularization algorithms, many were defined for procedural languages [68]. However, object-oriented programming languages bring some specific problems linked with late-binding and the fact that a package does not have to be systematically cohesive since it can be an extension of another one [104], [54].

Some approaches based on Formal Concept Analysis [99] show that such an analysis can be used to identify modules. However the presented example is small and not representative of real code. Other clustering algorithms [64], [65] have been proposed to identify modules [75], [86]. Once again, the specific characteristics of object-oriented programming are not taken into account. This is a challenge since object-oriented programming tends to scatter classes definitions over multiple packages and inheritance hierarchies. In addition, the existing algorithms or analyses often only work on toy applications. In the context of real applications, other constraints exist such as the least perturbation of the code, minimizing the hierarchy changes, paying attention to code ownership, layers, or library minimization. The approach will have to take into account these aspects.

Many different software metrics exist in the literature [73], [56], [61] such as the McCabe complexity metrics [81]. In the more specific case of object-oriented programming, assessing cohesion and coupling have been the focus of several metrics. However their success is rather questionable. For example, LCOM [43] has been highly criticized [55], [63], [62], [30], [39], [40]. Other approaches have been proposed such as RFC and CBO [43] to assess coupling between classes. However, many other metrics have not been the subject of careful analysis such as Data Abstraction Coupling (DAC) and Message Passing Coupling (MPC) [32], or some of metrics are not clearly specified (MCX, CCO, CCP, CRE) [73]. New cohesion measures were proposed [77], [89] taking into account class usage.

Research Agenda. We will work on the following items:

Characterization of “good” modularization. Any remodularization effort must use a quality function that allows the programmer to compare two possible decompositions of the system and choose which one represents a more desirable modularization. Remodularization consists in trying to maximize such a function. The typical function used by most researchers is some measure of cohesion/coupling. However, manual system modularization may rely on many different considerations: implemented functionalities, historical considerations, clients or markets served, ...We want to evaluate various modularization quality functions against existing modularizations to identify their respective strengths and weaknesses.

Cohesion and coupling metric evaluation and definition. Chidamber’s well-known cohesion metric named LCOM has been strongly criticized [55], [63], [62], [40]. However, the solutions rarely take into account that a class is an incremental definition and, as such, can exist in a several packages at once. For example, LCOM* flattens inheritance to determine the cohesion of a class. In addition, these metrics are not adapted to packages. We will thus work on the assessment of existing cohesion metrics for classes, defining new ones if necessary for packages, and assess coupling metrics as well [30], [39]. This work is also related to the notion of software quality treated below.
Build an empirical validation of DSM and enhancements. We want to assess Dependency Structure Matrix (DSM) to support remodularization. DSM is good to identify cyclic dependencies. Now we want to know if we can identify misplaced classes among groups of packages working as layers. For this purpose we will perform controlled experiments and in a second period apply DSM on one of the selected case studies. Based on these results, we will propose enhancements to the approach.

Layer identification. We want to propose an approach to identify layers based on a semi-automatic classification of package and class interrelationships that they contain. However, taking into account the wish or knowledge of the designer or maintainer should be supported. We will try to apply different algorithms and adapt them to the specific context of object-oriented programming [68].

3.1.3. Software Quality

Companies often look for the assessment of their software quality. Several models of software quality have been proposed: J.A. McCall [82] with his Factor-Criteria-Metrics has identified more than 50 candidate factors that may be used to assess software quality. Among those factors, only 11 were retained. Each of those has been characterized by 23 criteria that represent the internal project quality view. This approach is not easily used because of the high number of metrics —more than 300, some of which are not automatically computed. In an effort of conformance, the ISO (International Standardization Organization) and the IEC (International Electromenon Commission) jointly defined the ISO 9126 norm in 1999. This norm, currently being restructured, will be composed of 4 parts: quality model (ISO 9126-1), external metrology (ISO 9126-2), Internal metrology (ISO 9126-3), Usage quality of metrology (ISO 9126-4). There is also a body of work focusing on design evaluation as quality criteria [83], [91], [87] and new quality models: QMOOD is for example a hierarchical quality model which proposes to link directly quality criteria to software metrics based on object-oriented software metrics [31] while other works focus on linking different high level criteria with software code metrics [78], [79].

Research Agenda. Since software quality is fuzzy by definition and a lot of parameters should be taken into account we consider that defining precisely a unique notion of software quality is definitively a Graal in the realm of software engineering. The question is still relevant and important. We plan to work on the two following items in the context of the Squale project in contact with the Qualixo company:

Quality Model. We want to study the existing quality models and develop in particular models that take into account (1) the possible overlaps in the source of information —it is important to know whether a model measures the same aspects several times, using different metrics (2) the combination of indicators —often, software quality models happily combine metrics, but at the price of losing the explicit relationships between the indicator contributions.

3.2. Language Constructs for Modular Design

While the previous axis focuses on how to help remodularizing existing software, this second research axis aims at providing new language constructs to build more flexible and recomposable software. We will build on our work on traits [97], [52] and classboxes [33] but also start to work on new areas such as security in dynamic languages. We will work on the following points: (1) Traits: behavioral units and (2) Modularization as a support for security.

3.2.1. Traits-based program reuse

Context and Problems. Inheritance is well-known and accepted as a mechanism for reuse in object-oriented languages. Unfortunately, due to the coarse granularity of inheritance, it may be difficult to decompose an application into an optimal class hierarchy that maximizes software reuse. Existing schemes based on single inheritance, multiple inheritance, or mixins, all pose numerous problems for reuse.
To overcome these problems, we designed a new composition mechanism called Traits [97], [52]. Traits are pure units of behavior that can be composed to form classes or other traits. The trait composition mechanism is an alternative to multiple or mixin inheritance in which the composer has full control over the trait composition. The result enables more reuse than single inheritance without introducing the drawbacks of multiple or mixin inheritance. Several extensions of the model have been proposed [48], [90], [34], [53] and several type systems were defined [57], [98], [92], [72].

Traits are reusable building blocks that can be explicitly composed to share methods across unrelated class hierarchies. In their original form, traits do not contain state and cannot express visibility control for methods. Two extensions, stateful traits and freezable traits, have been proposed to overcome these limitations. However, these extensions are complex and not simple to implement.

**Research Agenda: Towards a pure trait language.** We plan distinct actions: (1) a large application of traits, (2) assessment of the existing trait models and (3) bootstrapping a pure trait language.

- To evaluate the expressiveness of traits, some hierarchies were refactored, showing code reuse [37]. However, such large refactorings, while valuable, may not exhibit all possible composition problems, since the hierarchies were previously expressed using single inheritance and following certain patterns. We want to redesign from scratch the collection library of Smalltalk (or part of it). Such a redesign should on the one hand demonstrate the added value of traits on a real large and redesigned library and on the other hand foster new ideas for the bootstrapping of a pure trait-based language.

In particular we want to reconsider the different models proposed (stateless [52], stateful [35], and freezable [53]) and their operators. We will compare these models by (1) implementing a trait-based collection hierarchy, (2) analyzing several existing applications that exhibit the need for traits —the Tweak UI library used in Sophie² and OpenCroquet³, the Icalendar implementation of Seaside⁴. Traits may be flattened [88]. This is a fundamental property that confers to traits their simplicity and expressiveness over Eiffel’s multiple inheritance. Keeping these aspects is one of our priority in forthcoming enhancements of traits.

- Alternative trait models. This work revisits the problem of adding state and visibility control to traits. Rather than extending the original trait model with additional operations, we use a fundamentally different approach by allowing traits to be lexically nested within other modules. This enables traits to express (shared) state and visibility control by hiding variables or methods in their lexical scope. Although the traits’ “flattening property” no longer holds when they can be lexically nested, the combination of traits with lexical nesting results in a simple and more expressive trait model. We formally specify the operational semantics of this combination. Lexically nested traits are fully implemented in AmbientTalk, where they are used among others in the development of a Morphic-like UI framework.

- We want to evaluate how inheritance can be replaced by traits to form a new object model. For this purpose we will design a minimal reflective kernel, inspired first from ObjVlisp but with trait composition [45], then from Smalltalk [59].

### 3.2.2. Reconciling Dynamic Languages and Security

**Context and Problems.** More and more applications require dynamic behavior such as modification of their own execution (often implemented using reflective features [67]). For example, F-script allows one to script Cocoa Mac-OS X applications and Lua is used in Adobe Photoshop. Now in addition more and more applications are updated on the fly, potentially loading untrusted code or simply broken code. Bytecode checking and static code verification are used to enable security, however such approaches do not really work in presence of dynamic languages and reflective features. Therefore there is a tension between a need for flexibility and for security —here, by security we mean a mix between confidentiality and integrity.

---

³ [http://www.opencroquet.org](http://www.opencroquet.org)
⁴ [http://www.seaside.st](http://www.seaside.st)
Research Agenda: A secure dynamic and reflective language. To solve this tension, we will work on Sure, a language where security is provided by construction: as an example, if the language does not offer field access and its reflective facilities are controlled, then the possibility to access and modify private data is controlled. In this context, layering and modularizing the meta-level [38], as well as controlling the access to reflective features [41], [42] are important challenges. We plan to:

- Study the security abstractions available in erights\(^5\) [85], [84], Java as well as classLoader strategies [71], [60].
- Categorize the different reflective features of languages such as CLOS [66], Python and Smalltalk [93] and identify suitable security mechanisms and infrastructure [58].
- Assess different security models (access rights, capabilities [94]...) and identify the ones adapted to our context as well as different access and right propagation.
- Define a language based on
  - the decomposition and restructuring of the reflective features [38],
  - the use encapsulation policies as a basis to restrict the interfaces of the controlled objects [96],
  - the definition of method modifiers to support privacy in the context of dynamic languages.

An open question is whether, instead of providing restricted interfaces, we could use traits to grant additional behavior to specific instances: without trait application, the instances would only exhibit default public behavior, but with additional traits applied, the instances would get extra behavior. We will develop Sure, a modular extension of the reflective kernel of Smalltalk (since it is one of the languages offering the largest set of reflective features such as pointer swapping, class changing, class definition...) [93].

\(^5\) http://www.erights.org
3. Scientific Foundations

3.1. Components and semantics

The primary foundations of the software component technology developed by SARDES relate to the component-based software engineering [110], and software architecture [108] fields. Nowadays, it is generally recognized that component-based software engineering and software architecture approaches are crucial to the development, deployment, management and maintenance of large, dependable software systems [57]. Several component models and associated architecture description languages have been devised over the past fifteen years; see e.g. [89] for an analysis of recent component models, and [93], [64] for surveys of architecture description languages.

To natively support configurability and adaptability in systems, SARDES component technology also draws from ideas in reflective languages [84], and reflective middleware [87], [62], [69]. Reflection can be used both to increase the separation of concerns in a system architecture, as pioneered by aspect-oriented programming [85], and to provide systematic means for modifying a system implementation.

The semantical foundations of component-based and reflective systems are not yet firmly established, however. Despite much work on formal foundations for component-based systems [90], [52], several questions remain open. For instance, notions of program equivalence when dealing with dynamically configurable capabilities, are far from being understood. To study the formal foundations of component-based technology, we try to model relevant constructs and capabilities in a process calculus, that is simple enough to formally analyze and reason about. This approach has been used successfully for the analysis of concurrency with the π-calculus [96], or the analysis of object-orientation [53]. Relevant developments for SARDES endeavours include behavioral theory and coinductive proof techniques [105], [103], process calculi with localities [65], [67], [70], and higher-order variants of the π-calculus [104], [77].

3.2. Open programming

Part of the language developments in SARDES concern the challenge of providing programming support for computer systems with continuously running services and applications, that operate at multiple physical and logical locations, that are constantly introduced, deployed, and combined, that interact, fail and evolve all the time. Programming such systems — called open programming by the designers of the Alice programming language [101] — is challenging because it requires the combination of several features, notably: (i) modularity, i.e. the ability to build systems by combining and composing multiple elements; (ii) security, i.e. the ability to deal with unknown and untrusted system elements, and to enforce if necessary their isolation from the rest of the system; (iii) distribution, i.e. the ability to build systems out of multiple elements executing separately on multiple interconnected machines, which operate at different speed and under different capacity constraints, and which may fail independently; (iv) concurrency, i.e. the ability to deal with multiple concurrent events, and non-sequential tasks; and (v) dynamicity, i.e. the ability to introduce new systems, as well as to remove, update and modify existing ones, possibly during their execution.

The rigorous study of programming features relate to the study of programming language constructs and semantics [97], [112], in general. Each of the features mentioned above has been, and continues to be, the subject of active research on its own. Combining them into a practical programming language with a well-defined formal semantics, however, is still an open question. Recent languages that provide relevant background for SARDES’ research are:

- For their support of dynamic notions of modules and software components: Acute [106], Alice [101], [102], ArchJava [54], Classages [91], Erlang [56], Oz [112], and Scala [98].
- For their security and failure management features: Acute, E [95], Erlang and Oz [68].
- For their support for concurrent and distributed execution, Acute, Alice, JoCaml [73], E, Erlang, Klaim [61], and Oz.
3.3. Software infrastructure

The SARDES approach to software infrastructure is both architecture-based and language-based: architecture-based for it relies on an explicit component structure for runtime reconfiguration, and language-based for it relies on a high-level type safe programming language as a basis for operating system and middleware construction. Exploiting high-level programming languages for operating system construction [109] has a long history, with systems such as Oberon [113], SPIN [60] or JX [74]. More recent and relevant developments for SARDES are:

- The developments around the Singularity project at Microsoft Research [72], [80], which illustrates the use of language-based software isolation for building a secure operating system kernel.
- The seL4 project [75], [86], which developed a formal verification of a modern operating system microkernel using the Isabelle/HOL theorem prover.
- The development of operating system kernels for multicore hardware architectures such as Corey [63] and Barrelfish [59].
- The development of efficient run-time for event-based programming on multicore systems such as libasync [114], [88].

3.4. System management and control

Management (or Administration) is the function that aims at maintaining a system’s ability to provide its specified services, with a prescribed quality of service. We approach management as a control activity, involving an event-reaction loop: the management system detects events that may alter the ability of the managed system to perform its function, and reacts to these events by trying to restore this ability. The operations performed under system and application administration include observation and monitoring, configuration and deployment, resource management, performance management, and fault management.

Up to now, administration tasks have mainly been performed in an ad-hoc fashion. A great deal of the knowledge needed for administration tasks is not formalized and is part of the administrators’ know-how and experience. As the size and complexity of the systems and applications are increasing, the costs related to administration are taking up a major part of the total information processing budgets, and the difficulty of the administration tasks tends to approach the limits of the administrators’ skills. For example, an analysis of the causes of failures of Internet services [99] shows that most of the service’s downtime may be attributed to management errors (e.g. wrong configuration), and that software failures come second. In the same vein, unexpected variations of the load are difficult to manage, since they require short reaction times, which human administrators are not able to achieve.

The above motivates a new approach, in which a significant part of management-related functions is performed automatically, with minimal human intervention. This is the goal of the so-called autonomic computing movement [82]. Several research projects [51] are active in this area. [83], [79] are recent surveys of the main research problems related to autonomic computing. Of particular importance for SARDES’ work are the issues associated with configuration, deployment and reconfiguration [71], and techniques for constructing control algorithms in the decision stage of administration feedback loops, including discrete control techniques [66], and continuous ones [76].

Management and control functions built by SARDES require also the development of distributed algorithms [92], [111] at different scales, from algorithms for multiprocessor architectures [78] to algorithms for cloud computing [94] and for dynamic peer-to-peer computing systems [55], [100]. Of particular relevance in the latter contexts are epidemic protocols such as gossip protocols [107] because of their natural resilience to node dynamicity or churn, an inherent scalability.
3. Scientific Foundations

3.1. Introduction

Our scientific foundations are grounded on two different dimensions, distributed collaborative systems supported by sophisticated data sharing mechanisms and service oriented computing with an emphasis on orchestration and on non functional properties.

Distributed collaborative systems enable distributed group work using computer technologies. Designing such systems require an expertise in the domain of Distributed Systems (DS) and in the Computer-supported collaborative work (CSCW) research area. Besides theoretical and technical aspects of distributed systems, design of distributed collaborative systems must take into account the human factor to offer solutions suitable for users and groups. SCORE team vision is to move away from a centralized authority based collaboration towards a decentralized collaboration where users have full control over their data that they can store locally and can decide with whom to share them. In this type of collaboration SCORE team investigated the issues of management of distributed shared data and coordination between users and groups.

Service oriented Computing [33] is an established domain in which the ECOO and now the SCORE team has been working on for a long time. It refers to the general discipline that studies the development of computer applications at the scale of the web. In this context, a service is an independent software program with a specific functional context and capabilities published as a service contract (or more traditionally an API). Service composition is the aggregation of a set of services whose interactions are coordinated. The scale, the autonomy of services, the heterogeneity and some well defined design principles underlying SoC opens new research questions that are at the basis of the SCORE problematic and that spans the disciplines of distributed computing, software engineering and CSCW. Our approach to contribute to the general vision of Service Oriented Computing and more generally to the emerging discipline of Service Science has been and is still to focus on the question of the efficient and flexible construction of reliable and secure high level services through the coordination/orchestration/composition of other services provided by distributed organizations or people.

3.2. Consistency Models for Distributed Collaborative Systems

Collaborative systems are distributed systems that allow users to share data. One important issue is to manage consistency of shared data according to concurrent access. Traditional consistency criteria such as locking, serializability, linearizability are not adequate for collaborative systems.

Causality, Convergence and Intention preservation (CCI) [36] are more suitable for developing middleware for collaborative applications.

We develop algorithms for ensuring CCI properties on collaborative distributed systems. Constraints on the algorithms are different according to the type of distributed system and type of data. The distributed system can be centralized, decentralized or peer-to-peer. The type of data can include strings, growable arrays, ordered trees, semantic graphs and multimedia data.

3.3. Optimistic Replication

Replication of data among different nodes of a network allows improving reliability, fault-tolerance, and availability. When data are mutable, consistency among the different replicas must be ensured. Pessimistic replication is based on the principle of single-copy consistency while optimistic replication allows the replicas to diverge during a short time period. The consistency model for optimistic replication [35] is called eventual consistency, meaning that replicas are guaranteed to converge to the same value when the system is idle.
Our research focuses on the two most promising families of optimistic replication algorithms for ensuring CCI:

- the operational transformation (OT) algorithms [31]
- the algorithms based on commutative replicated data types (CRDT) [34]

Operational transformation algorithms are based on the application of a transformation function when a remote modification is integrated into the local document. Integration algorithms are generic, being parameterized by operational transformation functions which depend on replicated document types. The advantage of these algorithms is their genericity. These algorithms can be applied to any data type and they can merge heterogeneous data in a uniform manner.

Commutative replicated data types is a new class of algorithms initiated by WOOT [32]. They ensure consistency of highly dynamic content on peer-to-peer networks. Unlike traditional optimistic replication algorithms, they can ensure consistency without concurrency control. CRDT algorithms rely on natively commutative operations defined on abstract data types such as lists or ordered trees. Thus, they do not require a merge algorithm or an integration procedure.

3.4. Business Process Management

Business Process Management (BPM) is considered as a core discipline behind Service Management and Computing. BPM, that includes the analysis, the modeling, the execution, the monitoring and the continuous improvement of enterprise processes is for us a central domain of studies.

A lot of efforts has been devoted in the past years to established standards business process models founded on well grounded theories (e.g. Petri Nets) that meet the needs of both business analyst but also of software engineers and software integrators. This has lead to heated debate as both points of view are very difficult to reconcile between the analyst side and the IT side. On one side, the business people in general require models that are easy to use and understand and that can be quickly adapted to exceptional situations. On the other side, IT people need models with an operational semantic in order to be able transform them into executable artifacts. Part of our work has been an attempt to reconcile these point of views, leading on one side to the Bonita product and more recently on our work in crisis management where the same people are designing, executing and monitoring the process as it executes. But more generally, and at a larger scale, we have been considering the problem of process spanning the barriers of organisations. This leads us to consider the more general problem of service composition as a way to coordinate inter organisational construction of application providing value based on the composition of lower level services [28].

3.5. Service Composition

More and more, we are considering processes as piece of software whose execution traverse the boundaries of organisations. This is especially true with service oriented computing where processes compose services produced by many organisations. We tackle this problem from very different perspectives, trying to find the best compromise between the need for privacy of internal processes from organisations and the necessity to publicize large part of them, proposing to distribute the execution and the orchestration of processes among the organisations themselves, and attempting to ensure transactional properties in this distributed setting [27].

Non functional aspects of service composition relate to all the properties and service agreements that one want to ensure and that are orthogonal to the actual business but that are important when a service is selected and integrated in a composition. This includes transactional context, security, privacy, and quality of service in general. Defining and orchestrating services on a large scale while providing the stakeholders with some strong guarantees on their execution is a first class problem for us. For a long time, we have proposed models and solutions to ensure that some properties (e.g. transactional properties) were guaranteed on process execution, either through design or through the definition of some protocols. Our work has also been extended to the problems of security, privacy and service level agreement among partners. These questions are still central in our work.
3. Scientific Foundations

3.1. Model Driven Engineering for Distributed Software

3.1.1. Software Product Lines

It is seldom the case nowadays that we can any longer deliver software systems with the assumption that one-size-fits-all. We have to handle many variants accounting not only for differences in product functionalities (range of products to be marketed at different prices), but also for differences in hardware (e.g.; graphic cards, display capacities, input devices), operating systems, localization, user preferences for GUI (“skins”). Obviously, we do not want to develop from scratch and independently all of the variants the marketing department wants. Furthermore, all of these variant may have many successive versions, leading to a two-dimensional vision of product-lines.

3.1.2. Object-Oriented Software Engineering

The object-oriented approach is now widespread for the analysis, the design, and the implementation of software systems. Rooted in the idea of modeling (through its origin in Simula), object-oriented analysis, design and implementation takes into account the incremental, iterative and evolutive nature of software development [52], [50]: large software system are seldom developed from scratch, and maintenance activities represent a large share of the overall development effort.

In the object-oriented standard approach, objects are instances of classes. A class encapsulates a single abstraction in a modular way. A class is both closed, in the sense that it can be readily instanciated and used by clients objects, and open, that is subject to extensions through inheritance [54].

3.1.3. Design Pattern

Since by definition objects are simple to design and understand, complexity in an object-oriented system is well known to be in the collaboration between objects, and large systems cannot be understood at the level of classes and objects. Still these complex collaborations are made of recurring patterns, called design patterns. The idea of systematically identifying and documenting design patterns as autonomous entities was born in the late 80’s. It was brought into the mainstream by such people as Beck, Ward, Coplien, Booch, Kerth, Johnson, etc. (known as the Hillside Group). However the main event in this emerging field was the publication, in 1995, of the book Design Patterns: Elements of Reusable Object Oriented Software by the so-called Gang of Four (GoF), that is E. Gamma, R. Helm, R. Johnson and J. Vlissides [51]. Today, design patterns are widely accepted as useful tools for guiding and documenting the design of object-oriented software systems. Design patterns play many roles in the development process. They provide a common vocabulary for design, they reduce system complexity by naming and defining abstractions, they constitute a base of experience for building reusable software, and they act as building blocks from which more complex designs can be built. Design patterns can be considered reusable micro-architectures that contribute to an overall system architecture. Ideally, they capture the intent behind a design by identifying the component objects, their collaborations, and the distribution of responsibilities. One of the challenges addressed in the Triskell project is to develop concepts and tools to allow their formal description and their automatic application.

3.1.4. Component

The object concept also provides the basis for software components, for which Szyperski’s definition [58] is now generally accepted, at least in the industry:

A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third party.
Component based software relies on assemblies of components. Such assemblies rely in turn on fundamental mechanisms such as precise definitions of the mutual responsibility of partner components, interaction means between components and their non-component environment and runtime support (e.g. .Net, EJB, Corba Component Model CCM, OSGI or Fractal).

Components help reducing costs by allowing reuse of application frameworks and components instead of redeveloping applications from scratch (product line approach). But more important, components offer the possibility to radically change the behaviors and services offered by an application by substitution or addition of new components, even a long time after deployment. This has a major impact of software lifecycle, which should now handle activities such as the design of component frameworks, the design of reusable components as deployment units, the validation of component compositions coming from various origins and the component life-cycle management.

Empirical methods without real component composition models have appeared during the emergence of a real component industry (at least in the Windows world). These methods are now clearly the cause of untractable validation and of integration problems that can not be transposed to more critical systems (see for example the accidental destruction of Ariane 501 [53]).

Providing solutions for formal component composition models and for verifiable quality (notion of trusted components) are especially relevant challenges. Also the methodological impact of component-based development (for example within the maturity model defined by the SEI) is also worth attention.

### 3.1.5. Contracts

Central to this trusted component notion is the idea of contract. A software contract captures mutual requirements and benefits among stake-holder components, for example between the client of a service and its suppliers (including subcomponents). Contracts strengthen and deepen interface specifications. Along the lines of abstract data type theory, a common way of specifying software contracts is to use boolean assertions called pre- and post-conditions for each service offered, as well as class invariants for defining general consistency properties. Then the contract reads as follows: The client should only ask a supplier for a service in a state where the class invariant and the precondition of the service are respected. In return, the supplier promises that the work specified in the post-condition will be done, and the class invariant is still respected. In this way rights and obligations of both client and supplier are clearly delineated, along with their responsibilities. This idea was first implemented in the Eiffel language [55] under the name Design by Contract, and is now available with a range of expressive power into several other programming languages (such as Java) and even in the Unified Modeling Language (UML) with the Object Constraint Language (OCL) [59]. However, the classical predicate based contracts are not enough to describe the requirements of modern applications. Those applications are distributed, interactive and they rely on resources with random quality of service. We have shown that classical contracts can be extended to take care of synchronization and extrafunctional properties of services (such as throughput, delays, etc) [49].

### 3.1.6. Models and Aspects

As in other sciences, we are increasingly resorting to modelling to master the complexity of modern software development. According to Jeff Rothenberg,

*Modeling, in the broadest sense, is the cost-effective use of something in place of something else for some cognitive purpose. It allows us to use something that is simpler, safer or cheaper than reality instead of reality for some purpose. A model represents reality for the given purpose; the model is an abstraction of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger and irreversibility of reality.*

So modeling is not just about expressing a solution at a higher abstraction level than code. This has been useful in the past (assembly languages abstracting away from machine code, 3GL abstracting over assembly languages, etc.) and it is still useful today to get a holistic view on a large C++ program. But modeling goes well beyond that.
Modeling is indeed one of the touchstones of any scientific activity (along with validating models with respect to experiments carried out in the real world). Note by the way that the specificity of engineering is that engineers build models of artefacts that usually do not exist yet (with the ultimate goal of building them).

In engineering, one wants to break down a complex system into as many models as needed in order to address all the relevant concerns in such a way that they become understandable enough. These models may be expressed with a general purpose modeling language such as the Unified Modeling Language (UML), or with Domain Specific Languages when it is more appropriate.

Each of these models can be seen as the abstraction of an aspect of reality for handling a given concern. The provision of effective means for handling such concerns makes it possible to establish critical trade-offs early on in the software life cycle, and to effectively manage variation points in the case of product-lines.

Note that in the Aspect Oriented Programming community, the notion of aspect is defined in a slightly more restricted way as the modularization of a cross-cutting concern. If we indeed have an already existing “main” decomposition paradigm (such as object orientation), there are many classes of concerns for which clear allocation into modules is not possible (hence the name “cross-cutting”). Examples include both allocating responsibility for providing certain kinds of functionality (such as logging) in a cohesive, loosely coupled fashion, as well as handling many non-functional requirements that are inherently cross-cutting e.g.; security, mobility, availability, distribution, resource management and real-time constraints.

However now that aspects become also popular outside of the mere programming world [56], there is a growing acceptance for a wider definition where an aspect is a concern that can be modularized. The motivation of these efforts is the systematic identification, modularization, representation, and composition of these concerns, with the ultimate goal of improving our ability to reason about the problem domain and the corresponding solution, reducing the size of software model and application code, development costs and maintenance time.

### 3.1.7. Design and Aspect Weaving

So really modeling is the activity of separating concerns in the problem domain, an activity also called analysis. If solutions to these concerns can be described as aspects, the design process can then be characterized as a weaving of these aspects into a detailed design model (also called the solution space). This is not new: this is actually what designers have been effectively doing forever. Most often however, the various aspects are not explicit, or when there are, it is in the form of informal descriptions. So the task of the designer is to do the weaving in her head more or less at once, and then produce the resulting detailed design as a big tangled program (even if one decomposition paradigm, such as functional or object-oriented, is used). While it works pretty well for small problems, it can become a major headache for bigger ones.

Note that the real challenge here is not on how to design the system to take a particular aspect into account: there is a huge design know-how in industry for that, often captured in the form of Design Patterns (see above). Taking into account more than one aspect as the same time is a little bit more tricky, but many large scale successful projects in industry are there to show us that engineers do ultimately manage to sort it out.

The real challenge in a product-line context is that the engineer wants to be able to change her mind on which version of which variant of any particular aspect she wants in the system. And she wants to do it cheaply, quickly and safely. For that, redoing by hand the tedious weaving of every aspect is not an option.

### 3.1.8. Model Driven Engineering

Usually in science, a model has a different nature that the thing it models (“do not take the map for the reality” as Sun Tse put it many centuries ago). Only in software and in linguistics a model has the same nature as the thing it models. In software at least, this opens the possibility to automatically derive software from its model. This property is well known from any compiler writer (and others), but it was recently made quite popular with an OMG initiative called the Model Driven Architecture (MDA). This requires that models are no longer informal, and that the weaving process is itself described as a program (which is as a matter of facts an executable meta-model) manipulating these models to produce a detailed design that can ultimately be transformed to code or at least test suites.
The OMG has built a meta-data management framework to support the MDA. It is mainly based on a unique M3 “meta-meta-model” called the Meta-Object Facility (MOF) and a library of M2 meta-models, such as the UML (or SPEM for software process engineering), in which the user can base his M1 model.

The MDA core idea is that it should be possible to capitalize on platform-independent models (PIM), and more or less automatically derive platform-specific models (PSM) –and ultimately code– from PIM through model transformations. But in some business areas involving fault-tolerant, distributed real-time computations, there is a growing concern that the added value of a company not only lies in its know-how of the business domain (the PIM) but also in the design know-how needed to make these systems work in the field (the transformation to go from PIM to PSM). Reasons making it complex to go from a simple and stable business model to a complex implementation include:

- Various modeling languages used beyond UML,
- As many points of views as stakeholders,
- Deliver software for (many) variants of a platform,
- Heterogeneity is the rule,
- Reuse technical solutions across large product lines (e.g. fault tolerance, security, etc.),
- Customize generic transformations,
- Compose reusable transformations,
- Evolve and maintain transformations for 15+ years.

This wider context is now known as Model Driven Engineering.
3. Scientific Foundations

3.1. Structuring of Applications for Scalability

Participants: Sylvain Contassot-Vivier, Jens Gustedt, Soumeya Leila Hernane, Thomas Jost, Wilfried Kirschenmann, Stéphane Vialle.

Large Scale Computing is a challenge under constant development that, almost by definition, will always try to reach the edge of what is possible at any given moment in time: in terms of the scale of the applications under consideration, in terms of the efficiency of implementations and in what concerns the optimized utilization of the resources that modern platforms provide or require. The complexity of all these aspects is currently increasing rapidly:

3.1.1. Diversity of platforms.

Design of processing hardware is diverging in many different directions. Nowadays we have SIMD registers inside processors, on-chip or off-chip accelerators (GPU, FPGA, vector-units), multi-cores and hyperthreading, multi-socket architectures, clusters, grids, clouds...The classical monolithic architecture of one-algorithm/one-implementation that solves a problem is obsolete in many cases. Algorithms (and the software that implements them) must deal with this variety of execution platforms robustly.

As we know, the “free lunch” for sequential algorithms provided by the increase of processor frequencies is over [41], we have to go parallel. But the “free lunch” is also over for many automatic or implicit adaption strategies between codes and platforms: e.g the best cache strategies can’t help applications that accesses memory randomly, or algorithms written for “simple” CPU (von Neumann model) have to be adapted substantially to run efficiently on vector units.

3.1.2. The communication bottleneck.

Communication and processing capacities evolve at a different pace, thus the communication bottleneck is always narrowing. An efficient data management is becoming more and more crucial.

Not many implicit data models have yet found their place in the HPC domain, because of a simple observation: latency issues easily kill the performance of such tools. In the best case, they will be able to hide latency by doing some intelligent caching and delayed updating. But they can never hide the bottleneck for bandwidth.

HPC was previously able to cope with the communication bottleneck by using an explicit model of communication, namely MPI. It has the advantage of imposing explicit points in code where some guarantees about the state of data can be given. It has the clear disadvantage that coherence of data between different participating entities is difficult to manage and is completely left to the programmer.

Here, our approach is and will be to timely request explicit actions (like MPI) that mark the availability of (or need for) data. Such explicit actions ease the coordination between tasks (coherence management) and allow the platform underneath the program to perform a proactive resource management.

3.1.3. Models of interdependence and consistency.

Interdependence of data between different tasks of an application and components of hardware will be crucial to ensure that developments will possibly scale on the ever diverging architectures. We have up to now presented such models (PRO, DHO, ORWL) and their implementations, and proved their validity for the context of SPMD-type algorithms.
Over the next years we will have to enlarge their spectrum of application. On the algorithm side we will have to move to heterogeneous computations combining different types of tasks in one application. For the architectures we will have to take into account the fact of increased heterogeneity, processors of different speed, multi-cores, accelerators (FPU, GPU, vector units), communication links of different bandwidth and latency, memory and generally storage capacity of different size, speed and access characteristics. The first implementations using ORWL in that context look particularly promising.

The models themselves will have to evolve to be better suited for more types of applications, such that they allow for a more fine-grained partial locking and access of objects. They should handle e.g collaborative editing or the modification of just some fields in a data structure. This work has already started with DHO which allows the locking of data ranges inside an object. But a more structured approach would certainly be necessary here to be usable more comfortably in applications.

3.1.4. Algorithmic paradigms

Concerning asynchronous algorithms, we have developed several versions of implementations, allowing us to precisely study the impact of our design choices. However, we are still convinced that improvements are possible, especially concerning the global convergence detection strategies. Typically, the way that detection is performed must take into account the architecture of the parallel system. We have observed that switching between asynchronous and synchronous iterations during the process can be better on small/middle size systems with high performance networks whereas a completely asynchronous process should be used on (very) large systems or with low performance networks. We are currently working on the design of a generic and non-intrusive way of implementing such a procedure in any parallel iterative algorithm.

Also, we would like to compare other variants of asynchronous algorithms, such as waveform relaxation. In that context, computations are not performed for each time step of the simulation but for an entire time interval. Then, the evolutions of the elements at the frontiers between processors sub-domains are exchanged in an asynchronous way. Although we have already studied such schemes in the past, we would like to evaluate how they behave on recent architectures, and how the models and software for data consistency mentioned above can be helpful in that context.

3.1.5. Cost models and accelerators

We have already designed some models that relate computation power and energy consumption. Our next goal is to design and implement an auto-tuning system that controls the application according to user defined optimization criteria (computation and/or energy performance). This implies the insertion of multi-schemes and/or multi-kernels into the application such that it will able to adapt its behavior to the requirements.

3.2. Experimentation Methodology

Participants: Sébastien Badia, Pierre-Nicolas Clauss, El Mehdi Fekari, Stéphane Genaud, Jens Gustedt, Marion Guthmuller, Lucas Nussbaum, Martin Quinson, Cristian Rosa, Luc Sarzyniec, Christophe Thiéry.

As emphasized above, our scientific objects are constituted by modern distributed systems. The scientific questions that we address mainly concern the performance and correction of these systems. Their specificities are their ever increasing size, complexity and dynamics. This has reached a point where it becomes practically impossible to fully understand these systems through the fine description of their components’ interaction. In that sense, computer science occupies a relatively unique position in epistemology since its scientific objects are human built artifacts, but become too complex to be directly understood by humans. This mandates a shift from the methodologies traditionally used in speculative sciences, where the knowledge is built a priori without experiments, to the ones of natural sciences, centered on the accumulation of knowledge through experiments.
These experiments can naturally be conducted on real platforms, but it is rarely practical to do so on real production platforms: the experiments can be disruptive to the production usage, and ongoing production can impose an uncontrollable bias on experiments. Direct execution on experimental facilities such as Grid’5000 alleviate these issues by providing a computing infrastructure similar to the targeted production platforms but dedicated to experiments and completely controllable. The obvious advantage of such direct execution is that it removes almost any experimental bias but it is naturally not applicable for some studies where the application to test or the target platform are not available. That is why simulation is very attractive to conduct what if analysis, for example to dimension a platform still to be built according to a given requirement, or to select the best algorithm for an application which is still to be written. Several intermediate steps exist between simulation and direct execution: benchmarking consists in experimenting with a real environment using synthetic applications while emulation consists in executing a real application on a platform model. The former is mainly used to devise some absolute insight about the environment without taking the application specificities into account. The latter is helpful to assess the behavior of an application on an environment that is not available [7].

These experimental methodologies are complementary, and the experimenters should combine them accordingly to benefit from their strengths in the process leading from an idea to a released product. Simulation is well adapted to the study of algorithms and helps in the realization of prototypes; direct execution on experimental facilities is an essential tool to change a prototype into an application; emulation permits to test the application in a wide range of conditions to transform it into a robust product.

Our team is active on most experimental methodologies (the only exception is benchmarking, since it consists in abstracting from the application to focus on the platforms while our whole approach is centered on the applications). This gives us an ideal position to attack the larger challenge of improving experimental methodology in distributed systems research, in order to put it on par with what is done in other sciences. We will ensure that our methods and tools support a top-quality and unified experimental process by bridging interfaces and validating our work through series of experiments spanning all methodologies.

This axis and the Structuring applications axis benefit from each other by building a positive feedback loop: the Structuring applications axis provides first-class use cases, while the Experimentation axis helps to confirm the design choices made in the first axis. Other use cases are provided by our ongoing collaborations with the production grids community, with industry and with the Grid’5000 ecosystem (through INRIA AEN Hemera).

3.2.1. Simulation and dynamic verification.

Our team plays a key role in the SimGrid project, a mature simulation toolkit widely used in the distributed computing community. Since over ten years, we work on the validity, scalability and robustness of our tool. In
In the future, we aim at extending further the applicability to Clouds and Exascale systems. Therefore, we will provide disk and memory models in addition to the already existing network and CPU models. We will also pursue our efforts on the tool’s scalability and efficiency. Interfaces constitute another important work axis, with the addition of specific APIs on top of our simulation kernel. They will provide the “syntactic sugar” needed to express algorithms of these communities. For example, virtual machines will be handled explicitly in the interface provided for Cloud studies. Similarly, we will pursue our work on an implementation of the MPI standard allowing to study real applications using that interface. This work will also be extended to other interfaces such as OpenMP or the ones developed in our team to structure the applications, e.g ORWL. During the next evaluation period, we also consider using our toolbox to give online performance predictions to the runtimes. It would allow these systems to improve their adaptability to the changing performance conditions experienced on the platform.

We recently integrated a model checking kernel in our tool to enable formal correction studies in addition to the practical performance studies enabled by simulation. Being able to study these two fundamental aspects of distributed applications from the same tool constitute a major advantage. In the next evaluation period, we will further work on this convergence for the study of correctness and performance using the same tools. Ensuring that they become usable on real applications constitute another challenge that we will address.

3.2.2. Experimentation on experimental facilities

During the last years, we have played a key role in the design and development of Grid’5000 by taking part in technical and design discussions and by managing several engineers working on the platform. We will pursue our involvement in the design of the testbed with a focus on ensuring that the testbed provides all the features needed for high-quality experimentation.

We will also work on experiment-supporting software, such as Kadeploy (software used to deploy user environments on Grid’5000) in INRIA ADT Kadeploy (2011-2013) and basic services that are common to most experiments (control of a large number of nodes, data management, etc.) in INRIA ADT Solfege (2011-2013).

Specifically, we will pursue the development of our emulation framework, Distem (based on our previous emulation framework, Wrekavoc). We will address new challenges such as multi-core and heterogeneous systems, add load generation and fault injection features, and increase its usability, to get it widely accepted and used within the community.
Most experiments performed on Grid’5000 are still controlled manually or, in the best case, with quick and dirty scripts. This severely limits the quality of experimental results since it is hard to reproduce experiments many times to ensure statistical validity or to fully describe an experimental setup which was not created by an automated process. It also limits the scale and/or complexity of the possible experiments. Together with researchers from the field of Business Process Management (BPM), we will work on enabling the use of Workflow Management Systems to perform experiments on distributed systems.

Overall, we hope that our work in this research axis will bring major contributions to the industrialization of experimentation on parallel and distributed systems.
3. Scientific Foundations

3.1. Modeling Platform Dynamics

Modeling the platform dynamics in a satisfying manner, in order to design and analyze efficient algorithms, is a major challenge. In distributed platforms, the performance of individual nodes (be they computing or communication resources) will fluctuate; in a fully dynamic platform, the set of available nodes will also change over time, and algorithms must take these changes into account if they are to be efficient.

There are basically two ways one can model such evolution: one can use a stochastic process, or some kind of adversary model.

In a stochastic model, the platform evolution is governed by some specific probability distribution. One obvious advantage of such a model is that it can be simulated and, in many well-studied cases, analyzed in detail. The two main disadvantages are that it can be hard to determine how much of the resulting algorithm performance comes from the specifics of the evolution process, and that estimating how realistic a given model is – none of the current project participants are metrology experts.

In an adversary model, it is assumed that these unpredictable changes are under the control of an adversary whose goal is to interfere with the algorithms efficiency. Major assumptions on the system’s behavior can be included in the form of restrictions on what this adversary can do (like maintaining such or such level of connectivity). Such models are typically more general than stochastic models, in that many stochastic models can be seen as a probabilistic specialization of a nondeterministic model (at least for bounded time intervals, and up to negligible probabilities of adopting “forbidden” behaviors).

Since we aim at proving guaranteed performance for our algorithms, we want to concentrate on suitably restricted adversary models. The main challenge in this direction is thus to describe sets of restricted behaviors that both capture realistic situations and make it possible to prove such guarantees.

3.2. Models for Platform Topology and Parameter Estimation

On the other hand, in order to establish complexity and approximation results, we also need to rely on a precise theoretical model of the targeted platforms.

- At a lower level, several models have been proposed to describe interference between several simultaneous communications. In the 1-port model, a node cannot simultaneously send to (and/or receive from) more than one node. Most of the “steady state” scheduling results have been obtained using this model. On the other hand, some authors propose to model incoming and outgoing communication from a node using fictitious incoming and outgoing links, whose bandwidths are fixed. The main advantage of this model, although it might be slightly less accurate, is that it does not require strong synchronization and that many scheduling problems can be expressed as multi-commodity flow problems, for which efficient decentralized algorithms are known. Another important issue is to model the bandwidth actually allocated to each communication when several communications compete for the same long-distance link.

- At a higher level, proving good approximation ratios on general graphs may be too difficult, and it has been observed that actual platforms often exhibit a simple structure. For instance, many real life networks satisfy small-world properties, and it has been proved, for instance, that greedy routing protocols on small world networks achieve good performance. It is therefore of interest to prove that logical (given by the interactions between hosts) and physical platforms (given by the network links) exhibit some structure in order to derive efficient algorithms.
3.3. General Framework for Validation

3.3.1. Low level modeling of communications

In the context of large scale dynamic platforms, it is unrealistic to determine precisely the actual topology and the contention of the underlying network at application level. Indeed, existing tools such as Alnem [60] are very much based on quasi-exhaustive determination of interferences, and it takes several days to determine the actual topology of a platform made up of a few tens of nodes. Given the dynamism of the platforms we target, we need to rely on less sophisticated models, whose parameters can be evaluated at runtime.

Therefore, we propose to model each node using a small set of parameters. This is related to the theoretical notion of distance labeling [59], and corresponds to assigning labels to the nodes, so that a cheap operation on the labels of two nodes provides an estimation of the value of a given parameter (the latency or the bandwidth between two nodes, for instance). Several solutions for performance estimation on the Internet are based on this notion, under the terminology of Network Coordinate Systems. Vivaldi [57], IDES [61] and Sequoia [62] are examples of such systems for latency estimation. In the case of bandwidth estimation, fewer solutions have been proposed. We have studied the last-mile model, in which we model each node by an incoming and an outgoing bandwidth and neglect interference that appears at the core of the network (Internet), in order to concentrate on local constraints.

3.3.2. Simulation

Once low level modeling has been obtained, it is crucial to be able to test the proposed algorithms. To do this, we will first rely on simulation rather than direct experimentation. Indeed, in order to be able to compare heuristics, it is necessary to execute those heuristics on the same platform. In particular, all changes in the topology or in the resource performance should occur at the same time during the execution of the different heuristics. In order to be able to replicate the same scenario several times, we need to rely on simulations. Moreover, a metric for providing approximation results in the case of dynamic platforms necessarily requires computing the optimal solution at each time step, which can be done off-line if all traces for the different resources are stored. Using simulation rather than experiments can be justified if the simulator itself has been proven valid. Moreover, the modeling of communications, processing and their interactions may be much more complex in the simulator than in the model used to provide a theoretical approximation ratio, such as in SimGrid. In particular, sophisticated TCP models for bandwidth sharing have been implemented in SimGRID.

During the course of the USS-SimGrid ANR Arpege project, the SimGrid simulation framework has been adapted to large scale environments. Thanks to hierarchical platform description, to simpler and more scalable network models, and to the possibility to distribute the simulation of several nodes, it is now possible to perform simulations of very large platforms (of the order of $10^5$ resources). This work will be continued in the ANR SONGS project, which aims at making SimGrid usable for Next Generation Systems (P2P, Grids, Clouds, HPC). In this context, simulation of exascale systems are envisioned, and we plan to develop models for platform dynamicity to allow realistic and reproducible experimentation of our algorithms.

3.3.3. Practical validation and scaling

Finally, we propose several applications that will be described in detail in Section 5. These applications cover a large set of fields (molecular dynamics, continuous integration...). All these applications will be developed and tested with an academic or industrial partner. In all these collaborations, our goal is to prove that the services that we propose in Section 4.1 can be integrated as steering tools in already developed software. Our goal is to assert the practical interest of the services we develop and then to integrate and to distribute them as a library for large scale computing.

At a lower level, in order to validate the models we propose, i.e. make sure that the predictions given by the model are close enough to the actual values, we need realistic datasets of network performance on large scale distributed platforms. Latency measurements are easiest to perform, and several datasets are available to researchers and serve as benchmarks to the community. Bandwidth datasets are more difficult to obtain, because of the measurement cost. As part of the bedibe software (see section 5.4), we have implemented
a script to perform such measurements on the Planet-Lab platform [56]. We plan to make these datasets available to the community so that they can be used as benchmarks to compare the different solutions proposed.

### 3.4. Efficient Queries and Compact Data Structures

The optimization schemes for content distribution processes or for handling standard queries require a good knowledge of the physical topology or performance (latencies, throughput, ...) of the network. Assuming that some rough estimate of the physical topology is given, former theoretical results described in Section 6.2 show how to pre-process the network so that local computations are performed efficiently. Due to the dynamism of large distributed platforms, some requirements on the coding of local data structures and the updating mechanism are needed. This last process is done using the maintenance of light virtual networks, so-called overlay networks (see Section 6.2). In our approach, we focus on:

- **Compact Routing tables.**
  Routing queries and broadcasting information on large scale platforms are tasks involving many basic message communications. The maximum performance objective imposes that basic messages are routed along paths of cost as low as possible. On the other hand, local routing decisions must be fast and the algorithms and data structures involved must support a certain amount of dynamism in the platform. Since the size of the data-structure impacts negatively the query and the update time, the space of the data-structure must be of limited size.

- **Local computations.**
  Although the size of the data structures is less constrained in comparison with P2P systems (due to security reasons), however, even in our collaborative framework, it is unrealistic that each node manages a complete view of the platform with the full resource characteristic. Thus, a node has to manage data structures concerning only a fraction of the whole system. In fact, a partial view of the network will be sufficient for many tasks: for instance, in order to compute the distance between two nodes a local and limited information available at the two nodes may suffice (distance labeling).

- **Overlay and small world networks.**
  The processes we consider can be highly dynamic. The preprocessing usually assumed takes polynomial time. Hence, when a new process arrives, it must be dealt with in an on-line fashion, i.e., we do not want to totally re-compute, and the (partial) re-computation has to be simple. In order to meet these requirements, overlay networks are normally implemented. These are light virtual networks, i.e., they are sparse and a local change of the physical network will only lead to a small change of the corresponding virtual network. As a result, small address books are sufficient at each node.
  A specific class of overlay networks are small-world networks. These are efficient overlay networks for (greedy) routing tasks assuming that distance requests can be performed easily.

- **Mobile Agent Computing.**
  Mobile Agent Computing has been proposed as a powerful paradigm to study distributed systems. Our purpose is to study the computational power of the mobile agent systems under various assumptions. Indeed, many models exist but little is known about their computational power. One major parameter describing a mobile agent model is the ability of the agents to interact. The most natural mobile agent computing problem is the exploration or mapping problem in which one or several mobile agents have to explore or map their environment. The rendezvous problem consists for two agents to meet at some unspecified node of the network. Two other fundamental problems deal with security, which is often the main concern of actual mobile agent systems. The first one consists in exploring the network in spite of harmful hosts that destroy incoming agents. An additional goal in this context is to locate the harmful host(s) to prevent further agent losses. We already mentioned the second problem related to security, which consists for the agents in capturing an intruder.
The goal is to enlarge the knowledge on the foundations of mobile agent computing. This will be done by developing new efficient algorithms for mobile agent systems and by proving impossibility results. This will also allow to compare the different models.

Of course, the main difficulty is to adapt the maintenance of local data structures to the dynamism of the network.
3. Scientific Foundations

3.1. Scheduling Strategies and Algorithm Design for Heterogeneous Platforms


Scheduling sets of computational tasks on distributed platforms is a key issue but a difficult problem. Although a large number of scheduling techniques and heuristics have been presented in the literature, most of them target only homogeneous resources. However, future computing systems, such as the computational Grid, are most likely to be widely distributed and strongly heterogeneous. Therefore, we consider the impact of heterogeneity on the design and analysis of scheduling techniques: how to enhance these techniques to efficiently address heterogeneous distributed platforms?

The traditional objective of scheduling algorithms is the following: given a task graph and a set of computing resources, or processors, map the tasks onto the processors, and order the execution of the tasks so that: (i) the task precedence constraints are satisfied; (ii) the resource constraints are satisfied; and (iii) a minimum schedule length is achieved. Task graph scheduling is usually studied using the so-called macro-dataflow model, which is widely used in the scheduling literature: see the survey papers [81], [90], [93], [94] and the references therein. This model was introduced for homogeneous processors, and has been (straightforwardly) extended to heterogeneous computing resources. In a word, there is a limited number of computing resources, or processors, to execute the tasks. Communication delays are taken into account as follows: let task $T$ be a predecessor of task $T'$ in the task graph; if both tasks are assigned to the same processor, no communication overhead is incurred, the execution of $T'$ can start immediately at the end of the execution of $T$; on the contrary, if $T$ and $T'$ are assigned to two different processors $P_i$ and $P_j$, a communication delay is incurred. More precisely, if $P_i$ completes the execution of $T$ at time-step $t$, then $P_j$ cannot start the execution of $T'$ before time-step $t + \text{comm}(T, T', P_i, P_j)$, where $\text{comm}(T, T', P_i, P_j)$ is the communication delay, which depends upon both tasks $T$ and $T'$, and both processors $P_i$ and $P_j$. Because memory accesses are typically several orders of magnitude cheaper than inter-processor communications, it is sensible to neglect them when $T$ and $T'$ are assigned to the same processor.

The major flaw of the macro-dataflow model is that communication resources are not limited in this model. Firstly, a processor can send (or receive) any number of messages in parallel, hence an unlimited number of communication ports is assumed (this explains the name macro-dataflow for the model). Secondly, the number of messages that can simultaneously circulate between processors is not bounded, hence an unlimited number of communications can simultaneously occur on a given link. In other words, the communication network is assumed to be contention-free, which of course is not realistic as soon as the number of processors exceeds a few units.

The general scheduling problem is far more complex than the traditional objective in the macro-dataflow model. Indeed, the nature of the scheduling problem depends on the type of tasks to be scheduled, on the platform architecture, and on the aim of the scheduling policy. The tasks may be independent (e.g., they represent jobs submitted by different users to a same system, or they represent occurrences of the same program run on independent inputs), or the tasks may be dependent (e.g., they represent the different phases of a same processing and they form a task graph). The platform may or may not have a hierarchical architecture (clusters of clusters vs. a single cluster), it may or may not be dedicated. Resources may be added to or may disappear from the platform at any time, or the platform may have a stable composition. The processing units may have the same characteristics (e.g., computational power, amount of memory, multi-port or only single-port communications support, etc.) or not. The communication links may have the same characteristics (e.g., bandwidths, latency, routing policy, etc.) or not. The aim of the scheduling policy can be to minimize the
overall execution time (makespan minimization), the throughput of processed tasks, etc. Finally, the set of all tasks to be scheduled may be known from the beginning, or new tasks may arrive all along the execution of the system (on-line scheduling).

In the GRAAL project, we investigate scheduling problems that are of practical interest in the context of large-scale distributed platforms. We assess the impact of the heterogeneity and volatility of the resources onto the scheduling strategies.

### 3.2. Scheduling for Parallel Sparse Direct Solvers and Combinatorial Scientific Computing

**Participants:** Guillaume Joslin, Maurice Brémond, Johannes Langguth, Jean-Yves L’Excellent, Bora Uçar, Mohamed Sid-Lakhdir.

The solution of sparse systems of linear equations (symmetric or unsymmetric, often with an irregular structure) is at the heart of many scientific applications arising in various domains such as geophysics, chemistry, electromagnetism, structural optimization, and computational fluid dynamics. The importance and diversity of the fields of applications are our main motivation to pursue research on sparse linear solvers. Furthermore, in order to solve hard problems that result from ever-increasing demand for accuracy in simulations, special attention must be paid to both memory usage and execution time on the most powerful parallel platforms (whose usage is necessary because of the volume of data and amount of computation required). This is done by specific algorithmic choices and scheduling techniques. From a complementary point of view, it is also necessary to be aware of the functionality requirements from the applications and from the users, so that robust solutions can be proposed for a large range of problems.

Because of their efficiency and robustness, direct methods (based on Gaussian elimination) are methods of choice to solve these types of problems. In this context, we are particularly interested in the multifrontal method [88], [89] for symmetric positive definite, general symmetric or unsymmetric problems, with numerical pivoting in order to ensure numerical accuracy. The existence of numerical pivoting induces dynamic updates in the data structures where the updates are not predictable with a static or symbolic analysis approach.

The multifrontal method is based on an elimination tree [92] which results (i) from the graph structure corresponding to the nonzero pattern of the problem to be solved, and (ii) from the order in which variables are eliminated. This tree provides the dependency graph of the computations and is exploited to define tasks that may be executed in parallel. In the multifrontal method, each node of the tree corresponds to a task (itself can be potentially parallel) that consists in the partial factorization of a dense matrix. This approach allows for a good locality and hence efficient use of cache memories.

We are especially interested in approaches that are intrinsically dynamic and asynchronous [1], [83], as these approaches can encapsulate numerical pivoting and can be adopted to various computer architectures. In addition to their numerical robustness, the algorithms are based on a dynamic and distributed management of the computational tasks, not so far from today’s peer-to-peer approaches: each process is responsible for providing work to some other processes and at the same time it acts as a worker for others. These algorithms are very interesting from the point of view of parallelism and in particular for the study of mapping and scheduling strategies for the following reasons:

- the associated task graphs are very irregular and can vary dynamically,
- they are currently used inside industrial applications, and
- the evolution of high performance platforms, to the more heterogeneous and less predictable ones, requires that applications adapt themselves, using a mixture of dynamic and static approaches, as our approach allows.
Our research in this field is strongly linked to the software package MUMPS (see Section 5.2) which is our main platform to experiment and validate new ideas and pursue new research directions. We are facing new challenges for very large problems (tens to hundreds of millions of equations) that occur nowadays in various application fields. The evolution of architectures towards clusters of multicore nodes and more and more parallelism is also a challenge that we are forced to face.

There are strong links between sparse direct methods and combinatorial scientific computing, which is more general. The aim of combinatorial scientific computing is to design combinatorial algorithms whose usage reduces the amount of resources needed for the solution of a target problem arising in scientific computing. The general approach is to identify issues that affect the performance of a scientific computing application (such as the memory use, the parallel speed up, etc.) and to develop combinatorial models and related algorithms to alleviate the issue. Our target scientific computing applications are the preprocessing phases of direct (in particular MUMPS), iterative, and hybrid methods for solving linear systems of equations, and the mapping of tasks (mostly the sub-tasks of such solvers) onto modern computing platforms. We will focus on the development and use of graph and hypergraph models, and related tools such as hypergraph partitioning, for load balancing and task mapping for parallel efficiency; and bipartite graph matching and vertex ordering for reducing the memory overhead and computational requirements of solvers. Although we direct our attention on these models and algorithms through the lens of linear system solvers, they are general enough to be applied to some other resource optimization problems.

3.3. Algorithms and Software Architectures for Service Oriented Platforms

Participants: Daniel Balouek, Nicolas Bard, Julien Bigot, Yves Caniou, Eddy Caron, Florent Chuffart, Simon Delamare, Frédéric Desprez, Gilles Fedak, Sylvain Gault, Haiwu He, Cristian Klein, Georges Markomanolis, Adrian Muresan, Christian Pérez, Vincent Pichon, Jonathan Rouzaud-Cornabas, Anthony Simonet, José Francisco Saray Villamizar, Bing Tang.

The fast evolution of hardware capabilities in terms of wide area communication as well as of machine virtualization leads to the requirement of another step in the abstraction of resources with respect to applications. Those large scale platforms based on the aggregation of large clusters (Grids), supercomputers, huge datacenters (Clouds) or collections of volunteer PCs (Desktop computing platforms) are now available for researchers of different fields of science as well as private companies. This variety of platforms and the way they are accessed have also an important impact on how applications are designed (i.e., the programming model used) as well as how applications are executed (i.e., the runtime/middleware system used). The access to these platforms is driven through the use of different services providing mandatory features such as security, resource discovery, virtualization, load-balancing, etc. Software as a Service (SaaS) has thus to play an important role in the future development of large scale applications. The overall idea is to consider the whole system, ranging from the resources to the application, as a set of services. Hence, a user application is an ordered set of instructions requiring and making uses of some services like for example an execution service. Such a service is also an application—but at the middleware level—that is proposing some services (here used by the user application) and potentially using other services like for example a scheduling service. This model based on services provided and/or offered is generalized within software component models which deal with composition issues as well as with deployment issues.

Our goal is to contribute to the design of programming models supporting a wide range of architectures and to their implementation by mastering the various algorithmic issues involved and by studying the impact on application-level algorithms. Ideally, an application should be written once; the complexity is to determine the adequate level of abstraction to provide a simple programming model to the developer while enabling efficient execution on a wide range of architectures. To achieve such a goal, the team plans to contribute at different level including programming models, distributed algorithms, deployment of services, services discovery, service composition and orchestration, large scale data management, etc.
3. Scientific Foundations

3.1. Large Scale Distributed Systems (LSDS)

What makes a fundamental difference between recent Global Computing systems (Seti@home), Grid (EGEE, TeraGrid) and former works on distributed systems is the large scale of these systems. This characteristic becomes also true for large scale parallel computers gathering tens of thousands of CPU cores. The notion of Large Scale is linked to a set of features that has to be taken into account in these systems. An example is the system dynamicity caused by node volatility: in Internet Computing Platforms (also called Desktop Grids), a non predictable number of nodes may leave the system at any time. Some recent results also report a very low MTTI (Mean Time To Interrupt) in top level supercomputers gathering 100,000+ CPU cores. Another example of characteristics is the complete lack of control of nodes connectivity. In Desktop Grid, we cannot assume that external administrator is able to intervene in the network setting of the nodes, especially their connection to Internet via NAT and Firewalls. This means that we have to deal with the in place infrastructure in terms of performance, heterogeneity, dynamicity and connectivity. These characteristics, associated with the requirement of scalability, establish a new research context in distributed systems. The Grand-Large project aims at investigating theoretically as well as experimentally the fundamental mechanisms of LSDS, especially for the high performance computing applications.

3.1.1. Computing on Large Scale Global Computing systems

Large scale parallel and distributed systems are mainly used in the context of Internet Computing. As a consequence, until Sept. 2007, Grand-Large has focused mainly on Desktop Grids. Desktop Grids are developed for computing (SETI@home, Folding@home, Decrypthon, etc.), file exchanges (Napster, Kazaa, eDonkey, Gnutella, etc.), networking experiments (PlanetLab, Porivo) and communications such as instant messaging and phone over IP (Jabber, Skype). In the High Performance Computing domain, LSDS have emerged while the community was considering clustering and hierarchical designs as good performance-cost tradeoffs. Nowadays, Internet Computing systems are still very popular (the BOINC platform is used to run over 40 Internet Computing projects and XtremWeb is used in production in three countries) and still raise important research issues.

Desktop Grid systems essentially extend the notion of computing beyond the frontier of administration domains. The very first paper discussing this type of systems [94] presented the Worm programs and several key ideas that are currently investigated in autonomous computing (self replication, migration, distributed coordination, etc.). LSDS inherit the principle of aggregating inexpensive, often already in place, resources, from past research in cycle stealing/resource sharing. Due to its high attractiveness, cycle stealing has been studied in many research projects like Condor [83], Glunix [76] and Mosix [55], to cite a few. A first approach to cross administration domains was proposed by Web Computing projects such as Jet [87], Charlotte [56], Javeline [70], Bayanian [92], SuperWeb [51], ParaWeb [63] and PopCorn [65]. These projects have emerged with Java, taking benefit of the virtual machine properties: high portability across heterogeneous hardware and OS, large diffusion of virtual machine in Web browsers and a strong security model associated with bytecode execution. Performance and functionality limitations are some of the fundamental motivations of the second generation of Global Computing systems like BOINC [53] and XtremWeb [72]. The second generation of Global Computing systems appeared in the form of generic middleware which allow scientists and programmers to design and set up their own distributed computing project. As a result, we have seen the emergence of large communities of volunteers and projects. Currently, Global Computing systems are among the largest distributed systems in the world. In the mean time, several studies succeeded to understand and enhance the performance of these systems, by characterizing the system resources in term of volatility and heterogeneity and by studying new scheduling heuristics to support new classes of applications: data-intensive, long running application with checkpoint, workflow, soft-real time
etc... However, despite these recent progresses, one can note that Global Computing systems are not yet part of high performance solution, commonly used by scientists. Recent researches to fulfill the requirements of Desktop Grids for high demanding users aim at redesigning Desktop Grid middleware by essentially turning a set of volatile nodes into a virtual cluster and allowing the deployment of regular HPC utilities (batch schedulers, parallel communication libraries, checkpoint services, etc...) on top of this virtual cluster. The new generation would permit a better integration in the environment of the scientists such as computational Grids, and consequently, would broaden the usage of Desktop Grid.

The high performance potential of LSDS platforms has also raised a significant interest in the industry. Performance demanding users are also interested by these platforms, considering their cost-performance ratio which is even lower than the one of clusters. Thus, several Desktop Grid platforms are daily used in production in large companies in the domains of pharmacology, petroleum, aerospace, etc.

Desktop Grids share with Grid a common objective: to extend the size and accessibility of a computing infrastructure beyond the limit of a single administration domain. In [73], the authors present the similarities and differences between Grid and Global Computing systems. Two important distinguishing parameters are the user community (professional or not) and the resource ownership (who own the resources and who is using them). From the system architecture perspective, we consider two main differences: the system scale and the lack of control of the participating resources. These two aspects have many consequences, at least on the architecture of system components, the deployment methods, programming models, security (trust) and more generally on the theoretical properties achievable by the system.

Beside Desktop Grids and Grids, large scale parallel computers with tens of thousands (and even hundreds of thousands) of CPU cores are emerging with scalability issues similar to the one of Internet Computing systems: fault tolerance at large scale, large scale data movements, tools and languages. Grand-Large is gradually considering the application of selected research results, in the domain of large scale parallel computers, in particular for the fault tolerance and language topics.

3.1.2. Building a Large Scale Distributed System

This set of studies considers the XtremWeb project as the basis for research, development and experimentation. This LSDS middleware is already operational. This set gathers 4 studies aiming at improving the mechanisms and enlarging the functionalities of LSDS dedicated to computing. The first study considers the architecture of the resource discovery engine which, in principle, is close to an indexing system. The second study concerns the storage and movements of data between the participants of a LSDS. The third study concerns the issue of scheduling in LSDS in the context of multiple users and applications. Finally the last study seeks to improve the performance and reduce the resource cost of the MPICH-V fault tolerant MPI for desktop grids.

3.1.2.1. The resource discovery engine

A multi-users/multi-applications LSDS for computing would be in principle very close to a P2P file sharing system such as Napster [93], Gnutella [93] and Kazaa [82], except that the shared resource is the CPUs instead of files. The scale and lack of control are common features of the two kinds of systems. Thus, it is likely that solutions sharing fundamental mechanisms will be adopted, such as lower level communication protocols, resource publishing, resource discovery and distributed coordination. As an example, recent P2P projects have proposed distributed indexing systems like CAN [90], CHORD [95], PASTRY [91] and TAPESTRY [100] that could be used for resource discovery in a LSDS dedicated to computing.

The resource discovery engine is composed of a publishing system and a discovery engine, which allow a client of the system to discover the participating nodes offering some desired services. Currently, there is as much resource discovery architectures as LSDS and P2P systems. The architecture of a resource discovery engine is derived from some expected features such as speed of research, speed of reconfiguration, volatility tolerance, anonymity, limited use of the network, matching between the topologies of the underlying network and the virtual overlay network.
This study focuses on the first objective: to build a highly reliable and stable overlay network supporting the higher level services. The overlay network must be robust enough to survive unexpected behaviors (like malicious behaviors) or failures of the underlying network. Unfortunately it is well known that under specific assumptions, a system cannot solve even simples tasks with malicious participants. So, we focus the study on designing overlay algorithms for transient failures. A transient failure accepts any kind of behavior from the system, for a limited time. When failures stop, the system will eventually provide its normal service again.

A traditional way to cope with transient failures are self-stabilizing systems [71]. Existing self-stabilizing algorithms use an underlying network that is not compatible with LSDS. They assume that processors know their list of neighbors, which does not fit the P2P requirements. Our work proposes a new model for designing self-stabilizing algorithms without making this assumption, then we design, prove and evaluate overlay networks self-stabilizing algorithms in this model.

3.1.2.2. Fault Tolerant MPI

MPICH-V is a research effort with theoretical studies, experimental evaluations and pragmatic implementations aiming to provide a MPI implementation based on MPICH [85], featuring multiple fault tolerant protocols.

There is a long history of research in fault tolerance for distributed systems. We can distinguish the automatic/transparent approach from the manual/user controlled approach. The first approach relies either on coordinated checkpointing (global snapshot) or uncoordinated checkpointing associated with message logging. A well known algorithm for the first approach has been proposed by Chandy and Lamport [67]. This algorithm requires restarting all processes even if only one process crashes. So it is believed not to scale well. Several strategies have been proposed for message logging: optimistic [97], pessimistic [52], causal [98]. Several optimizations have been studied for the three strategies. The general context of our study is high performance computing on large platforms. One of the most used programming environments for such platforms is MPI.

Within the MPICH-V project, we have developed and published several original fault tolerant protocols for MPI: MPICH-V1 [60], MPICH-V2 [61], MPICH-Vcausal, MPICH-Vcl [62], MPICH-Pcl. The two first protocols rely on uncoordinated checkpointing associated with either remote pessimistic message logging or sender based pessimistic message logging. We have demonstrated that MPICH-V2 outperforms MPICH-V1. MPICH-Vcl implements a coordinated checkpoint strategy (Chandy-Lamport) removing the need of message logging. MPICH-V2 and Vcl are concurrent protocols for large clusters. We have compared them considering a new parameter for evaluating the merits of fault tolerant protocols: the impact of the fault frequency on the performance. We have demonstrated that the stress of the checkpoint server is the fundamental source of performance differences between the two techniques. MPICH-Vcausal implements a causal message logging protocols, removing the need for waiting acknowledgement in contrary to MPICH-V2. MPICH-Pcl is a blocking implementation of the Vcl protocol. Under the considered experimental conditions, message logging becomes more relevant than coordinated checkpoint when the fault frequency reaches 1 fault every 4 hours, for a cluster of 100 nodes sharing a single checkpoint server, considering a data set of 1 GB on each node and a 100 Mb/s network.

Multiple important events arose from this research topic. A new open source implementation of the MPI-2 standard was born during the evolution of the MPICH-V project, namely OpenMPI. OpenMPI is the result of the alliance of many MPI projects in the USA, and we are working to port our fault tolerance algorithms both into OpenMPI and MPICH.

Grids becoming more popular and accessible than ever, parallel applications developers now consider them as possible targets for computing demanding applications. MPI being the de-facto standard for the programming of parallel applications, many projects of MPI for the Grid appeared these last years. We contribute to this new way of using MPI through a European Project in which we intend to grid-enable OpenMPI and provide new fault-tolerance approaches fitted for the grid.

When introducing Fault-Tolerance in MPI libraries, one of the most neglected component is the runtime environment. Indeed, the traditional approach consists in restarting the whole application and runtime environment
in case of failure. A more efficient approach could be to implement a fault-tolerant runtime environment, capable of coping with failures at its level, thus avoiding the restart of this part of the application. The benefits would be a quicker restart time, and a better control of the application. However, in order to build a fault-tolerant runtime environment for MPI, new topologies, more connected, and more stable, must be integrated in the runtime environment.

For traditional parallel machines of large scale (like large scale clusters), we also continue our investigation of the various fault tolerance protocols, by designing, implementing and evaluating new protocols in the MPICH-V project.

### 3.2. Volatility and Reliability Processing

In a global computing application, users voluntarily lend the machines, during the period they don’t use them. When they want to reuse the machines, it is essential to give them back immediately. We assume that there is no time for saving the state of the computation (for example because the user is shooting down his machine). Because the computer may not be available again, it is necessary to organize checkpoints. When the owner takes control of his machine, one must be able to continue the computation on another computer from a checkpoint as near as possible from the interrupted state.

The problems raised by this way of managing computations are numerous and difficult. They can be put into two categories: synchronization and repartition problems.

- **Synchronization problems (example).** Assume that the machine that is supposed to continue the computation is fixed and has a recent checkpoint. It would be easy to consider that this local checkpoint is a component of a global checkpoint and to simply rerun the computation. But on one hand the scalability and on the other hand the frequency of disconnections make the use of a global checkpoint totally unrealistic. Then the checkpoints have to be local and the problem of synchronizing the recovery machine with the application is raised.

- **Repartition problems (example).** As it is also unrealistic to wait for the computer to be available again before rerunning the interrupted application, one has to design a virtual machine organization, where a single virtual machine is implemented as several real ones. With too few real machines for a virtual one, one can produce starvation; with too many, the efficiency is not optimal. The good solution is certainly in a dynamic organization.

These types of problems are not new ([74]). They have been studied deeply and many algorithmic solutions and implementations are available. What is new here and makes these old solutions not usable is scalability. Any solution involving centralization is impossible to use in practice. Previous works validated on former networks can not be reused.

### 3.2.1. Reliability Processing

We voluntarily presented in a separate section the volatility problem because of its specificity both with respect to type of failures and to frequency of failures. But in a general manner, as any distributed system, a global computing system has to resist to a large set of failures, from crash failures to Byzantine failures, that are related to incorrect software or even malicious actions (unfortunately, this hypothesis has to be considered as shown by DECRYPTTHON project or the use of erroneous clients in SETI@HOME project), with in between, transient failures such as loss of message duplication. On the other hand, failures related accidental or malicious memory corruptions have to be considered because they are directly related to the very nature of the Internet. Traditionally, two approaches (masking and non-masking) have been used to deal with reliability problems. A masking solution hides the failures to the user, while a non-masking one may let the user notice that failures occur. Here again, there exists a large literature on the subject (cf. [84], [96], [71] for surveys). Masking techniques, generally based on consensus, are not scalable because they systematically use generalized broadcasting. The self-stabilizing approach (a non-masking solution) is well adapted (specifically its time adaptive version, cf. [81], [80], [57], [58], [75]) for three main reasons:
1. Low overhead when stabilized. Once the system is stabilized, the overhead for maintaining correction is low because it only involves communications between neighbours.

2. Good adaptivity to the reliability level. Except when considering a system that is continuously under attacks, self-stabilization provides very satisfying solutions. The fact that during the stabilization phase, the correctness of the system is not necessarily satisfied is not a problem for many kinds of applications.

3. Lack of global administration of the system. A peer to peer system does not admit a centralized administrator that would be recognized by all components. A human intervention is thus not feasible and the system has to recover by itself from the failures of one or several components, that is precisely the feature of self-stabilizing systems.

We propose:

1. To study the reliability problems arising from a global computing system, and to design self-stabilizing solutions, with a special care for the overhead.

2. For problem that can be solved despite continuously unreliable environment (such as information retrieval in a network), to propose solutions that minimize the overhead in space and time resulting from the failures when they involve few components of the system.

3. For most critical modules, to study the possibility to use consensus based methods.

4. To build an adequate model for dealing with the trade-off between reliability and cost.

3.3. Parallel Programming on Peer-to-Peer Platforms (P5)

Several scientific applications, traditionally computed on classical parallel supercomputers, may now be adapted for geographically distributed heterogeneous resources. Large scale P2P systems are alternative computing facilities to solve grand challenge applications.

Peer-to-Peer computing paradigm for large scale scientific and engineering applications is emerging as a new potential solution for end-user scientists and engineers. We have to experiment and to evaluate such programming to be able to propose the larger possible virtualization of the underlying complexity for the end-user.

3.3.1. Large Scale Computational Sciences and Engineering

Parallel and distributed scientific application developments and resource managements in these environments are a new and complex undertaking. In scientific computation, the validity of calculations, the numerical stability, the choices of methods and software are depending of properties of each peer and its software and hardware environments; which are known only at run time and are non-deterministic. The research to obtain acceptable frameworks, methodologies, languages and tools to allow end-users to solve accurately their applications in this context is capital for the future of this programming paradigm.

GRID scientific and engineering computing exists already since more than a decade. Since the last few years, the scale of the problem sizes and the global complexity of the applications increase rapidly. The scientific simulation approach is now general in many scientific domains, in addition to theoretical and experimental aspects, often link to more classic methods. Several applications would be computed on world-spread networks of heterogeneous computers using some web-based Application Server Provider (ASP) dedicated to targeted scientific domains. New very strategic domains, such as Nanotechnologies, Climatology or Life Sciences, are in the forefront of these applications. The development in this very important domain and the leadership in many scientific domains will depend in a close future to the ability to experiment very large scale simulation on adequate systems [ 79 ]. The P2P scientific programming is a potential solution, which is based on existing computers and networks. The present scientific applications on such systems are only concerning problems which are mainly data independents: i.e. each peer does not communicate with the others.
P2P programming has to develop parallel programming paradigms which allow more complex dependencies between computing resources. This challenge is an important goal to be able to solve large scientific applications. The results would also be extrapolated toward future petascale heterogeneous hierarchically designed supercomputers.

3.3.2. Experimentations and Evaluations

We have followed two tracks. First, we did experiments on large P2P platforms in order to obtain a realistic evaluation of the performance we can expect. Second, we have set some hypothesis on peers, networks, and scheduling in order to have theoretical evaluations of the potential performance. Then, we have chosen a classical linear algebra method well-adapted to large granularity parallelism and asynchronous scheduling: the block Gauss-Jordan method to invert dense very large matrices. We have also chosen the calculation of one matrix polynomial, which generates computation schemes similar to many linear algebra iterative methods, well-adapted for very large sparse matrices. Thus, we were able to theoretically evaluate the potential throughput with respect to several parameters such as the matrix size and the multicast network speed.

Since the beginning of the evaluations, we experimented with those parallel methods on a few dozen peer XtremWeb P2P Platforms. We continue these experiments on larger platforms in order to compare these results to the theoretical ones. Then, we would be able to extrapolate and obtain potential performance for some scientific applications.

Recently, we also experimented several Krylov based method, such as the Lanczos and GMRES methods on several grids, such as a French-Japanese grid using hundred of PC in France and 4 clusters at the University of Tsukuba. We also experimented on GRID5000 the same methods. We currently use several middleware such as Xtremweb, OmniRPC and Condor. We also begin some experimentations on the Tsubame supercomputer in collaboration with the TITech (Tokyo Institute of Technologies) in order to compare our grid approaches and the High performance one on an hybrid supercomputer.

Experimentations and evaluation for several linear algebra methods for large matrices on P2P systems will always be developed all along the Grand Large project, to be able to confront the different results to the reality of the existing platforms.

As a challenge, we would like, in several months, to efficiently invert a dense matrix of size one million using a several thousand peer platform. We are already inverting very large dense matrices on Grid5000 but more efficient scheduler and a larger number of processors are required to this challenge.

Beyond the experimentations and the evaluations, we propose the basis of a methodology to efficiently program such platforms, which allow us to define languages, tools and interface for the end-user.

3.3.3. Languages, Tools and Interface

The underlying complexity of the Large Scale P2P programming has to be mainly virtualized for the end-user. We have to propose an interface between the end-user and the middleware which may extract the end-user expertise or propose an on-the-shelf general solution. Targeted applications concern very large scientific problems which have to be developed using component technologies and up-to-dated software technologies.

We introduced the YML framework and language which allows to describe dependencies between components. We introduced different classes of components, depending of the level of abstraction, which are associated with divers parts of the framework. A component catalogue is managed by an administrator and/or the end-users. Another catalogue is managed with respect to the experimental platform and the middleware criteria. A front-end part is completely independent of any middleware or testbed, and a back-end part is developed for each targeted middleware/platform couple. A YML scheduler is adapted for each of the targeted systems.

The YML framework and language propose a solution to develop scientific applications to P2P and GRID platform. An end-user can directly develop programs using this framework. Nevertheless, many end-users would prefer avoid programming at the component and dependency graph level. Then, an interface has to be proposed soon, using the YML framework. This interface may be dedicated to a special scientific domain to be able to focus on the end-user vocabulary and P2P programming knowledge. We plan to develop such
version based on the YML framework and language. The first targeted scientific domain will be very large linear algebra for dense or sparse matrices.

3.4. Methodology for Large Scale Distributed Systems

Research in the context of LSDS involves understanding large scale phenomena from the theoretical point of view up to the experimental one under real life conditions.

One key aspects of the impact of large scale on LSDS is the emergence of phenomena which are not co-ordinated, intended or expected. These phenomena are the results of the combination of static and dynamic features of each component of LSDS: nodes (hardware, OS, workload, volatility), network (topology, congestion, fault), applications (algorithm, parameters, errors), users (behavior, number, friendly/aggressive).

Validating current and next generation of distributed systems targeting large-scale infrastructures is a complex task. Several methodologies are possible. However, experimental evaluations on real testbeds are unavoidable in the life-cycle of a distributed middleware prototype. In particular, performing such real experiments in a rigorous way requires to benchmark developed prototypes at larger and larger scales. Fulfilling this requirement is mandatory in order to fully observe and understand the behaviors of distributed systems. Such evaluations are indeed mandatory to validate (or not!) proposed models of these distributed systems, as well as to elaborate new models. Therefore, to enable an experimentally-driven approach for the design of next generation of large scale distributed systems, developing appropriate evaluation tools is an open challenge.

Fundamental aspects of LSDS as well as the development of middleware platforms are already existing in Grand-Large. Grand-Large aims at gathering several complementary techniques to study the impact of large scale in LSDS: observation tools, simulation, emulation and experimentation on real platforms.

3.4.1. Observation tools

Observation tools are mandatory to understand and extract the main influencing characteristics of a distributed system, especially at large scale. Observation tools produce data helping the design of many key mechanisms in a distributed system: fault tolerance, scheduling, etc. We pursue the objective of developing and deploying a large scale observation tool (XtremLab) capturing the behavior of thousands of nodes participating to popular Desktop Grid projects. The collected data will be stored, analyzed and used as reference in a simulator (SIMBOINC).

3.4.2. Tool for scalability evaluations

Several Grid and P2P systems simulators have been developed by other teams: SimGrid [66], GridSim [64], Briks [50]. All these simulators considers relatively small scale Grids. They have not been designed to scale and simulate 10 K to 100 K nodes. Other simulators have been designed for large multi-agents systems such as Swarm [86] but many of them considers synchronous systems where the system evolution is guided by phases. In the P2P field, ad hoc many simulators have been developed, mainly for routing in DHT. Emulation is another tool for experimenting systems and networks with a higher degree of realism. Compared to simulation, emulation can be used to study systems or networks 1 or 2 orders of magnitude smaller in terms of number of components. However, emulation runs the actual OS/middleware/applications on actual platform. Compared to real testbed, emulation considers conducting the experiments on a fully controlled platform where all static and dynamic parameters can be controlled and managed precisely. Another advantage of emulation over real testbed is the capacity to reproduce experimental conditions. Several implementations/configurations of the system components can be compared fairly by evaluating them under the similar static and dynamic conditions. Grand-Large is leading one of the largest Emulator project in Europe called Grid explorer (French funding). This project has built and used a 1K CPUs cluster as hardware platform and gathers 24 experiments of 80 researchers belonging to 13 different laboratories. Experiments concerned developing the emulator itself and use of the emulator to explore LSDS issues. In term of emulation tool, the main outcome of Grid explorer is the V-DS system, using virtualization techniques to fold a virtual distributed system 50 times larger than the actual execution platform. V-DS aims at discovering, understanding and managing implicit uncoordinated large scale phenomena. Grid Explorer is still in use within the Grid’5000 platform and serves the community of 400 users 7 days a week and 24h a day.
3.4.3. Real life testbeds: extreme realism

The study of actual performance and connectivity mechanisms of Desktop Grids needs some particular testbed where actual middleware and applications can be run under real scale and real life conditions. Grand-Large is developing DSL-Lab, an experimental platform distributed on 50 sites (actual home of the participants) and using the actual DSL network as the connection between the nodes. Running experiments over DSL-Lab put the piece of software to study under extremely realistic conditions in terms of connectivity (NAT, Firewalls), performance (node and network), performance symmetry (DSL Network is not symmetric), etc.

To investigate real distributed system at large scale (Grids, Desktop Grids, P2P systems), under real life conditions, only a real platform (featuring several thousands of nodes), running the actual distributed system can provide enough details to clearly understand the performance and technical limits of a piece of software. Grand-Large members are strongly involved (as Project Director) in the French Grid5000 project which intents to deploy an experimental Grid testbed for computer scientists. This testbed features about 4000 CPUs gathering the resources of about 9 clusters geographically distributed over France. The clusters will be connected by a high speed network (Renater 10G). Grand-Large is the leading team in Grid5000, chairing the steering committee. As the Principal Investigator of the project, Grand-Large has taken some strong design decisions that nowadays give a real added value of Grid5000 compared to all other existing Grids: reconfiguration and isolation. From these two features, Grid5000 provides the capability to reproduce experimental conditions and thus experimental results, which is the cornerstone of any scientific instrument.

3.5. High Performance Scientific Computing

This research is in the area of high performance scientific computing, and in particular in parallel matrix algorithms. This is a subject of crucial importance for numerical simulations as well as other scientific and industrial applications, in which linear algebra problems arise frequently. The modern numerical simulations coupled with ever growing and more powerful computational platforms have been a major driving force behind a progress in numerous areas as different as fundamental science, technical/technological applications, life sciences.

The main focus of this research is on the design of efficient, portable linear algebra algorithms, such that solving a large set of linear equations or a least squares problem. The characteristics of the matrices commonly encountered in this situations can vary significantly, as are the computational platforms used for the calculations. Nonetheless two common trends are easily discernible. First, the problems to solve are larger and larger, since the numerical simulations are using higher resolution. Second, the architecture of today’s supercomputers is getting very complex, and so the developed algorithms need to be adapted to these new architectures.

3.5.1. Communication avoiding algorithms for numerical linear algebra

Since 2007, we work on a novel approach to dense and sparse linear algebra algorithms, which aims at minimizing the communication, in terms of both its volume and a number of transferred messages. This research is motivated by technological trends showing an increasing communication cost. Its main goal is to reformulate and redesign linear algebra algorithms so that they are optimal in an amount of the communication they perform, while retaining the numerical stability. The work here involves both theoretical investigation and practical coding on diverse computational platforms. We refer to the new algorithms as communication avoiding algorithms [6], [9]. In our team we focus on communication avoiding algorithms for dense direct methods as well as sparse iterative methods.

The theoretical investigation focuses on identifying lower bounds on communication for different operations in linear algebra, where communication refers to data movement between processors in the parallel case, and to data movement between different levels of memory hierarchy in the sequential case. The lower bounds are used to study the existing algorithms, understand their communication bottlenecks, and design new algorithms that attain them.
This research focuses on the design of linear algebra algorithms that minimize the cost of communication. Communication costs include both latency and bandwidth, whether between processors on a parallel computer or between memory hierarchy levels on a sequential machine. The stability of the new algorithms represents an important part of this work.

3.5.2. Preconditioning techniques

Solving a sparse linear system of equations is the most time consuming operation at the heart of many scientific applications, and therefore it has received a lot of attention over the years. While direct methods are robust, they are often prohibitive because of their time and memory requirements. Iterative methods are widely used because of their limited memory requirements, but they need an efficient preconditioner to accelerate their convergence. In this direction of research we focus on preconditioning techniques for solving large sparse systems.

One of the main challenges that we address is the scalability of existing methods as incomplete LU factorizations or Schwarz-based approaches, for which the number of iterations increases significantly with the problem size or with the number of processors. This is often due to the presence of several low frequency modes that hinder the convergence of the iterative method. To address this problem, we study direction preserving solvers in the context of multilevel filtering LU decompositions. A judicious choice for the directions to be preserved through filtering allows us to alleviate the effect of low frequency modes on the convergence. While preconditioners and their scalability are studied by many other groups, our approach of direction preserving and filtering is studied in only very few other groups in the world (as Lawrence Livermore National Laboratory, Frankfurt University, Pennsylvania State University).

3.5.3. Fast linear algebra solvers based on randomization

Linear algebra calculations can be enhanced by statistical techniques in the case of a square linear system $Ax = b$ where $A$ is a general or symmetric indefinite matrix \cite{54} & \cite{1}. Thanks to a random transformation of $A$, it is possible to avoid pivoting and then to reduce the amount of communication. Numerical experiments show that this randomization can be performed at a very affordable computational price while providing us with a satisfying accuracy when compared to partial pivoting. This random transformation called Partial Random Butterfly Transformation (PRBT) is optimized in terms of data storage and flops count. A PRBT solver for LU factorization (and for $LDL^T$ factorization on multicore) has been developed. This solver takes advantage of the latest generation of hybrid multicore/GPU machines and gives better Gflop/s performance than existing factorization routines.

3.5.4. Sensitivity analysis of linear algebra problems

We derive closed formulas for the condition number of a linear function of the total least squares solution \cite{2}. Given an over determined linear systems $Ax = b$, we show that this condition number can be computed using the singular values and the right singular vectors of $[A, b]$ and $A$. We also provide an upper bound that requires the computation of the largest and the smallest singular value of $[A, b]$ and the smallest singular value of $A$. In numerical experiments, we compare these values with condition estimates from the literature.
3. Scientific Foundations

3.1. Introduction

The methodological component of HiePACS concerns the expertise for the design as well as the efficient and scalable implementation of highly parallel numerical algorithms to perform frontier simulations. In order to address these computational challenges a hierarchical organization of the research is considered. In this bottom-up approach, we first consider in Section 3.2 generic topics concerning high performance computational science. The activities described in this section are transversal to the overall project and its outcome will support all the other research activities at various levels in order to ensure the parallel scalability of the algorithms. The aim of this activity is not to study general purpose solution but rather to address these problems in close relation with specialists of the field in order to adapt and tune advanced approaches in our algorithmic designs. The next activity, described in Section 3.3, is related to the study of parallel linear algebra techniques that currently appear as promising approaches to tackle huge problems on millions of cores. We highlight the linear problems (linear systems or eigenproblems) because they are in many large scale applications the main computational intensive numerical kernels and often the main performance bottleneck. These parallel numerical techniques will be the basis of both academic and industrial collaborations described in Section 4.2 and Section 4.3, but will also be closely related to some functionalities developed in the parallel fast multipole activity described in Section 3.4. Finally, as the accuracy of the physical models increases, there is a real need to go for parallel efficient algorithm implementation for multiphysics and multiscale modelling in particular in the context of code coupling. The challenges associated with this activity will be addressed in the framework of the activity described in Section 3.5.

3.2. High-performance computing on next generation architectures


The research directions proposed in HiePACS are strongly influenced by both the applications we are studying and the architectures that we target (i.e., massively parallel architectures, ...). Our main goal is to study the methodology needed to efficiently exploit the new generation of high-performance computers with all the constraints that it induces. To achieve this high-performance with complex applications we have to study both algorithmic problems and the impact of the architectures on the algorithm design. From the application point of view, the project will be interested in multiresolution, multiscale and hierarchical approaches which lead to multi-level parallelism schemes. This hierarchical parallelism approach is necessary to achieve good performance and high-scalability on modern massively parallel platforms. In this context, more specific algorithmic problems are very important to obtain high performance. Indeed, the kind of applications we are interested in are often based on data redistribution for example (e.g. code coupling applications). This well-known issue becomes very challenging with the increase of both the number of computational nodes and the amount of data. Thus, we have both to study new algorithms and to adapt the existing ones. In addition, some issues like task scheduling have to be restudied in this new context. It is important to note that the work done in this area will be applied for example in the context of code coupling (see Section 3.5).
Considering the complexity of modern architectures like massively parallel architectures (i.e., Blue Gene-like platforms) or new generation heterogeneous multicore architectures, task scheduling becomes a challenging problem which is central to obtain a high efficiency. Of course, this work requires the use/design of scheduling algorithms and models specifically to tackle our target problem. This has to be done in collaboration with our colleagues in the scheduling community like for example O. Beaumont (Inria CEPAGE Project-Team). It is important to note that this topic is strongly linked to the underlying programming model. Indeed, considering multicore architectures, it has appeared, in the last five years, that the best programming model is an approach mixing multi-threading within computational nodes and message passing between them. In the last five years, a lot of work has been developed in the high-performance computing community to understand what is critical to efficiently exploit massively multicore platforms that will appear in the near future. It appeared that the key for the performance is firstly the grain of computations. Indeed, in such platforms the grain of the parallelism must be small so that we can feed all the processors with a sufficient amount of work. It is thus very crucial for us to design new high performance tools for scientific computing in this new context. This will be done in the context of our solvers, for example, to adapt to this new parallel scheme. Secondly, the larger the number of cores inside a node, the more complex the memory hierarchy. This remark impacts the behaviour of the algorithms within the node. Indeed, on this kind of platforms, NUMA effects will be more and more problematic. Thus, it is very important to study and design data-aware algorithms which take into account the affinity between computational threads and the data they access. This is particularly important in the context of our high-performance tools. Note that this work has to be based on an intelligent cooperative underlying run-time (like the marcel thread library developed by the Inria RUNTIME Project-Team) which allows a fine management of data distribution within a node.

Another very important issue concerns high-performance computing using “heterogeneous” resources within a computational node. Indeed, with the emergence of the GPU and the use of more specific co-processors (like clearspeed cards, ...), it is important for our algorithms to efficiently exploit these new kind of architectures. To adapt our algorithms and tools to these accelerators, we need to identify what can be done on the GPU for example and what cannot. Note that recent results in the field have shown the interest of using both regular cores and GPU to perform computations. Note also that in opposition to the case of the parallelism granularity needed by regular multicore architectures, GPU requires coarser grain parallelism. Thus, making both GPU and regular cores work all together will lead to two types of tasks in terms of granularity. This represents a challenging problem especially in terms of scheduling. From this perspective, in the context of the PhD of Andra Hugo, we investigate new approaches for composing parallel applications within a runtime system for heterogeneous platforms. The main goal of this work is to build an improved runtime system which is able to deal with parallel tasks (which may use different parallelization schemes or even different parallelization supports). Our final goal would be to have high performance solvers and tools which can efficiently run on all these types of complex architectures by exploiting all the resources of the platform (even if they are heterogeneous).

In order to achieve an advanced knowledge concerning the design of efficient computational kernels to be used on our high performance algorithms and codes, we will develop research activities first on regular frameworks before extending them to more irregular and complex situations. In particular, we will work first on optimized dense linear algebra kernels and we will use them in our more complicated hybrid solvers for sparse linear algebra and in our fast multipole algorithms for interaction computations. In this context, we will participate to the development of those kernels in collaboration with groups specialized in dense linear algebra. In particular, we intend develop a strong collaboration with the group of Jack Dongarra at the University Of Tennessee. The objectives will be to develop dense linear algebra algorithms and libraries for multicore architectures in the context the PLASMA project (http://icl.cs.utk.edu/plasma/) and for GPU and hybrid multicore/GPU architectures in the context of the MAGMA project (http://icl.cs.utk.edu/magma/). The framework that hosts all these research activities is the associated team MORSE (http://www.inria.fr/en/teams/morse).

A more prospective objective is to study the fault tolerance in the context of large-scale scientific applications for massively parallel architectures. Indeed, with the increase of the number of computational cores per node, the probability of a hardware crash on a core is dramatically increased. This represents a crucial problem that needs to be addressed. However, we will only study it at the algorithmic/application level even if it needed...
lower-level mechanisms (at OS level or even hardware level). Of course, this work can be done at lower levels (at operating system) level for example but we do believe that handling faults at the application level provides more knowledge about what has to be done (at application level we know what is critical and what is not). The approach that we will follow will be based on the use of a combination of fault-tolerant implementations of the run-time environments we use (like for example FT-MPI) and an adaptation of our algorithms to try to manage this kind of faults. This topic represents a very long range objective which needs to be addressed to guaranty the robustness of our solvers and applications. In that respect, we are involved in a ANR-Blanc project entitles RESCUE jointly with two other Inria EPI, namely GRAAL and GRAND-LARGE. The main objective of the RESCUE project is to develop new algorithmic techniques and software tools to solve the exascale resilience problem. Solving this problem implies a departure from current approaches, and calls for yet-to-be- discovered algorithms, protocols and software tools.

Finally, it is important to note that the main goal of HiePACS is to design tools and algorithms that will be used within complex simulation frameworks on next-generation parallel machines. Thus, we intend with our partners to use the proposed approach in complex scientific codes and to validate them within very large scale simulations.

3.3. High performance solvers for large linear algebra problems

Participants: Emmanuel Agullo, Mikko Byckling, Mathieu Chanaud, Olivier Coulaud, Iain Duff, Luc Giraud, Abdou Guermouche, Andrea Hugo, Yan-Fei Jing, Matthieu Lecouvez, Yohan Lee-Tin-Yien, Jean Roman, Pablo Salas Medina, Stojce Nakov, Xavier Vasseur, Mawussi Zounon.

Starting with the developments of basic linear algebra kernels tuned for various classes of computers, a significant knowledge on the basic concepts for implementations on high-performance scientific computers has been accumulated. Further knowledge has been acquired through the design of more sophisticated linear algebra algorithms fully exploiting those basic intensive computational kernels. In that context, we still look at the development of new computing platforms and their associated programming tools. This enables us to identify the possible bottlenecks of new computer architectures (memory path, various level of caches, inter processor or node network) and to propose ways to overcome them in algorithmic design. With the goal of designing efficient scalable linear algebra solvers for large scale applications, various tracks will be followed in order to investigate different complementary approaches. Sparse direct solvers have been for years the methods of choice for solving linear systems of equations, it is nowadays admitted that such approaches are not scalable neither from a computational complexity nor from a memory view point for large problems such as those arising from the discretization of large 3D PDE problems. Although we will not contribute directly to this activity, we will use parallel sparse direct solvers as building boxes for the design of some of our parallel algorithms such as the hybrid solvers described in the sequel of this section. Our activities in that context will mainly address preconditioned Krylov subspace methods; both components, preconditioner and Krylov solvers, will be investigated.

3.3.1. Hybrid direct/iterative solvers based on algebraic domain decomposition techniques

One route to the parallel scalable solution of large sparse linear systems in parallel scientific computing is the use of hybrid methods that combine direct and iterative methods. These techniques inherit the advantages of each approach, namely the limited amount of memory and natural parallelization for the iterative component and the numerical robustness of the direct part. The general underlying ideas are not new since they have been intensively used to design domain decomposition techniques; those approaches cover a fairly large range of computing techniques for the numerical solution of partial differential equations (PDEs) in time and space. Generally speaking, it refers to the splitting of the computational domain into sub-domains with or without overlap. The splitting strategy is generally governed by various constraints/objectives but the main one is to express parallelism. The numerical properties of the PDEs to be solved are usually intensively exploited at the continuous or discrete levels to design the numerical algorithms so that the resulting specialized technique will only work for the class of linear systems associated with the targeted PDE.
In that context, we attempt to apply to general unstructured linear systems domain decomposition ideas. More precisely, we will consider numerical techniques based on a non-overlapping decomposition of the graph associated with the sparse matrices. The vertex separator, built by a graph partitioner, will define the interface variables that will be solved iteratively using a Schur complement techniques, while the variables associated with the internal sub-graphs will be handled by a sparse direct solver. Although the Schur complement system is usually more tractable than the original problem by an iterative technique, preconditioning treatment is still required. For that purpose, the algebraic additive Schwarz technique initially developed for the solution of linear systems arising from the discretization of elliptic and parabolic PDE's will be extended. Linear systems where the associated matrices are symmetric in pattern will be first studied but extension to unsymmetric matrices will be latter considered. The main focus will be on difficult problems (including non-symmetric and indefinite ones) where it is harder to prevent growth in the number of iterations with the number of subdomains when considering massively parallel platforms. In that respect, we will consider algorithms that exploit several sources and grains of parallelism to achieve high computational throughput. This activity may involve collaborations with developers of sparse direct solvers as well as with developers of run-time systems and will lead to the development to the library MaPHyS (see Section 5.2). Some specific aspects, such as mixed MPI-thread implementation for the computer science aspects and techniques for indefinite system for the numerical aspects will be investigated in the framework of a France Berkeley Fund project granted that started last year.

3.3.2. Full geometric multigrid method for 3D Maxwell equations

The multigrid methods are among the most promising numerical techniques to solve large linear system of equations arising from the discretization of PDE’s. Their ideal scalabilities, linear growth of memory and floating-point operations with the number of unknowns, for solving elliptic equations make them very appealing for petascale computing and a lot of research works in the recent years has been devoted to the extension to other types of PDE.

In this work (Ph. D. of Mathieu Chanaud in collaboration with CEA/CESTA), we have considered a full geometric multigrid solver for the solution of methodology for solving large linear systems arising from Maxwell equations discretized with first-order Nédelec elements on fully unstructured meshes. This solver combines a parallel sparse direct solver and full multigrid cycles. The goal of this method is to compute the solution for problems defined on fine irregular meshes with minimal overhead costs when compared to the cost of applying a classical direct solver on the coarse mesh. Mathieu Chanaud defended his PhD in October 2011.

The direct solver can handle linear systems with up to a few tens of million unknowns, but this size is limited by the computer memory, so that finer problem resolutions that often occur in practice cannot be handled by this direct solver.

3.3.3. Linear Krylov solvers

Preconditioning is the main focus of the two activities described above. They aim at speeding up the convergence of a Krylov subspace method that is the complementary component involved in the solvers of interest for us. In that framework, we believe that various aspects deserve to be investigated; we will consider the following ones:

Preconditioned block Krylov solvers for multiple right-hand sides. In many large scientific and industrial applications, one has to solve a sequence of linear systems with several right-hand sides given simultaneously or in sequence (radar cross section calculation in electromagnetism, various source locations in seismic, parametric studies in general, ...). For “simultaneous” right-hand sides, the solvers of choice have been for years based on matrix factorizations as the factorization is performed once and simple and cheap block forward/backward substitutions are then performed. In order to effectively propose alternative to such solvers, we need to have efficient preconditioned Krylov subspace solvers. In that framework, block Krylov approaches, where the Krylov spaces associated with each right-hand side are shared to enlarge the search space will be considered. They are not only attractive because of this numerical feature (larger search space), but also from an implementation point of view. Their block-structures exhibit nice features with respect to data
locality and re-usability that comply with the memory constraint of multicore architectures. For right-hand
sides available one after each other, various strategies that exploit the information available in the sequence
of Krylov spaces (e.g. spectral information) will be considered that include for instance technique to perform
incremental update of the preconditioner or to built augmented Krylov subspaces. In that context, Yan-Fei
Jing, who joint HiePACS as post-doc, is investigating how reliable block Arnoldi procedure can be combined
with deflated restarted block GMRES technique.

**Flexible Krylov subspace methods with recycling techniques.** In many situations, it has been observed
that significant convergence improvements can be achieved in preconditioned Krylov subspace methods by
enriching them with some spectral information. On the other hand effective preconditioning strategies are
often designed where the preconditioner varies from one step to the next (e.g. in domain decomposition
methods, when approximate solvers are considered for the interior problems, or more generally for block
preconditioning technique where approximate block solution are used) so that a flexible Krylov solver is
required. In that context, we intend to investigate how numerical techniques implementing subspace recycling
and/or incremental preconditioning can be extended and adapted to cope with this situation of flexible
preconditioning; that is, how can we numerically benefit from the preconditioning implementation flexibility.

**Krylov solver for complex symmetric non-Hermitian matrices.** In material physics when the absorption
spectrum of a molecule due to an exterior field is computed, we have to solve for each frequency a
dense linear system where the matrix depends on the frequency. The sequence of matrices are complex
symmetric non-Hermitian. While a direct approach can be used for small molecules, a Krylov subspace
solver must be considered for larger molecules. Typically, Lanczos-type methods are used to solve these
systems but the convergence is often slow. Based on our earlier experience on preconditioning techniques for
dense complex symmetric non-Hermitian linear system in electromagnetism, we are interested in designing
new preconditioners for this class of material physics applications. A first track will consist in building
preconditioners on sparsified approximation of the matrix as well as computing incremental updates, eg.
Sherman-Morrison type, of the preconditioner when the frequency varies. This action will be developed in
the framework of the research activity described in Section 4.2.

**Approximate factoring of the inverse.** When the matrix of a given sparse linear system of equations is known
to be nonsingular, the computation of approximate factors for the inverse constitutes an algebraic approach to
preconditioning. The main aim is to combine standard preconditioning ideas with sparse approximate inverse
approximation to have implicitly dense approximate inverse approximations. Theory has been developed and
encouraging numerical experiments have been obtained on a set of sparse matrices of small to medium size.
We plan to propose a parallel implementation of the construction of the preconditioner and to investigate its
efficiency on real-life problems. Extension of this technique to build a sparse approximation of the Schur
complement for algebraic domain decomposition has also been investigated and could be integrated in the
MaPhySpackage in the future.

**Extension or modification of Krylov subspace algorithms for multicore architectures.** Finally to match as
much as possible to the computer architecture evolution and get as much as possible performance out of the
computer, a particular attention will be paid to adapt, extend or develop numerical schemes that comply with
the efficiency constraints associated with the available computers. Nowadays, multicore architectures seem
to become widely used, where memory latency and bandwidth are the main bottlenecks; investigations on
communication avoiding techniques will be undertaken in the framework of preconditioned Krylov subspace
solvers as a general guideline for all the items mentioned above.

**Eigensolvers.** Many eigensolvers also rely on Krylov subspace techniques. Naturally some links exist between
the Krylov subspace linear solvers and the Krylov subspace eigensolvers. We plan to study the computation
of eigenvalue problems with respect to the following three different axes:

- Exploiting the link between Krylov subspace methods for linear system solution and eigensolvers,
  we intend to develop advanced iterative linear methods based on Krylov subspace methods that
  use some spectral information to build part of a subspace to be recycled, either though space
  augmentation or through preconditioner update. This spectral information may correspond to a
certain part of the spectrum of the original large matrix or to some approximations of the eigenvalues obtained by solving a reduced eigenproblem. This technique will also be investigated in the framework of block Krylov subspace methods.

• In the framework of an FP7 Marie project (MyPlanet), we intend to study parallel robust nonlinear quadratic eigensolvers. It is a crucial question in numerous technologies like the stability and vibration analysis in classical structural mechanics. The first research action consists in enhancing the robustness of the linear eigensolver and to consider shift invert technique to tackle difficult problems out of reach with the current technique. One of the main constraint in that framework is to design matrix-free technique to limit the memory consumption of the complete solver. For the nonlinear part different approaches ranging from simple nonlinear stationary iterations to Newton’s type approaches will be considered.

• In the context of the calculation of the ground state of an atomistic system, eigenvalue computation is a critical step; more accurate and more efficient parallel and scalable eigensolvers are required (see Section 4.2).

3.4. High performance Fast Multipole Method for N-body problems

Participants: Bérenger Bramas, Arnaud Etcheverry, Olivier Coulaud, Pierre Fortin, Luc Giraud, Jean Roman.

In most scientific computing applications considered nowadays as computational challenges (like biological and material systems, astrophysics or electromagnetism), the introduction of hierarchical methods based on an octree structure has dramatically reduced the amount of computation needed to simulate those systems for a given error tolerance. For instance, in the N-body problem arising from these application fields, we must compute all pairwise interactions among N objects (particles, lines, ...) at every timestep. Among these methods, the Fast Multipole Method (FMM) developed for gravitational potentials in astrophysics and for electrostatic (coulombic) potentials in molecular simulations solves this N-body problem for any given precision with \(O(N)\) runtime complexity against \(O(N^2)\) for the direct computation.

The potential field is decomposed in a near field part, directly computed, and a far field part approximated thanks to multipole and local expansions. In the former ScAlApplix project, we introduced a matrix formulation of the FMM that exploits the cache hierarchy on a processor through the Basic Linear Algebra Subprograms (BLAS). Moreover, we developed a parallel adaptive version of the FMM algorithm for heterogeneous particle distributions, which is very efficient on parallel clusters of SMP nodes. Finally on such computers, we developed the first hybrid MPI-thread algorithm, which enables to reach better parallel efficiency and better memory scalability. We plan to work on the following points in HiePACS.

3.4.1. Improvement of calculation efficiency

Nowadays, the high performance computing community is examining alternative architectures that address the limitations of modern cache-based designs. GPU (Graphics Processing Units) and the Cell processor have thus already been used in astrophysics and in molecular dynamics. The Fast Multipole Method has also been implemented on GPU. We intend to examine the potential of using these forthcoming processors as a building block for high-end parallel computing in N-body calculations. More precisely, we want to take advantage of our specific underlying BLAS routines to obtain an efficient and easily portable FMM for these new architectures. Algorithmic issues such as dynamic load balancing among heterogeneous cores will also have to be solved in order to gather all the available computation power. This research action will be conducted on close connection with the activity described in Section 3.2.

3.4.2. Non uniform distributions

In many applications arising from material physics or astrophysics, the distribution of the data is highly non uniform and the data can grow between two time steps. As mentioned previously, we have proposed a hybrid MPI-thread algorithm to exploit the data locality within each node. We plan to further improve the load balancing for highly non uniform particle distributions with small computation grain thanks to dynamic load balancing at the thread level and thanks to a load balancing correction over several simulation time steps at the process level.
3.4.3. Fast Multipole Method for dislocation operators

The engine that we develop will be extended to new potentials arising from material physics such as those used in dislocation simulations. The interaction between dislocations is long ranged \( (O(1/r)) \) and anisotropic, leading to severe computational challenges for large-scale simulations. Several approaches based on the FMM or based on spatial decomposition in boxes are proposed to speed-up the computation. In dislocation codes, the calculation of the interaction forces between dislocations is still the most CPU time consuming. This computation has to be improved to obtain faster and more accurate simulations. Moreover, in such simulations, the number of dislocations grows while the phenomenon occurs and these dislocations are not uniformly distributed in the domain. This means that strategies to dynamically balance the computational load are crucial to achieve high performance. Funded by the ANR-OPTIDIS, Arnaud Etcheverry started a PhD in October to study parallel scalable FMM techniques for the dislocation calculations.

3.4.4. Fast Multipole Method for boundary element methods

The boundary element method (BEM) is a well known solution of boundary value problems appearing in various fields of physics. With this approach, we only have to solve an integral equation on the boundary. This implies an interaction that decreases in space, but results in the solution of a dense linear system with \( O(N^3) \) complexity. The FMM calculation that performs the matrix-vector product enables the use of Krylov subspace methods. Based on the parallel data distribution of the underlying octree implemented to perform the FMM, parallel preconditioners can be designed that exploit the local interaction matrices computed at the finest level of the octree. This research action will be conducted on close connection with the activity described in Section 3.3. Following our earlier experience, we plan to first consider approximate inverse preconditioners that can efficiently exploit these data structures.

3.5. Efficient algorithmics for code coupling in complex simulations

Participants: Mohamed Abdoul Asize, Olivier Coulaud, Aurélien Esnard, Jean Roman, Jérôme Soumagne, Clément Vuchener.

Many important physical phenomena in material physics and climatology are inherently complex applications. They often use multi-physics or multi-scale approaches, that couple different models and codes. The key idea is to reuse available legacy codes through a coupling framework instead of merging them into a standalone application. There is typically one model per different scale or physics; and each model is implemented by a parallel code. For instance, to model a crack propagation, one uses a molecular dynamic code to represent the atomistic scale and an elasticity code using a finite element method to represent the continuum scale. Indeed, fully microscopic simulations of most domains of interest are not computationally feasible. Combining such different scales or physics are still a challenge to reach high performance and scalability. If the model aspects are often well studied, there are several open algorithmic problems, that we plan to investigate in the HiePACS project-team.

The experience that we have acquired in the ScAlApplix project through the activities in crack propagation simulations with LibMultiScale and in M-by-N computational steering (coupling simulation with parallel visualization tools) with EPSI shows us that if the model aspect was well studied, several problems in parallel or distributed algorithms are still open and not well studied. In the context of code coupling in HiePACS we want to contribute more precisely to the following points.

3.5.1. Efficient schemes for multiscale simulations

As mentioned previously, many important physical phenomena, such as material deformation and failure (see Section 4.2), are inherently multiscale processes that cannot always be modeled via continuum model. Fully microscopic simulations of most domains of interest are not computationally feasible. Therefore, researchers must look at multiscale methods that couple micro models and macro models. Combining different scales such as quantum-atomic or atomic, mesoscale and continuum, are still a challenge to obtain efficient and accurate schemes that efficiently and effectively exchange information between the different scales. We are currently involved in two national research projects (ANR), that focus on multiscale schemes. More precisely,
the models that we start to study are the quantum to atomic coupling (QM/MM coupling) in the NOSSI ANR and the atomic to dislocation coupling in the OPTIDIS ANR (proposal for the 2010 COSINUS call of the French ANR).

3.5.2. Load-balancing of complex coupled simulations based on the hypergraph model

One most important issue is undoubtedly the problem of load-balancing of the whole coupled simulation. Indeed, the naive balancing of each code on its own can lead to important imbalance in the coupling area. Another connected problem we plan to investigate is the problem of resource allocation. This is particularly important for the global coupling efficiency, because each code involved in the coupling can be more or less computationally intensive, and there is a good trade-off to find between resources assigned to codes to avoid that one of them wait for the others.

The performance of the coupled codes depends on how the data are well distributed on the processors. Generally, the data distributions of each code are built independently from each other to obtain the best load-balancing. But once the codes are coupled, the naive use of these decompositions can lead to important imbalance in the coupling area. Therefore, the modeling of the whole coupling is crucial to improve the performance and to ensure a good scalability. The goal is to find the best data distribution for the whole coupled codes and not only for each standalone code. One idea is to use an hypergraph model that will incorporate information about the coupling itself. Then, we expect the greater expressiveness of hypergraph will enable us to perform a coupling-aware partitioning in order to improve the load-balancing of the whole coupled simulation.

Another connected problem we plan to investigate is the problem of resource allocation. This is particularly important for the global coupling efficiency and scalability, because each code involved in the coupling can be more or less computationally intensive, and there is a good trade-off to find between resources assigned to codes to avoid that one of them wait for the others. Typically, if we have a given number of processors and two coupled codes, how to split the processors among each code?

Moreover, the load-balancing of modern parallel adaptive simulations raises a crucial issue when the problem size varies during execution. In such cases, it could be convenient to dynamically adapt the number of resources used at runtime. However, most of previous works on repartitioning only consider a constant number of resources. We plan to design new repartitioning algorithm based on an hypergraph model that can handle a variable number of processors. Furthermore, this kind of algorithms could be used for the dynamic balancing of a coupled simulation, in the case where the whole number of resources is fixed but can change for each code.

3.5.3. Steering and interacting with complex coupled simulations

The computational steering is an effort to make the typical simulation work-flow (modelling, computing, analyzing) more efficient, by providing online visualization and interactive steering over the on-going computational processes. The online visualization appears very useful to monitor and to detect possible errors in long-running applications, and the interactive steering allows the researcher to alter simulation parameters on-the-fly and to immediately receive feedback on their effects. Thus, the scientist gains an additional insight in the simulation regarding to the cause-and-effect relationship.

In the ScAlApplix project, we have studied this problem in the case where both the simulation and the visualization can be parallel, what we call M-by-N computational steering, and we have developed a software environment called EPSN (see Section 5.3 ). More recently, we have proposed a model for the steering of complex coupled simulations and one important conclusion we have from these previous works is that the steering problem can be conveniently modeled as a coupling problem between one or more parallel simulation codes and one visualization code, that can be parallel as well. We propose in HiePACS to revisit the steering problem as a coupling problem and we expect to reuse the new redistribution algorithms developped in the context of code coupling for the purpose of M-by-N steering. We expect such an approach will enable to steer massively-parallel simulations. Another point we plan to study is the monitoring and interaction with resources, in order to perform user-directed checkpoint/restart or user-directed load-balancing at runtime.
In several applications, it is often very useful either to visualize the results of the ongoing simulation before writing it to disk, or to steer the simulation by modifying some parameters and visualize the impact of these modifications interactively. Nowadays, high performance computing simulations use many computing nodes, that perform I/O using the widely used HDF5 file format. One of the problems is now to use real-time visualization using high performance computing. In that respect we need to efficiently combine very large parallel simulation systems with parallel visualization systems. The originality of this approach is the use of the HDF5 file format to write in a distributed shared memory (DSM); so that the data can be read from the upper part of the visualization pipeline. This leads to define a relevant steering model based on a DSM. It implies finding a way to write/read data efficiently in this DSM, and steer the simulation. This work is developed in collaboration with the Swiss National Supercomputing Centre (CSCS).

As concerns the interaction aspect, we are interested in providing new mechanisms to interact with the simulation directly through the visualization. For instance in the ANR NOSSI, in order to speed up the computation we are interested in rotating a molecule in a cavity or in moving it from one cavity to another within the crystal lattice. To perform safely such interactions a model of the interaction in our steering framework is necessary to keep the data coherency in the simulation. Another point we plan to study is the monitoring and interaction with resources, in order to perform user-directed checkpoint/restart or user-directed load balancing at runtime.
3. Scientific Foundations

3.1. Our goals and methodology

Managing data at large scales is paramount nowadays and many application areas exhibit a need for efficient scaling to huge data sizes: data mining applications [55], multimedia applications [46], database-oriented applications ([49], [67], [62]), bioinformatic applications, etc. In such contexts, one important goal is to provide mechanisms allowing to transparently manage massive data blocks (e.g., of several terabytes), while providing efficient, fine-grain access to small parts of the data.

The overall goal of the KerData team is to bring a substantial contribution to the effort of the research community to address the above challenges. More specifically, to support the large-scale execution of the applications we described, KerData aims to design and implement distributed algorithms for scalable data storage and input/output management for efficient large-scale data processing. We target two main execution infrastructures: cloud platforms and post-Petascale HPC supercomputers. We are also looking at other kinds of infrastructures (that we are considering as secondary), e.g. hybrid platforms combining enterprise desktop grids extended to cloud platforms.

Our approach relies on building prototypes and on their large-scale experimental validation on real testbeds and experimental platforms. In our current projects, our target platforms include: the Grid’5000 testbed, Amazon and Microsoft’s Azure commercial clouds, public clouds based on open-source IaaS toolkits such as Nimbus and OpenNebula. In the HPC area we have access to the Jaguar and Kraken supercomputers (ranked 3rd and 11th respectively in the Top 500 supercomputer list). Last but not least, our methodology includes large-scale validations of our solutions with real-life applications, such as the ones described in Section 4.1. To this purpose, we have started to build partnerships with the application communities that can potentially benefit from our contributions and we will continue to do so in future collaborative projects.

3.2. Transparent, distributed data sharing

The management of massive data blocks naturally requires the use of data fragmentation and of distributed storage. Grid infrastructures, typically built by aggregating distributed resources that may belong to different administration domains, were built during the last years with the goal of providing an appropriate solution. When considering the existing approaches to grid data management, we can notice that most of them heavily rely on explicit data localization and on explicit transfers of large amounts of data across the distributed architecture: GridFTP [40], LDR [35], Chirp [29], IBP [41], NeST [42], etc. Managing huge amounts of data in such an explicit way at a very large scale makes the design of grid applications much more complex. One key issue to be addressed is therefore the transparency with respect to data localization and data movements. Such a transparency is highly suitable, as it alleviates the user of the need to handle data localization and transfers.

3.3. Managing massive unstructured data under heavy concurrency on large-scale distributed infrastructures

3.3.1. Massive unstructured data: BLOBs

Studies show more than 80% [53] of data globally in circulation is unstructured. On the other hand, data sizes increase at a dramatic level with more than 1 TB of data gathered per week in common scenarios for some production applications (e.g., medical experiments [65]). Finally, on Post-Petascale HPC machines, the use of huge storage objects is also currently being considered as a promising alternative to today’s dominant
approaches to data management. Indeed, these approaches rely on very large numbers of small files, and using huge storage objects reduces the corresponding metadata overhead of the file system. Such huge unstructured data are stored as binary large objects (BLOBs) that may continuously be updated by applications. However, traditional databases or file systems can hardly cope in an efficient way with BLOBs which grow to huge sizes.

3.3.2. Scalable processing of massive data: heavy access concurrency

To address the scalability issue, specialized abstractions like MapReduce [47] and Pig-Latin [63] propose high-level data processing frameworks intended to hide the details of parallelization from the user. Such platforms are implemented on top of huge object storage platforms. They target high performance by optimizing the parallel execution of the computation. This leads to heavy access concurrency to the BLOBs, thus the need for the storage layer to offer support in this regard. Parallel and distributed file systems also consider using objects for low-level storage (see next subsection [48], [69], [51]). In other application areas, huge BLOBs need to be used concurrently at the highest level layers of applications directly: high-energy physics, multimedia processing [46] or astronomy.

3.3.3. Versioning

When addressing the problem of storing and efficiently accessing very large unstructured data objects [60], [65] in a distributed environment, a challenging case is the one where data is mutable and potentially accessed by a very large number of concurrent, distributed processes. In this context, versioning is an important feature. Not only it allows to roll back data changes when desired, but it also enables cheap branching (possibly recursively): the same computation may proceed independently on different versions of the BLOB. Versioning should obviously not impact access performance to the object significantly, given that objects are under constant heavy access concurrency. On the other hand, versioning leads to increased storage space usage and becomes a major concern when the data size itself is huge. Versioning efficiency thus refers to both access performance under heavy load and reasonably acceptable overhead of storage space.

3.4. Towards scalable, BLOB-based distributed file systems

Recent research [50] emphasizes a clear move currently in progress from a block-based interface to an object-based interface in storage architectures. The goal is to enable scalable, self-managed storage networks by moving low-level functionalities such as space management to storage devices or to storage server, accessed through a standard object interface. This move has a direct impact on the design of today’s distributed file systems: object-based file system would then store data rather as objects than as unstructured data blocks. According to [50], this move may eliminate nearly 90% of management workload which was the major obstacle limiting file systems’ scalability and performance.

Two approaches exploit this idea. In the first approach, the data objects are stored and manipulated directly by a new type of storage device called object-based storage device (OSD). This approach requires an evolution of the hardware, in order to allow high-level object operations to be delegated to the storage device. Examples of parallel/distributed file systems following this approach are Lustre [66] and Ceph [69]. Recently, research efforts [48] have explored the feasibility and the possible benefits of integrating OSDs into parallel file systems, such as PVFS [45].

The second approach does not rely on the presence of OSDs, but still tries to benefit from an object-based approach to improve performance and scalability: files are structured as a set of objects that are stored on storage servers. Google File System [51], and HDFS (Hadoop File System) [33] illustrate this approach.

3.5. Emerging large-scale infrastructures for distributed applications

During the last few years, research and development in the area of large-scale distributed computing led to the clear emergence of several types of physical execution infrastructures for large-scale distributed applications.
3.5.1. Cloud computing infrastructures

The cloud computing model \cite{68}, \cite{59}, \cite{44} is gaining serious interest from both industry and academia in the area of large-scale distributed computing. It provides a new paradigm for managing computing resources: instead of buying and managing hardware, users rent virtual machines and storage space.

Various cloud software stacks have been proposed by leading industry companies, like Google, Amazon or Yahoo!. They aim at providing fully configurable virtual machines or virtual storage (IaaS: Infrastructure-as-a-Service), higher-level services including programming environments such as MapReduce \cite{47} (PaaS: Platform-as-a-Service \cite{31}, \cite{36}) or community-specific applications (SaaS: Software-as-a-Service \cite{32}, \cite{37}). On the academic side, two of the most visible projects in this area are Nimbus \cite{38}, \cite{57} from the Argonne National Lab (USA) and OpenNebula \cite{39}, which aim at providing a reference implementation for a IaaS.

In the context of the emerging cloud infrastructures, some of the most critical open issues relate to data management. Providing the users with the possibility to store and process data on externalized, virtual resources from the cloud requires simultaneously investigating important aspects related to security, efficiency and quality of service. To this purpose, it clearly becomes necessary to create mechanisms able to provide feedback about the state of the storage system along with the underlying physical infrastructure. The information thus monitored, can further be fed back into the storage system and used by self-managing engines, in order to enable an autonomic behavior \cite{58}, \cite{64}, \cite{54}, possibly with several goals such as self-configuration, self-optimization, or self-healing. Exploring ways to address the main challenges raised by data storage and management on cloud infrastructures is the major factor that motivated the creation of the KerData research team INRIA RENNES – BRETAGNE ATLANTIQUE. These topics are at the heart of our involvement in several projects that we are leading in the area of cloud storage: MapReduce (see Section 6.1), AzureBrain (see Section 6.1), DataCloud@work (see Section 6.3).

3.5.2. Petascale infrastructures

In 2011, a new NSF-funded Petascale computing system, Blue Waters, will go online at the University of Illinois. Blue Waters is expected to be the most powerful supercomputer in the world for open scientific research when it comes online. It will be the first system of its kind to sustain one-Petaflop performance on a range of science and engineering applications. The goal of this facility is to open up new possibilities in science and engineering. It provides unheard computational capability. It makes it possible for investigators to tackle much larger and more complex research challenges across a wide spectrum of domains: predict the behavior of complex biological systems, understand how the cosmos evolved after the Big Bang, design new materials at the atomic level, predict the behavior of hurricanes and tornadoes, and simulate complex engineered systems like the power distribution system and airplanes and automobiles.

To reach sustained-Petascale performance, machines like Blue Waters relies on advanced, dedicated technologies at several levels: processor, memory subsystem, interconnect, operating system, programming environment, system administration tools. In this context, data management is again a critical issue that highly impacts the application behavior and its overall performance. Petascale supercomputers exhibit specific architectural features (e.g., a multi-level memory hierarchy scalable to tens to hundreds of thousands of codes) that needs to be specifically taken into account. Providing scalable data throughput on such unprecedented scales is clearly an open challenge today. In this context, we are investigating techniques to achieve concurrency-optimized I/O in collaboration with teams from the National Center for Supercomputing Applications (NCSA/UIUC) in the framework of the Joint INRIA-UIUC for Petascale Computing (see Section 6.6).

3.6. Emerging programming models for scalable data-management

MapReduce is a parallel programming paradigm successfully used by large Internet service providers to perform computations on massive amounts of data. A computation takes a set of input key/value pairs, and produces a set of output key/value pairs. The user of a MapReduce library expresses the computation as two functions: map, that processes a key/value pair to generate a set of intermediate key/value pairs, and reduce, that merges all intermediate values associated with the same intermediate key. The framework takes care of
splitting the input data, scheduling the jobs’ component tasks, monitoring them and re-executing the failed ones. After being strongly promoted by Google, it has also been implemented by the open source community through the Hadoop project, maintained by the Apache Foundation and supported by Yahoo! and even by Google itself. This model is currently getting more and more popular as a solution for rapid implementation of distributed data-intensive applications. The key strength of the MapReduce model is its inherently high degree of potential parallelism that should enable processing of Petabytes of data in a couple of hours on large clusters consisting of several thousand nodes.

At the core of the MapReduce frameworks stays a key component: the storage layer. To enable massively parallel data processing to a high degree over a large number of nodes, the storage layer must meet a series of specific requirements. Firstly, since data is stored in huge files, the computation will have to efficiently process small parts of these huge files concurrently. Thus, the storage layer is expected to provide efficient fine-grain access to the files. Secondly, the storage layer must be able to sustain a high throughput in spite of heavy access concurrency to the same file, as thousands of clients simultaneously access data.

These critical needs of data-intensive distributed applications have not been addressed by classical, POSIX-compliant distributed file systems. Therefore, specialized file systems have been designed, such as HDFS, the default storage layer of Hadoop. HDFS has however some difficulties in sustaining a high throughput in the case of concurrent accesses to the same file. Amazon’s cloud computing initiative, Elastic MapReduce, employs Hadoop on their Elastic Compute Cloud infrastructure (EC2) and inherits these limitations. The storage back-end used by Hadoop is Amazon’s Simple Storage Service (S3), which provides limited support for concurrent accesses to shared data. Moreover, many desirable features are missing altogether, such as the support for versioning and for concurrent updates to the same file. Finally, another important requirement for the storage layer is its ability to expose an interface that enables the application to be data-location aware. This is critical in order to allow the scheduler to use this information to place computation tasks close to the data and thus reduce network traffic, contributing to a better global data throughput. These topics are at the core of KerData’s contribution to the MapReduce ANR project and to the Hemera large wingspan project, both started in 2010, see Section 8.2.
3. Scientific Foundations

3.1. Large System Modeling and Analysis

Participants: Bruno Gaujal, Derrick Kondo, Arnaud Legrand, Panayotis Mertikopoulos, Florence Perronnin, Brigitte Plateau, Olivier Richard, Corinne Touati, Jean-Marc Vincent.

3.1.1. Simulation of distributed systems

Since the advent of distributed computer systems, an active field of research has been the investigation of scheduling strategies for parallel applications. The common approach is to employ scheduling heuristics that approximate an optimal schedule. Unfortunately, it is often impossible to obtain analytical results to compare the efficiency of these heuristics. One possibility is to conduct large numbers of back-to-back experiments on real platforms. While this is possible on tightly-coupled platforms, it is infeasible on modern distributed platforms (i.e. Grids or peer-to-peer environments) as it is labor-intensive and does not enable repeatable results. The solution is to resort to simulations.

3.1.1.1. Flow Simulations

To make simulations of large systems efficient and trustful, we have used flow simulations (where streams of packets are abstracted into flows). SimGRID is a simulation platform that not only enable one to get repeatable results but also make it possible to explore wide ranges of platform and application scenarios.

3.1.1.2. Perfect Simulation

Using a constructive representation of a Markovian queuing network based on events (often called GSMPs), we have designed a perfect simulation algorithms computing samples distributed according to the stationary distribution of the Markov process with no bias. The tools based on our algorithms ($\psi$) can sample the stationary measure of Markov processes using directly the queuing network description. Some monotone networks with up to $10^{50}$ states can be handled within minutes over a regular PC.

3.1.2. Fluid models and mean field limits

When the size of systems grows very large, one may use asymptotic techniques to get a faithful estimate of their behavior. One such tools is mean field analysis and fluid limits, that can be used at a modeling and simulation level. Proving that large discrete dynamic systems can be approximated by continuous dynamics uses the theory of stochastic approximation pioneered by Michel Benaim or population dynamics introduced by Thomas Kurtz and others. We have extended the stochastic approximation approach to take into account discontinuities in the dynamics as well as to tackle optimization issues.

Recent applications include call centers and peer to peer systems. where the mean field approach helps to get a better understanding of the behavior of the system and to solve several optimization problems. Another application concerns task brokering in desktop grids taking into account statistical features of tasks as well as of the availability of the processors. Mean field has also been applied to the performance evaluation of work stealing in large systems and to model central/local controllers as well as knitting systems.

3.1.3. Discrete Event Systems

The interaction of several processes through synchronization, competition or superposition within a distributed system is a big source of difficulties because it induces a state space explosion and a non-linear dynamic behavior. The use of exotic algebra, such as (min,max,plus) can help. Highly synchronous systems become linear in this framework and therefore are amenable to formal solutions. More complicated systems are neither linear in (max,plus) nor in the classical algebra. Several qualitative properties have been established for a large class of such systems called free-choice Petri nets (sub-additivity, monotonicity or convexity properties). Such qualitative properties are sometimes enough to assess the class of routing policies optimizing the global behavior of the system. They are also useful to design efficient numerical tools computing their asymptotic behavior.
The worst case analysis of networks can also be done using the (max,plus) machinery, called network calculus or real time calculus in this context.

### 3.1.4. Game Theory

Resources in large-scale distributed platforms (grid computing platforms, enterprise networks, peer-to-peer systems) are shared by a number of users having conflicting interests who are thus prone to act selfishly. A natural framework for studying such non-cooperative individual decision-making is game theory. In particular, game theory models the decentralized nature of decision-making.

It is well known that such non-cooperative behaviors can lead to important inefficiencies and unfairness. In other words, individual optimizations often result in global resource waste. In the context of game theory, a situation in which all users selfishly optimize their own utility is known as a Nash equilibrium or Wardrop equilibrium. In such equilibria, no user has interest in unilaterally deviating from its strategy. Such policies are thus very natural to seek in fully distributed systems and have some stability properties. However, a possible consequence is the Braess paradox in which the increase of resource happens at the expense of every user. This is why, the study of the occurrence and degree of such inefficiency is of crucial interest. Up until now, little is known about general conditions for optimality or degree of efficiency of these equilibria, in a general setting.

Many techniques have been developed to enforce some form of collaboration and improve these equilibria. In this context, it is generally prohibitive to take joint decisions so that a global optimization cannot be achieved. A possible option relies on the establishment of virtual prices, also called shadow prices in congestion networks. These prices ensure a rational use of resources. Equilibria can also be improved by advising policies to mobiles such that any user that does not follow these pieces of advice will necessarily penalize herself (correlated equilibria).

### 3.2. Management of Large Architectures

**Participants:** Derrick Kondo, Arnaud Legrand, Vania Marangozova-Martin, Olivier Richard, Corinne Touati.

#### 3.2.1. Instrumentation, analysis and prediction tools

To understand complex distributed systems, one has to provide reliable measurements together with accurate models before applying this understanding to improve system design.

Our approach for instrumentation of distributed systems (embedded systems as well as multi-core machines or distributed systems) relies on quality of service criteria. In particular, we focus on non-obtrusiveness and experimental reproducibility.

Our approach for analysis is to use statistical methods with experimental data of real systems to understand their normal or abnormal behavior. With that approach we are able to predict availability of very large systems (with more than 100,000 nodes), to design cost-aware resource management (based on mathematical modeling and performance evaluation of target architectures), and to propose several scheduling policies tailored for unreliable and shared resources.

#### 3.2.2. Fairness in large-scale distributed systems

Large-scale distributed platforms (Grid computing platforms, enterprise networks, peer-to-peer systems) result from the collaboration of many people. Thus, the scaling evolution we are facing is not only dealing with the amount of data and the number of computers but also with the number of users and the diversity of their behavior. In a high-performance computing framework, the rationale behind this joining of forces is that most users need a larger amount of resources than what they have on their own. Some only need these resources for a limited amount of time. On the opposite some others need as many resources as possible but do not have particular deadlines. Some may have mainly tightly-coupled applications while some others may have mostly embarrassingly parallel applications. The variety of user profiles makes resources sharing a challenge. However resources have to be fairly shared between users, otherwise users will leave the group and join another one. Large-scale systems therefore have a real need for fairness and this notion is missing from classical scheduling models.
3.2.3. Tools to operate clusters

The MESCAL project-team studies and develops a set of tools designed to help the installation and the use of a cluster of PCs. The first version had been developed for the Icluster1 platform exploitation. The main tools are a scalable tool for cloning nodes (KA-DEPLOY) and a parallel launcher based on the TAKTUK project (now developed by the MOAIS project-team). Many interesting issues have been raised by the use of the first versions among which we can mention environment deployment, robustness and batch scheduler integration. A second generation of these tools is thus under development to meet these requirements.

KA-DEPLOY has been retained as the primary deployment tool for the experimental national grid GRID’5000.

3.2.4. Simple and scalable batch scheduler for clusters and grids

Most known batch schedulers (PBS, LSF, Condor, ...) are of old-fashioned conception, built in a monolithic way, with the purpose of fulfilling most of the exploitation needs. This results in systems of high software complexity (150,000 lines of code for OpenPBS), offering a growing number of functions that are, most of the time, not used. In such a context, it becomes hard to control both the robustness and the scalability of the whole system.

OAR is an attempt to address these issues. Firstly, OAR is written in a very high level language (Perl) and makes intensive use of high level tools (MySql and TAKTUK), thereby resulting in a concise code (around 5000 lines of code) easy to maintain and extend. This small code as well as the choice of widespread tools (MySql) are essential elements that ensure a strong robustness of the system. Secondly, OAR makes use of SQL requests to perform most of its job management tasks thereby getting advantage of the strong scalability of most database management tools. Such scalability is further improved in OAR by making use of TAKTUK to manage nodes themselves.

3.3. Migration and resilience

Participants: Yves Denneulin, Jean-François Méhaut.

Making a distributed system reliable has been and remains an active research domain. Nonetheless this has not so far lead to results usable in an intranet or federal architecture for computing. Most propositions address only a given application or service. This may be due to the fact that until clusters and intranet architectures arose, it was obvious that client and server nodes were independent. So, a fault or a predictable disconnection on most of the nodes did not lead to a complete failure of the system. This is not the case in parallel scientific computing where a fault on a node can lead to a data loss on thousands of other nodes. The reliability of the system is hence a crucial point. MESCAL’s work on this topic is based on the idea that each process in a parallel application will be executed by a group of nodes instead of a single node: when the node in charge of a process fails, another in the same group can replace it in a transparent way for the application.

There are two main problems to be solved in order to achieve this objective. The first one is the ability to migrate processes of a parallel, and thus communicating, application without enforcing modifications. The second one is the ability to maintain a group structure in a completely distributed way. The first one relies on a close interaction with the underlying operating systems and networks, since processes can be migrated in the middle of a communication. This can only be done by knowing how to save and replay later all ongoing communications, independently of the communication pattern. Freezing a process to restore it on another node is also an operation that requires collaboration of the operating system and a good knowledge of its internals. The other main problem (keeping a group structure) belongs to the distributed algorithms domain and is of a much higher level nature.

3.4. Large scale data management

Participants: Yves Denneulin, Vania Marangozova-Martin, Jean-François Méhaut.
In order to use large data, it is necessary (but not always sufficient, as seen later) to efficiently store and transfer them to a given site (a set of nodes) where it is going to be used. The first step toward this achievement is the construction of a file system that is an extension of NFS for the grid environment. The second step is an efficient transfer tool that provides throughput close to optimal (i.e. the capacity of the underlying hardware).

3.4.1. Fast distributed storage over a cluster

Our goal here is to design a distributed file system for clusters that enables one to store data over a set of nodes (instead of a single one). It was designed to permit the usage of a set of disks to optimize memory allocations. It is important for performance and simplicity that this new file system has little overhead for access and updates. From a user point of view, it is used just as a classical NFS. From the server point of view, however, the storage is distributed over several nodes (possibly including the users).

3.4.2. Reliable distribution of data

Storage distribution on a large set of disks raises the reliability problem: more disks mean a higher fault rate. To address this problem we introduced in NFSP a redundancy on the IODs, the storage nodes by defining VIOD, Virtual IOD, which is a set of IODs that contain exactly the same data. So when an IOD fails another one can serve the same data and continuity of service is insured though. This doesn’t modify the way the file-system is used by the clients: distribution and replication remain transparent. Several consistency protocols are proposed with various levels of performance; they all enforce at least the NFS consistency which is expected by the client.
3. Scientific Foundations

3.1. Scheduling

Participants: Pierre-François Dutot, Guillaume Huard, Grégory Mounié, Jean-Louis Roch, Denis Trystram, Frédéric Wagner.

The goal of this theme is to determine adequate multi-criteria objectives which are efficient (precision, reactivity, speed) and to study scheduling algorithms to reach these objectives.

In the context of parallel and distributed processing, the term scheduling is used with many acceptations. In general, scheduling means assigning tasks of a program (or processes) to the various components of a system (processors, communication links).

Researchers within MOAIS have been working on this subject for many years. They are known for their multiple contributions for determining the target dates and processors the tasks of a parallel program should be executed; especially regarding execution models (taking into account inter-task communications or any other system features) and the design of efficient algorithms (for which there exists a performance guarantee relative to the optimal scheduling).

Parallel tasks model and extensions. We have contributed to the definition and promotion of modern task models: parallel moldable tasks and divisible load. For both models, we have developed new techniques to derive efficient scheduling algorithms (with a good performance guaranty). We proposed recently some extensions taking into account machine unavailabilities (reservations).

Multi-objective Optimization. A natural question while designing practical scheduling algorithms is "which criterion should be optimized?". Most existing works have been developed for minimizing the makespan (time of the latest tasks to be executed). This objective corresponds to a system administrator view who wants to be able to complete all the waiting jobs as soon as possible. The user, from his/her point of view, would be more interested in minimizing the average of the completion times (called minsum) of the whole set of submitted jobs. There exist several other objectives which may be pertinent for specific use. We worked on the problem of designing scheduling algorithms that optimize simultaneously several objectives with a theoretical guarantee on each objective. The main issue is that most of the policies are good for one criterion but bad for another one.

We have proposed an algorithm that is guaranteed for both makespan and minsum. This algorithm has been implemented for managing the resources of a cluster of the regional grid CIMENT. More recently, we extended such analysis to other objectives (makespan and reliability). We concentrate now on finding good algorithms able to schedule a set of jobs with a large variety of objectives simultaneously. For hard problems, we propose approximation of Pareto curves (best compromizes).

Incertainties. Most of the new execution supports are characterized by a higher complexity in predicting the parameters (high versatility in desktop grids, machine crash, communication congestion, cache effects, etc.). We studied some time ago the impact of uncertainties on the scheduling algorithms. There are several ways for dealing with this problem: First, it is possible to design robust algorithms that can optimized a problem over a set of scenarii, another solution is to design flexible algorithms. Finally, we promote semi on-line approaches that start from an optimized off-line solution computed on an initial data set and updated during the execution on the "perturbed" data (stability analysis).

Game Theory. Game Theory is a framework that can be used for obtaining good solution of both previous problems (multi-objective optimization and incertain data). On the first hand, it can be used as a complement of multi-objective analysis. On the other hand, it can take into account the uncertainties. We are curently working at formalizing the concept of cooperation.
Scheduling for optimizing parallel time and memory space. It is well known that parallel time and memory space are two antagonists criteria. However, for many scientific computations, the use of parallel architectures is motivated by increasing both the computation power and the memory space. Also, scheduling for optimizing both parallel time and memory space targets an important multicriteria objective. Based on the analysis of the dataflow related to the execution, we have proposed a scheduling algorithm with provable performance.

Coarse-grain scheduling of fine grain multithreaded computations on heterogeneous platforms. Designing multi-objective scheduling algorithms is a transversal problem. Work-stealing scheduling is well studied for fine grain multithreaded computations with a small critical time: the speed-up is asymptotically optimal. However, since the number of tasks to manage is huge, the control of the scheduling is expensive. We proposed a generalized lock-free cactus stack execution mechanism, to extend previous results, mainly from Cilk, based on the work-first principle for strict multi-threaded computations on SMPs to general multithreaded computations with dataflow dependencies. The main result is that optimizing the sequential local executions of tasks enables to amortize the overhead of scheduling. This distributed work-stealing scheduling algorithm has been implemented in Kaapi.

3.2. Adaptive Parallel and Distributed Algorithms Design

Participants: François Broquedis, Pierre-François Dutot, Thierry Gautier, Guillaume Huard, Bruno Raffin, Jean-Louis Roch, Denis Trystram, Frédéric Wagner.

This theme deals with the analysis and the design of algorithmic schemes that control (statically or dynamically) the grain of interactive applications.

The classical approach consists in setting in advance the number of processors for an application, the execution being limited to the use of these processors. This approach is restricted to a constant number of identical resources and for regular computations. To deal with irregularity (data and/or computations on the one hand; heterogeneous and/or dynamical resources on the other hand), an alternate approach consists in adapting the potential parallelism degree to the one suited to the resources. Two cases are distinguished:

- in the classical bottom-up approach, the application provides fine grain tasks; then those tasks are clustered to obtain a minimal parallel degree.
- the top-down approach (Cilk, Cilk+, TBB, Hood, Athapascan) is based on a work-stealing scheduling driven by idle resources. A local sequential depth-first execution of tasks is favored when recursive parallelism is available.

Ideally, a good parallel execution can be viewed as a flow of computations flowing through resources with no control overhead. To minimize control overhead, the application has to be adapted: a parallel algorithm on $p$ resources is not efficient on $q < p$ resources. On one processor, the scheduler should execute a sequential algorithm instead of emulating a parallel one. Then, the scheduler should adapt resource availability by changing its underlying algorithm. This first way of adapting granularity is implemented by Kaapi (default work-stealing schedule based on work-first principle).

However, this adaptation is restrictive. More generally, the algorithm should adapt itself at runtime to improve its performance by decreasing the overheads induced by parallelism, namely the arithmetic operations and communications. This motivates the development of new parallel algorithmic schemes that enable the scheduler to control the distribution between computation and communication (grain) in the application to find the good balance between parallelism and synchronizations. MOAIS has exhibited several techniques to manage adaptivity from an algorithmic point of view:

- amortization of the number of global synchronizations required in an iteration (for the evaluation of a stopping criterion);
- adaptive deployment of an application based on on-line discovery and performance measurements of communication links;
- generic recursive cascading of two kind of algorithms: a sequential one, to provide efficient executions on the local resource, and a parallel one that enables an idle resource to extract parallelism to dynamically suit the degree of parallelism to the available resources.
The generic underlying approach consists in finding a good mix of various algorithms, what is often called a "poly-algorithm". Particular instances of this approach are Atlas library (performance benchmark are used to decide at compile time the best block size and instruction interleaving for sequential matrix product) and FFTW library (at run time, the best recursive splitting of the FFT butterfly scheme is precomputed by dynamic programming). Both cases rely on pre-benchmarking of the algorithms. Our approach is more general in the sense that it also enables to tune the granularity at any time during execution. The objective is to develop processor oblivious algorithms: similarly to cache oblivious algorithms, we define a parallel algorithm as processor-oblivious if no program variable that depends on architecture parameters, such as the number or processors or their respective speeds, needs to be tuned to minimize the algorithm runtime.

We have applied this technique to develop processor oblivious algorithms for several applications with provable performance: iterated and prefix sum (partial sums) computations, stream computations (cipher and hd-video transformation), 3D image reconstruction (based on the concurrent usage of multi-core and GPU), loop computations with early termination. Finally, to validate these novel parallel computation schemes, we developed a tool named KRASH. This tool is able to generate dynamic CPU load in a reproducible way on many-cores machines. Thus, by providing the same experimental conditions to several parallel applications, it enables users to evaluate the efficiency of resource uses for each approach.

This adaptation technique is now integrated in softwares that we are developing with external partners within contracts. In particular, in partnership with STM within the Minalogic SCEPTRE contract we have developed a specific optimized C interface, dedicated to stream computation for multi-processor system on chips (MPSoC); this interface is named AWS (Adaptive Work-Stealing).

Besides, we developed a parallel implementation of the C++ Standard Template Library STL on top of Kaapi; this library, named KaSTL, provides adaptive parallel algorithms for distributed containers (such as transform, foreach and findif on vectors). By optimizing the work-stealing to our adaptive algorithm scheme, a new non-blocking (wait-free) implementation of Kaapi has been designed. A first prototype of this C library, named X-Kaapi, The benchmarks experimented on SMPs and NUMAs architectures provides good performances with respect to concurrent libraries MCSTL, PaSTL, Intel TBB, and Cilk+, while improving the grain where parallelism can be exploited.

Extensions concern the development of algorithms that are both cache and processor oblivious. The processor algorithms proposed for prefix sums and segmentation of an array are cache oblivious too. We are currently working on sorting and mesh partitioning within a collaboration with the CEA.

3.3. Interactivity

Participants: Vincent Danjean, Pierre-François Dutot, Thierry Gautier, Bruno Raffin, Jean-Louis Roch.

The goal of this theme is to develop approaches to tackle interactivity in the context of large scale distributed applications.

We distinguish 2 types of interactions. A user can interact with an application having only little insight about the internal details of the program running. This is typically the case for a virtual reality application where the user just manipulates 3D objects. We have a "user-in-the-loop". In opposite, we have an "expert -in-the-loop" if the user is an expert that knows the limits of the program that is being executed and that he can interacts with it to steer the execution. This is the case for instance when the user can change some parameters during the execution to improve the convergence of a computation.

3.3.1. User-in-the-loop

Some applications, like virtual reality applications, must comply with interactivity constraints. The user should be able to observe and interact with the application with an acceptable reaction delay. To reach this goal the user is often ready to accept a lower level of details. To execute such application on a distributed architecture requires to balance the workload and activation frequency of the different tasks. The goal is to optimize CPU and network resource use to get as close as possible to the reactivity/level of detail the user expect.
Virtual reality environments significantly improve the quality of the interaction by providing advanced interfaces. The display surface provided by multiple projectors in CAVE-like systems for instance, allows a high resolution rendering on a large surface. Stereoscopic visualization gives an information of depth. Sound and haptic systems (force feedback) can provide extra information in addition to visualized data. However driving such an environment requires an important computation power and raises difficult issues of synchronization to maintain the overall application coherent while guaranteeing a good latency, bandwidth (or refresh rate) and level of details. We define the coherency as the fact that the information provided to the different user senses at a given moment are related to the same simulated time.

Today’s availability of high performance commodity components including networks, CPUs as well as graphics or sound cards make it possible to build large clusters or grid environments providing the necessary resources to enlarge the class of applications that can aspire to an interactive execution. However the approaches usually used for mid size parallel machines are not adapted. Typically, there exist two different approaches to handle data exchange between the processes (or threads). The synchronous (or FIFO) approach ensures all messages sent are received in the order they were sent. In this case, a process cannot compute a new state if all incoming buffers do not store at least one message each. As a consequence, the application refresh rate is driven by the slowest process. This can be improved if the user knows the relative speed of each module and specify a read frequency on each of the incoming buffers. This approach ensures a strong coherency but impact on latency. This is the approach commonly used to ensure the global coherency of the images displayed in multi-projector environments. The other approach, the asynchronous one, comes from sampling systems. The producer updates data in a shared buffer asynchronously read by the consumer. Some updates may be lost if the consumer is slower than the producer. The process refresh rates are therefore totally independent. Latency is improved as produced data are consumed as soon as possible, but no coherency is ensured. This approach is commonly used when coupling haptic and visualization systems. A fine tuning of the application usually leads to satisfactory results where the user does not experience major incoherences. However, in both cases, increasing the number of computing nodes quickly makes infeasible hand tuning to keep coherency and good performance.

We propose to develop techniques to manage a distributed interactive application regarding the following criteria:

- latency (the application reactivity);
- refresh rate (the application continuity);
- coherency (between the different components);
- level of detail (the precision of computations).

We developed a programming environment, called FlowVR, that enables the expression and realization of loosen but controlled coherency policies between data flows. The goal is to give users the possibility to express a large variety of coherency policies from a strong coherency based on a synchronous approach to an uncontrolled coherency based on an asynchronous approach. It enables the user to loosen coherency where it is acceptable, to improve asynchronism and thus performance. This approach maximizes the refresh rate and minimizes the latency given the coherency policy and a fixed level of details. It still requires the user to tune many parameters. In a second step, we are planning to explore auto-adaptive techniques that enable to decrease the number of parameters that must be user tuned. The goal is to take into account (possibly dynamically) user specified high level parameters like target latencies, bandwidths and levels of details, and to have the system automatically adapt to reach a trade-off given the user wishes and the resources available. Issues include multi-criterion optimizations, adaptive algorithmic schemes, distributed decision making, global stability and balance of the regulation effort.

### 3.3.2. Expert-in-the-loop

Some applications can be interactively guided by an expert who may give advices or answer specific questions to hasten a problem resolution. A theoretical framework has been developed in the last decade to define precisely the complexity of a problem when interactions with an expert is allowed. We are studying these
interactive proof systems and interactive complexity classes in order to define efficient interactive algorithms
dedicated to scheduling problems. This, in particular, applies to load-balancing of interactive simulations when
a user interaction can generate a sudden surge of imbalance which could be easily predicted by an operator.

3.4. Adaptive middleware for code coupling and data movements

Participants: Vincent Danjean, Thierry Gautier, Clément Pernet, Bruno Raffin, Jean-Louis Roch, Frédéric
Wagner.

This theme deals with the design and implementation of programming interfaces in order to achieve an efficient
coupling of distributed components.

The implementation of interactive simulation application requires to assemble together various software
components and to ensure a semantic on the displayed result. To take into account functional aspects of the
computation (inputs, outputs) as well as non functional aspects (bandwidth, latency, persistence), elementary
actions (method invocation, communication) have to be coordinated in order to meet some performance
objective (precision, quality, fluidity, etc). In such a context the scheduling algorithm plays an important role
to adapt the computational power of a cluster architecture to the dynamic behavior due to the interactivity.
Whatever the scheduling algorithm is, it is fundamental to enable the control of the simulation. The purpose
of this research theme is to specify the semantics of the operators that perform components assembling and to
develop a prototype to experiment our proposals on real architectures and applications.

3.4.1. Application Programming Interface

The specification of an API to compose interactive simulation application requires to characterize the
components and the interaction between components. The respect of causality between elementary events
ensures, at the application level, that a reader will see the last write with respect to an order. Such a consistency
should be defined at the level of the application to control the events ordered by a chain of causality. For
instance, one of the result of Athapascan was to prove that a data flow consistency is more efficient than other
ones because it generates fewer messages. Beyond causality based interactions, new models of interaction
should be studied to capture non predictable events (delay of communication, capture of image) while ensuring
a semantic.

Our methodology is based on the characterization of interactions required between components in the context
of an interactive simulation application. For instance, criteria could be coherency of visualization, degree of
interactivity. Beyond such characterization we hope to provide an operational semantic of interactions (at least
well suited and understood by usage) and a cost model. Moreover they should be preserved by composition to
predict the cost of an execution for part of the application.

The main result relies on a computable representation of the future of an execution; representations such as
macro data flow are well suited because they explicit which data are required by a task. Such a representation
can be built at runtime by an interpretation technique: the execution of a function call is differed by computing
beforehand at runtime a graph of tasks that represents the (future) calls to execute.

3.4.2. Kernel for Asynchronous, Adaptive, Parallel and Interactive Application

Managing the complexity related to fine grain components and reaching high efficiency on a cluster architec-
ture require to consider a dynamic behavior. Also, the runtime kernel is based on a representation of the
execution: data flow graph with attributes for each node and efficient operators will be the basis for our soft-
ware. This kernel has to be specialized for the considered applications. The low layer of the kernel has features
to transfer data and to perform remote signalization efficiently. Well known techniques and legacy code have
to be reused. For instance, multithreading, asynchronous invocation, overlapping of latency by computing,
parallel communication and parallel algorithms for collective operations are fundamental techniques to reach
performance. Because the choice of the scheduling algorithm depends on the application and the architec-
ture, the kernel will provide an causally connected representation of the system that is running. This allows
to specialize the computation of a good schedule of the data flow graph by providing algorithms (scheduling
algorithms for instance) that compute on this (causally connected) representation: any modification of the representation is turned into a modification on the system (the parallel program under execution). Moreover, the kernel provides a set of basic operators to manipulate the graph (e.g. computes a partition from a schedule, remapping tasks, ...) to allow to control a distributed execution.
3. Scientific Foundations

3.1. Runtime Systems Evolution

This research project takes place within the context of high-performance computing. It seeks to contribute to the design and implementation of parallel runtime systems that shall serve as a basis for the implementation of high-level parallel middleware. Today, the implementation of such software (programming environments, numerical libraries, parallel language compilers, parallel virtual machines, etc.) has become so complex that the use of portable, low-level runtime systems is unavoidable.

Our research project centers on three main directions:

Mastering large, hierarchical multiprocessor machines  With the beginning of the new century, computer makers have initiated a long term move of integrating more and more processing units, as an answer to the frequency wall hit by the technology. This integration cannot be made in a basic, planar scheme beyond a couple of processing units for scalability reasons. Instead, vendors have to resort to organize those processing units following some hierarchical structure scheme. A level in the hierarchy is then materialized by small groups of units sharing some common local cache or memory bank. Memory accesses outside the locality of the group are still possible thanks to bus-level consistency mechanisms but are significantly more expensive than local accesses, which, by definition, characterizes NUMA architectures.

Thus, the task scheduler must feed an increasing number of processing units with work to execute and data to process while keeping the rate of penalized memory accesses as low as possible. False sharing, ping-pong effects, data vs task locality mismatches, and even task vs task locality mismatches between tightly synchronizing activities are examples of the numerous sources of overhead that may arise if threads and data are not distributed properly by the scheduler. To avoid these pitfalls, the scheduler therefore needs accurate information both about the computing platform layout it is running on and about the structure and activities relationships of the application it is scheduling.

As quoted by Gao et al. [59], we believe it is important to expose domain-specific knowledge semantics to the various software components in order to organize computation according to the application and architecture. Indeed, the whole software stack, from the application to the scheduler, should be involved in the parallelizing, scheduling and locality adaptation decisions by providing useful information to the other components. Unfortunately, most operating systems only provide a poor scheduling API that does not allow applications to transmit valuable hints to the system.

This is why we investigate new approaches in the design of thread schedulers, focusing on high-level abstractions to both model hierarchical architectures and describe the structure of applications' parallelism. In particular, we have introduced the bubble scheduling concept [14] that helps to structure relations between threads in a way that can be efficiently exploited by the underlying thread scheduler. Bubbles express the inherent parallel structure of multithreaded applications: they are abstractions for grouping threads which “work together” in a recursive way. We are exploring how to dynamically schedule these irregular nested sets of threads on hierarchical machines [7], the key challenge being to schedule related threads as closely as possible in order to benefit from cache effects and avoid NUMA penalties. We are also exploring how to improve the transfer of scheduling hints from the programming environment to the runtime system, to achieve better computation efficiency.
This is also the reason why we explore new languages and compiler optimizations to better use domain specific information. In the ANR project PetaQCD, we propose a new domain specific language, QIRAL, to generate parallel codes from high level formulations for Lattice QCD problems. QIRAL describes the formulation of the algorithms, of the matrices and preconditions used in this domain and generalizes languages such as SPIRAL used in auto-tuning library generator for signal processing applications. Lattice QCD applications require huge amount of processing power, on multinode, multi-core with GPUs. Simulation codes require to find new algorithms and efficient parallelization. So far, the difficulties for orchestrating parallelism efficiently hinder algorithmic exploration. The objective of QIRAL is to decouple algorithm exploration with parallelism description. Compiling QIRAL uses rewriting techniques for algorithm exploration, parallelization techniques for parallel code generation and potentially, runtime support to orchestrate this parallelism. Results of this work are submitted to publication.

For parallel programs running on multicores, measuring reliable performance and determining performance stability is becoming a key issue: indeed, a number of hardware mechanisms may cause performance instability from one run to the other. Thread migration, memory contention (on any level of the cache hierarchy), scheduling policy of the runtime can introduce some variation, independently of the program input. A speed-up is interesting only it corresponds to a performance that can be obtained through repeated execution of the application. Very few research efforts have been made in the identification of program optimization/runtime policy/hardware mechanisms that may introduce performance instability. We studied in [61] on a large set of OpenMP benchmarks performance variations, identified the mechanisms causing them and showing the need for better strategies for measuring speed-ups. Following this effort, we developed inside the tool MAQAO (Modular Assembler Quality Analyzer and Optimizer), the precise analysis of the interactions between OpenMP threads, through static analysis of binary codes and memory tracing. In particular, the influence of thread affinity is estimated and the tool proposes hints to the user to improve its OpenMP codes.

Aside from greedyly invading all these new cores, demanding HPC applications now throw excited glances at the appealing computing power left unharvested inside the graphical processing units (GPUs). A strong demand is arising from the application programmers to be given means to access this power without bearing an unaffordable burden on the portability side. Efforts have already been made by the community in this respect but the tools provided still are rather close to the hardware, if not to the metal. Hence, we decided to launch some investigations on addressing this issue. In particular, we have designed a programming environment named STARPU that enables the programmer to offload tasks onto such heterogeneous processing units and gives that programmer tools to fit tasks to processing units capability, tools to efficiently manage data moves to and from the offloading hardware and handles the scheduling of such tasks all in an abstracted, portable manner. The challenge here is to take into account the intricacies of all computation unit: not only the computation power is heterogeneous among the machine, but data transfers themselves have various behavior depending on the machine architecture and GPUs capabilities, and thus have to be taken into account to get the best performance from the underlying machine. As a consequence, STARPU not only pays attention to fully exploit each of the different computational resources at the same time by properly mapping tasks in a dynamic manner according to their computation power and task behavior by the means of scheduling policies, but it also provides a distributed shared-memory library that makes it possible to manipulate data across heterogeneous multicore architectures in a high-level fashion while being optimized according to the machine possibilities.

Optimizing communications over high performance clusters and grids Using a large panel of mechanisms such as user-mode communications, zero-copy transactions and communication operation offload, the critical path in sending and receiving a packet over high speed networks has been drastically reduced over the years. Recent implementations of the MPI standard, which have been carefully designed to directly map basic point-to-point requests onto the underlying low-level interfaces, almost reach the same level of performance for very basic point-to-point messaging requests. How-
ever more complex requests such as non-contiguous messages are left mostly unattended, and even more so are the irregular and multifold communication schemes. The intent of the work on our NEWMADELEINE communication engine, for instance, is to address this situation thoroughly. The NEWMADELEINE optimization layer delivers much better performance on complex communication schemes with negligible overhead on basic single packet point-to-point requests. Through Mad-MPI, our proof-of-concept implementation of a subset of the MPI API, we intend to show that MPI applications can also benefit from the NEWMADELEINE communication engine.

The increasing number of cores in cluster nodes also raises the importance of intra-node communication. Our KNEM software module aims at offering optimized communication strategies for this special case and let the above MPI implementations benefit from dedicated models depending on process placement and hardware characteristics.

Moreover, the convergence between specialized high-speed networks and traditional ETHERNET networks leads to the need to adapt former software and hardware innovations to new message-passing stacks. Our work on the OPEN-MX software is carried out in this context.

Regarding larger scale configurations (clusters of clusters, grids), we intend to propose new models, principles and mechanisms that should allow to combine communication handling, threads scheduling and I/O event monitoring on such architectures, both in a portable and efficient way. We particularly intend to study the introduction of new runtime system functionalities to ease the development of code-coupling distributed applications, while minimizing their unavoidable negative impact on the application performance.

Integrating Communications and Multithreading   Asynchronism is becoming ubiquitous in modern communication runtimes. Complex optimizations based on online analysis of the communication schemes and on the de-coupling of the request submission vs processing. Flow multiplexing or transparent heterogeneous networking also imply an active role of the runtime system request submit and process. And communication overlap as well as reactiveness are critical. Since network request cost is in the order of magnitude of several thousands CPU cycles at least, independent computations should not get blocked by an ongoing network transaction. This is even more true with the increasingly dense SMP, multicore, SMT architectures where many computing units share a few NICs. Since portability is one of the most important requirements for communication runtime systems, the usual approach to implement asynchronous processing is to use threads (such as Posix threads). Popular communication runtimes indeed are starting to make use of threads internally and also allow applications to also be multithreaded. Low level communication libraries also make use of multithreading. Such an introduction of threads inside communication subsystems is not going without troubles however. The fact that multithreading is still usually optional with these runtimes is symptomatic of the difficulty to get the benefits of multithreading in the context of networking without suffering from the potential drawbacks. We advocate the importance of the cooperation between the asynchronous event management code and the thread scheduling code in order to avoid such disadvantages. We intend to propose a framework for symbiotically combining both approaches inside a new generic I/O event manager.
3. Scientific Foundations

3.1. Introduction

The scientific foundations of our work are those of network design and network analysis. Specifically, this concerns the principles of packet switching and in particular of IP networks (protocol design, protocol testing, routing, scheduling techniques), and the mathematical and algorithmic aspects of the associated problems, on which our methods and tools are based.

These foundations are described in the following paragraphs. We begin by a subsection dedicated to Quality of Service (QoS) and Quality of Experience (QoE), since they can be seen as unifying concepts in our activities. Then we briefly describe the specific sub-area of models’ evaluation and about the particular multidisciplinary domain of network economics.

3.2. Quality of Service and Quality of Experience

Since it is difficult to develop as many communication solutions as possible applications, the scientific and technological communities aim towards providing general services allowing to give to each application or user a set of properties nowadays called “Quality of Service” (QoS), a terminology lacking a precise definition. This QoS concept takes different forms according to the type of communication service and the aspects which matter for a given application: for performance it comes through specific metrics (delays, jitter, throughput, ...), for dependability it also comes through appropriate metrics: reliability, availability, or vulnerability, in the case for instance of WAN (Wide Area Network) topologies, etc.

QoS is at the heart of our research activities: we look for methods to obtain specific “levels” of QoS and for techniques to evaluate the associated metrics. Our ultimate goal is to provide tools (mathematical tools and/or algorithms, under appropriate software “containers” or not) allowing users and/or applications to attain specific levels of QoS, or to improve the provided QoS, if we think of a particular system, with an optimal use of the resources available. Obtaining a good QoS level is a very general objective. It leads to many different areas, depending on the systems, applications and specific goals being considered. Our team works on several of these areas. We also investigate the impact of network QoS on multimedia payloads to reduce the impact of congestion.

Some important aspects of the behavior of modern communication systems have subjective components: the quality of a video stream or an audio signal, as perceived by the user, is related to some of the previous mentioned parameters (packet loss, delays, ...) but in an extremely complex way. We are interested in analyzing these types of flows from this user-oriented point of view. We focus on the user perceived quality, the main component of what is nowadays called Quality of Experience (in short, QoE), to underline the fact that, in this case, we want to center the analysis on the user. In this context, we have a global project called PSQA, which stands for Pseudo-Subjective Quality Assessment, and which refers to a methodology allowing to automatically measuring the QoE (see 6.5 ).

Another special case to which we devote research efforts in the team is the analysis of qualitative properties related to interoperability assessment. This refers to the act of determining if end-to-end functionality between at least two communicating systems is as required by the base standards for those systems. Conformance is the act of determining to what extent a single component conforms to the individual requirements of the standard it is based on. Our purpose is to provide such a formal framework (methods, algorithms and tools) for interoperability assessment, in order to help in obtaining efficient interoperability test suites for new generation networks, mainly around IPv6-related protocols. The interoperability test suites generation is based on specifications (standards and/or RFCs) of network components and protocols to be tested.
3.3. Stochastic modeling

The scientific foundations of our modeling activities are composed of stochastic processes theory and, in particular, Markov processes, queuing theory, stochastic graphs theory, etc. The objectives are either to develop numerical solutions, or analytical ones, or possibly discrete event simulation or Monte Carlo (and Quasi-Monte Carlo) techniques. We are always interested in models’ evaluation techniques for dependability and performability analysis, both in static (network reliability) and dynamic contexts (depending on the fact that time plays an explicit role in the analysis or not). We look at systems from the classical so-called call level, leading to standard models (for instance, queues or networks of queues) and also at the burst level, leading to fluid models.

In recent years, our work on the design of the topologies of WANs led us to optimization techniques, in particular in the case of very large optimization problems, usually formulated in terms of graphs. The associated methods we are interested in are composed of simulated annealing, genetic algorithms, TABU search, etc. For the time being, we have obtained our best results with GRASP techniques.

Network pricing is a good example of a multi-disciplinary research activity half-way between applied mathematics, economy and networking, centered on stochastic modeling issues. Indeed, the Internet is facing a tremendous increase of its traffic volume. As a consequence, real users complain that large data transfers take too long, without any possibility to improve this by themselves (by paying more, for instance). A possible solution to cope with congestion is to increase the link capacities; however, many authors consider that this is not a viable solution as the network must respond to an increasing demand (and experience has shown that demand of bandwidth has always been ahead of supply), especially now that the Internet is becoming a commercial network. Furthermore, incentives for a fair utilization between customers are not included in the current Internet. For these reasons, it has been suggested that the current flat-rate fees, where customers pay a subscription and obtain an unlimited usage, should be replaced by usage-based fees. Besides, the future Internet will carry heterogeneous flows such as video, voice, email, web, file transfers and remote login among others. Each of these applications requires a different level of QoS: for example, video needs very small delays and packet losses, voice requires small delays but can afford some packet losses, email can afford delay (within a given bound) while file transfer needs a good average throughput and remote login requires small round-trip times. Some pricing incentives should exist so that each user does not always choose the best QoS for her application and so that the final result is a fair utilization of the bandwidth. On the other hand, we need to be aware of the trade-off between engineering efficiency and economic efficiency; for example, traffic measurements can help in improving the management of the network but is a costly option. These are some of the various aspects often present in the pricing problems we address in our work. More recently, we have switched to the more general field of network economics, dealing with the economic behavior of users, service providers and content providers, as well as their relations.
3. Scientific Foundations

3.1. Overview of the needed paradigms

Management of telecommunications networks and services, and Web services, involves the following algorithmic tasks:

Observing, monitoring, and testing large distributed systems: Alarm or message correlation is one of the five basic tasks in network and service management. It consists in causally relating the various alarms collected throughout the considered infrastructure—be it a network or a service sitting on top of a transport infrastructure. Fault management requires in particular reconstructing the set of all state histories that can explain a given log of observations. Testing amounts to understanding and analyzing the responses of a network or service to a given set of stimuli; stimuli are generally selected according to given test purposes. All these are variants of the general problem of observing a network or service. Networks and services are large distributed systems, and we aim at observing them in a distributed way as well, namely: logs are collected in a distributed way and observation is performed by a distributed set of supervising peers.

Quality of Service (QoS) evaluation, negotiation, and monitoring: QoS issues are a well established topic for single domain networks or services, for various protocols — e.g., Diffserv for IP. Performance evaluation techniques are used that follow a “closed world” point of view: the modeling involves the overall traffic, and resource characteristics are assumed known. These approaches extend to some telecommunication services as well, e.g., when considering (G)MPLS over an IP network layer.

However, for higher level applications, including composite Web services (also called orchestrations, this approach to QoS is no longer valid. For instance, an orchestration using other Web services has no knowledge of how many users are calling the same Web services. In addition, it has no knowledge of the transport resources it is using. Therefore, the well developed “closed world” approach can no longer be used. Contract based approaches are considered instead, in which a given orchestration offers promises to its users on the basis of promises it has from its subcontracting services. In this context, contract composition becomes a central issue. Monitoring is needed to check for possible breaching of the contract. Countermeasures would consist in reconfiguring the orchestration by replacing the failed subcontracted services by alternative ones.

The DistribCom team focuses on the algorithms supporting the above tasks. Therefore models providing an adequate framework are fundamental. We focus on models of discrete systems, not models of streams or fluid types of models. And we address the distributed and asynchronous nature of the underlying systems by using models involving only local, not global, states, and local, not global, time. These models are reviewed in section 3.2. We use these mathematical models to support our algorithms and we use them also to study and develop formalisms of Web services orchestrations and workflow management in a more general setting.

3.2. Models of concurrency: nets, scenarios, event structures, and their variants

For Finite State Machines (FSM), a large body of theory has been developed to address problems such as: observation (the inference of hidden state trajectories from incomplete observations), control, diagnosis, and learning. These are difficult problems, even for simple models such as FSM’s. One of the research tracks of DistribCom consists in extending such theories to distributed systems involving concurrency, i.e., systems in which both time and states are local, not global. For such systems, even very basic concepts such as “trajectories” or “executions” need to be deeply revisited. Computer scientists have for a long time recognized this topic of concurrent and distributed systems as a central one. In this section, we briefly introduce the reader to the models of scenarios, event structures, nets, languages of scenarios, graph grammars, and their variants.
3.2.1. Scenarios.

The simplest concept related to concurrency is that of a finite execution of a distributed machine. To this end, scenarios have been informally used by telecom engineers for a long time. In scenarios, so-called “instances” exchange asynchronous messages, thus creating events that are totally ordered on a given instance, and only partially ordered by causality on different instances (emission and reception of a message are causally related). The formalization of scenarios was introduced by the work done in the framework of ITU and OMG on High-level Message Sequence Charts and on UML Sequence Diagrams in the last ten years, see [67], [73]. This allowed in particular to formally define infinite scenarios, and to enhance them with variables, guards, etc [77], [75], [76]. Today, scenarios are routinely offered by UML and related software modeling tools.

3.2.2. Event structures.

The next step is to model sets of finite executions of a distributed machine. Event structures were invented by Glynn Winskel and co-authors in 1980 [72], [78]. Executions are sets of events that are partially ordered by a causality relation. Event structures collect all the executions by superimposing shared prefixes. Events not belonging to a same execution are said in conflict. Events that are neither causally related nor in conflict are called concurrent. Concurrent processes model the “parallel progress” of components. Categories of event structures have been defined, with associated morphisms, products, and co-products, see [79]. Products and co-products formalize the concepts of parallel composition and “union” of event structures, respectively. This provides the needed apparatus for composing and projecting (or abstracting) systems. Event structures have been mostly used to give the semantics of various formalisms or languages, such as Petri nets, CCS, CSP, etc [72], [78]. We in DistribCom make a nonstandard use of these, e.g., we use them as a structure to compute and express the solutions of observation or diagnosis problems, for concurrent systems.

3.2.3. Nets and languages of scenarios.

The next step is to have finite representations of systems having possibly infinite executions. In DistribCom, we use two such formalisms: Petri nets [74], [59] and languages of scenarios such as High-level Message Sequence Charts (HMSC) [67], [76]. Petri nets are well known, at least in their basic form, we do not introduce them here. We use so-called safe Petri Nets, in which markings are boolean (tokens can be either 0 or 1); and we use also variants, see below.

3.2.4. Extensions and variants.

Two extensions of the basic concepts of nets or scenario languages are useful for us. Nets or scenario languages enriched with variables, actions, and guards, are useful to model general concurrent and distributed dynamical systems in which a certain discrete abstraction of the control is represented by means of a net or a scenario language. Manipulating such symbolic nets requires using abstraction techniques. Time Petri nets and network of timed automata are particular cases of symbolic nets. Probabilistic Nets or event structures: Whereas a huge literature exists on stochastic Petri nets or stochastic process algebras (in computer science), randomizing concurrent models, i.e., with ω’s being concurrent trajectories, not sequential ones, has been addressed only since the 21st century. We have contributed to this new area of research.

3.2.5. Handling dynamic changes in the systems.

The last and perhaps most important issue, for our applications, is the handling of dynamic changes in the systems model. This is motivated by the constant use of dynamic reconfigurations in management systems. Extensions of net models have been proposed to capture this, for example the dynamic nets of Vladimiro Sassone [58] and net systems [60]. For the moment, such models lack a suitable theory of unfoldings.

3.3. Modal logics for distributed systems

Modal logics are a family of logics that were developed originally to reason about different modalities occurring in natural language, such as for example the modality of knowledge (epistemic logic), the modalities of obligation and permission (deontic logic) and the modality of time (temporal logic). Temporal logics (CTL, LTL, μ-calculus...) are the most prominent (modal) logics used in computer science nowadays, especially in the field of verification.
3.3.1. Epistemic logic and distributed systems.

In the 1980’s, epistemic logic was propounded by computer scientists such as Fagin, Halpern, Moses and Vardi to address problems in distributed systems, resulting in the TARK conference series (Theoretical Aspects of Rationality and Knowledge) and the books [63], [70]. This interest in epistemic logic was due to their observation that the notion of knowledge plays a central role in the informal reasoning used in the design of distributed protocols. This lead these authors to “hope that a theory of knowledge, communication and action will prove rich enough to provide general foundations for a unified theoretical treatment of distributed systems” [66]. The research pursued in DistribCom follows this line of thought, although we also strive to feed and confront our theoretical developments with actual problems stemming from diverse areas of application of distributed systems.

In [63], the behaviour of a distributed system is represented by a set of runs, each run being a possible execution of the distributed system, determined by a given protocol. Processors are called agents and their partial observation of the system is represented at any point in the run by indistinguishability relations between local states of different runs (the local state of a processor represents the state of this processor at a moment of time). This model was used to show for example that the specific notion of common knowledge of epistemic logic is necessary to reach agreement and to coordinate actions [66]. Dynamic Epistemic Logic (DEL) is another logical framework that can be used to represent and reason about distributed systems (connections between these two logical frameworks were made in [80]). DEL deals with the representation of global states of synchronous distributed systems. The global state of the system at a moment in time is represented directly by means of an epistemic model. Events occurring in this distributed system are represented by means of event models and their effects on the local states of agents (processors) are represented by means of a product update.

The contributions in this sub-module are described in Section 6.4.

3.3.2. Deontic logic and privacy in distributed systems.

We also use deontic logic in combination with epistemic logic for the formalization of privacy regulations. We intend to use this formalization to reason about privacy in the composition of web-services. The combination of these two modal logics can be used to express statements such as “it is forbidden for agent 1 to know that agent 2 sent message m” or “if agent 1 is an administrator of the system, then it is permitted for him to know information i”. This provides a formal language very close to the natural language used in actual privacy regulations by law legislators. In the long run, we expect this formal language to be used at the level of interfaces of the web-service in order to:

1. check that the privacy policy declared by the web-service on its interface is indeed compliant (coherent) with respect to the privacy regulations expressed by law makers;
2. check that the web-service does enforce and apply the privacy policy it has declared on its interface.

The contributions in this sub-module are described in Section 6.9.

3.4. Statistical Model Checking

Complex systems pose two particular challenges to formal verification: (i) the non-determinism caused by concurrency and unpredictable environmental conditions and (ii) the size of the state space. Our interest is probabilistic model checking, that can verify intricate details of a system’s dynamical behaviour and where non-determinism is handled by assigning probabilistic distributions to unknowns and quantifying results with a probability. Exact probabilistic model checking quantifies these probabilities to the limit of numerical precision by an exhaustive exploration of the state space, but is restricted by what can be conveniently stored in memory. Our focus is therefore statistical model checking (SMC), that avoids an explicit representation of the state space by building a statistical model of the executions of a system and giving results within confidence bounds. The key challenges of this approach are to reduce the length (simulation steps and cpu time) and number of simulation traces necessary to achieve a result with given confidence. Rare properties pose a particular problem in this respect, since they are not only difficult to observe but their probability is
difficult to bound. A further goal is to make a tool where the choice of modelling language and logic are flexible.
DNET Team

3. Scientific Foundations

3.1. Sensor network, distributed measure and distributed processing

**Participants:** Guillaume Chelius, Eric Fleury, Andreea Chis, Clément Burin des Rosiers, Sandrine Avakian, Guillaume Roche, Fabien Jammes.

**Glossary**

**Sensor network for distributed embedded measure.** In order to gather information on the dynamic of a specific physical phenomena, a distributed embedded measure must be performed. The quality of the measure is crucial and largely impacts the analysis. Moreover, by conducting and controlling the measure and its bias during the experiment, one may adapt and optimize the analysis.

Sensors networks offer an efficient way to measure physical phenomena at various space and time scales. The important challenge is to take advantage of a communicating sensor node that can be associated to a physical object in order to design a large scale distributed measurement system that can monitor and sense the physical world. Given a target application, the goal is to design adequate sensor nodes and to set up the way they communicate, cooperate and collect their data in order to fulfill the application constraints. Fulfilling this goal requires the development of theoretical and practical techniques to help the dimensioning and deployment of such distributed sensing tools, to manage the distributed measures and to perform efficient and reliable distributed computing on top of the network. With these main tasks in mind, we define the following objectives:

**Design of a global sensing tool.** Based on the deployment context, we should propose a methodology to design the most appropriate and accurate measurement architecture that matches the application constraints. Heterogeneity is a fundamental, beneficial quality of distributed sensing tool, not just a problem to overcome. Heterogeneous sensing systems are more immune to the weaknesses of sensing modalities and more robust against defective, missing, or malicious data sources than even carefully designed homogeneous systems. However, data heterogeneity also presents main challenges when trying to integrate data from many different sensors.

**Measure characterization and dimensioning.** Measure characterization and dimensioning must take into account the different correlations in space and time that exist between all sensors. The challenge is due to the heterogeneous data resolution. Moreover, data is generally multi modal and multi scale with possible irregularities and offer much correlation in time and/or space. Data sets collected by various sensors may be characterized by the lack of most common simple statistical properties such as stationary, linearity, or Gaussianity. Relevant time scales may be difficult to identify, or may even not exist. Observed properties have non-trivial relations and even the choice of the time scale granularity that should be used in the measure and the analysis is a complex problem as it may bias the analysis in an uncontrolled way.

**In-network distributed processing.** As some computation can be delocalized closer to the sensed phenomena, in the sensor nodes, space/time correlations can be exploited in order to optimize the amount of data sent through the network.

3.2. Statistical Characterization of Complex Interaction Networks

**Participants:** Guillaume Chelius, Eric Fleury, Adrien Friggeri.

**Glossary**

**Evolving networks can be regarded as "out of equilibrium" systems.** Indeed, their dynamics is typically characterized by non standard and intricate statistical properties, such as non-stationarity, long range memory effects, intricate space and time correlations.
The dynamics of complex networks often exhibit no preferred time scale or equivalently involve a whole range of scales and are characterized by a scaling or scale invariance property. Another important aspect of network dynamics resides in the fact that the sensors measure information of different nature. For instance, in the MOSAR project, inter-individual contacts are registered, together with the health status of each individual, and the time evolution of the resistance to antibiotics of the various strains analyzed. Moreover, such information is collected with different and unsynchronized resolutions in both time and space. This property, referred to as multi-modality, is generic and central in most dynamical networks. With these main challenges in mind, we define the following objectives.

From "primitive" to "analyzable" data: Observables. The various and numerous modalities of information collected on the network generate a huge "primitive" data set. It has first to be processed to extract "analyzable data", which can be envisioned with different time and space resolutions: it can concern either local quantities, such as the number of contacts of each individual, pair-wise contact times and durations, or global measures, e.g., the fluctuations of the average connectivity. The first research direction consists therefore in identifying, from the "primitive data", a set of "analyzable data" whose relevance and meaningfulness for the analysis of network dynamic and network diffusion phenomena will need to be assessed. Such "analyzable data" needs also to be extracted from large "primitive data" set with "reasonable" complexity, memory and computational loads.

Granularity and resolution. The corresponding data will take the form of time-series, "condensing" network dynamics description at various granularity levels, both in time and space. For instance, the existence of a contact between two individuals can be seen as a link in a network of contacts. Contact networks corresponding to contact sequences aggregated at different analysis scales (potentially ranging from hours to days or weeks) can be built. However, it is so far unclear to which extent the choice of the analysis scale impacts the relevance of network dynamics description and analysis. An interesting and open issue lies in the understanding of the evolution of the network from a set of isolated contacts (when analyzed with low resolution) to a globally interconnected ensemble of individuals (at large analysis scale). In general, this raises the question of selecting the adequate level of granularity at which the dynamics should be analyzed. This difficult problem is further complicated by the multi-modality of the data, with potentially different time resolutions.

(non-)Stationarity. Stationarity of the data is another crucial issue. Usually, stationarity is understood as a time invariance of statistical properties. This very strong definition is difficult to assess in practice. Recent efforts have put forward a more operational concept of relative stationarity in which an observation scale is explicitly included. The present research project will take advantage of such methodologies and extend them to the network dynamics context. The rationale is to compare local and global statistical properties at a given observation scale in time, a strategy that can be adapted to the various time series that can be extracted from the data graphs so as to capture their dynamics. This approach can be given a statistical significance via a test based on a data-driven characterization of the null hypothesis of stationarity.

Dependencies, correlations and causality. To analyze and understand network dynamics, it is essential that (statistical) dependencies, correlations and causalities can be assessed among the different components of the "analyzable data". For instance, in the MOSAR framework, it is crucial to assess the form and nature of the dependencies and causalities between the time series reflecting e.g., the evolution along time of the strain resistance to antibiotics and the fluctuations at the inter-contact level. However, the multimodal nature of the collected information together with its complex statistical properties turns this issue into a challenging task. Therefore, Task1 will also address the design of statistical tools that specifically aim at measuring dependency strengths and causality directions amongst multivariate signals presenting these difficulties. The objective is to provide elements of answers to natural yet key questions such as: Does a given property observed on different components of the data result from a same and single network mechanism controlling the ensemble or rather stem from different and independent causes? Do correlations observed on one instance of information (e.g., topological) command correlations for other modalities? Can directionality in correlations
(causality) be inferred amongst the different components of multivariate data? These should also shed complementary lights on the difficulties and issues associated to the identification of "important" nodes or links...

### 3.3. Theory and Structural Dynamic Properties of dynamic Networks

**Participants:** Guillaume Chelius, Christophe Crespelle, Eric Fleury, Qinna Wang.

**Glossary**

**Characterization of the dynamics of complex networks.** We need to focus on intrinsic properties of evolving/dynamic complex networks. New notions (as opposed to classical static graph properties) have to be introduced: rate of vertices or links appearances or disappearances, the duration of link presences or absences. Moreover, more specific properties related to the dynamics have to be defined and are somehow related to the way to model a dynamic graph.

To go further in the Classical graph notions like the definition of path, connected components and \(k\)-core have to be revisited in this context. Advanced properties need also to be defined in order to apprehend the intrinsic dynamic structural issues of such dynamic graphs. The notion of communities (dense group of nodes) is important in any social / interaction network context and may play an important role within an epidemic process. To transpose the static graph community concept into the dynamical graph framework is a challenging task and appears necessary in order to better understand how the structure of graphs evolves in time. In these context we define the following objectives:

**Toward a dynamic graph model and theory.** We want to design new notions, methods and models for the analysis of dynamic graphs. For the static case, graph theory has defined a vast and consistent set of notions and methods such as paths, flows, centrality measures. These notions and methods are completely lacking for the study of dynamic graphs. We aim at providing such notions in order to study the structure of graphs evolving in time and the phenomenon taking place on these dynamic graphs. Our approach relies on describing a dynamic graph by a series of graphs which are the snapshots of the state of the graph at different moments of its life. This object is often poorly used: most works focus on the structure of each graph in the series. Doing so, one completely forget the relationships between the graphs of the series. We believe that these relationships encompass the essence of the structure of the dynamic and we place it at the very center of our approach. Thus, we put much effort on developing graph notions able to deal with a series of graphs instead of dealing with a single graph. These notions must capture the temporal causality of the series and the non trivial relationships between its graphs. Our final goal is to provide a set of the notions and indicators to describe the dynamics of a network in a meaningful way, just like complex networks theory does for static complex networks.

**Dynamic communities.** The detection of dynamic communities is particularly appealing to describe dynamic networks. In order to extend the static case, one may apply existing community detection methods to successive snapshots of dynamic networks. This is however not totally satisfying for two main reasons: first, this would take a large amount of time (directly proportional to the data span); moreover, having a temporal succession of independent communities is not sufficient and we lose valuable information and dependencies. We also need to investigate the temporal links, study the time granularity and look for time periods that could be compressed within a single snapshot.

**Tools for dynamic graph visualization.** Designing generic and pure graph visualization tools is clearly out of the scope of the D-NET project. Efficient graph drawing tools or network analysis toolkit/software are now available (e.g., GUESS, TULIP, Sonivis, Network Workbench). However, the drawback of most softwares is that the dynamics is not taken into account. Since we will study the hierarchy of dynamics through the definition of communities we plan to extend graph drawing methods by using the communities’ structures. We also plan to handle the time evolution in the network analysis toolkit. A tool like TULIP is well designed and could be improved by allowing operations (selection, grouping, sub graph computation...) to take place on the time dimension as well.
GANG Project-Team (section vide)
3. Scientific Foundations

3.1. Analytical information theory

Participant: Philippe Jacquet.

Glossary

Information theory Branch of mathematics dedicated to the quantification of the performance of a medium to carry information. Initiated by Shannon in 1948.

Abstract. Information theory and analytical methods play a central role in the networking technology. It identifies the key parameter that must be quantified in order to characterize the performance of a network.

The analytical information theory is part of the foundations of the Hipercom project. This is a tool box that has been collected and adapted from the areas of the analysis of algorithms and the information theory. It provides powerful tool for the analysis of telecommunication algorithms. The analysis of the behavior of such algorithms in their asymptotic range are fundamental in order to identify their critical parts. It helps to design and properly scale the protocols. Application of analytical information theory ranges from channel capacity computations, compression algorithm performance evaluation, predictor designs.

3.2. Methodology of telecommunication algorithm evaluation

Participants: Cédric Adjih, Emmanuel Baccelli, Thomas Clausen, Philippe Jacquet, Salman Malik, Yacine Mezali, Pascale Minet, Paul Mühlethaler, Yasser Toor.

Glossary

Power laws probability distributions that decays has inverse power of the variable for large values of the variable. Power laws are frequent in economic and statistical analysis (see Pareto law). Simple models such as Poisson processes and finite state Markov processes don’t generate distributions with power laws.

We develop our performance evaluation tools towards deterministic performance and probabilistic performance. Our tools range from mathematical analysis to simulation and real life experiment of telecommunication algorithms.

One cannot design good algorithms without good evaluation models. Hipercom project team has an historically strong experience in performance evaluation of telecommunication systems, notably when they have multiple access media. We consider two main methodologies:

- Deterministic performance analysis,
- Probabilistic performance analysis

In the deterministic analysis, the evaluation consists to identify and quantify the worst case scenario for an algorithm in a given context. For example to evaluate an end-to-end delay. Mathematically it consists into handling a \((\max,+)^\) algebra. Since such algebra is not commutative, the complexity of the evaluation of an end-to-end delay frequently grows exponentially with the number of constraints. Therefore the main issue in the deterministic evaluation of performance is to find bounds easier to compute in order to have practical results in realistic situations.
In the probabilistic analysis of performance, one evaluate the behavior of an algorithm under a set of parameters that follows a stochastic model. For example traffic may be randomly generated, nodes may move randomly on a map. The pioneer works in this area come from Knuth (1973) who has systemized this branch. In the domain of telecommunication, the domain has started a significant rise with the appearance of the problematic of collision resolution in a multiple access medium. With the rise of wireless communication, new interesting problems have been investigated.

The analysis of algorithm can rely on analytical methodology which provides the better insight but is practical in very simplistic models. Simulation tools can be used to refine results in more complicated models. At the end of the line, we proceed with real life experiments. To simplify, experiments check the algorithms with 10 nodes in maximum, simulations with 100 nodes maximum, analytical tools with more 1,000 nodes, so that the full range of applicability of the algorithms is investigated.

3.3. Network traffic and architecture models

Participants: Cédric Adjih, Philippe Jacquet, Aline Carneiro Viana, Salman Malik, Yacine Mezali.

Abstract. Network models are important. We consider four model problems: topology, mobility, dynamics and traffic models.

One needs good and realistic models of communication scenarios in order to provide pertinent performance evaluation of protocols. The models must assess the following key points:

- The architecture and topology: the way the nodes are structured within the network
- The mobility: the way the nodes move
- The dynamics: the way the nodes change status
- The traffic: the way the nodes communicate

For the architecture there are several scales. At the internet scale it is important to identify the patterns which dictate the node arrangement. For example the internet topology involves many power law distribution in node degree, link capacities, round trip delays. These parameters have a strong impact in the performance of the global network. At a smaller scale there is also the question how the nodes are connected in a wireless network. There is a significant difference between indoor and outdoor networks. The two kinds of networks differ on wave propagation. In indoor networks, the obstacles such as walls, furniture, etc, are the main source of signal attenuations. In outdoor networks the main source of signal attenuation is the distance to the emitter. This lead to very different models which vary between the random graph model for indoor networks to the unit graph model for outdoor networks.

The mobility model is very important for wireless network. The way nodes move may impact the performance of the network. For example it determines when the network splits in distinct connected components or when these components merge. With random graph models, the mobility model can be limited to the definition of a link status holding time. With unit disk model the mobility model will be defined according to random speed and direction during random times or random distances. There are some minor complications on the border of the map.

The node dynamic addresses the elements that change inside the node. For example its autonomy, its bandwidth requirement, the status of server, client, etc. Pair to pair networks involve a large class of users who frequently change status. In a mobile ad hoc network, nodes may change status just by entering a coverage area, or because some other nodes leaves the coverage area.
The traffic model is very most important. There are plenty literature about traffic models which arose when Poisson models was shown not to be accurate for real traffics, on web or on local area networks. Natural traffic shows long range dependences that don’t exist in Poisson traffic. There are still strong issues about the origin of this long range dependences which are debated, however they have a great impact on network performance since congestions are more frequent. The origin are either from the distribution of file sizes exchanged over the net, or from the protocols used to exchange them. One way to model the various size is to consider on/off sources. Every time a node is on it transfers a file of various size. The TCP protocol has also an impact since it keeps a memory on the network traffic. One way to describe it is to use an on/off model (a source sending packets in transmission windows) and to look at the superposition of these on/off sources.

3.4. Algorithm conception and implementation

Participants: Cédric Adjih, Aline Carneiro Viana, Emmanuel Baccelli, Thomas Clausen, Philippe Jacquet, Pascale Minet, Paul Mühlenthaler, Yasser Toor, Ichrak Amdouni, Ridha Soua, Erwan Livolant.

Abstract. Algorithms are designed with focal point on performance. The algorithms we specify in detail range between medium access control to admission control and quality of service management.

The conception of algorithms is an important focus of the project team. We specify algorithms in the perspective of achieving the best performance for communication. We also strive to embed those algorithms in protocols that involve the most legacy from existing technologies (Operating systems, internet, Wifi). Our aim with this respect is to allow code implementations for real life experiment or imbedded simulation with existing network simulators. The algorithm specified by the project ranges from multiple access schemes, wireless ad hoc routing, mobile multicast management, Quality of Service and admission controls. In any of these cases the design emphasizes the notions of performance, robustness and flexibility. For example, a flooding technique in mobile ad hoc network should be performing such to save bandwidth but should not stick too much close to optimal in order to be more reactive to frequent topology changes. Some telecommunication problems have NP hard optimal solution, and an implementable algorithm should be portable on very low power processing unit (e.g. sensors). Compromise are found are quantified with respect to the optimal solution.
3. Scientific Foundations

3.1. Evolutionary needs in network and service management

The foundation of the MADYNES research activity is the ever increasing need for automated monitoring and control within networked environments. This need is mainly due to the increasing dependency of both people and goods towards communication infrastructures as well as the growing demand towards services of higher quality. Because of its strategic importance and crucial requirements for interoperability, the management models were constructed in the context of strong standardization activities by many different organizations over the last 15 years. This has led to the design of most of the paradigms used in today's deployed approaches. These paradigms are the Manager/Agent interaction model, the Information Model paradigm and its container, together with a naming infrastructure called the Management Information Base. In addition to this structure, five functional areas known under the FCAPS\(^1\) acronym are associated to these standards.

While these models were well suited for the specific application domains for which they were designed (telecommunication networks or dedicated protocol stacks), they all show the same limits. Especially they are unable:

1. to deal with any form of dynamicity in the managed environment,
2. to master the complexity, the operating mode and the heterogeneity of the emerging services,
3. to scale to new networks and service environments.

These three limits are observed in all five functional areas of the management domain (fault, configuration, accounting, performance and security) and represent the major challenges when it comes to enable effective automated management and control of devices, networks and services in the next decade.

MADYNES addresses these challenges by focusing on the design of management models that rely on inherently dynamic and evolving environments. The project is centered around two core activities. These activities are, as mentioned in the previous section, the design of an autonomous management framework and its application to three of the standard functional areas namely security, configuration and performance.

3.2. Autonomous management

3.2.1. Models and methods for a self-management plane

Self organization and automation are fundamental requirements within the management plane in today’s dynamic environments. It is necessary to automate the management processes and enable management frameworks to operate in time sensitive evolving networks and service environments. The automation of the organization of devices, software components, networks and services is investigated in many research projects and has already led to several solution proposals. While these proposals are successful at several layers, like IP auto-configuration or service discovery and binding facilities, they did not enhance the management plane at all. For example, while self-configuration of IP devices is commonplace, no solution exists that provides strong support to the management plane to configure itself (e.g. finding the manager to which an agent has to send traps or organizing the access control based on locality or any other context information). So, this area represents a major challenge in extending current management approaches so that they become self-organized.

Our approach is bottom-up and consists in identifying those parameters and framework elements (manager data, information model sharing, agent parameters, protocol settings, ...) that need dynamic configuration and self-organization (like the address of a trap sink). For these parameters and their instantiation in various management frameworks (SNMP, Netconf, WBEM, ...), we investigate and elaborate novel approaches enabling fully automated setup and operation in the management plane.

\(^1\)Fault, Configuration, Accounting, Performance and Security
3.2.2. Design and evaluation of P2P-based management architectures

Over the last years, several models have emerged and gained wide acceptance in the networking and service world. Among them, the overlay networks together with the P2P paradigms appear to be very promising. Since they rely mainly on fully decentralized models, they offer excellent fault tolerance and have a real potential to achieve high scalability. Mainly deployed in the content delivery and the cooperation and distributed computation disciplines, they seem to offer all features required by a management framework that needs to operate in a dynamic world. This potential however needs an in depth investigation because these models have also many characteristics that are unusual in management (e.g. a fast and uncontrolled evolution of the topology or the existence of a distributed trust relationship framework rather than a standard centralized security framework).

Our approach envisions how a complete redesign of a management framework is done given the characteristics of the underlying P2P and overlay services. Among the topics of interest we study the concept of management information and operations routing within a management overlay as well as the distribution of management functions in a multi-manager/agent P2P environment. The functional areas targeted in our approach by the P2P model are network and service configuration and distributed monitoring. The models are to be evaluated against highly dynamic frameworks such as ad-hoc environments (network or application level) and mobile devices.

3.2.3. Integration of management information

Representation, specification and integration of management information models form a foundation for network and service management and remains an open research domain. The design and specification of new models is mainly driven by the appearance of new protocols, services and usage patterns. These need to be managed and exposed through well designed management information models. Integration activities are driven by the multiplication of various management approaches. To enable automated management, these approaches need to inter-operate which is not the case today.

The MADYNES approach to this problem of modeling and representation of management information aims at:

1. enabling application developers to establish their management interface in the same workspace, with the same notations and concepts as the ones used to develop their application,
2. fostering the use of standard models (at least the structure and semantics of well defined models),
3. designing a naming structure that allows the routing of management information in an overlay management plane, and
4. evaluating new approaches for management information integration especially based on management ontologies and semantic information models.

3.2.4. Modeling and benchmarking of management infrastructures and activities

The impact of a management approach on the efficiency of the managed service is highly dependent on three factors:

- the distribution of the considered service and their associated management tasks,
- the management patterns used (e.g. monitoring frequency, granularity of the management information considered),
- the cost in terms of resources these considered functions have on the managed element (e.g. method call overhead, management memory footprint).

While the first factor was investigated in several research projects so far, none of the other two were investigated at all. The lack of such benchmarking data and models simply makes the objective evaluation of the operational costs of a management approach impossible. This may be acceptable in backbone networks where processing and communication resources can be tuned very easily (albeit sometimes at a non negligible cost). This is not true in constrained environments like devices constrained by battery or processing power as found in wireless networks for which the lack of management cost models is a serious concern.
MADYNES addresses this problem from multiple viewpoints: communication patterns, processing and memory resources consumption. Our goal is to provide management patterns combining several management technologies if needed so as to optimize the resources consumed by the management activity imposed by the operating environment.

Therefore, we establish abacuses for management frameworks and in parallel we collect data on current management practice. These data will form the core of the “Constraints-based management tuning activity” that we are working on and can be used for rigorous comparison among distribution and processing of management activities.

### 3.3. Functional areas

#### 3.3.1. Security management

Securing the management plane is vital. While several proposals are already integrated in the existing management frameworks, they are rarely used. This is due to the fact that these approaches are completely detached from the enterprise security framework. As a consequence, the management framework is “managed” separately with different models; this represents a huge overhead. Moreover the current approaches to security in the management plane are not inter-operable at all, multiplying the operational costs in a heterogeneous management framework.

The primary goal of the research in this activity is the design and the validation of a security framework for the management plane that will be open and capable to integrate the security services provided in today’s management architectures. Management security interoperability is of major importance in this activity.

Our activity in this area aims at designing a generic security model in the context of multi-party / multi-technology management interactions. Therefore, we develop research on the following directions:

1. Abstraction of the various access control mechanisms that exist in today’s management frameworks. We are particularly interested in extending these models so that they support event-driven management, which is not the case for most of them today.
2. Extension of policy and trust models to ease and to ensure coordination among managers towards one agent or a subset of the management tree. Provisional policies are of great interest to us in this context.
3. Evaluation of the adequacy of key distribution architectures to the needs of the management plane as well as selecting reputation models to be used in the management of highly dynamic environments (e.g. multicast groups, ad-hoc networks).

A strong requirement towards the future generic model is that it needs to be instantiated (with potential restrictions) into standard management platforms like SNMP, WBEM or Netconf and to allow interoperability in environments where these approaches coexist and even cooperate. A typical example of this is the security of an integration agent which is located in two management worlds.

Since 2006 we have also started an activity on security assessment. The objective is to investigate new methods and models for validating the security of large scale dynamic networks and services. The first targeted service is VoIP.

#### 3.3.2. Configuration: automation of service configuration and provisioning

Configuration covers many processes which are all important to enable dynamic networks. Within our research activity, we focus on the operation of tuning the parameters of a service in an automated way. This is done together with the activation topics of configuration management and the monitoring information collected from the underlying infrastructure. Some approaches exist today to automate part of the configuration process (download of a configuration file at boot time within a router, on demand code deployment in service platforms). While these approaches are interesting they all suffer from the same limits, namely:

1. they rely on specific service life cycle models,
2. they use proprietary interfaces and protocols.
These two basic limits have high impacts on service dynamics in a heterogeneous environment.

We follow two research directions in the topic of configuration management. The first one aims at establishing an abstract life-cycle model for either a service, a device or a network configuration and to associate with this model a generic command and programming interface. This is done in a way similar to what is proposed in the area of call control in initiatives such as Parlay or OSA.

In addition to the investigation of the life-cycle model, we work on technology support for distributing and exchanging configuration management information. Especially, we investigate policy-driven approaches for representing configurations and constraints while we study XML-based protocols for coordinating distribution and synchronization. Off and online validation of configuration data is also part of this effort.

3.3.3. Performance and availability monitoring

Performance management is one of the most important and deployed management function. It is crucial for any service which is bound to an agreement about the expected delivery level. Performance management needs models, metrics, associated instrumentation, data collection and aggregation infrastructures and advanced data analysis algorithms.

Today, a programmable approach for end-to-end service performance measurement in a client server environment exists. This approach, called Application Response Measurement (ARM) defines a model including an abstract definition of a unit of work and related performance records; it offers an API to application developers which allows easy integration of measurement within their distributed application. While this approach is interesting, it is only a first step toward the automation of performance management.

We are investigating two specific aspects. First we are working on the coupling and possible automation of performance measurement models with the upper service level agreement and specification levels. Second we are working on the mapping of these high level requirements to the lower level of instrumentation and actual data collection processes available in the network. More specifically we are interested in providing automated mapping of service level parameters to monitoring and measurement capabilities. We also envision automated deployment and/or activation of performance measurement sensors based on the mapped parameters. This activity also incorporates self-instrumentation (and when possible on the fly instrumentation) of software components for performance monitoring purpose.
3. Scientific Foundations

3.1. Scientific Foundations

The main mathematical tools and formalisms used in MAESTRO include:

- theory of stochastic processes: Markov process, point process, Palm measure, large deviations, branching process, mean-field approximation;
- theory of dynamical discrete-event systems: queues, fluid approximation;
- theory of control and scheduling: dynamic programming, Markov decision process, game theory, deterministic and stochastic scheduling, pathwise comparison;
- theory of singular perturbations;
- random matrix theory.
3. Scientific Foundations

3.1. Scientific Foundations

The project develops tools and theory in the following domains: Discrete Mathematics (in particular Graph Theory), Algorithmics, Combinatorial Optimization and Simulation.

Typically, a telecommunication network (or an interconnection network) is modeled by a graph. A vertex may represent either a processor or a router or any of the following: a switch, a radio device, a site or a person. An edge (or arc) corresponds to a connection between the elements represented by the vertices (logical or physical connection). We can associate more information both to the vertices (for example what kind of switch is used, optical or not, number of ports, equipment cost) and to the edges (weights which might correspond to length, cost, bandwidth, capacity) or colors (modeling either wavelengths or frequencies or failures) etc. Depending on the application, various models can be defined and have to be specified. This modeling part is an important task. To solve the problems, we manage, when possible, to find polynomial algorithms. For example, a maximum set of disjoint paths between two given vertices is by Menger’s theorem equal to the minimum cardinality of a cut. This problem can be solved in polynomial time using graph theoretic tools or flow theory or linear programming. On the contrary, determining whether in a directed graph there exists a pair of disjoint paths, one from \( s_1 \) to \( t_1 \) and the other from \( s_2 \) to \( t_2 \), is an NP-complete problem, and so are all the problems which aim at minimizing the cost of a network which can satisfy certain traffic requirements. In addition to deterministic hypotheses (for example if a connection fails it is considered as definitely down and not intermittently), the project started recently to consider probabilistic ones.

Graph coloring is an example of concept which appears in various contexts: WDM networks where colors represent wavelengths, radio networks where colors represent frequencies, fault tolerance where colors represent shared risk resource groups, and scheduling problems. Another tool concerns the development of new algorithmic aspects like parameterized algorithms.
3. Scientific Foundations

3.1. Experimental approach to Networking

Based on a practical view, the Planète approach to address the above research topics is to design new communication protocols or mechanisms, to implement and to evaluate them either by simulation or by experimentation on real network platforms (such as PlanetLab and OneLab). Our work includes a substantial technological component since we implement our mechanisms in pre-operational systems and we also develop applications that integrate the designed mechanisms as experimentation and demonstration tools. We also work on the design and development of networking experimentation tools such as the ns-3 network simulator and experimental platforms. We work in close collaboration with research and development industrial teams.

In addition to our experimentation and deployment specificities, we closely work with researchers from various domains to broaden the range of techniques we can apply to networks. In particular, we apply techniques of the information and queuing theories to evaluate the performance of protocols and systems. The collaboration with physicists and mathematicians is, from our point of view, a promising approach to find solutions that will build the future of the Internet.

In order to carry out our approach as well as possible, it is important to attend and contribute to IETF (Internet Engineering Task Force) and other standardization bodies meetings on a regular basis, in order to propose and discuss our ideas in the working groups related to our topics of interests.
3. Scientific Foundations

3.1. Design and Analysis of Algorithms

The general goal of the research in this domain is of designing algorithms to analyze and control the traffic of communication networks. The team is currently involved in the design of algorithms to allocate bandwidth in optical networks and also to allocate resources in content-centric networks. See the corresponding sections below.

3.2. Scaling of Markov Processes

The growing complexity of communication networks makes it more difficult to apply classical mathematical methods. For a one/two-dimensional Markov process describing the evolution of some network, it is sometimes possible to write down the equilibrium equations and to solve them. The key idea to overcome these difficulties is to consider the system in limit regimes. This list of possible renormalization procedures is, of course, not exhaustive. The advantages of these methods lie in their flexibility to various situations and to the interesting theoretical problems they raised.

A fluid limit scaling is a particularly important means to scale a Markov process. It is related to the first order behavior of the process and, roughly speaking, amounts to a functional law of large numbers for the system considered.

A fluid limit keeps the main characteristics of the initial stochastic process while some second order stochastic fluctuations disappear. In "good" cases, a fluid limit is a deterministic function, obtained as the solution of some ordinary differential equation. As can be expected, the general situation is somewhat more complicated. These ideas of rescaling stochastic processes have emerged recently in the analysis of stochastic networks, to study their ergodicity properties in particular.
RESO Project-Team

3. Scientific Foundations

3.1. Expertise

We follow a transversal approach that goes from the assessment of the systems to the implementation of the proposed analytical networking solutions on large-scale realistic platforms. The RESO team leverages 2 main assets to work out the raised challenges:

- a multidisciplinary competence comprising:
  - specialists in the design and the implementation of protocol architectures for advanced high performance cluster networks, long distance networks and the Internet
  - specialists of distributed systems and algorithms (wireless)
  - researchers in statistical signal processing, performance evaluation and queuing theory,
  - an experimented computer engineering team.

- an original and pioneer experimental facility relying on a large scale, independent and fully recon-figurable network plate-form (Grid5000), and on a versatile and scalable metrology system which permits fine packet-grain captures at 10Gbps and the measurement of energy consumed by the processing nodes.

3.2. Optimized Protocol implementations and networking equipments

Participants: Laurent Lefèvre, Paulo Gonçalves, Jean-Patrick Gelas, Olivier Glück, Fabienne Anhalt, Guilherme Koslovski.

The key enabling factor of new network services is programmability at every level; that is the ability for new software capabilities to self-configure themselves over the network. We explore the concept, "dynamic programming enablers" for dynamic service driven configuration of communication resources. Dynamic programming enablers apply to an executable service that is injected and activated into the network system elements to create the new functionality at runtime. The basic idea is to enable trusted parties (users, operators, and service providers) to activate management-specific service and network components into a specific platform. We study mechanisms and infrastructures required to support these components. We aim at providing new functionality to services using Internet facilities, addressing the self-management operations in differentiated and integrated services. The goal is the enhancement of the creation and the management (customization, delivery, execution and stop) of Internet services.

3.3. Quality of Service and Transport layer for Future Networks

Participants: Isabelle Guérin-Lassous, Paulo Gonçalves, Thomas Begin, Laurent Lefèvre, Guilherme Koslovski, Anne-Cécile Orgerie, Thiago Abreu, Doreid Ammar.

Recently, network operators have observed profound changes in the daily traffic. The number of applications that generate traffic has tremendously grown up. Numerous delay-sensitive applications (e.g., Telephony over IP) and resource-intensive applications (e.g., streaming video) are constantly emerging. This steady increase of applications, combined with intensive use, has significantly affected the utilization level of networking resources, and it might, ultimately, cause significant network congestions and performance disruptions. Therefore, quality of service issues are still open problems that require adapted solutions.
We investigate quality of service solutions for future networks. In particular, we focus our studies on the evaluation of key parameters for quality of service solutions, like for instance the remaining available bandwidth for wireless networks or the use of routers’ links in function of time and space. The integration of the dynamics of these parameters is one of our main goals. This work implies the design of algorithms and performance evaluation studies carried out analytically, by simulation or by experimentation.

3.4. Network services for high demanding applications

Participants: Laurent Lefèvre, Paulo Gonçalves, Olivier Glück, Jean-Patrick Gelas, Matthieu Imbert, Mohammed Diouri, Ashley Chonka.

In a grid environment, two key points in the communication layers have to be considered in order to execute efficiently high performance applications: the heterogeneity of high-speed interconnections composing the grid and the Wide Area Network used to achieve inter-site communications. Our expertise to explore new mechanisms for improving the application performance when executed on the grid, relies on MPI applications. In particular, we gained insights on how to mutualize the performances of several interconnected high-speed networks and how to leverage this factorization to improve a single execution outcome of a MPI application. As one major difficulty dwells in an efficient and reliable communication between the heterogeneous agents of a grid environment, we developed solid skills in handling communication mechanisms that are resilient to heterogeneous and long-distance interconnections.

3.5. Energy aware software components for large scale distributed systems

Participants: Laurent Lefèvre, Anne-Cécile Orgerie, Olivier Glück, Jean-Patrick Gelas, Mohammed Diouri, Ghislain Landry Tsafack, Olivier Mornard, Maxime Morel.

In large scale distributed systems (Datacenters, Grids, Clouds and Networks) it is now a crucial challenge to propose software environments able to support recent improvements in green leverages (ON/OFF, DVFS...), to monitor and expose resource usage and to support energy efficient approaches. RESO explores the proposition of energy aware schedulers (in Orgerie’s PhD), performance and energy prediction (in Tsafack’s PhD), energy usage simulators (ECOFEN tool) and green exascale infrastructures (Diouri’s PhD-) in the context of datacenters (PrimeEnergyIT project), clouds (CompatibleOne Project) and networks (GreenTouch project).

3.6. High Speed Network’s metrology

Participants: Paulo Gonçalves, Laurent Lefèvre, Thomas Begin, Matthieu Imbert, Shubhabraya Roy, Ashley Chonka, Anne-Cécile Orgerie, Ghislain Landry Tsafack.

Metrology of wide-area computer networks (i.e. the deployment of a series of tools allowing for collecting relevant information regarding the system status), is a discipline recently introduced in the context of networks, and which undergoes constant developments. In our case, this activity consists in measuring along time and space: (i) the nature and the amount of exchanged information between the constituents of a system ; (ii) the energy consumption by the different equipments of the system. In the one hand, it is then a matter of using the collected data to forecast the network load evolution, so as to dynamically anticipate congestion, and more widely, to guarantee a given application dependent Quality of Service. On the other hand, we strive at optimizing the resources usage from an energy efficiency viewpoint and at designing transport protocols for green networking.

We ground our approach on our large scale, fully controllable and configurable experimental facility (Grid5000+MetroFlux [53]+ShowWatts+WattM) to validate, to better understand and to extend anterior results that were either heuristically observed or theoretically derived. Conversely, we perform realistic experiments, under prescribed and reproducible conditions, to get new insights into the statistical specificities of internet traffic, and to precisely identify the role of the network parameters [52].

3.7. Statistical signal processing

Participants: Paulo Gonçalves, Thomas Begin, Shubhabraya Roy.
From a statistical signal processing viewpoint, collected traces correspond to (multivariate) time series principally characterized by non-properties: non-Gaussianity, non-stationarity, non-linearities, absence of a characteristic time scale (scale invariance). Our research activity is undertaking the development of reliable signal analysis and modeling tools aimed at identifying these (non-)properties in the specific context of computer network traffic. In the course, we intend to clarify the importance of granularity of measurements. More precisely, our interest lies in:

- Sampling theory;
- Large deviations principles;
- Estimation theory;
- Stochastic processes;
- Machine learning;
- Multi-resolution analysis.

3.8. Performance evaluation

Participants: Thomas Begin, Paulo Gonçalves, Shubhabraya Roy.

Broadly speaking, performance evaluation aims at quantifying the behavior of a system. To do this, we frequently have to rely our analysis on a theoretical model rather than directly observing the behavior of the system. Several reasons may explain this choice: the system is not instrumented or not available for measurements, analytical results are often faster to obtain and brings more insight, the system may still be at prototype stage, etc.

Constructive modeling basically consists to mimic the internal operations of a system in a theoretical model. To hold the complexity of the model at a tractable level, it is common to represent complex internal mechanisms by random variables so that the resulting model is stochastic. The choice of distributions for random variables is often driven by the our expertise on the system. We attempt to devise models that can be represented as Markov chains or as queueing model as this will ease their subsequent resolution.

The resolution of the theoretical model provides numerical values for customary performance parameters such as the steady-state distribution of the number of requests (packets) in the system, the average throughput, the rate of losses, the mean utilization rate of a resources, etc. Regarding the nature of the model and its complexity, we set up entirely (sometimes approximate) analytical solutions, numerical solutions or discrete-event simulations to assess the values of the sought performance parameters.
3. Scientific Foundations

3.1. General statement

SWING aims at supporting the extensive spawning of radio systems thanks to spontaneous, cooperative and self-organization mechanisms which have to offer higher QoS such as system capacity or real-time capabilities under complementary constraints such as security or energy efficiency. Its research fields cover flexible radio node design, agile radio resource sharing, and autonomous wireless networking. These three main research axes are completed by three cross-layer actions that are optimization, security, and prototyping.

As explained in the previous section, these three complementary research axes flexible radio nodes, agile radio resource sharing and autonomous radio networking protocols present strong interactions and have to be considered as interdependent characteristics of a unique objective: finding nodes and network architectures that will make Internet of Things (IoT) a robust and efficient reality.

3.2. Flexible Radio Node design

To cope with terminal flexibility, the terminals’ complexity is constantly increasing with the multiplicity of radio interfaces on the market. It is nowadays impossible to develop an all-standards-compliant terminal since most standards are permanently under development. Therefore, the wireless convergence at the application level is possible only by superposing several radio interfaces within the same terminal. This strategy is neither cost efficient, nor optimal from the compactness points of view, and prohibiting any system adaptability. Further, the time to market of the new technologies is also slowed down, because the future evolutions are conditioned to the usual backward compatibility that often limits the capabilities of new technologies.

Since the seminal paper of J. Mitola in 1991, the concept of Software Radio appeared progressively as a key technology to offer adaptable and full-compliant radio systems. If a full software radio system remains futuristic, some software defined radio architectures are already on the market (picochip, EVP,Lyrtech SFF SDR, GNU radio platforms). Moreover, the concept of cognitive radio requires that the terminal is aware of the available radio resources on a wide set of frequencies. Several problems are still open:

- What will be the emerging hardware paradigm for software radio? Multi-processor on chip are now available but there are room for many experimental study to focus the target software radio computation model.
- What will be the programming model for software defined radio? What will be the emerging waveform description language? Some attempts have been made to define a radio virtual machine but research direction are still widely open.
- wireless nodes design offering to the upper layers the required agile radio capabilities for self-optimizing networks is still an open problem, especially to consider simultaneously the three objectives: QoS, Energy consumption and Security.
- What kind of RF architecture do software defined radio need? Wideband or multiband capabilities that are mandatory for dynamic spectrum sensing and resource management strongly constrain the RF design.

The first challenge is then the optimization of wireless nodes with respect to our three common objectives (QoS, Energy consumption and Security) while offering to the upper layers the required agile radio capabilities for self-optimizing networks.
3.3. Agile Radio Resource Sharing

For current systems, the radio access reliability is guaranteed by the strict reservation of a band to a specific application. This band is further divided into multiple sub-bands, allocated to specific service providers, base stations or mobile users. Such resource reservation is really too restrictive and a less stringent approach is mandatory to improve the radio resource usage efficiency.

On the opposite, the total deregulation already effective in the 2.4 GHz ISM band, already shows the inefficiency of actual uncoordinated resource sharing principles: the coexistence of more and more systems will quickly lead to unacceptable strong interference levels and hence major malfunctions. Therefore, the solutions will originate from a self-adaptive behavior. To achieve a true self-adaptive behavior, the radio nodes should be able to analyze their radio environment thanks to cognitive radio capabilities as mentioned in the previous challenge. Based on this knowledge, each radio node has to decide which resource it can use without impending the other (primary) users' performance. Both sensing scheduling followed by transmission scheduling necessitate the design of cognitive medium access procedures (C-MAC). If centralized approaches are possible, they however suffer from the lack of adaptability and strong algorithmic complexity. Promising approaches rather rely on distributed algorithms where nodes take decisions on their own knowledge. Specific mechanisms are required to ensure robustness and stability.

Complementarity, and beyond a simple decision of using a radio resource, cooperative approaches between heterogeneous radio nodes open still more perspectives to increase the system efficiency. An important issue relies on exploiting the relay channel concept introduced in 1971 by van der Meulen but mor recently extended to multiple sources and relays, while introducing simultaneously network coding capabilities. This problem is really open as the capacity bounds are not even known.

The second challenge consists in proposing distributed algorithms for resource sharing and cooperation, optimized with respect to our three common objectives (QoS, Energy consumption and Security) and exploiting cognitive radio.

3.4. Autonomous Wireless Networking

The previously described mechanisms allow to manage efficiently the radio resource in the neighborhood of a node by taking into account the different wireless interactions. Next, to support the expected massive deployment of wireless nodes on machines or things, the routing issue becomes a very complex problem and should be revisited in the context of distributed wireless networks, particularly if we want to take benefit from agile radio, opportunistic radio links and non-symmetric neighbors. Because of the large-scale dimension of the networks we consider that centralized approaches should be dismissed to the benefit of distributed and localized protocols: based on local information and local interactions, the aim is to synthesize a global behavior in terms of routing, data gathering, etc. The most important issues deal with capacity and resource management, activity scheduling, topology control and protocols adaptability to the evolution of the network topology. Because such features need to be human-free, they are often referred to as the self-* paradigm which will drive our research effort. Hence, cooperation among nodes is also a tool that can be considered at the networking layer. However, such cooperative techniques will be carefully designed since they can trigger additional overhead in the network thus reducing the benefits of adaptability. Since network topologies are constantly evolving due to the mobility of the nodes and the variability of the radio links properties, fault-tolerant protocols are needed to guarantee robustness and self-stabilization. Finally, this self-* paradigm opens critical security problems since transmission reliability and security will rely on many intermediate relays. We will look further to find mechanisms to ensure safe data transmissions in a fully interactive and reconfigurable digital world.

The third challenge deals with routing and self-organization mechanisms in large scale heterogeneous networks, optimized with respect to our three constraints (QoS, Energy consumption and Security), while exploiting cognitive and agile radio mechanisms.
3.5. Cross-layer approaches

Many teams over the world are dealing with either one or another of the previously mentioned research domains. However, putting together all skills to research some new concepts at the interface between these domains is less common. The bottom-up strategy will promote the development of realistic models from the node to the network behavior in terms of energy, complexity or performance evaluation. This strategy will also contribute to evaluate the impact of new node designs at the network level. The top-down strategy allows to define realistic scenarios, to develop large scale system requirements and specifications. This strategy will also promote the development of realistic interference models. Other aspects are truly transverse. For instance, ensuring security in distributed networks needs to be designed at the different levels, from the PHY layer security paradigm up to distributed mechanisms over large scale networks.

Our cross-layer framework is driven from three complementary points of view:

Performance and optimization – Performance evaluation and global optimization define a cross-layer axis of our project. In this action, we will be able to merge our contributions on smart wireless networks modeling using combinatorial and stochastic modeling tools. Global optimization is meant to describe system-wide behaviors and provide theoretical bounds on its performance, both by benchmarking the existing solutions and by guiding their improvement which will foster new developments. Our global optimization framework will progressively account for the software radio capabilities of the radio nodes, the properties of resource sharing algorithms and new self-* protocols. Realistic models of the wireless medium will be included, as well as refined models of adaptive protocols. This action will lead to three results: realistic models of smart wireless networks properties, global optimization and performance bounds derivation as well as distributed sub-optimal but feasible algorithms. This cross-layer axis on optimization is a necessity for developing new approaches and tools that are both efficient, provably reliable and relevant to the inherent cross-layer, dynamic and statistical nature of the systems under study.

Security – Security is one of the main cross-layer challenges of the SWING project. Security must be envisioned at each level, from hardware to routing protocols, in order to guarantee an end-to-end comprehensive security strategy. Moreover, in the context of embedded architectures, security related processing must be maintained to the least acceptable energy cost. The main challenges will then be the design of new energy efficient cryptographic primitives (in hardware and in software), the design of security mechanisms for routing protocols in order to preserve the networks from some specific attacks. The band deregulation and the on-the-fly adaptation reduces dangerously the access security. If cooperative mechanisms have to be used, the security of the various applications must be simultaneously guaranteed. Thus, security must be considered from a cross-layer perspective to allow cooperation at the physical layer while still protecting from malicious data access.

Prototyping – In SWING, we aim at addressing the challenges of smart wireless networks not only from a theoretical point of view, but also from a practical one, using simulations and prototypes. From our past experience, we acquired and developed several simulation tools. The CITI laboratory is also equipped with up-to-date radio design platforms allowing to test the embedded software radio systems, evaluate MIMO communications and perform real radio channel measurements. These skills have been acquired thanks to strong partnerships with the industrial community, which we plan to expand via new cooperations with Orange Labs, Alcatel-Lucent and other partners.
3. Scientific Foundations

3.1. Scientific Foundations

- **Modeling and performance analysis of wireless networks.** Our main focus was on cellular networks, mobile ad hoc networks (MANETs) and their vehicular variants called VANETs. Our main advances about wireless networks have been based on the development of analytical tools for their performance analysis and on new results from network information theory. Concerning cellular networks, the main questions bear on coverage and capacity in large CDMA networks when taking intercell interferences and power control into account. Our main focus has been on the design of: 1) a strategy for the densification and parameterization of UMTS and future OFDM networks that is optimized for both voice and data traffic; 2) new self organization and self optimization protocols for cellular networks e.g. for power control, sub-carrier selection, load balancing, etc.

Concerning MANETs, we investigated MAC layer scheduling algorithms, routing algorithms and power control. The MAC protocols we considered are based on Aloha and CSMA as well as their cognitive radio extensions. We investigated opportunistic routing schemes for MANETs and VANETs. The focus was on cross layer optimizations allowing one to maximize the transport capacity of multihop networks.

- **Theory of network dynamics.** TREC is pursuing the analysis of network dynamics by algebraic methods. The mathematical tools are those of discrete event dynamical systems: semi-rings, and in particular network calculus, ergodic theory, perfect simulation, stochastic comparison, inverse problems, large deviations, etc. Network calculus gives results on worst-case performance evaluation; ergodic theory is used to assess the stability of discrete event dynamical systems; inverse problem methods are used to estimate some network parameters from external observations and to design network probing strategies.

- **The development of stochastic geometry and random geometric graphs tools.** Stochastic geometry is a rich branch of applied probability which allows one to quantify random phenomena on the plane or in higher dimension. It is intrinsically related to the theory of point processes and also to random geometric graphs. Our research is centered on the development of a methodology for the analysis, the synthesis, the optimization and the comparison of architectures and protocols to be used in wireless communication networks. The main strength of this method is its capacity for taking into account the specific properties of wireless links, as well as the fundamental question of scalability.

- **Combinatorial optimization and analysis of algorithms.** In this research direction started in 2007, we build upon our expertise on random trees and graphs and our collaboration with D. Aldous in Berkeley. Sparse graph structures have proved useful in a number of applications from information processing tasks to the modeling of social networks. We obtained new results in this research direction: computation of the asymptotic for the rank of the adjacency matrix of random graphs, computation of the matching number and the b-matching number of large graphs. We also applied our result to design bipartite graph structures for efficient balancing of heterogeneous loads and to analyze the flooding time in random graphs.

- **Economics of networks.** The premise of this relatively new direction of research, developed jointly with Jean Bolot [SPRINT ATL and then TECHNICOLOR] is that economic incentives drive the development and deployment of technology. Such incentives exist if there is a market where suppliers and buyers can meet. In today’s Internet, such a market is missing. We started by looking at the general problem of security on Internet from an economic perspective. A new research direction...
started on the economic value of user localization in wireless networks. This led to an Infocom’11 paper. We also built on our expertise in random graphs to derive new insights concerning diffusion and cascading behavior in random (possibly clustered) networks.