

AutoGrid: Towards an Autonomic Grid Middleware

Marcio Augusto Sekeff Sallem and Stanley Araújo de Sousa
Universidade Federal do Maranhão
Programa de Pos-Graduação em Engenharia de Eletricidade
Avenida dos Portugueses, S/N, Bacanga
São Luis, MA, Brasil
Email: marciosallem@lsd.ufma.br, stanleyaraujo@oi.com.br

Francisco José da Silva e Silva
Universidade Federal do Maranhão
Departamento de Informática
Avenida dos Portugueses, S/N, Bacanga
São Luis, MA, Brasil
Email: fssilva@deinf.ufma.br

Abstract—Computer grids have drawn great attention of academic and enterprise communities, becoming an attractive alternative for the execution of applications that demand huge computational power, allowing the integration of computational resources spread through different administrative domains. However, grids exhibit high variation of resource availability, node instability, variations on load distribution, and heterogeneity of computational devices and network technology. Due to those characteristics, grid management and configuration is error-prone and almost impracticable to be performed solely by human beings.

This paper describes AutoGrid, an autonomic grid middleware built using Adapta reconfiguration framework and runtime system. AutoGrid introduces self-managing capabilities to the Integrate grid middleware, such as: context-awareness, self-healing, self-optimization and self-configuration. This paper also presents insights and experiments that show the benefits towards an autonomic grid infrastructure.

I. INTRODUCTION

A computer grid comprises a hardware and software infrastructure that allows integration and sharing of distributed resources, such as software, data and peripherals, inside and among institutions. This computational infrastructure has drawn great attention of academic and enterprise communities, becoming an attractive alternative for execution of applications that demand huge computational power, and allowing the integration of computational resources spread through different administrative domains.

Computer grids have been used to solve problems in varied areas of scientific, enterprise, and industrial activities, for example: computational biology, image processing for medical diagnosis, weather forecast, high energy physics, marketing simulations, and oil prospection. Grid computing empowers the conception of a new generation of applications that allow combining computations, experiments, observations, and data got in real time. The phenomena modeled by these applications require diverse software components whose compositions and interactions are extremely dynamic. Moreover, the grid infrastructure is also heterogeneous and dynamic, aggregating a great amount of computation and communication resources, databases and, sometimes, sensors and specific peripherals. The dynamism can be observed in terms of high variation in resource availability, node instability, and workload variations in nodes and network links.

The dynamic nature of the grid infrastructure, its high scalability and great heterogeneity have turned impracticable its configuration, maintenance and recovery in case of failures solely by human beings. Several recent research projects [13], [10] have recognized the necessity of providing a greater autonomy to grid systems, which is one of the greatest challenges for the new generation of this kind of middleware. The term autonomic computing has been used to denote a system that exhibits functional properties, such as [8]:

- Context-awareness: the system must be aware of its execution environment and be able to react to environmental changes;
- Self-Optimization: the system should be able to detect performance degradation and intelligently perform self-optimization actions;
- Self-Healing: the system must be aware of potential problems, and should be able to reconfigure itself in order to continue to function smoothly and to recover from failures using diverse and adaptive failure-handling techniques;
- Self-Configuration: the system must have the ability to dynamically adjust its resources based on its state and the state of its execution environment.

The AutoGrid project, currently being developed at the Federal University of Maranhão, Brazil, main goal is the development of a robust and self-managing autonomic grid system. The AutoGrid project uses the Integrate grid middleware [7] as the foundation for its implementation, incorporating autonomic mechanisms to its infrastructure in order to make its configuration and administration independent from human intervention. Our research primarily focuses on adding to Integrate four autonomic properties: context awareness, self-healing, self-optimization, and self-configuration.

This paper presents the current state of the AutoGrid project, its architecture and implementation highlights. It is organized as follows: Section II illustrates other approaches to autonomic grids. Section III describes the architecture of the AutoGrid middleware, its main components and interactions. Section IV exhibits AutoGrid autonomic mechanisms, while Section V draw our conclusions and discusses AutoGrid future work directions.

II. RELATED WORK

Project AutoMate [1] investigates models and architectures to enable the development and execution of self-managing grid applications. AutoMate is a framework that provides key solutions to autonomic formulation, composition and runtime management of applications on the grid; it is not an autonomic grid middleware like AutoGrid. AutoMate autonomic grid applications are dynamically and opportunistically composed from autonomic elements, which comprise components, services or applications that consistently configure, manage, adapt and optimize their own execution. Autonomic elements also export information about their behavior, resource requirements, performance, interactivity and adaptability to system and application dynamics. Finally, AutoMate extends the current grid middleware infrastructure and adds key services on top of it to support the policy, content and context driven execution and management of autonomic applications.

OptimalGrid [11] aims to simplify creating and managing large-scale, connected, parallel grid applications. On the other hand, AutoGrid focuses on more general applications, such as: sequential applications that are composed of a binary that executes on a single machine; parametric applications which are also composed of a single binary, but multiple copies of it are executed on different nodes with different input parameters or data sets; and the BSP applications developed according to the Bulk Synchronous Parallel computing model [16]. OptimalGrid handles, among other things, runtime management, dynamic level of parallelism, dynamic load balancing and even system fault tolerance and recovery. OptimalGrid also includes autonomic grid functionality: 1) self-configuring to calculate an initial optimal assignment of the number of compute nodes to solve a problem, and the load distribution among those nodes; 2) self-optimizing in rebalancing load according to current network status and computation power, or in replacing algorithms to achieve high performance; 3) self-healing which introduces a fault tolerance layer, so node failures do not represent the loss of the whole computation.

The Grid Application Development Software (GrADS) [2] goal is to simplify distributed heterogeneous computing, exploring the scientific and technical problems that must be solved to make it easier for users to develop, execute, and tune applications on the Grid. GrADS comprises a performance monitoring library and tools for constructing applications from grid-aware components. [3] implements scheduling policies for dynamically migrating applications on the GrADS infrastructure, in response to load system changes and application characteristics. Currently, AutoGrid does not support application migration; however, it can dynamically choose the grid scheduling algorithm used, and also has self-healing and self-configuration features. GrADS self-optimization mechanism main components are: 1) Migrator that migrates applications on the GrADS grid system, 2) Contract Monitor that monitors application progress, and 3) Rescheduler that decides when to migrate.

III. AUTOGRID ARCHITECTURE

AutoGrid is an autonomic grid middleware that augments Integrate [7] middleware with self-managing capabilities. Integrate is a multi-university effort to build a novel grid computing middleware infrastructure to leverage the idle computing power of personal workstations for the execution of computationally intensive parallel applications. The basic architectural unit of an Integrate grid is the cluster, a collection of machines usually connected by a local network. Clusters can be organized in a hierarchy, encompassing a large number of machines. Each cluster contains a Cluster Manager node that executes Integrate components responsible for managing the cluster computing resources and for inter-cluster communication. Other cluster nodes are called Workstations, which export part of its resources to Grid users. They can be shared or dedicated machines.

In order to achieve self-manageability, AutoGrid middleware components are implemented using Adapta [17], a framework for developing self-adaptive applications that decouples the code that governs the business rules from the code responsible for adaptation. The framework architecture is based on computational reflection, where Adapta comprises the application meta-level while the base level consists of the application business objects. Adapta is also a runtime system regularly monitoring the executing environment and notifying events to registered components whenever a resource availability condition is detected.

Adapta introduces a well-defined reconfiguration language for each aspect of the reconfiguration process (monitoring, environment change detection, and application reconfiguration). The reconfiguration language is XML-based and its statements are stored externally to the application, describing the component object model. Application developers and administrators can alter a component object model by simply modifying its corresponding reconfiguration statements. Modifications to the object model are interpreted and built on runtime, using the Adaptive-Object Model (AOM) pattern [18]. Therefore, each Adapta component is also flexible and reconfigurable.

A. AutoGrid Main Components

AutoGrid reuses several Integrate components and refactors others, introducing autonomic features. The main components of AutoGrid architecture are:

- Application Submission and Control Tool (ASCT): a graphical user interface that allows users to submit applications and control their execution;
- Application Repository (AR): stores the code of applications that can be executed on the grid;
- Local Resource Manager (LRM): a component that runs in each cluster node responsible for instantiating and executing applications scheduled to the node;
- Monitoring Service (MS): collects information about the state of grid node resources, such as memory, CPU, disk, network usage and applications. Each resource comprise one or more properties, which are monitorable attributes,

such as, available memory, CPU load usage, network bandwidth and latency, amount of application threads;

- Local Event Service (LES): receives notifications from colocated MSs and notifies registered components when a specific resource availability condition is detected;
- Event Processing System (EPS): a distributed event service that detects composite events from different event sources (grid nodes);
- Dynamic Reconfiguration System (DyReS): the adaptation engine that applies reconfiguration actions to the application in respond to environmental changes. It also coordinates dependent components during the reconfiguration process;
- Global Resource Manager (GRM): manages the cluster resources by receiving notifications of resource usage from the MSs (through an information update protocol), and runs the scheduler that allocates tasks to nodes based on resources availability. It is augmented with self-managing capabilities using DyReS adaptation engine;
- Execution Manager (EM): maintains information about each application submission, such as its state, executing node, input and output parameters, submission and conclusion timestamps. It also coordinates the application recovery process, in case of failures.
- Stable Storage: a distributed data repository that stores the state of applications (checkpoints).

IV. AUTOGRID AUTONOMIC FEATURES

AutoGrid comprises four autonomic features: (a) context-awareness, comprising the monitoring of grid resources and the detection of environmental changes; (b) self-configuration, that is based on computational reflection and comprises the self-representation of the grid software structure and the support for reconfiguration actions; (c) self-healing, comprising a flexible mechanism for application fault-tolerance; and (d) self-optimization, through an adaptive replacement of the scheduling algorithm.

A. AutoGrid Context-awareness Mechanism

Figure 1 illustrates the interactions among AutoGrid components during the whole reconfiguration process. To achieve autonomic behavior, it is necessary that the grid middleware senses its own executing environment, which includes individual node parameters, such as, CPU availability, memory used, disk space, and global grid parameters, such as application income rate and the mean time between failures. For that matter, the Monitoring Service (MS) regularly inspects the underlying hardware and the executing environment on every grid node (1) using monitor objects, which are individually assigned to a single property. Monitors can be dynamically instantiated to introduce new monitoring requirements not known at design-time or replaced on the fly to cope with the diversity of computational platforms. Each property also has a set of operating ranges, for example, one could use the

following operation ranges for monitoring the CPU load usage: [0%, 40%), [40%, 75%), and [75%, 100%]. The MS notifies a collocated Local Event Service (LES) whenever there is a change on the operating range of a property (2).

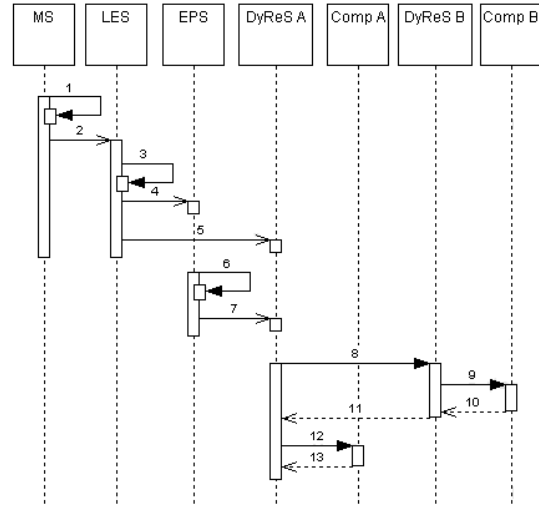


Fig. 1. The reconfiguration process

Upon a significant resource change notification, the LES evaluates if a determined resource availability condition occurred (3). Event evaluation is taken carefully to minimize the amount of messages sent to nodes and it is based on a boolean expression provided by the grid developer as part of the event definition. To trigger an event notification, the corresponding boolean expression must stay *true* during an amount of time specified by the user, known as the *duration time*. The duration time avoids generating notifications when temporary situations occur, such as a resource usage peak (e.g. a CPU use peak, triggered by starting a heavy program). Once an event is detected, LES notifies the Event Processing System (EPS) (4) and all subscribed components embodied by a DyReS instance (5). Event notifications include the host that generated the event, a timestamp, and property values specified at event definition.

EPS is a distributed event service that detects composite events (6) from different event sources (distributed nodes). EPS is required whenever the decision to reconfigure grid components should consider the combination of events detected on distinct grid nodes. For example, dynamic load balancing algorithms based on application migration should take into account the CPU usage of every grid node and also the network status. Upon detecting a distributed event, the EPS notifies subscribed components embodied by a DyReS instance (7). In order to avoid typical distributed systems issues, such as message loss, duplication, or ordering EPS uses three processing parameters:

- Detection window, which indicates the amount of time that a received event is valid, based on its timestamp;

- Scheduling time, which indicates the amount of time to wait for processing an event, helping to maintain the correct event ordering;
- Concurrence time, which indicates the amount of time that two events should be considered concurrent and treated simultaneously.

The context-awareness mechanism finishes its cycle once local or composite events are notified to all subscribed DyReS instances. Next step is self-configuration.

B. AutoGrid Self-configuration Mechanism

Each adaptive component in the AutoGrid architecture is augmented with a DyReS instance that corresponds to its meta-level. The DyReS is responsible for receiving event notifications (from LES or EPS), and initiating reconfiguration on behalf of its component. Currently, DyReS can perform two reconfiguration actions: dynamic change of application parameters and dynamic replacement of application algorithms with a well-defined state transfer protocol. However, DyReS instances can be extended by expert users to introduce new actions, such as component addition, removal, or replacement.

The grid middleware may require that a reconfiguration taken on a single component be synchronized amongst dependent components. For synchronization purposes, DyReS instances manage component dependencies by maintaining references to other DyReS instances, based on [12]. DyRes comprises *hooks*, that represent the components it depends upon, as well as *clients*, that references dependent components. For example, on figure 1, *DyReS A* is a hook to *DyReS B*, which means *DyReS B* is *DyReS A* client. Synchronization notifications flow through this dependency chain formed by hooks and clients, arriving as event notifications at a component (8). *DyReS B* instance (that receives the synchronization notification) must perform the reconfiguration on behalf of its component (9). When reconfiguration finishes (10), *DyReS B* responds to its hook that it finished reconfiguration (11). Finally, *DyReS A* reconfigures its own component (12) and (13).

The synchronization protocol shown uses a *timeout* attribute to avoid that the synchronization process extends itself indefinitely, due to a component crash or network traffic conditions.

1) *Change of Application Parameters*: The parameter updating mechanism uses a callback method approach. The grid middleware developer introduces, during design, callback methods to be invoked on every class that has an updatable parameter. AutoGrid obtains the callback reference and dynamically invokes it using the new values informed on the reconfiguration action. Parameter updating is required to dynamically change the MTBF value used by the replication mechanism and alter the interval between checkpoints, as shall be seen on the self-healing feature section.

2) *Replacement of Application Algorithms*: The algorithm replacement procedure requires a proxy that introduces an indirection layer above replaceable objects, keeping adaptation transparent to clients. The proxy also manages state transfer

during the substitution process. The state consists on a set of variables, that represents the current computation of the active algorithm. Since variables are class fields, the proxy can inspect the object implementing the active algorithm, store its state, and load it inside another object. Algorithm replacement is necessary to dynamically substitute the current grid scheduling algorithm for one that is better suited to the executing environment, as can be seen on the self-optimization feature section.

Adapta uses a lazy approach for object replacement, instead of an eager one [15]. The lazy approach allows the already running algorithm to complete its execution before being replaced, while the eager one immediately suspends the algorithm execution, performs the replacement, and resumes it from the point where it was suspended. The lazy approach advantage is that it always reaches a valid state, which is not guaranteed by the eager approach, since not every random execution point (a transient state) is a valid state into the new object.

C. AutoGrid Self-healing Mechanism

Computational grids are highly prone to failures due to several facts, such as its dynamic nature or grid autonomy. In order to ensure the continuity of applications executions, diverse failure handling techniques strategies can be applied, which includes:

- Retrying, which restarts an application from scratch;
- Replication, which submits the same application for execution a number of times, generating various application replicas. When one of the replicas finishes, the grid middleware must discard (or ignore) the others and return the results to the requesting user;
- Checkpointing, which periodically saves the state of the computation in a stable storage during the failure free execution time. Upon a failure, the application restarts from the last saved point (a checkpoint). The adoption of this technique introduces an overhead to the normal application execution time. More details about AutoGrid checkpointing mechanism can be found on [5], [4].

Hwang et al [9] presented the tradeoff among failure handling techniques implemented on a grid (retrying, checkpointing, replication, and replication with checkpointing), considering several different execution environments scenarios, which involve various failure handling parameters, such as the failure-free execution time or the mean time between failures (MTBF). AutoGrid self-healing mechanism could automatically select the most appropriate failure-handling mechanism based on the parameters presented; and it also could adjust the time interval between consecutive checkpoints, according to the amount of computation.

Currently, AutoGrid version can dynamically decide the amount of replicas to be generated for a given application submission based on the execution environment MTBF and the application mean execution time. The Global Resource

Manager (GRM) is the component in the AutoGrid architecture that manages fault-tolerance. GRM was augmented with DyReS to dynamically modify the MTBF value and recalculate the amount of replicas. The algorithm used for that purpose attempts to approximate as much as possible the application execution time to its mean execution time using as less replicas as possible.

We measured the benefits of varying the amount of replicas with a set of simulations. Figures 2 and 3 show the values obtained for the application estimated execution time considering the amount of generated replicas and the execution environment MTBF. The simulations considered an application execution time in the absence of failures of 18 and 36 hours for figures 2 and 3, respectively. By analyzing the simulation results, we can conclude that: a) as the failure rate increases (lower MTBF values), a higher amount of replicas is necessary to keep the application execution time inside an acceptable range (a tolerance value); b) if we fix the MTBF value, its not profitable, after a certain point, to increase the amount of replicas, since the gain would be minimum while more Grid resources would be used. For example, on figure 2 when the MTBF is 4 or 8, using 2, 4, or 9 replicas would make no significant difference to the application execution time. On figure 3 the same thing can be seen when the MTBF is 8; c) some times great gains can be achieved by slightly varying the number of replicas. For instance, on figure 2 we can see an enormous advantage by using 2 replicas instead of 1 when the MTBF is between 0.5 and 2. Also on figure 3 the same benefit is obtained when the MTBF is between 2 and 4.



Fig. 2. Mean Execution Time of 18 hours

The simulation results clearly indicate the benefits of altering the amount of application replicas as the MTBF and application execution time varies. On the other side, the dynamic nature of the Grid execution environment makes it very difficult (if not impossible) for a Grid user or administrator to manually decide and set the best amount of replicas to be generated for each application submission. AutoGrid contains a monitor that periodically measures the current Grid MTBF based on a database generated by the Execution Manager component. This database contains information about each application execution, such as: the global submission



Fig. 3. Mean Execution Time of 36 hours

and conclusion timestamps and detailed information about the execution of each process that comprises the application execution, including data concerning eventual failures that may have occurred.

D. AutoGrid Self-optimization Mechanism

The computational grid comprise a highly dynamic environment. Dynamism is more noticeable on opportunistic grids, that leverages idle computing power of personal workstations for executing computationally-intensive parallel applications. Because grid nodes are not dedicated, it is common that applications assigned to resources perform poorly than expected. Load balancing techniques, adaptive scheduling and dynamic re-scheduling are important topics to be investigated.

Maheswaran et al [14] quantifies the relative performance of scheduling heuristics. It discusses that the choice of a particular scheduling algorithm is a function of factors, such as the arrival rate of the tasks, the optimization heuristic used (e.g., optimizing makespan versus optimizing average sharing penalty). It uses simulations to prove that there is always an algorithm that is more appropriate for a given environmental situation.

Dong et al [6] refers to an adaptive solution to the scheduling problem as the one in which the algorithms and parameters used to make scheduling decisions change dynamically according to the previous, current and/or future resource status. It also points that adaptive scheduling can optimize overall Grid performance and minimize application response time.

Based on those studies, dynamically replacing grid scheduling algorithms is profitable, and AutoGrid provides the means for that. For example, one could use the MCT (minimum completion time), an on-line heuristic, instead of min-min, which is a batch heuristic, when the arrival rate of the tasks is above a certain threshold. Another important attribute that affects selection of the appropriate grid scheduling algorithm is resource utilization; for example, one could use the max-min heuristic to maximize concurrency if resource usage is low. During algorithm replacement, AutoGrid manages state transfer, which takes into account the queue of applications stored by the algorithm.

The Global Resource Manager (GRM) comprise AutoGrid scheduler. GRM consists of a set of scheduling algorithms and knowledge of the appropriate moment to trigger the dynamic switch. Investigations are being made to measure the exact moment of replacement, according to the application arrival rate, resource usage, network status and other environmental properties.

V. CONCLUSIONS AND FUTURE WORK

Computational grids have become an attractive alternative for executing parallel and distributed applications that demand high computational power, and also for integrating computational resources spread through different administrative domains. However, the dynamic nature of the grid infrastructure, its high scalability and great heterogeneity has turn impracticable its configuration, maintenance and recovery in case of failures solely by humans beings.

This paper presented AutoGrid, a self-managing autonomic grid system that uses Integrate grid middleware as its foundation. AutoGrid is capable of (re)configuring itself according to context data regularly observed from the executing environment. AutoGrid also features self-healing, in its ability to recover from application failures without human intervention, and self-optimization, by minimizing application response time while optimizing overall Grid performance.

AutoGrid design emphasizes openness and flexibility. Therefore, its monitoring infrastructure allows to introduce, remove and replace context monitors according to the deployment environment. Moreover, the adaptation engine itself is extensible which permits that advanced users design new adaptive mechanisms to achieve different and more robust autonomic features.

AutoGrid autonomic features (context-awareness, self-configuration, self-healing and self-optimization) are being evaluated using experiments and simulations that clearly indicate that adaptive approaches bring considerable benefits to the grid infrastructure.

We are currently introducing into AutoGrid a set of grid scheduling algorithms which are selected on runtime according to the execution environment; also, we are measuring the benefits of adaptive scheduling to overall grid performance. We are investigating other self-optimization mechanisms, such as dynamic load balancing and re-scheduling of applications. AutoGrid context-awareness feature is being augmented with interfaces that export context-data to grid applications. Finally, we are extending AutoGrid adaptation engine with a component migration mechanism.

ACKNOWLEDGMENT

AutoGrid is part of the Integrate2 project, supported by the Brazilian Federal Research Agency, CNPq, grant No. 55.0094/05-9.

REFERENCES

- [1] M. Agarwal, V. Bhat, H. Liu, V. Matossian, V. Putty, C. Schmidt, G. Zhang, L. Zhen, and M. Parashar. Automate: Enabling autonomic grid applications. Technical Report TR-269, The Applied Software Systems Laboratory, Department of Electrical and Computer Engineering, Rutgers University, 2003.
- [2] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crumme, D. Reed, L. Torczon, and R. Wolski. The grads project: Software support for high-level grid application development. *Int. J. High Perform. Comput. Appl.*, 15(4):327–344, 2001.
- [3] E. Caron, A. Chis, and A. Su. Plug-in scheduler design for a distributed grid environment. In *MCG '06: Proceedings of the 4th international workshop on Middleware for grid computing*, page 6, New York, NY, USA, 2006. ACM Press.
- [4] R. Y. de Camargo, A. Goldchleger, F. Kon, and A. Goldman. Checkpointing-based rollback recovery for parallel applications on the integrate grid middleware. In *MGC '04: Proceedings of the 2nd workshop on Middleware for grid computing*, pages 35–40, New York, NY, USA, 2004. ACM Press.
- [5] R. Y. de Camargo, F. Kon, and R. Cerqueira. Strategies for checkpoint storage on opportunistic grids. *IEEE Distributed Systems Online*, 7(9):1, 2006.
- [6] F. Dong and S. G. Akl. Scheduling algorithms for grid computing: State of the art and open problems. Technical Report 2006-504, School of Computing, Queens University, Kingston, Ontario, January 2006.
- [7] A. Goldchleger, F. Kon, A. Goldman, M. Finger, and G. C. Bezerra. Integrate: Object-oriented grid middleware leveraging idle computing power of desktop machines. *Concurrency and Computation: Practice and Experience*, 16(5):449–459, March 2004.
- [8] S. Hariri, B. Khargharia, H. Chen, J. Yang, Y. Zhang, M. Parashar, and H. Liu. The autonomic computing paradigm. *Cluster Computing*, 9(1):5–17, 2006.
- [9] S. Hwang and C. Kesselman. Gridworkflow: A flexible failure handling framework for the grid. In *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*, page 126, Washington, DC, USA, 2003. IEEE Computer Society.
- [10] J. Kaufman, T. Lehman, G. Deen, and J. Thomas. Optimal-grid: Autonomic computing on the grid. URL: <http://www-128.ibm.com/developerworks/library/gr-opgrid/>.
- [11] J. Kaufman, T. Lehman, G. Deen, and J. Thomas. Optimalgrid – autonomic computing on the grid, June 2003.
- [12] F. Kon. *Automatic Configuration of Component-Based Distributed Systems*. PhD thesis, University of Illinois, Urbana, Illinois, 2000.
- [13] P. Leong, C. Miao, and F. L. B. Sung. Agent mediated autonomic service orchestration in grid environment. In *3rd IEEE International Conference on Industrial Informatics (INDIN)*, pages 148–153, 2005.
- [14] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *HCW '99: Proceedings of the Eighth Heterogeneous Computing Workshop*, page 30, Washington, DC, USA, 1999. IEEE Computer Society.
- [15] A. Mukhija and M. Glinz. Runtime adaptation of applications through dynamic recomposition of components. In *ARCS*, pages 124–138, 2005.
- [16] J. B. Pinheiro, R. Camargo, A. Goldchleger, and F. Kon. Integrate: a tool for executing parallel applications on a grid for opportunistic computing. In *Proceedings of the 23th Brazilian Symposium on Computer Networks (SBRC Tools Track)*, Fortaleza-CE, Brasil, May 2005.
- [17] M. A. S. Sallem and F. J. da Silva e Silva. Adapta: a framework for dynamic reconfiguration of distributed applications. In *ARM '06: Proceedings of the 5th workshop on Adaptive and reflective middleware (ARM '06)*, page 10, New York, NY, USA, 2006. ACM Press.
- [18] J. W. Yoder and R. E. Johnson. The adaptive object-model architectural style. In *Proceedings of the IFIP 17th World Computer Congress TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture*, pages 3–27, Deventer, The Netherlands, The Netherlands, 2002. Kluwer, B.V.