UNIVERSITÀ DI PISA

DIPARTIMENTO DI INFORMATICA

TECHNICAL REPORT: TR-09-20

An approach to Mobile Grid platforms for the development and support of complex ubiquitous applications

Carlo Bertolli, Daniele Buono, Gabriele Mencagli
and Marco Vanneschi

October 29, 2009

ADDRESS: Largo B. Pontecorvo 3, 56127 Pisa, Italy. TEL: +39 050 2212700, FAX: +39 050 2212726

ABSTRACT

Several complex and time-critical applications require the existence of novel distributed, heterogeneous and dynamic platforms composed of a variety of fixed and mobile processing nodes and networks. Such platforms, that can be called Pervasive Mobile Grids, aim to merge the features of Pervasive Computing and High-performance Grid Computing onto a new emerging paradigm. In this Chapter we study a methodology for designing high-performance distributed computations, able to exploit the heterogeneity and dynamicity of Pervasive Grids, by expressing Adaptivity and Context Awareness directly at the application level. We describe a programming model approach, and we compare it with other existing research works in the field of Pervasive Mobile Computing, discussing the rationales of the requirements and the features of a novel programming model for the target platforms and applications. In order to exemplify the proposed methodology we introduce our evaluation framework ASSISTANT, and we provide some interesting future directions in this research field.

INTRODUCTION

An increasing number of critical applications require the existence of novel distributed, heterogeneous and dynamic ICT platforms composed of a variety of fixed and mobile processing nodes and networks. Notable examples of such applications are (but not limited to) risk and emergency management, disaster prevention, homeland security and *i*-mobility. These platforms are characterized by full virtualization of ubiquitous computing resources, data and knowledge bases and services, embedded systems, PDA devices, wearable computers and sensors, interconnected through fixed, mobile and ad-hoc networks. Wireless-based platforms, enabling the robust, flexible and efficient cooperation of mobile components, including both software components and human operators, are of special interest. Users themselves are part of the distributed platform. These platforms, that aim to merge the features of *Pervasive Computing* and of *Grid Computing* onto a new emerging paradigm for heterogeneous distributed platforms, can be called *Pervasive Mobile Grids* (Hingne, Joshi, Finin, Kargupta, & Houstis, 2003; Priol & Vanneschi, 2008).

Figure 1 shows an abstract view of a Pervasive Grid platform focusing on the heterogeneity of computing resources and on interconnection network technologies.

The Pervasive Grid paradigm implies the development, deployment, execution and management of applications that, in general, are dynamic in nature. Dynamicity concerns the number and the specific identification of cooperating components, the deployment and composition of the most suitable versions of software components, processing and networking resources and services, i.e., both the quantity and the quality of the application components to achieve the needed *Quality of Service* (QoS). The specification and requirements of QoS itself are varying dynamically during the application, according to the *user intentions* and to the information produced by sensors and services, as well as according to the monitored state and performance of networks and nodes.

The general reference point for this kind of platforms is the *Grid* paradigm (Bermam, Fox, & Hey, 2003; Foster & Kesselman, 2003) which, by definition, aims to enabling the access, selection and aggregation of a variety of distributed and heterogeneous resources and services. However, though notable advancements have been achieved in recent years, current Grid technology is not yet able to supply the needed software tools with the features of high adaptivity, ubiquity, proactivity, self-organization, scalability and performance, interoperability, as well as fault tolerance and security, of the emerging applications running on a very large number of fixed and mobile nodes connected by various kinds of networks.

Pervasive Grid applications include *data- and compute-intensive processing* (e.g. forecasting and decision support models) not only for off-line centralized activities, but also for on-line and decentralized activities. Consider the execution of software components performing a forecasting model or of a decision support system model, which are critical compute-intensive activities to be executed respecting operational real-time deadlines. In “normal” connectivity

conditions we are able to execute these components on a centralized server, exploiting its processing power to achieve the highest performance as possible. Critical conditions in the application scenario (e.g. in emergency management) can lead to different user requirements (e.g. increasing the performance to complete the forecasting computation within a given, new deadline). Changes in network conditions (e.g. network failures or congestion situations) can lead to the necessity to execute a version of the application model directly on spatially local resources which are available to the users (e.g. personnel, rescuers, emergency managers and stakeholders): when central servers are not available or reachable, such resources are interface nodes and/or mobile devices themselves. In such cases, the forecasting model can be executed on different or additional computing resources, including a set of distributed mobile resources running different application software versions which are specifically defined and designed to exploit such kind of resources. In other words, in this scenario it is important to assure the service continuity, adapting the application to different user requirements but also to the so-called *context*: the actual conditions of the both the surrounding environment and the computing and communication platform. So the key-issue is the definition of programming paradigms, models, and frameworks to design and develop these kinds of complex and dynamic applications, focusing on *Adaptivity* and *Context Awareness* as crucial issues to be solved and to be integrated with high-performance and real-time features.

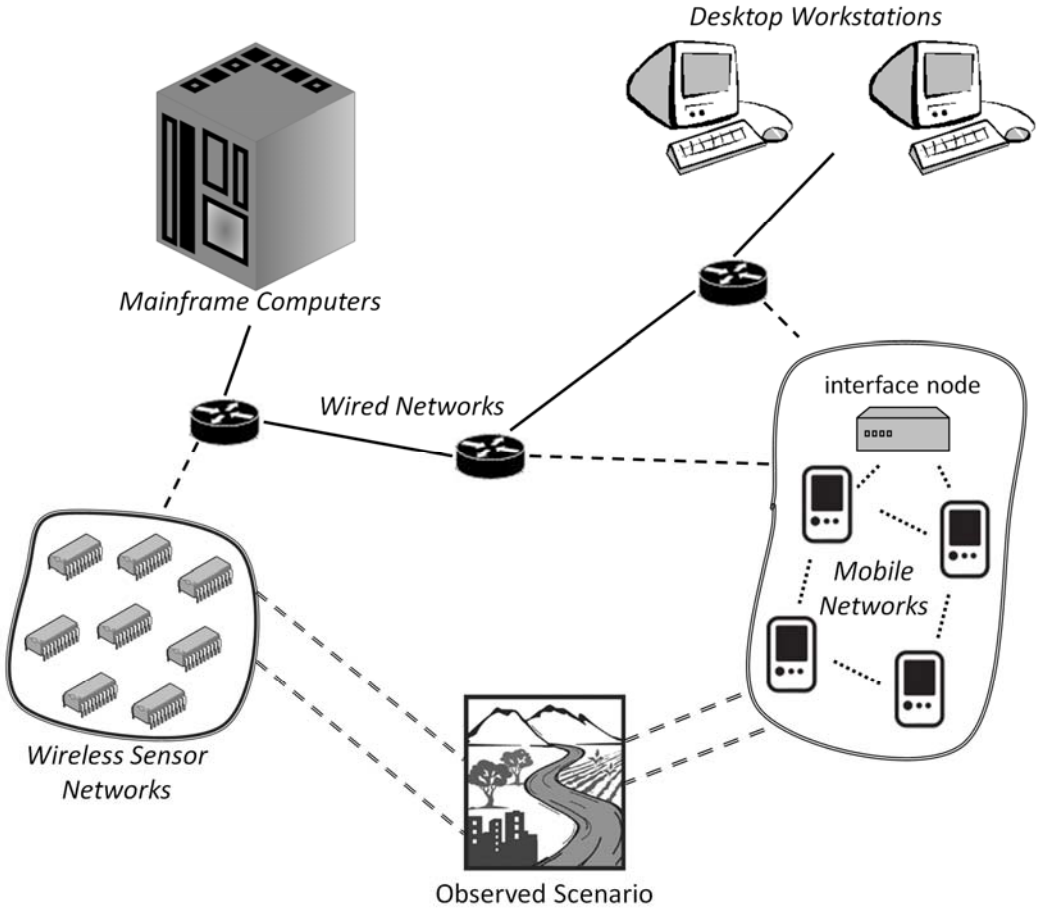

Figure 1. A schematic view of a Pervasive Grid infrastructure.

So, in Pervasive Grids various, heterogeneous fixed and mobile computers (e.g. PDA, wearable devices, new generation handphones) and networks must be able to capillary provide users with the necessary services in various connectivity, processing and location-based conditions. According to the current trends in computer technology, interface nodes and mobile devices can be equipped with very powerful, parallel computing resources, such as *multi-/many-core* components or GPUs, thus *rendering the embedding of compute intensive functions quite feasible at low power dissipation*. These devices can be part of self-configuring ad-hoc/mesh networks, in such a way that they can cooperatively form a

distributed embedded system executing specific application components, as well as being are able to cooperate with centralized servers (e.g. a workstation cluster) and wired networks.

In this Chapter we study a methodology for designing high-performance computations, able to exploit the heterogeneity and dynamicity of Pervasive Grids, by expressing Adaptivity and Context Awareness directly at the application level. We describe a programming model approach, and we compare it with other existing research works in the field of Pervasive Mobile Computing, discussing the rationales of the requirements and the features of a novel programming model for the target platforms and applications. As a consequence we discuss the advantages of a programming model methodology with respect to some standard middleware-based solutions. As a concrete example, we introduce a partial view of the ASSISTANT programming model, which is intended to be a first starting-point to express the identified main features of the Pervasive Grid approach. Finally we describe a set of interesting and opened research problems in the field of complex ubiquitous applications, concerning with a unified approach to define processing and communication strategies and with different methodologies to express application adaptivity and context-awareness.

BACKGROUND

In this section we describe the state of the art concerning adaptive and context-aware applications for pervasive distributed platforms. In particular our objective is to describe how adaptivity and dynamicity are expressed, focusing on the expressiveness of different approaches.

In many cases adaptivity is expressed *at the run-time support level* only. During the execution, the run-time system can select different protocols, algorithms or alternative implementations of the same mechanisms, in response to specific events which describe the actual context situation. This support level is often called *middleware* (Emmerich, 2003): a set of common services, operating on lower level resources, utilized by distributed cooperating application components. At the middleware level adaptivity can be expressed by a proper static or dynamic selection of different service implementations, or by setting specific parameters of configurable primitives. In many instances of this approach all the reconfigurations and adaptation processes are fully invisible to the applications.

In other research works adaptivity is a key-issue which is directly expressed *at the application level*. Mechanisms and tools are provided that allow programmers to define how their applications can be reconfigured and what the sensed events are. Applications can be defined in such a way that multiple *versions* of the same component or module are defined, and a proper *version selection strategy* must be expressed by the programmer. So, adaptation strategies and policies are directly part of the application semantics, which can be characterized by a functional part and a control logic (manager) expressing the adaptive behavior of the application.

Odyssey (Noble et al., 1997; Noble, 2000) is a research framework for the definition of *mobile* applications able to adapt their behavior, and especially their resource utilization, according to the actual state of the surrounding execution environment. The framework features run-time reconfigurations which are noticed by the final users as a change in the application execution *quality*. In Odyssey this quality concept is called *fidelity*: a fidelity decrease leads to a lower utilization of system resources (e.g. memory occupation and battery consumption). The framework periodically controls these system resources, and interacts with applications raising or lowering the corresponding fidelity levels. In the case of Odyssey, all these reconfigurations are automatically activated by the run-time system without any user intervention. For instance, in a media player application the fidelity can be the available compression of the played audio file, which can be dynamically selected according to the actual available network bandwidth.

In Odyssey applications are composed of two distinct parts: the first one produces input data according to a certain fidelity level, and the second one executes the visualization activities on

the previous data. The first part of each application is managed by a set of specific framework components called *Warden*. Each warden produces data with the predefined fidelity level, and they are coordinated by a unique entity called *Viceroy*. The viceroy is responsible for centralized resource management, for monitoring the resource utilization level and it handles incoming application requests routing them to the proper wardern. If the actual resource level is outside a defined range (i.e. window of tolerance), applications are notified via upcalls. Applications respond to these notifications by changing their fidelity level and using different wardens. The communications and interactions between the two application parts are managed by a kernel module (*Interceptor*), which extends the operation system features providing resource monitoring activities.

In Odyssey adaptivity is performed by a collaborative interaction between the run-time system (i.e. operating system or middleware) and the individual applications. This approach encourages a coordinated adaptivity between different applications which is not completely subsumed by the run-time system. As a counterpart, the fidelity concept (which is a key-point of this approach) is application-dependent: in general, it is not possible to define generic fidelity variation strategies which can be parametrically configured for every applications. Another relevant consideration is the narrow relationship between the fidelity level and the quality of visualized data: the mobile parts of Odyssey applications exploit only visualization activities. This assumption can be restrictive when we consider more complex applications involving an intensive cooperation between computation, communication and visualization. In this case adaptivity must concern not only the quality of the visualized data, but also optimized algorithms, protocols and the performance of critical computations.

In ***Aura*** (Garlan, Siewiorek, Smailagic, & Steenkiste, 2002) the *heterogeneity* of Pervasive Grid platforms is the main issue that has been faced. For each resource type proper applications exist, which make it possible to fully exploit the underlying device features. As an example a word-editor for a smartphone has probably less features than a standard one, but it is able to utilize the device touchpad. In Aura adaptivity is expressed introducing the abstract concept of *Task*: a specific work that a user has submitted to the system (e.g. write a document). A task can be completed by many applications (called services or *Suppliers*), and the framework dynamically utilizes the most-suitable service. The framework executes all the support activities to migrate a task from an application to another. Consider the following situation: a user must prepare his presentation for a meeting and he uses the personal computer localized in his office. Then the user is late, so he must leave the office and complete his presentation by using a mobile device (e.g. his PDA device). Aura framework takes care of all the necessary reconfiguration and adaptation processes. So, the user's partial work is automatically transferred to his PDA and transformed for the mobile application.

Aura framework is composed of a set of different layers. The task manager (called *Prism*) analyzes context information (e.g. user location and motion) guessing the user intentions. Context data are obtained by means of a *Context Observer* (i.e. a set of sensor devices and the corresponding raw data interpretation activities). Service Suppliers represent all the services that are able to execute a specific submitted task. They are implemented by wrapping existing applications providing the predefined Aura interfaces. These interfaces make it possible to extract all the useful information from the actual utilized service, and employ this information as a partial computed task which can be completed by a different supplier.

In this framework application adaptivity is expressed by selecting the most proper service, according to environmental data (e.g. the user location) obtained from sensor devices. It is an example of adaptivity mainly expressed at the run-time system level: each service supplier is a standard application not aware of any adaptation process. The run-time support decides the service selection strategies by using interpreted context data, but this is not directly part of the application semantics. In particular Aura essentially considers very simple applications (e.g. write a presentation). On the other hand, if we consider more complex mobile applications (e.g. executing a forecasting model for disaster prevention), transferring a partial computed

task to a different supplier can be a critical issue. As an alternative to Aura's approach, programmers could exploit the structure of the computation providing the transformations and the adaptivity logic necessary to complete a partial task by using a different supplier (e.g. changing the sequential algorithm and/or the parallelism pattern).

Cortex (Chang, Hee Kim, & Kim, 2007) is a programming model for adaptive context-aware applications, focusing on *time-critical* distributed applications (e.g. automatic car control systems and air traffic control avoiding collisions). For these applications it is very critical to properly manage the system response time without any centralization point in the underlying system architecture and adapting the application components to lead the system into a safe state, even in case of unexpected environmental changes. As an example, an air traffic system controls thousands of airplanes during their taking-off and landing phases, preserving the safe distances and avoiding traffic congestions.

In Cortex an application is composed of a set of *Sentient Objects*. Each object is a small context-aware system which can cooperate with the other objects by means of asynchronous events. A sentient object has a set of sensors to obtain context data and a set of actuators (i.e. physical devices capable of real-world actuations). Sensor data can pre-processed executing data-fusion techniques and interpreted by using a specific hierarchical *Context Model*. The most important part is the *Inference Engine*: interpreted context data are utilized to infer new facts and situations by using a set of rules which the programmer can express in *CLISP* (C Language Integrated Production System).

Cortex is a very interesting approach to context-aware systems, especially in the case of developing applications capable of perceiving the state of the surrounding environment, operating independently of human control, and being proactive (i.e. being anticipatory and taking own decisions without the user intervention). This research work presents many positive features, though it is mainly an ad-hoc solution for mobile control systems. Programming the inference engine by means of CLISP rules and using the corresponding context model can be a difficult task, critical for the system response and the adaptive behavior of applications. It requires very skilled programmers and the management code could be very difficult to be reused for other applications.

MB++ (Lillethun, Hilley, Horrigan, & Ramachandran, 2007) is a framework for developing *compute-intensive* applications in Pervasive Grid environments. Such applications are pervasive (i.e. designed for small mobile devices) *and* require also the execution of high-performance computations performed by HPC centralized resources (e.g. a cluster architecture). Typical examples are transformations on data streams (e.g. data-fusion, format conversion, feature extraction and classifications). These applications are described as data-flow graphs, whose nodes are transformations on data streams and the results are visualized by mobile nodes. An example of MB++ application is a metropolitan-area emergency response infrastructure. A large set of input data are obtained from pervasive and sensor devices: e.g. traffic cameras, mobile devices from local police and alarms located in specific buildings. These data are made available for monitoring activities, but they are also useful for executing complex real-time analysis (e.g. forecasting models and decision support systems) by using HPC centralized resources.

MB++ system architecture is composed of some clients, which are mobile devices producing or consuming information, and a set of HPC resources which execute the main system components: the *Type Server*, the *Stream Server* and a set of *Transformation Engines*. Type server dynamically manages data type definitions for each stream and all the transformation requests received from the clients. Stream server is responsible for executing data-flow graphs submitted by clients. A *Scheduler*, inside the stream server, enqueues the received graphs in specific command queues for each transformation engine. A transformation engine is executed on each HPC resource present in the system. The stream server allocates data-flow

graphs (or part of them) onto a set of transformation engines, whereas the source code of corresponding transformations are provided by the type server.

MB++ is one of the first research works focusing on high-performance computations in pervasive scenarios. The data-flow graph assignment is performed statically by the stream server when the graphs are allocated for the first time. So, in specific situations, we are not able to obtain a load-balancing execution as in other approaches (Danelutto & Dazzi, 2006). In MB++ adaptivity and context awareness are not expressed and there are no interactions between mobile devices (except those with the stream server). In particular client mobile devices execute only pre-processing or post-processing activities, whereas data-flow graphs can be executed on HPC resources only. In many other critical scenarios, such as emergency response systems, we require also the possibility to dynamically execute real-time intensive computations on a distributed set of localized mobile resources.

In this section we have presented the actual state of the art concerning adaptive and context-aware systems. From our point of view there is not a unified approach for programming large pervasive grid infrastructures, especially for defining time-critical ubiquitous applications. Some research works focus on HPC computations in real-time environments, but in these approaches the "pervasive part" of application definition is essentially missing. It means that there are no tools, programming constructs or methodologies to manage and define interactions with sensor devices and to manage context information by means of proper knowledge models. Other research works achieve the necessary expressiveness to define context-aware and adaptive applications, but they do not face on intensive real-time computations performed by HPC centralized resources nor by distributed systems of mobile devices.

A PROGRAMMING MODEL APPROACH

Programming Pervasive Grid environments

The development of complex and time-critical applications for Pervasive Grids requires a novel approach which has not been completely faced in the previous research works. This approach must be characterized by a strong synergy between two different research fields: *Pervasive Computing* (Weiser, 1999; Hansmann, Merk, Nicklous, & Stober, 2003) and *Grid Computing* (Berman et al., 2003; Foster & Kasselman 2003). Both of them consist of a set of methodologies to define applications and systems for heterogeneous distributed execution environments, but this common objective is faced by adopting very different points of view. Pervasive Computing is centered upon the creation of systems characterized by a multitude of heterogeneous ubiquitous computing and communication resources, whose integration aims to offering seamless services to the users according to their current needs and intentions. In this scenario the main issue is to provide a complete integration between the final users and the surrounding execution platform. Currently, many Pervasive Computing projects favor an infrastructured approach based on some *middleware* architectures. On the other hand, Grid Computing focuses on the efficient execution of compute-intensive processes by using geographically distributed computing platforms. In this field, techniques to deal with the heterogeneity and the dynamicity of network and computing resources (e.g. scheduling, load balancing, data management) are more oriented towards the achievement of given levels of performance, efficiency and security.

Next generation Pervasive Grid platforms (Priol & Vanneschi, 2007) are still at the beginning: the integration of traditional applications *and* ubiquitous applications and devices is a field still requiring intensive theoretic and experimental research. The integration must provide *a proper combination of high-performance programming models and pervasive computing frameworks*, in such a way to express a QoS-driven adaptive behavior for critical high-performance applications. In the remaining part of this section we will identify the main features of this novel approach.

In the previous section we have described some research works concerning adaptive and context-aware systems for pervasive platforms. These approaches are fundamental for our purposes, although they are suitable for classes of pervasive infrastructures (e.g. smart houses and control systems) characterized by static environments only, like a room or a building, in which some centralized resources are identified. This assumption has simplified the system design (e.g. Odyssey and Aura), since critical components and support mechanisms can be performed by fixed entities, exploiting the necessary coordination between all the system resources including the mobile ones. Novel approaches must consider fully decentralized and mobile solutions, characterized by applications able to adapt their behavior according to the actual state of the application environment *and* of the *execution* environment: that is, the current performance and availability of networks and computing nodes are of special interest in the context definition.

Adaptivity makes it possible to face the dynamicity of the surrounding computing platform and to achieve and maintain specific QoS levels. We consider the term *QoS* as a set of metrics, reflecting the experienced behavior of an application such as: its memory occupation, battery consumption, the estimated performance (service time, response time), as well as the user degree of satisfaction, e.g. the precision of computed results. From this point of view the QoS concept is very similar as the fidelity level in Odyssey, but with crucial differences. First of all it is not only concerned with the quality of visualized data, but *all non-functional properties of applications* can be involved. A notable example is (but it is not limited to) the performance of an intensive computation which can be mapped onto different kind of nodes: the computation can adapt its performance by changing the number of utilized computing nodes (i.e. parallelism degree) and networks, the mapping between application modules and corresponding utilized resources, or modifying the behavior of some specific components (using different algorithms or parallelization schemes).

We want to study how to describe and design applications that are dynamically self-reconfiguring during their execution life. *Reconfigurations* can be triggered by analyzing monitored performance metrics and the actual state of the execution environment (e.g. node or network failures, presence of new available mobile nodes, or emergency conditions). So applications must be aware of their execution context (i.e. *Context Awareness*) obtaining this information by using proper monitoring services or exploiting sensor devices. A Pervasive Grid programming model must offer the necessary *programming constructs and methodologies* to describe the reconfigurations and the interactions between application components and the context data providers, interpreting also raw data by using proper Context Models: e.g. ontology-based approaches (Gruber, 1993; Uschold & Grunninger, 1996), key-value approaches or logic-based models (Baldauf, Dustdar, & Rosenberg, 2007).

We have identified the three main features to achieve the necessary expressiveness for programming complex ubiquitous applications: *high-performance*, *adaptivity* and *context awareness*. We focus on each of these individual issues in the remaining part of this section.

Features of a Programming Model approach

Expressing HPC computations

In large-scale distributed environments the development of high-performance dynamic applications is characterized by two distinct approaches: a low-level approach by using directly Grid middleware services, as stated in Mache (2006), and a high-level approach by using high-level programming models.

In the former case applications utilize some middleware services directly to control the Grid resources, leaving the programmer the full knowledge of middleware adaptation mechanisms and the full responsibility of their utilization.

In the high-level approach, instead, a uniform approach is provided: strategies to drive dynamic adaptivity are expressed in the same high-level formalism of the programming model, without having to deal with the implementation of adaptation mechanisms, in the same

way in which the programmer has no visibility of the implementation of the traditional programming constructs. This approach has several interesting features, in particular it reduces the design and development phases of complex ubiquitous applications and, at the same time, a good trade-off between programmability and performance can be achieved.

A high-level approach is the only solution to one of the most crucial issues in high-performance applications design, i.e. the so-called *performance portability*: defining parallel programs having a reasonable expectation about their performance, and in general their behavior, when they are executed on different architectures (e.g. a multiprocessor, a workstation cluster, a distributed system of pervasive devices or multicore components). Performance portability is even more important in Pervasive Grids, that must be able to dynamically reconfigure the applications onto very different and heterogeneous computing and communication resources.
Structured Parallel Programming (Cole, 2004) is a considerable high-level approach for developing highly-portable parallel applications. In this approach parallel programs are expressed by using well-known abstract *parallelism schemes* (e.g. task-farm, pipeline, data-parallel, divide&conquer), for which the implementation of communication and computation patterns are known. Performance portability can be exploited by using proper *performance models* for each specific scheme, which make it possible to measure and dynamically modify the application performance and its resource utilization (e.g. performance and memory utilization, battery consumption for mobile nodes). This feature renders it feasible the definition of efficient fault-tolerance (Bertolli, 2009) and adaptivity (Vanneschi & Veraldi, 2007) high-performance mechanisms, which, as seen in the Background section, are not present in other pervasive computing projects.

Expressing Adaptivity and application reconfigurations
Structured parallel programming is a valuable starting point, however it is not sufficient for a Pervasive Grid programming model, that must be characterized also by *reconfiguration mechanisms* to achieve adaptivity. We distinguish two kinds of reconfigurations: functional and non-functional ones.

Non-functional reconfigurations preserve the application semantics and involve non-functional parameters of a computation (e.g. its memory utilization, its performance, or power consumption). In parallel processing projects and in pervasive computing projects (notably Aura) an “invisible” approach to adaptivity is adopted, i.e. delegating the reconfiguration actions to the run time system, without introducing specific mechanisms visible to the programmer. However, *an invisible approach is not sufficient for complex ubiquitous applications*. Suppose to have an intensive computation which is processed on a centralized HPC server. Due to some events related to the state of the surrounding execution platform, we could require the migration of this computation onto a set of mobile intelligent devices. This migration is a complex operation, concerning not only simple technological issues (e.g. changing the data format and migrating a partial task, as in Aura), but also concerning the relevant differences of new available resources and their efficient exploitation. A parallel computation for a cluster architecture could not be efficiently executed on a set of mobile nodes, due to their possible limitations, such as memory and processing capacity, or the performance offered by their mobile interconnection networks. In this case, a reconfiguration approach can exploit a specific property of Structured Parallel Programming: we can *change the composition* of different parallelization schemes without modifying the computation semantics (Vanneschi, 2002), for example the parallelism degree, the data partitioning scheme, the aggregation/disaggregation of program modules according known cost models. In this way we are able to express multiple compatible behaviors of a certain application part, replacing it without modifying the other parts.

Functional reconfigurations consist in providing a set of different *versions* of the same application or component, each one suitable for specific context situations (e.g. mapping onto specific available resources or when some network conditions occur). All these versions have

a *different but compatible semantics*: they can exploit different sequential algorithms, different parallelization schemes or optimizations, but preserving the component's interfaces in such a way that the selection of a different version does not modify the behavior of the global application. Again, the run-time system is not able to decide the proper version selection strategy in an invisible way. Instead, the programmer is directly involved in defining the mapping between different context situations and corresponding functional reconfigurations: for this purpose, *specific programming constructs* for reconfigurations are provided by the programming model..

In conclusion, both for functional and non-functional reconfigurations, adaptivity is not completely application-transparent, since the programmer must be aware of the adaptation process, i.e. similarly to the a*pplication-aware adaptivity* in Odyssey, but according to an approach which is not limited to the quality of visualized data, but includes the quality any application phase..

Exploiting the Context knowledge

Context awareness is a common issue in many pervasive frameworks. Context-aware applications are able to adapt their behavior without explicit user intervention, improving the application usability by taking environmental information into account. For example Aura applications can be *location-aware*, observing the user motion and reacting to this information. In general, the term *context* can be defined as "any information that can be used to characterize the situation of entities (i.e. whether a person, place, or an object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves" (Dey, 2001, p. 3).

In Pervasive Grids. the application must adapt its behavior mainly considering the application context (e.g. the state of a flood and the identified damages), as well as the context of computing and network resources (e.g. values of the communication and computation bandwidth and/or latency, availability and connectivity), which can lead to different execution requirements (e.g. improving the performance of a real-time forecasting computation in such a way that it can be completed until a specific deadline), thus to proper application reconfigurations.

In a Context-aware system three aspects are very important: how to obtain context-related data, how to represent and manage this information, and how to use this data to trigger proper application reconfigurations.

Context-related raw data can be acquired by sensor devices, failure detectors and monitoring services. In many approaches low-level details about this acquisition process are hided from the applications. In some middleware solutions (Chen, Finin, & Joshi, 2003) a *Context Server* is introduced. It is a fixed entity, which gathers all the sensor data applying proper context models to extract implicit knowledge, in such a way to represent a centralized view about the entire application execution context. This approach encourages a hierarchical system architecture with one or many centralized support-level components. This solution is no longer admissible in large-scale dynamic pervasive infrastructures: a programming model approach must face the explicit definition of the so-called "context logic" of each application. Each application component must be a context-aware adaptive unit, exploiting also parallel computations. To define its behavior, the programming model must offer all the necessary programming constructs to express its "*functional logic*" (i.e. different versions), its "*control logic*" (i.e. mapping between context situations and corresponding reconfigurations) and also the necessary "*context logic*" (i.e. what context data are sensed, how they are interpreted and how to define the necessary context situations).

To *express control and context logics* many methodologies can be utilized. One common solution consists in defining these logics with a set of *Event-Condition-Action rules* (ECA). An event defines a context-related situation, the condition is a boolean expression on the local state of the computation, and the corresponding action is a proper reconfiguration operation. Each application component has a set of adaptation rules (i.e. the adaptation policy). The

control logic of the component identifies the activated rules and performs the corresponding reconfigurations (e.g. according to a non-deterministic choice). These reconfiguration actions can be exploited when the adaptive computation reaches specific *reconfiguration safe-points* (Bertolli, 2009). The rule definition requires also to express the meaning of the corresponding interesting context events (i.e. the context logic). Low-level context data can be obtained from some primitive providers (which we call *context interfaces*) and they can be interpreted by the context logic to express high-level events utilized in corresponding adaptation rules. These high-level events can be defined by using different methodologies: in the simplest case they are first-order logic expressions (which are our starting-point), in other approaches it is possible to exploit some logic languages as Event Calculus (Kowalski & Sergot, 1986) or Fuzzy Logic (Cao, Xing, Chan, Feng, & Jin, 2005).

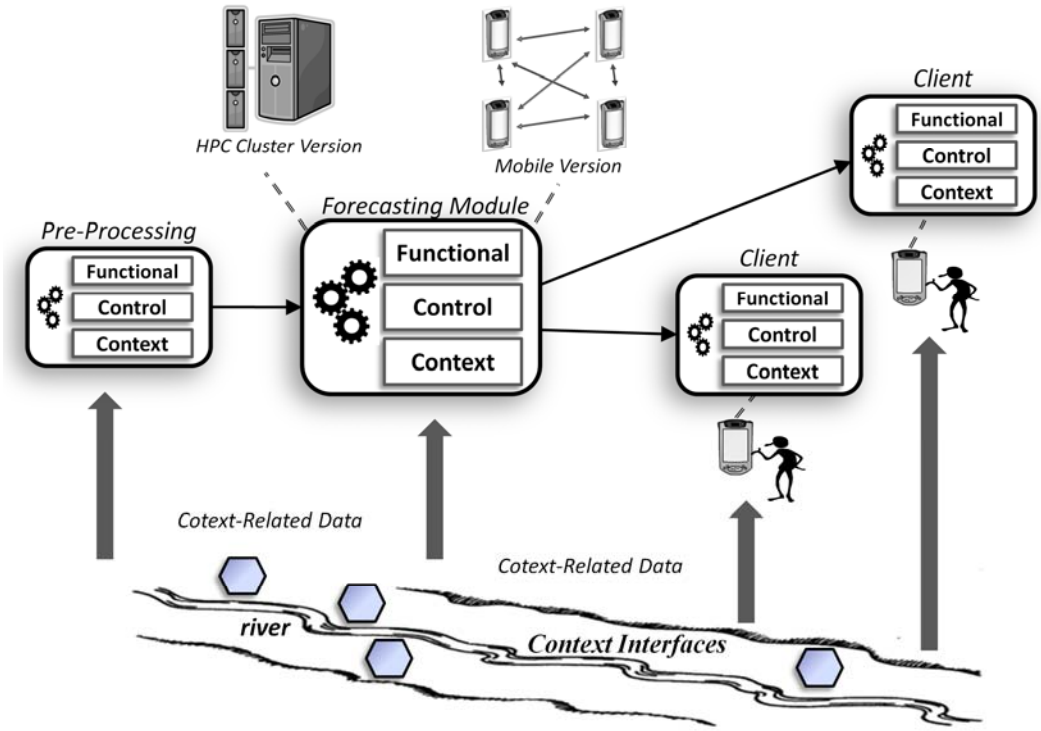

Figure 2. An HPC, Adaptive and Context-Aware application for flood prevention.

Figure 2 shows an abstract view of a complex ubiquitous application for river flood management, defined by using the described programming model features The application is composed of a set of interconnected adaptive and context-aware components and primitive context interfaces. Each component is characterized by the three logics which express its semantics according to an integrated approach of parallel programming methodologies and adaptive context-aware solutions.

Adaptivity and Context Awareness in a HPC programming model

In order to exemplify the proposed methodology we introduce our evaluation framework **ASSISTANT** (*ASSIST with Adaptivity and Context Awareness*) programming model (Fantacci, Vanneschi, Bertolli, Mencagli, & Tarchi, 2009; Bertolli, Buono, Mencagli, & Vanneschi, 2009) developed in the context of the research project In.Sy.Eme (*Integrated System for Emergency*).

The starting point is the previous experience in the **ASSIST** programming model (Vanneschi, 2002). ASSIST is a programming environment oriented to the development of parallel and distributed high-performance applications. An ASSIST application is described as a graph of modules, each one exploiting a sequential or a parallel computation, and communicating via typed data streams. The *Parmod* construct allows the programmer to instantiate and to

configure a parallel module according to any scheme of Structured Parallel Programming even in complex and compound forms (e.g. general or dedicated farms, data-parallelism with static or dynamic communication stencils). ASSIST has been developed for parallel architectures such as shared memory multiprocessors and workstation clusters, and also for high-performance Grid platforms. ASSIST partially faces the dynamicity and heterogeneity of the execution environment, offering a limited approach to application-transparent adaptivity (Aldinucci, Danelutto, & Vanneschi, 2006) concerning the non-functional reconfiguration of the number of concurrent processes (Vanneschi & Veraldi, 2007) which execute a parallel module.

ASSISTANT targets adaptivity and context-awareness of ASSIST programs by allowing programmers to express how the parallel computation evolves reacting to specified events. The new *Parmod* construct is extended to include all the three logics of an adaptive parallel module: *functional*, *control* and *context* logics.
The functional logic support the design of all the different *versions* of the same module, expressed in the ASSIST syntax.
The control logic (Parmod manager) support the design of adaptation strategies, i.e. the *functional and non-functional reconfigurations* performed to adapt the Parmod behavior in response to specified events. Control logics of different application Parmods can interact by means of *control events.*
The context logic includes the *context event* definitions. The programmer can specify events which correspond to sensor data monitoring the environment, as well as network and nodes performance and state. Events related to the dynamic state of the computation (e.g. the service time of a Parmod) can also be specified. All these events can be obtained by proper primitive context interfaces.

The concept of *adaptive versions* is expressed by means of the *operation* construct. Each Parmod can include multiple operations featuring the same input and output interfaces. Each operation includes its own part of functional, control and context logics of the Parmod in which it is defined. Therefore each operation is characterized by its own parallel algorithm, but also its own control and context logics. A Parmod has a *global state* shared between its different operations and it can define the events which it is interested to sense. Semantically, only one operation for each Parmod can be currently activated by its control logic. When a Parmod is started, a user-specified *initial* operation is performed. During the execution, the context logic of a Parmod, or the control logic of other modules, can notify one or more events. The control logic exploits a mapping between these events and reconfiguration actions, defined by the programmer, to either select a new operation to be executed, or execute non-functional reconfigurations e.g. modifying the parallelism degree by means of the *parallelism* construct.
The control logic of an ASSISTANT Parmod can be described as an *automaton*: internal states are the operations of the Parmod, input states are the admissible boolean event expressions, and the output states are the corresponding reconfigurations that must be performed. Figure 3 shows an automaton example.

In the example, the initial operation is OP_0. If the predicate concerning event EV_0 is true, we continue the execution of OP_0 performing non-functional reconfigurations (e.g. modifying its parallelism degree): i.e., self-transition, starting and ending in the same internal state, correspond to non-functional reconfigurations. Consider now the transition from OP_0 to OP_1 fired by a second predicate concerning event EV_1. In this case a functional reconfiguration is involved, a so-called *operation switching*. This switching can include pre- and post-elaborations: for instance, we can reach some consistent state before moving from OP_0 to OP_1 in order to allow the former operation to start from a partially computed result, instead of from the beginning.

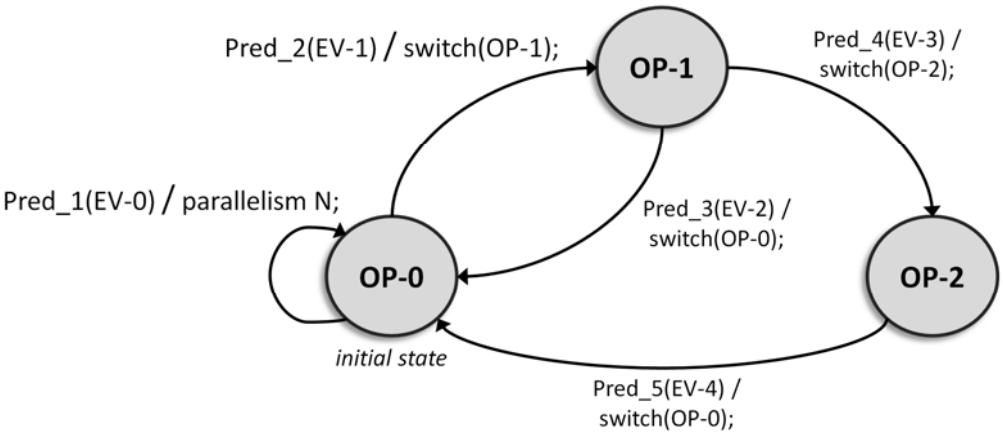

Figure 3. An example of an Event-Operation automaton.

Distinct operations can be defined to be efficiently executed on distinct computation and communication resources: for example OP_0 on a workstation cluster, and OP_1 on a distributed configuration of mobile or interface nodes connected by a wireless network. Event EV_0 can correspond to the request to decrease the service time of the cluster version, and EV_1 to the disconnection of some mobile devices from the central server and to the request of a user to have granted, even in this situation, a certain continuity of service.
When the new operation must be executed on a different set of resources, all the necessary *deployment* activities must be executed by the run-time support which implements this kind of reconfigurations. This behavior is expressed in each operation of a Parmod by using the *on_event* construct. Syntactically, the programmer makes use of nondeterministic clauses whose general structure is described as a typical Event-Condition-Action rule.

FUTURE RESEARCH DIRECTIONS

There are many opened research directions concerning the design and the development of complex ubiquitous applications.

A general class of research problems corresponds to the evaluation of the various mentioned approaches to context awareness in relation to the expressiveness and efficiency of the programming model.

A crucial issue concerns the methodology utilized to express adaptivity and the semantics of the functional logic - control logic interaction of a same application module. This interaction pattern recalls the classical approach of *Control Theory*, in which a system is monitored triggering proper actions to maintain specific output requirements. From this point of view the functional reconfigurations due to context changes are similar to a feedforward control system, whereas non-functional reconfigurations can be described as a feedback control system in which application non-functional properties are monitored (e.g. memory occupation, battery consumption, and the estimated performance). Though some research works (Kokar, Baclawski, & Eracar, 1999) introduce the mentioned analogy, this research direction is still opened especially to formalize a well-defined semantics for adaptive systems.

Moreover, in Pervasive Grids transformations from raw data to user-oriented information are implemented through a sort of *interleaved chain of processing and communication phases*. Several communication strategies can be explicitly programmed according to different parallel algorithms, especially for scheduling, resource allocation and heterogeneous networks, each one exploiting different requirements in terms of the provided latency, throughput and fault tolerance features. These algorithms consider multiple parameters, and the corresponding optimization problems need high computational capabilities. Traditionally, communication activities are implemented in commercial dedicated network devices through a best effort approach, but to achieve a more flexible and scalable solution these activities

could be explicitly programmed according to the specific application needs and requirements. So, in a completely integrated approach, application-aware adaptivity must concern not only the processing phases, but also the possibility to program communication activities expressing the corresponding adaptation strategies. Some first implementations in Network Processors (Venkatachalam, Chandra, & Yavatkar, 2003; Antichi et al., 2009) encourage this research direction.

CONCLUSION

Complex ubiquitous applications exploit the high-degree of heterogeneity and dynamicity of Pervasive Grids, and, at the same time, they often require high-performance. The consequence of these features is the necessary integration between high-performance capability and adaptive and context-aware behavior which applications must be able to express. Limitations and drawbacks of existing approaches to pervasive applications can be removed by a high-level high-performance programming model approach, which allows programmers to express high-performance adaptive and context-aware computations in an integrated fashion. We have described this approach and discussed it with respect to existing solutions mainly based on some middleware infrastructure. In the final part we have introduced our experimental framework ASSISTANT which can be considered a research framework in which theoretical and experimental studies on this methodology can be carried on. Some future directions in the field of High Performance Pervasive Computing, and Pervasive Mobile Grids, have been introduced, which represent some interesting guidelines for future research works.

REFERENCES

Aldinucci, M., Danelutto, M., & Vanneschi, M. (2006, February). *Autonomic QoS in ASSIST Grid-Aware components*. Paper presented at the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, Montbeliard-Sochaux, France.

Antichi, G., Callegari, C., Coppola, M., Ficara, D., Giordano, S., Meneghin, M., . . . Vanneschi, M. (2009, July). *A High Level Development, Modeling and Simulation Methodology for Complex Multicore Network Processors*. Paper presented at the International Symposium on Performance Evaluation of Computer and Telecommunication Systems, Istanbul, Turkey.

Baldauf, M., Dustdar, S., & Rosenberg, F. (2007). A survey on Context-Aware systems. *International Journal of Ad Hoc Ubiquitous Computing*, 2(4), 263-277.

Berman, F., Fox, G., & Hey, A. J. G. (2003). *Grid computing: Making the global infrastructure a reality*. New York, USA: John Wiley & Sons.

Bertolli, C. (2009). *Fault tolerance for High-Performance application using structured parallelism models*. Saarbrücken, Germany: VDM Verlag.

Bertolli, C., Buono, D., Mencagli, G., & Vanneschi, M. (2009, September). *Expressing adaptivity and context awareness in the ASSISTANT programming model*. Paper presented at the third International ICST Conference on Autonomic Computing and Communication Systems, Limassol, Cyprus.

Cao, J., Xing, N., Chan, A. T. S., Feng, Y., & Jin, B. (2005, August). *Service adaptation using fuzzy theory in Context-Aware mobile computing middleware*. Paper presented at the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, Hong Kong, China.

Chang, J. W., Hee Kim. J.,. & Kim, Y. K. (2007, June). *Design of overall architecture supporting Context-Aware application services in pervasive computing*. Paper presented at the 2007 International Conference on Embedded Systems & Applications, Las Vegas, USA.

Chen, H., Finin, T., & Joshi, A. (2003). An ontology for context-aware pervasive computing environments. *The Knowledge Engineering Review*, 3(18), 197-207.

Cole, M. (2004). Bringing skeletons out of the closet: a pragmatic manifesto for skeletal parallel programming. *Parallel Computing*, 30(3), 389-406.

Danelutto, M. & Dazzi, P. (2006, May). *Joint structured/unstructured parallelism exploitation in Muskel*. Paper presented at the 6th International Conference on Computational Science, Reading, UK.

Dey, A. K. (2001). Understanding and using context. *Personal and Ubiquitous Computing*, 5(1), 4-7.

Emmerich, W. (2000, June). *Software engineering and middleware: a roadmap*. Paper presented at the 22nd International Conference on Software Engineering, New York, USA.

Fantacci, R., Vanneschi, M., Bertolli, C., Mencagli, G., & Tarchi, D. (2009). Next generation grids and wireless communication networks: towards a novel integrated approach. *Wireless Communication and Mobile Computing*, 9(4), 445-467.

Foster, I. & Kesselman, C. (2003). *The Grid 2: Blueprint for a new computing infrastructure*. San Franscisco, USA: Morgan Kaufmann Publishers.

Garlan, D., Siewiorek, D., Smailagic, A., & Steenkiste, P. (2002). Project Aura: Toward distraction-free pervasive computing. *Pervasive Computing*, 1(2), 22-31.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2), 199-220.

Hansmann, U., Merk, L., Nicklous, M. S., & Stober, T., (2003). *Pervasive computing: The mobile world* (2nd ed.). Berlin, Germany: Springer-Verlag.

Hingne, V., Joshi, A., Finin, T., Kargupta, H., & Houstis, E. (2003, April). *Towards a pervasive grid*. Paper presented at the 17th International Symposium on Parallel and Distributed Processing, Nice, France.

Kokar, M. M., Baclawski, K., & Eracar, Y. A. (1999). Control Theory-Based Foundations of Self-Controlling Software. *IEEE Intelligent Systems*, 14(3), 37-45.

Kowalski, R. & Sergot, M. (1986). A Logic-Based calculus of events. *New Generation Computing*, 4(1), 67-95.

Lillethun, D. J., Hilley, D., Horrigan, S., & Ramachandran, U. (2007, August). *Mb++: An integrated architecture for pervasive computing and high-performance computing*. Paper presented at the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, Daegu, Korea.

Mache, J. (2006). Hands on grid computing with Globus toolkit 4. *Computing Sciences in Colleges*, 22(2), 99-100.

Noble, B. (2000). System support for mobile, adaptive applications. *Personal Communications*, 7(1), 44-49.

Noble, B. D., Satyanarayanan, M., Narayanan, D., Tilton, J. E., Flinn, J., & Walker, K. R. (1997). Agile Application-Aware adaptation for mobility. *ACM SIGOPS Operating Systems Review*, 31(5), 276-287.

Priol, T. & Vanneschi, M. (Eds.). (2007). *Towards next generation grids. Proceedings of the CoreGRID symposium 2007*. Berlin, Germany: Springer Verlag.

Priol, T. & Vanneschi, M. (Eds.). (2008). *From grids to service and pervasive computing. Proceedings of the CoreGRID symposium 2008*. Berlin, Germany: Springer Verlag.

Uschold, M. & Gruninger, M. (1996). Ontologies: principles, methods and applications. *Knowledge Engineering Review*, 11(2), 93-136.

Vanneschi, M. (2002). The programming model of Assist, an environment for parallel and distributed portable applications. *Parallel Computing*, 28(12), 1709-1732.

Vanneschi, M. & Veraldi, L. (2007). Dynamicity in distributed applications: issues, problems and the Assist approach. *Parallel Computing*, 33(12), 822-845.

Venkatachalam, M., Chandra, P., & Yavatkar, R. (2003). A highly flexible, distributed multiprocessor architecture for network processing. *Journal of Computer and Telecommunications Networking*, 41(5), 563-586.

Weiser, M. (1999). The computer for the 21st century. *SIGMOBILE Mobile Computing and Communications Review*, 3(3), 3-11.

KEY TERMS & DEFINITIONS

Grid: a distributed computing infrastructure for coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations, able to guarantee planned levels of Quality of Service.

Shared or distributed memory architectures: parallel architectures in which the processors interact via a shared memory space at some level (*multiprocessor*), or via an input-output communication structure (*multicomputer* or *cluster*). In *multicore* technology, a multiprocessor or multicomputer is integrated on a single chip.

Parallelism scheme (parallelism paradigm): a structure of a parallel computation with precise semantics in terms of distribution of processing and data. Each scheme is characterized by efficient implementations and related cost models (optimal degree of parallelism, service time, completion time, speed-up or scalability).

Farm (also called *master-worker*)**:** a parallelism scheme characterized by the replication of functions into several identical copies (workers), with an additional scheduling functionality able to balance the processing load of the workers with respect to a stream of tasks.

Data-parallelism: a parallelism scheme characterized by the replication of functions and by the partitioning of the related data structures into several workers, with additional functionalities for data scattering, gathering, and possibly multicasting. In some cases workers operate independently of each other (*map* scheme), in other cases they need to interact during the computation (*stencil* scheme).

Adaptivity: feature of a system/application able to dynamically modify its behavior and/or its structure in order to exploit the available computing and communication resources, with the goal of achieving a planned level of Quality of Service and of satisfying the user intentions.

Context Awareness: feature of a system/application able to dynamically know the functional and non-functional characteristics of the context in which it operates. *Context* is defined as "any information that can be used to characterize the situation of entities (i.e. whether a person, place or an object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves." (Dey, 2001, p. 3).