A Taxonomy for Desktop Grids from Users' Perspective

Monica Vlădoiu and Zoran Constantinescu

Abstract— We present here our taxonomy of desktop grid systems and some hints on how to use it to solve real-world problems. We have customized this taxonomy according to user' perspective to counterbalance the existent taxonomies that are mainly focused on the developers' viewpoint. Our taxonomy is three-level and hierarchical. The first level refers to infrastructure and includes resource type, the platform that runs at the provider, scalability and security issues. The second one includes conceptual model, architecture and data model. The last level concerns aspects related to software: application type, administrator privileges, architecture of the support operating system, and licensing. A table with the classification of the main desktop grids according to this taxonomy is provided. We have been applying here our experience in developing a desktop grid system (QADPZ) in order to satisfy the demands of the specialists in scientific computing and visualization wrt. user goals, needs and restrictions. Examining few typical application scenarios has eased crafting a user-centric taxonomy. We hope that our approach will help promote the introduced taxonomy as a practice for its potential users.

Index Terms— **desktop grid taxonomy, grid computing, parallel and distributed computing, user's perspective**

I. Introduction

Continued exponential technology improvements, new collaborative modalities enabled by the quasi-ubiquitous Internet, and the demands of increasingly complex problems have fuelled a revolution in science and engineering. The new modes of inquiry (data-intensive science, simulation-based science, remote access to experimental apparatus and virtual community science) constitute an ambitious vision for the future, which realization will require long-term investments of financial resources and of intellectual resources by those who must build and apply the necessary global information infrastructure. With the rapid advances in IT, especially supercomputing, commodity and grid computing, every scientist and engineer will benefit from an advanced simulation kit that will make analysis, product development, and design both optimal and cost effective. Thus it becomes possible to investigate incredibly complex dynamics by means of ever more realistic simulations. However, this brings with it vast amounts of multi-dimensional data. To analyze these data is useful yet difficult and expensive.

Manuscript received March 22, 2008.

Monica Vlădoiu is with the Departament of Informatics, PG University of Ploieşti Romania, Bd. Bucureşti, Nr. 39, Ploieşti, Romania; e-mail: (mvladoiu@upg-ploiesti.ro, monica@unde.ro).

Zoran Constantinescu is on leave from Norwegian University of Science and Technology, Trondheim, Norway. (e-mail: zoran@unde.ro).

The remarkable performances of the major volunteer computing projects, as SETI@home [1], clearly demonstrate the usefulness of harvesting cycles over the Internet. The attractiveness of exploiting such systems is further reinforced by the fact that costs are highly distributed: every volunteer supports his or her resources (hardware, power costs and internet connections) while the benefited entity provides management infrastructures (network bandwidth, servers and management services) receiving in exchange a massive and otherwise unaffordable computing power. Fortunately, the usefulness of such computing is not limited to major high throughput public projects. Many institutions, ranging from academics to enterprises, hold vast number of desktop machines and could benefit from exploiting their idle cycles.

The availability of several desktop grid platforms have smoothened the setup, management and exploitation of desktop grid systems. Currently, several platforms exist ranging from academic projects such as BOINC [2], XtremWeb [3] and SZTAKI [4] to commercial solutions like Entropia [5], United Devices [6] and Platform Computing [7]. This plethora of platforms has contributed to the explosion of new Desktop Grids (DGs) and related projects, not only over the Internet but also at an institutional level (e.g. a university campus). Nevertheless, DG computing is still under heavy research conceptualization, and development. There are still many aspects to clarify and solve: security issues, scheduling, volatile environment, sabotage-tolerance, integration with Grid, decentralization, taxonomies etc

We present here our three-level hierarchical DG taxonomy. The first level refers to infrastructure and hosts resource type, the platform that runs at the provider, scalability and security. The second one includes conceptual model, architecture and data model. The last level concerns software (SW): application type, architecture of the support operating system (OS), the need for administrator privileges, and licensing. A table with the classification of the main DGs according to this taxonomy is provided as well. We have developed this taxonomy from the users' perspective, as a result of our interaction with the scientists from Scientific Computing and Visualization during the development of QADPZ, an early open source system for DG computing, which enables users from a local network or Internet to share their resources [8].

II. Related Work

DG systems are relatively new, therefore the need for appropriate taxonomies wrt. infrastructure, computing model, hardware architecture, communication mechanism, software architecture, scheduling, resources, licensing etc. is still present. There are some valuable such works in the literature. Hamscher *et al.* [9] proposed centralized, hierarchical, and decentralized scheduling architectures for Grid. Sarmenta [10] classified volunteer computing into application-based and web-based. Krauter *et al.* [11] proposed a four-faceted (scheduling organization, state estimation, rescheduling and scheduling policy) taxonomy for describing resource management architectures and a survey of grid resource management systems. Grids can be computational grids (distributed supercomputing, high throughput), data grids, and service grids (on demand, collaborative and multimedia). Chien *et al.* [12] see DGs as Internet grid and enterprise grid. Ali *et al.* [13] characterized resource allocation heuristics for heterogeneous computing systems according to workload, platform, and mapping strategy. Yu and Buyya [14] proposed a four-featured taxonomy of scientific workflow scheduling for Grid computing: architecture, decision-making, planning scheme, and scheduling strategies. Yeo and Buyya [15] proposed a taxonomy of market-based resource management system for utility-driven cluster computing, wrt to job model (processing type, composition, QoS specification, QoS update) and resource allocation model (domain, update and QoS support). Venugopal *et al.* [16] proposed a scheduling taxonomy for data grids according to application model, scope, data replication, utility function and locality. Chu *et al.* [17] classified DGs into the first (application specific systems), the second (multi-application systems), and the third generation (modular and service-oriented systems). Choi *et al.*, [18] categorize DGs to organization, platform, scale, and resource provider properties.

Capello [19] provides a DG taxonomy from various points of view: scale/connectivity (LAN, MAN, WAN), architecture (scheduler – push/pull, data – P2P/data server/coordinator, node – PC/cluster), application domain (computation, data, communication, networking, multiple), resource discovery (centralized, distributed), coordination (simple, multiple), programming models (bag of tasks, master worker, MPI), multi-user and multi-application, along with a classification table for several distributed systems wrt to the taxonomy.

Choi *et al.* [20] present a DG taxonomy and its mapping to some state-of-the-art systems to identify their distinctive elements, strong points, weaknesses, and challenges. The taxonomy emphasizes system, application, resource, and scheduler. From the system viewpoint, DGs are categorized wrt organization (centralized, distributed), scale (Internet or LAN), platform (web- or middleware-based), and resource provider (volunteer, enterprise). As for application, the taxonomy includes type (data- or computation-intensive), dependency (independent, flow- or execution-dependent), divisibility (fixed, divisible), submission patterns (deterministic, non-deterministic) and QoS (turnaround time, result correctness, priority, price, restrictions and preferences). Resources are classified wrt altruism (altruistic, egoistic), dedication (dedicated, volatile), scale (LAN, Internet), state change (static, dynamic), trust (trustworthy, malicious), failure (reliable, faulty), heterogeneity, registration patterns (deterministic, non-deterministic) and QoS (price, load sharing or balancing). Finally, the scheduler perspective is taken into account for various features: organization (centralized, distributed, hierarchical), policy (simple – FCFS or random; model-based – deterministic, economy or mathematical; heuristics – reputation- or state-based), grouping, object (application- or resource-oriented), dynamism, mode (scheduler- or resource provider-initiated), trust, incentive scheduling, load sharing or balancing (work stealing/pull or redistribution/push), fault tolerance, adaptive scheduling and goals (turnaround time, deadline, throughput, price, security, load sharing and balancing, trust, incentive and reliability). Particularly, the proposed taxonomy deals with volunteer's key properties (such as volatility, dedication, reputation, trust, etc.) in a DG environment and considers the resource grouping (construction of computational overlay network), result certification, and reputation/incentive scheduling aspects.

III. HIERARCHICAL TAXONOMY OF DGs

One fair question to ask is *Why a new taxonomy?…* from user's perspective... The answer lies on this short fishing story [21]: "to go fishing one needs, first of all, equipment – the poles and lines, hooks, sinkers, floaters, and bait. Without this fishing is .. impossible. Possessing equipment and knowing how to employ it, however, does not guarantee success. Choosing the wrong one from all that is available – the wrong hook or an inappropriate bait probably means turning up empty-handed, even if one handles that hook or bait magnificently". Learning from this, we introduce here our three-layer hierarchical DG taxonomy. The first level refers to infrastructure and includes the resource type, the platform that runs at the provider, scalability and security issues. On the second level we have the conceptual model, architecture and data model. The last level concerns aspects wrt software: application type, architecture of the support OS and whether a license is needed or not. Further, a table with the classification of the main DGs according to this taxonomy is provided.

L1. Infrastructure: resource, platform, scalability, security

Resource type specifies how resources are provided to the system. There are two main trends: volunteer and enterprise. Volunteer DG is based on voluntary participants, while enterprise DG is based on non-voluntary participants usually within a corporation, research lab or university. Mostly, volunteer DG relies on Internet, while enterprise DG is LAN-based. Volunteer DG is more volatile, malicious, and faulty, whereas enterprise DG is more controllable because its resource providers are located in the same administrative domain. Typical examples of volunteer DG are SETI@home, BOINC, XtremWeb, and Bayanihan [22], and enterprise DG are Entropia and Condor [23]. Based on the **platform** running on the resource provider, DGs can be *web-based*, where the applications are run into the web browser (Java applets or ActiveX controls), or *middleware based,* where the user must install a specific middleware, that provides the functionality and services required to execute computing applications on the provider's resource. In the former, the users only need to load a specific web page, containing an applet, which is automatically downloaded and executed by the resource provider. Typical web-based examples are Bayanihan, Javelin [24], while middleware-based are SETI@home, BOINC, XtremWeb, Entropia and Condor.

Scalability divides DGs into two groups: *Internet-based* and *LAN-based*. Internet-based DGs are characterized by anonymous resource providers, connectivity issues (firewall, NAT, dynamic addressing, possibly poor bandwidth and unreliable connection), possibly malicious resources, and high security risks. In contrast, LAN-based DGs show more constant and reliable connectivity, lower security risks or under certain degree of control. Mainly, volunteer DGs fall in the first group, and enterprise DGs in to the second one.

Security in DGs deals with aspects of access to the computational resources through authentication and authorization techniques, and access to the computational data, input and results, by providing data integrity and encryption. Since computations are run in open and non-trustable environments, it is necessary to protect the integrity of data and to validate the computation results. Hardware and software mishaps as well as malicious volunteers can falsify the outcome of computations, rendering the results useless. Thus, a major concern of middleware tools supporting DG computation is to provide results validation and sabotage tolerance mechanisms. Without a sabotage detection mechanism, a malicious user can potentially undermine a computation that may have been executing for weeks or even months. In contrast, applications executed over more controlled clusters offer some reliability and trustability.

L2. Models: computing model, architecture, data model

According to the **computing model** we can group DGs into two main categories: one is the typical, *master-worker computing model*, consisting of independent tasks, and the other one involves *parallel paradigms* with communication between the tasks. The master-worker (M-W) model includes a master (server) process which sends tasks to a set of worker processes, then each worker makes some kind of computation on some tasks, a computation that generally requires a variable and unpredictable time. The master then waits for the answer from each individual worker before sending a new task to that worker. This is a typical form of embarrassingly parallel pattern, where tasks are mutually independent, and can be executed in parallel. The other category involves tasks which depend on each other: there is either an execution flow between the tasks, such that one task needs to be executed only after other tasks are finished (typically accomplished using some sort of task-dependency graph), or the tasks are run in parallel, with data communication between each task (typical paradigms involved are PVM, MPI, BSP).

DGs can be categorized into *centralized*, *hierarchical* and *peer-to-peer* (distributed) according to the **architecture** of the components of each system. A centralized DG consists of a central server, where resource providers donate computing resources during their idle time, and job submitters send their computing requests (jobs). Usually a job is divided into smaller, independent computing units, called tasks, with their own input data. The server distributes these tasks to the available resources, based on some scheduling algorithm. Typical examples are BOINC, XtremWeb, Entropia etc. In a hierarchical DG, desktop grids on the lower level can ask for work from higher level, or vice versa, DGs on the higher level can send work to the lower levels. The control of work at the higher level can be realized with priority handling at the lower level. A basic DG can be configured to participate in a hierarchy, that is, to connect to a higher-level instance of DG (parent node in the tree of the hierarchy). When the child node (a stand-alone DG) has less work than resources available, it asks for work from the parent. The parent node can see the child as one powerful client. An example of such hierarchical DG is the SZTAKI. A somewhat similar approach is present in the Condor, featuring a mechanism for sharing resources among Condor pools (groups of computing resources), called *flocking*. By using this technique, a Condor pool is able to accept job requests that are forwarded from a remote pool. However, the main drawback is the static configuration: to use flocking, the Condor pools must be manually configured.

In a peer-to-peer DG, there is no central server, in contrast with the centralized type. Resource providers have only partial information of other providers. They are also responsible for constructing the computational overlay network and for scheduling a job in a distributed way, according to each other capability, availability, reputation or trust. The reliability and performance of such P2P systems depend on how the overlay network is constructed, because there is no reliable central server. Such systems are CCOF, Messor, Paradropper, and Organic Grid.

Data model concerns classifying of DGs based on how computational data (input/output data) is transferred between the components of the DG. We are concerned here with data communication between job submitter and resource provider on one hand, and between different resource providers on the other hand, in the situation when communication between running tasks is required (parallel models). We identified three data model types: *middleware*, *data servers*, and *direct communication*. In the first situation, using the middleware, which connects the two components, transfers data. This could be the master (server) in a centralized configuration, or all the involved nodes in a P2P configuration. The downside of this approach is the bottleneck possibility in the case of large data sets, which could affect other communication between the components (control, discovery, status, etc.).

In the data server model, all the data is transferred using another type of component, which is a repository of both input and output data: a data server. In this case, the job submitter is responsible for uploading the input data to the data server, and for retrieving the results, while the job running on the resource provider's computer is responsible for downloading the input data and storing the results on the server after finishing the job. This model has the advantage of moving the burden of data transfer (communication and complexity) from the central node to a more dedicated, and optimized component. However, there is a complexity added in maintaining such a data server, which in some situations might not be necessary.

The third data model involves direct data communication between the components. This could be done either by using a common network file system, where each component has access to it, by using a distributed file sharing mechanism (P2P Bittorrent), or by using lower lever network based communication for data transfer. The type of direct data communication could be chosen based on the amount of data transferred, and the frequency with which data transfers occur. We can also have the situation when the submitted job contains also the input data for the computation.

L3. SW: application, architecture, administration, license

SW applications to be run on the DGs can be of different types. The SETI project is typical of the Internet computing domain in that its application is both dedicated and vertically integrated. In other words, no generic use of the computing resources is or can be made, and all dataflow between the computing resource and the data server uses proprietary APIs (dedicated pre-defined applications). This includes legacy applications, which already exist and are inherited from languages, platforms, and techniques earlier than current technology. Most enterprises that use computers have legacy applications that serve critical business needs. In order to run them some kind of virtualization could be necessary, depending on the complexity of the application and the resources it needs (third party applications or libraries, file system access, specific OS, etc.). This could range from simple, virtual file systems, to more complex virtual environments (virtual machines emulating an OS).

Another class includes applications written in a high level or interpreted programming language, like Lisp, Perl, Java, where in order to run the program, a specific run-time environment should be present. The computing jobs must be distributed according to each processing resource's capabilities, and provide the appropriate starting mechanism. Web-based and Java-based systems have their own limitations. One of these is the historically slow execution speed of the Java virtual machine (JVM) that executes the platform-independent bytecode. Another problem comes from the security restrictions imposed on Java applets that prevent applets from accessing local storage space or communicating with machines other than the host from which they came. Together, these two problems may limit the performance and scalability of Java-based systems.

A whole class of application includes those where the programs could be compiled in a programming language (C/C++, Fortran), and where additional support for desktop grids could be included. This allows fine tuning the application in term of computing performance, but requires an API from the DG to be provided. This includes also parallel applications, where different communication paradigms are required (message passing, shared memory, etc.).

A last type refers to lightweight programs, highly optimised for performance, and which are specific to each desktop grid. For example, computational applications could be made in form of plugins (or shared libraries), which contains only the computational problem, the rest of the communication, file access, and other access to resources are handled by the supporting middle layer application running on the resource provider. In this case, a more complex abstraction and API are needed from the underlying DG system.

SW platform in DGs concerns with the OS that is running on different system components. This could be Linux, or Unix versions (BSD, IRIX, etc.) when resource providers are nodes from a cluster, or Linux, Windows, Mac, if resource providers are desktop computers. Thus, a DG should have support for the different OS. Many DGs are based on Java for portability.

SW administration - during the QADPZ development, we have learned that another restriction of existing systems, especially middleware based, is that each resource provider needs to install a runtime module as administrator. This poses some issues regarding data integrity and accessibility on providers computers. QADPZ tries to overcome this by allowing the middleware module to run as a non-priviledged user to the local system [25]. **SW license** can be necessary for commercial DGs or not in case of open source DGs.

Our taxonomy is worthwhile both theoretically, as it includes facets that other taxonomies ignore for the time being (e.g. hierarchical architecture, fixed set of pre-defined applications for volunteer computing projects, administration privileges and so on), and practically, being an instrument for choosing the most appropriate desktop grid for the problem to be solved. Below we present some scenarios [26] for use of desktop grids in real world situation and follow them with some hints about how to choose the best solution.

Scenario 1. A holding that want decide on the placement of a new unit invokes a sophisticated financial forecasting model from an Application Service Provider (ASP), providing it with access to appropriate proprietary historical data from a corporate database that is kept at a storage service provider. During the decision-making meeting, *what-if* scenarios are run collaboratively and interactively, even though the division heads are located worldwide. The scenarios must meet desired security and performance requirements.

Scenario 2. An industrial consortium formed to develop a feasibility study for a next-generation supersonic spacecraft undertakes a highly accurate multidisciplinary simulation of the entire spacecraft that integrates proprietary software components developed by different participants, with each component operating on that participant's computers and having access to appropriate design databases and other data made available to the consortium by its members.

Scenario 3. A crisis management team responds to a toxic waste accident by estimating the spread of the waste (using local weather and soil models), by determining the impact based on population location, and creates a short-term plan, and a task emergency response personnel by planning and coordinating evacuation, notifying hospitals, etc.

Scenario 4. A large-scale Internet game consists of many virtual worlds, each with its own physical features and laws. Each world may have a large number of inhabitants that interact with each other and move from one world to another. Worlds may expand to accommodate population growth. New simulation technology to model the physical laws of the world will be needed. Simulations need to be coupled to see what happens when worlds collide.

Hints to choose the most suitable DG for a given problem: we first look at the first column, first entry (resource) and if the project is requested to have robustness and reliability (major issue for first 3 scenarios) we would better choose the enterprise DG as it overcome the volatility of volunteer computing. More, it has accountability and, depending on the type of the organization, lacks anonymity (except for universities or alike organizations). On the second choice (middleware vs. web-based), if the ensuring of control and security is crucial we should go for middleware platform (first 3 scenarios), while for the 4th scenario we could use both. Though, we must remind that the enterprise desktop grid is limited in power, and the volunteer computing has

Table 1. Classification of the main desktop grid systems according to the taxonomy

	Infrastructure	**Models**	**Software**
DG system	**Resource** **Platform** **Scalability** **Security**	**Computing model** **Architecture** **Data communication model**	**SW application** **SW platform** **SW administration** **SW license**
distributed.net	- volunteer - middleware - Internet - trust	- master-worker (M-W) - centralized - data server	- set of dedicated only - all OS - non admin - closed
Entropia	- volunteer - middleware - Internet - trust	- master-worker - centralized - data server	- set of dedicated only - Windows - non admin - closed
SETI@home	- volunteer - middleware - Internet - trust	- master-worker - centralized - data server	- set of dedicated only - Linux, Win, Mac - non admin - closed
Bayanihan	- volunteer - web-based - Internet - Java sandbox	- master-worker - centralized - middleware	- Java applet - all OS (Java) - non admin - open source
Condor	- enterprise - middleware - LAN, Internet - authentication	- M-W, PVM, MPI - centralized, (hierarchical) - file system	- legacy, script, compiled - Linux, Win, Mac - admin - license
XtremWeb	- enterprise - middleware - LAN, Internet? - authentication	- M-W, MPI - centralized, (hierarchical) - middleware	- Java applet - all OS (Java) - admin? - open source
QADPZ	- enterprise - middleware - LAN, Internet - authentication	- M-W, MPI, PVM - centralized - file system, data server	- legacy,script,compiled, lightweight - Linux,Win,Mac,Unix - non admin, admin - open source
BOINC	- enterprise - middleware - LAN, Internet - authentication	- M-W - centralized - data server	- legacy, script, compiled - Linux,Win,Mac,Solars - admin - open source
SZTAKI LDG (BOINC based)	- enterprise - middleware - LAN, Internet - authentication	- M-W - hierarchical - data server	- legacy, script, compiled - Linux,Win, Mac, Solaris - admin - open source
Javelin Javelin++	- volunteer - web-based - Internet - Java sandbox	- M-W - centralized - middleware	- Java applet - all OS (Java) - non admin - open source

virtually unlimited resources. As for the scale and security, probably the best option for the first two problems is the LAN solution as it ensures privacy and keeps the secrets of the application away from un-authorized eyes. The last two, on the other hand can go both ways. The models from the second column are strongly influenced by the nature and complexity of the application. One choice would be suitable in the case of an application that can be broken in small tasks that can run parallel, with no communication between them (master-worker), and another for a different type of application, in which tasks can communicate with each other (Message Passing Interface - MPI). Moreover, if the application needs a huge computational power, we would probably prefer a hierarchical DG, as it can borrow power from third parties. As for the data communication model, one has to consider the difficulty of developing the software that will manipulate the data and the technical limitations (within a virtual file system vs. "back and forth" from a data server). The main difference in the usage of institutional DGs relatively to public ones lies in the dimension of the application that can be tackled. In fact, while public projects usually embrace massive applications made up of an enormous number of tasks, institutional DGs (much more limited in resources) are better matched for small size applications. So, whereas in public volunteer projects importance is on the number of tasks carried out per time unit (throughput), users of institutional desktop grids are normally more interested in a fast execution of their applications, seeking fast turnaround time.

The last column is easier to work with as many of the issues involved here are known before starting to solve a given problem: we have our own application or we want to run a pre-defined one, if we have our own, which kind it is (Java applet, legacy, script etc.), what platform we use (Linux, Windows, Mac etc.), what are the needed administration privileges (admin or user), whether we are interested in access to the source code or not, and finally if we need a desktop grid for which a commercial license is requested.

IV. Conclusions and future work

Scientists are becoming familiar with desktop programs capable of presenting interactive models of molecules. The field of bioinformatics and the field of cheminformatics make a heavy use of these visualization engines for interpreting lab data and for training. Medical imaging is a huge application domain for scientific visualization with an emphasis on enhancing imaging results graphically, e.g. using pseudo-coloring or overlaying of plots. Real-time visualization can serve to simultaneously image analysis results within or beside an analyzed (e.g. segmented) scan. Data visualization techniques are now commonly used to provide business intelligence. Performance metrics and key performance indicators are displayed on an interactive digital dashboard. Business executives use these software systems to monitor the status of their business results and activities.

All users of scientific computing and visualization have an interest in better hardware, software and integrated systems, and much of what has being developed was shared by a number of scientific and engineering disciplines up to a point, but with very large costs that were accessible only to large research facilities (e.g. SGI visualization servers and large PC clusters). Because of the huge number of PCs in the world, desktop grid and volunteer computing can (and do) supply more computing power to science than does any other type of computing. This power enables scientific research that could not be done otherwise. This advantage will increase over time, because the laws of economics dictate that consumer electronics (PCs and game consoles) will advance faster than more specialized products, and that there will simply be more of them. Volunteer computing power cannot be bought; it must be earned. A research project that has limited funding but large public appeal (such as SETI@home) can get huge computing power. In contrast, traditional supercomputers are extremely expensive, and are available only for applications that can afford them (for example, nuclear weapon design or intelligence). Desktop grid and volunteer computing encourage public interest in science, and provides the public with voice in determining the directions of scientific research.

This paper emphasizes the need for appropriate taxonomies to help the potential user to choose from the variety of DGs that are now available for public use. We have applied here our experience in developing QADPZ to satisfy the demands of the specialists in scientific computing and visualization. Their feedback has been carefully used in improving QADPZ and the current taxonomy. We have customized it according to user' perspective to counterbalance the many existing taxonomies that are focused on developers' viewpoint. This taxonomy could be further extended, as the desktop grids evolve, with autonomic computing features (QADPZ presents the following autonomic capabilities: self-management, self-configuration, self-optimization and self-healing [8]).

Taxonomies continue to grow in importance as the DGs mature. We have tried to think them in terms of user goals, needs and restrictions. Examining few typical application scenarios has eased crafting a user-centric taxonomy. We hope that our approach will help promote the introduced taxonomy as a practice for its potential users.

References

[1] SETI@home (March, 2008) [online] http://setiathome.ssl.berkeley.edu/
[2] BOINC (March, 2008) [online] Available: http://boinc.berkeley.edu/.
[3] XtremWeb (March, 2008) [online] http://www.lri.fr/~fedak/XtremWeb
[4]SZTAKI (March,2008) [online] Availablehttp://szdg.lpds.sztaki.hu/szdg/
[5] Entropia (December, 2003), [online] Available: www.entropia.com
[6] United Devices (May 2007) [online] http://distributedcomputing.info/
[7] Platform Computing (March 2008) [online] http://www.platform.com/
[8] Z. Constantinescu, "Towards an autonomic distributed computing environment", in *Proc. of 14th Database and Expert Systems Applications Workshops*, September 2003, Prague, Czech Republic
[9] V. Hamscher *et al.*, "Evaluation of job-scheduling strategies for grid computing", in *Proceedings of the First IEEE/ACM International Workshop on Grid Computing (Grid 2000),* LNCS 1971, Springer-Verlag, pp. 191-202, (Dec., 2000), Bangalore, India
[10] L. F. G. Sarmenta, "Volunteer computing" Ph.D. thesis, 2001, MIT, Cambridge, USA
[11] K. Krauter, R. Buyya, M. Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing", in *Software – Practice and Experience*, 2002, 32: pp. 135–164
[12] A. Chien *et al.*, "Entropia: architecture and performance of an enterprise desktop grid system", in *Journal of Parallel and Distributed Computing*, 63, 5, May 2003, pp. 597-610.
[13] S. Ali *et al.*, "Characterizing resource allocation heuristics for heterogeneous computing systems", in *Advances in Computers: Parallel, Distributed and Pervasive Computing*, 2005, 63, pp. 93-129
[14] J. Yu, R. Buyya, "A taxonomy of scientific workflow systems for grid computing", in *SIGMOD Record*, *Special Issue on Scientific Workflows* 34, 3, pp. 44-49.
[15] C. S. Yeo, R. Buyya, "A taxonomy of market-based resource management systems for utility-driven cluster computing", *Software: Practice and Experience* 36, 13 (Nov., 2006), pp. 1381-1419
[16] S. Venugopal, R. Buyya, R., K. Ramamohanarao, "A taxonomy of data grids for distributed data sharing, management and processing", in *ACM Computing Surveys* 38, 1, March 2006, pp.1-53
[17] X. Chu *et al.,* "Aneka: Next-generation enterprise grid platform for e-science and e-business applications", in *Proceedings of the 3rd International Conference on e-Science and Grid Computing (e-Science 2007),* IEEE CS Press, pp. 151-159, Dec. 2007, Bangalore, India
[18] S. Choi *et al*., "Characterizing and Classifying Desktop Grid", in *Proc. of 7th IEEE International Symposium on Cluster Computing and the Grid (CCGRID 2007)*, pp. 743-748, May 2007, Rio de Janeiro, Brazil
[19] F. Cappello, "3rd generation desktop grids", in *Proc. of the 1st XtremWeb Users Group Workshop (XW'07),* Feb. 2007, Hammamet, Tunisia
[20] S. Choi *et al.,* "A Taxonomy of Desktop Grids and its Mapping to State-of-the-Art Systems", in *ACM Computing Surveys*, [online] Available www.gridbus.org/reports/DesktopGridTaxonomy2008.pdf
[21] SAGEPUB (March, 2008). A practical evaluation taxonomy. [online] Available: www.sagepub.com/upm-data/5047_Chen_Chapter_3.pdf
[22] Bayanihan (March, 2008) [online] http://bayanihancomputing.net/
[23] Condor (March, 2008) [online] http://www.cs.wisc.edu/condor/
[24] Javelin (May 2005) [online] Available http://javelin.cs.ucsb.edu/
[25] QADPZ (March, 2008) [online] Available http://qadpz.sourceforge.net
[26] I. Foster, C. Kesselman, *The grid : blueprint for a new computing infrastructure*, Boston, Morgan Kaufmann, 2004