An Economics-Driven Dynamic Search Approach for Self-Managing Power in Software Architectures

(Search-Based Software Engineering - SBSE Track)

1st Author
1st author's affiliation
1st line of address
2nd line of address
Telephone number, incl. country code
1st author's email address

2nd Author
2nd author's affiliation
1st line of address
2nd line of address
Telephone number, incl. country code
2nd E-mail

3rd Author
3rd author's affiliation
1st line of address
2nd line of address
Telephone number, incl. country code
3rd E-mail

ABSTRACT

One of the novel contributions of this paper extends the current static Search-Based Software Engineering (SBSE) work by introducing an economics inspired approach, which exploits market-based control, to "dynamic" or "online" software engineering problems. Self-adaptive energy consumption and management in software architectures in relation to dynamic changes in scalability requirements is used as a case study to illustrate how market-based control can deal with dynamic SBSE problems. We suggest an architectural layer, which incorporate the market-based approach. This layer dynamically optimizes energy consumption of a runtime architectural instance by continually monitoring and matching the resource requirements (demand) with the resource availability (supply) in relation to various scalability scenarios. Simulation of the approach on a client-server architecture shows that the layer can dynamically search and improve the power savings while maintaining the desired scalability requirements of a software system. The contribution pioneers new line of research in dynamic SBSE explicating green-aware self-managed software architectures.

Keywords

Green software engineering, dynamic search-based software engineering, software architecture, power management, and self-adaptation.

1. INTRODUCTION

Monitoring and self-managing the tradeoffs between scalability provision and power consumption at runtime *dynamically* evolves the software syste*m*. Dynamic evolution refers to the changes, which occur while the system is operational [5]. This is demanding and requires that system evolve dynamically and the adaptation for self-managing the trade-off occurs at runtime. Current research and practice pursue software architectures as the appropriate level of abstraction for evaluating, reasoning about, managing and facilitating the *dynamic change and evolution* of complex software systems, explicitly accounting for the dependability requirements. We advocate an architectural based approach to monitor and self-manage such tradeoffs, as such is believed to offer the potential benefits of generality (i.e. the underlying concepts and principles are applicable to a wide range of application domains); appropriate level of abstraction to tackle the complexity of the problem (i.e. software architecture can provide the appropriate level of abstraction in describing dynamic changes as opposed to algorithmic level); potential for scalability (i.e. facilitating the use in large-scale complex applications), and providing opportunities to facilitate automated analyses benefiting from Architecture Description Languages analyses(ADLs).

The application of search based optimization techniques to software engineering has recently witnessed intense activity right across the life-cycle from requirements engineering, project planning and cost estimation through testing, to automated maintenance, service oriented software engineering, compiler optimization and quality assessment. A wide range of "classical" optimization and search techniques has been used. These are mainly local search, simulated annealing, genetic algorithms and genetic programming. As mentioned by Harman [9], all applications of SBSE has so far been concerned with 'static' or 'offline' optimization problems. In "static"

problems, the algorithm is executed off line in order to find a solution to the problem in hand. In "dynamic" optimization or "on line" SBSE, solutions are repeatedly generated in real time and applied during the lifetime of the execution of the system to which the solution applies [9]. This is in line with the work in dynamic optimization, which is directly relevant to dynamic SBSE[10][11]. Moreover, the runtime nature of dynamic SBSE does require lightweight, simple, scalable, and computation inexpensive optimization techniques. This is necessary as the search is continuously active and may be initiated at various time intervals of the execution not only for seeking to find an optimal solution to the said problem, but rather, it may seek to improve upon the current runtime situation. Classical optimization and search techniques may be ineffective and expensive to use in addressing the dynamic SBSE requirements for the representation of the problem and the definition of the fitness function are mere active at runtime and volatile with respect to time.

This paper makes novel contribution to SBSE on three fronts: (i) the contribution concerns with the dynamic runtime search for an optimal solution for the problem of dynamic monitoring and self-management of software architectures in addressing the tradeoffs between scalability provision and power consumption at runtime; (ii) The contribution extends the current static SBSE work by introducing an economics inspired approach, using market-based theory, to online dynamic optimization of software engineering problems. Self-adaptive energy consumption management in software architectures is used as a case to illustrate how market based approaches can effectively deal with dynamic software engineering problems. The contribution suggests an architectural layer, which makes a novel use of the classical supply and demand (market-based economic theory) keeping the dynamic energy management process and its optimization simple, intuitive, and appealing. This layer dynamically optimizes energy utilization of a runtime architectural instance by continually monitoring and matching the resource requirements (demand) with the resource availability (supply) in relation to various scalability scenarios. In this context, we draw on an analogy with the supply/demand concepts of a market economy with the resource availability and resource demand of a computer system. By scaling up/down the resource availability to match with its demand, we can avoid unnecessary wastages of power due to inaccurate resource provisioning. Simulation results show that our framework dynamically searches and improves the power savings, for web based client-server architecture while maintaining the desired scalability requirements of a system. The suggested architectural layer provides the promise to be integrated with existing architecture styles for making them green-aware; (iii) most importantly, the contribution pioneers new line of research in dynamic SBSE explicating green-aware software architectures and engineering, where dynamic SBSE can provides new opportunities to the filed of dynamic self-management of software architectures in relation to both power and dependability provisions. This initiative feeds into the long-term and green-aware vision of helping in reducing power consumption and CO2 emissions in ICT infrastructures.

The rest of the paper is organized as follows: Section 2 discusses the motivation for the research. Section 3 describes a market-based approach to dynamic SBSE for managing the tradeoffs between scalability demands and power consumption. The experimental results and evaluation are presented in section 4. Related work is discussed in section 5. Section 6 concludes.

2. MOTIVATIVATION

Software systems architects are continually faced with the challenge of scaling up software systems architectures to support constantly growing load of users' processing needs and data. Scaling up the architectures to meet these needs does certainly introduce additional energy cost. For example, to meet the scalability requirements, additional hardware and/or software resources may need to be deployed. Reducing the energy demands in such architectures while meeting the scalability requirements is a challenging dynamic runtime optimization problem. We explicate the attention to power as an architectural constraint/property that need to be analyzed in relation to scalability.

We argue that the software engineering should be green aware, where the software engineering and design activities should not only be judged by their technical merits, but also by their contributions to energy savings. In particular, the software system architecture appears to be the appropriate level of abstraction to address green-aware concerns. We argue that there is a pragmatic need for new software architectural layer, which could be easily integrated with existing styles for self-managing the trade-offs between scalability and power. The power consumption can be minimized by only provisioning the required amount of resource at any given point of time. For example, architecture can be scaled only when the demand for the resource increases. Classical market-based economic theory is appealing for addressing this problem effectively in the context of supply/demand.

We model power, scalability and their tradeoffs from an economics-driven software engineering perspective [17]. We view dynamic evolution of software architectures, while accommodating scalability with minimal power as

one of investing/managing valuable resources with the goal of maximizing the value added. The value added is reduced power cost, while achieving desirable scalability levels. The fundamental premise is that the architecture (and their components) experiences no uniform workloads exhibiting variation in scalability requirements during its operation time. Such an assumption is valid for most systems, both when considered in isolation and when internetworked (e.g. cloud, ultra-large-scale software systems). A second assumption is that it is possible to monitor, with a certain degree of confidence, the fluctuations of workload and their power as we show in the proposed framework. Such observation and prediction should not consume significant energy, however. In this context, we argue that dynamic change and evolution of software architectures is a value-seeking and value-maximizing process, where the architecture is undergoing a dynamic change (at runtime) and seeking value. We treat scalability provision and their power consumption as scare resources, which need to be dynamically optimized. Performing a runtime search for best architectural instances, which address the scalability requirements with minimal power is a problem, which is appealing to dynamic' or 'on line' Search-based Software Engineering (SBSE), where the optimization problem is rapidly changing and the current best solution must be continually adapted. The goal is to maximize the value added and select the optimal execution plans.

3. PROPOSED APPROACH

Our proposed solution is a conceptual architectural level framework, which incorporates the dynamic/runtime search for managing the tradeoffs between power and scalability for an architecture instance, as depicted in figure 1. The framework draws an analogy with the supply/demand market economic theory in performing the dynamic search. In the next subsections, we present a brief overview of supply demand theory. We then describe the proposed architecture, which incorporate the economics-driven dynamic search and we elaborate on the exploited analogy. We discuss various runtime scenarios arising from its application.

3.1 Economic theory

A market economy is described using two terms: Demand and Supply. Demand is defined as the quantity of goods consumers are willing to buy. Supply is defined as the quantity of goods the suppliers are willing to produce. Inflation scenario occurs when the supply of a good could not meet the demand. Similarly, a recession scenario occurs when the supply is in excess with demand. In order to keep the economy in equilibrium (neither inflation nor recession), the supply should always be matched with the demand. At this point, all the goods supplied are consumed, without any wastage.

3.2 Proposed Architecture

We build on an analogy with the classical economic theory of 3.1 to perform the dynamic search for an optimal architectural instance at runtime, which maintain the desirable level of scalability while saving energy. Figure 1 depicts the architecture of our proposed model. As shown in the figure, the economic layer of our framework is composed of four components. The Supply Manager (SM) and Demand Manager (DM) components deal with the supply i.e. resource availability and demand i.e. resource requirement concerns of a system. (The resource availability can be defined as number of nodes in a cluster and the resource requirements can be defined as number of incoming requests per second.) The Supply/Demand Coordinator (Coordinator) controls these two components to reach the equilibrium state where the resource availability matches its demand. In this state, unnecessary wastage of power due to resource over-provisioning can be avoided. For example, the Coordinator notifies the SM to decrease the resource availability whenever the resource demand decreases and vice versa. The additional strategic planning component is responsible for handling change management in the framework. The translational layer is used to map the abstract architectural concerns with its implementation concerns. The system layer represents the underlying runtime architectural instance, which is being power, managed. The sensors and actuators, attached to the system, monitor and control the system specific values. A detailed view of the components of economic layer is given below.

Demand Manager: It collects the resource demand information of the underlying system through the sensors. For example, the resource demand of client server architecture can be determined by the number of pending requests in the queue. This information is collected for a specified time interval, consolidated and transformed into an abstract index value called *demand index*. The demand index is analogous to the number of goods demanded in the context of economic theory. The *demand index* for a web based client-server architecture can be calculated as follows: number of pending requests/(number of active nodes * 100). This component also performs actions on the underlying system such as blocking the incoming requests from distributing to the server for processing, upon receiving notification from the Coordinator.

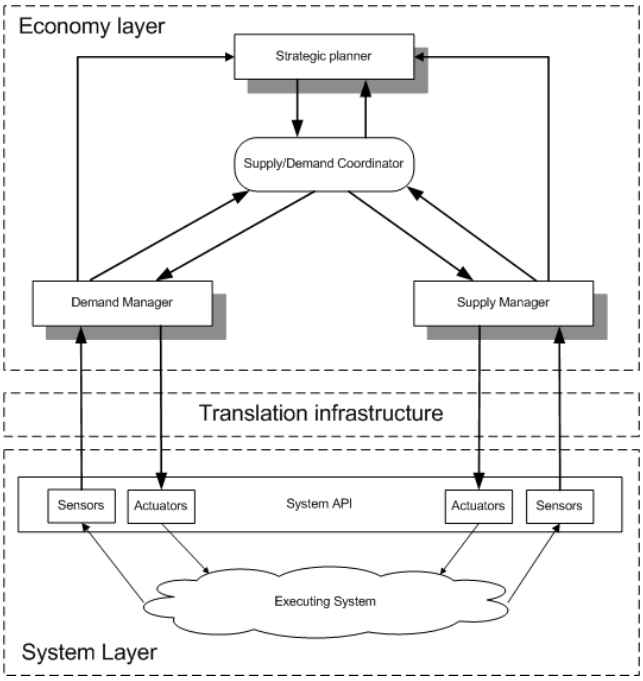

Figure 1: Economic framework

Supply Manager: It collects the resource availability information of the underlying system through the sensors. For example, the resource availability of cluster based client-server architecture can be determined by the number of active nodes and their respective loads in the cluster. This information is collected for a specified time interval, consolidated and transformed into an abstract value called *supply index*. The *supply index* is analogous to the number of goods supplied in the context of economic theory. The *supply index* for a web based cluster server is calculated by the ratio of cumulative available capacity of the cluster in percentage (%) and the total capacity of the cluster in percentage (%). It also performs actions such as increasing or decreasing resource levels, on the underlying system, upon receiving notification from the Coordinator.

Supply/Demand Coordinator (Coordinator): The coordinator is the abstract component, which acts as a mediator between the SM and DM by continuously monitoring the supply and demand indices. It consists of predefined configurations of what action to be taken when the *demand index* varies relative to the *supply index*. It obtains the index values from both SM and DM, compares them, chooses an appropriate action (to match the supply with the demand) and controls the SM and DM to execute the action. For example, whenever it encounters an *inflation* or *recession constraint*, it compares the supply and demand indices, decides whether to increase or decrease the resource supply by notifying SM and DM to take action accordingly.

Strategic Planner: This component produces change management plans upon request from Coordinator. For example, it reconfigures the framework when the underlying system is integrated with external resource, on the fly, to meet unanticipated demand. It can also recreate failed components of the framework to address the fault-tolerance.

Next section describes about how these components work interactively to manage power in scalable client server architecture.

3.3 Working Scenarios

We describe an architecture scenario, which utilizes the proposed power management layer. We use an instance of client server architecture which consists of a serverCluster of replicated nodes to meet the scalability requirements. The client requests are sent to the server for processing through a request queue. Power saving can be achieved in such an environment by provisioning the resource according to the demand i.e. by dynamically switching off some of the nodes under lighter load conditions to save power. Likewise, scaling up can be achieved by dynamically switching on some of the nodes under burst load conditions. Our proposed framework automates this process efficiently and dynamically using supply and demand theory.

The DM controls the demand layer of the cluster which is its request queue. The SM controls the supply layer of the cluster, which is the ServerCluster itself. Both the layers should have respective sensors and actuators attached to them for monitoring and controlling the changes respectively. The DM monitors the request queue and calculates the *demand index*. Similarly, the SM monitors the load in each cluster node and calculates the *supply index* at regular intervals. The SM and DM regularly update the latest index values to the Coordinator, which keeps track of the same, anticipating for constraint violations.

Inflation Scenario: An inflation scenario occurs when the *supply index* value reaches below 0.2 (80% of the resources are utilized in the cluster). Assume that there is one node active in the cluster which is running at 85% of its load. Assume also that there are 110 requests in the queue. A node can process up to 100 requests at its peak load. The *demand index* at this point of time is calculated as 110/(1*100) = 1.1 and the *supply index* is calculated as (100-85)/(100) = 0.15. As the supply index is less than 0.2 (more than 80% of the cluster is utilized), an *inflation constraint* is flagged at the Coordinator. When the *inflation constraint* is flagged, the Coordinator compares the *supply index* with *demand index* to see if the *demand index* exceeds

supply index by more than 0.2 (0.2 is the threshold difference between supply and demand index configured in the Coordinator, but it can be varied according to the underlying system), which means that the supply cannot meet the demand requirements. If this condition occurs, the Coordinator starts self adaptation process by notifying the SM to increase the supply and wait for an acknowledgement from SM upon completion of the action. It also notifies the DM to decrease the demand temporarily, as overloading the cluster may increase the response time.

After receiving notification, the SM controls the underlying cluster (with the help of the actuator) to activate a new node and DM controls the request queue of the underlying cluster (with the help of actuator) to stop sending the low priority requests to the cluster. Once a new node is activated, SM updates the Coordinator with an acknowledgment and the new *supply index* value which is (200-85)/(200) = 0.575. The Coordinator in turn compares this with the new *demand index* obtained from DM which is 110/(2*100) = 0.55. Since the difference is now below 0.2, it updates the demand manager to normalize the demand distribution (start distributing the low priority requests). Thus, minimal resource configuration is sufficient for the architecture at initialization in order to save power. The architecture can be scaled dynamically in the later stage as the demand increases.

Recession Scenario: Recession scenario occurs when the *supply index* goes above 0.4 (only 60% of the cluster load is utilized). Let us assume that there are three active nodes in the cluster, which are running respectively at 40%, 50% and 60% of the load. Assume that the number of requests that are queued in the system at present is 60. Demand index at this point is 60/(3*100) = 0.2 and the supply index is ((100-40) + (100-50) + (100-60))/300 = 0.5. As the *supply index* is above 0.4, a *recession constraint* is flagged in the Coordinator. When the *recession constraint* is flagged, the Coordinator compares the two indices to see if *demand index* exceeds the *supply index* by 0.2, which means there is excess resource available. If this condition occurs, the Coordinator starts the self-adaptation process by notifying the SM to decrease the supply and waits for the acknowledgement. After receiving the notification, the SM deactivates a node which is running at 40% load by transferring its load into other nodes with 20% each. Now, the new supply index is (100-(50+20)) + (100-(60+20)) / 200 = 0.25 and the new demand index is 60/(2*100) = 0.3. The difference is below 0.2. Thus, the opportunity of reducing the resource availability is utilized effectively to conserve power.

Higher inflation/lower recession scenarios: After notifying the SM to increase the supply due to an *inflation constraint*, the coordinator continues monitoring the latest supply and demand indices. If the demand index grows at a higher rate, for example, when the new supply cannot match the current demand, it notifies the SM to scale up the resource further without waiting for the acknowledgment for the previous notification. Thus, more than one node is activated simultaneously which reduces the activation time by almost half and hence improves the overall performance. Similarly, it notifies the SM to scale down the resources further when the demand index decreases rapidly. Deactivating simultaneous nodes without any delay will result in significant power saving. It also shows how effectively our framework responds to burst increase in the demand by scaling the architecture at higher rate.

These different scenarios clearly show the effectiveness of our approach in managing the trade-offs between scalability and power. There are other scenarios where the *supply index* is below 0.2 or above 0.4, but the difference between the indices is not more than 0.2. In this case, the actual supply of resources is still met with the current demand and hence no self-adaptation is required. Also, there are chances where the difference between the indices may be more than 0.2, but the *supply index* is maintained between 0.2 and 0.4. In this case, the resource utilization is still optimal (60% to 80%) and hence no self-adaptation is required.

4. PRELIMINARY RESULTS AND EVALUATION

To test the feasibility of our approach, we have implemented the framework, which incorporate the dynamic economics-driven search. The components of the proposed architectural layer were implemented as C# classes with multi-threading feature to monitor and take actions simultaneously. We have simulated a client-server architecture by means of a cluster-based web application with eight replicated nodes. We have simulated the web clients and generated the requests in different patterns such as sudden burst and gradual increase in load. Figure 2 shows the evolution of the cluster for various resource demands levels, and consequently power, at different time intervals. The result demonstrates the effectiveness of our approach in dealing with both scalability requirements and power conservation by scaling up and down the architecture according to the resource needed. From figure 2, we observe that the bursts of incoming requests (between 900-1000th sec) are handled effectively by adding more than one server simultaneously. Furthermore, the simulation revealed the delay involved in increasing the supply,

which is equal to the time taken to activate a node. Though this latency is noticeable for single node activation, the delay is effectively utilized while activating more than one node (higher inflation scenario).

Our approach appears to be scalable, simple and intuitive to use in addressing the dynamic requirements for our search in matching supply with demand at runtime. The representation of the problem is handled by the economic layer of the framework and the corresponding SM, DM, and the Supply/Demand coordinator and strategic planner. The definition of the fitness function is mere active and dynamic at runtime. Interestingly the equilibrium thought in matching supply to demand and/or in tuning the recession and inflation scenarios to equilibrium is grounded and implicit in the application of the theory; henceforth, making it simple and intuitive to use.

Though we experimented only with client/server architecture, our approach could be integrated with other architecture styles. The standalone nature of the architectural layer makes it reusable across different architectures. Our approach can also scale efficiently for larger architectural styles, which abstract more than one subsystem. For example, supply and demand index values of each subsystem are accompanied with a unique ID such that more than one subsystem can be handled effectively by a single framework.

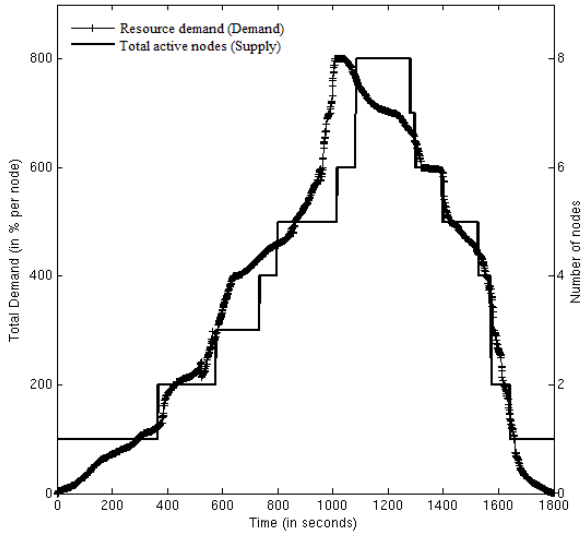

Figure 2: Cluster evolution on applying economic theory

5. RELATED WORK

A self-managed software architecture is one in which components automatically configure their interaction in a way that is compatible with an overall architectural specification and achieves the goal of the system. There has been a number of ongoing work on self-managed software architectures, as reported in the roadmap of Kramer and Magee [5] and documented in related workshops, such as SEAMS (Software Engineering for Adaptive and Self-Managing Systems), WOSS (Workshop on Self-healing and Self-managed Software) and other workshops affiliated with CDS (Configurable Distributed Systems). Most notably, [13] pioneered research in dynamic change and evolution of software architectures. [13][14] proposed associated analysis techniques and language for dynamic changes in software architectures and their initial steps towards self-management were reported in [21]. [15] has extensively researched the problem; they propose a language and associated environment for architecture-based development and evolution. [19] describes an architecture evolution manager, which assists the infrastructure in run-time adaptation and self healing. [16] describes an architectural approach focusing on adaptation and evolution management. [20] describes architecture models supporting self-healing. [18] propose an infrastructure for both component- and application-level reconfiguration using a hierarchy of managers. Although the research work discussed over the years has provided much that is useful in contributing towards self-management, it has not yet tackled the problem of self-managing the tradeoffs between power and dependability at the software architecture level.

Search based optimization techniques have been applied to various problems in software engineering from requirements engineering, project planning and cost estimation through testing, to automated maintenance, service oriented software engineering, compiler optimization and quality assessment. The Software Engineering by Automated Search (SEBASE) information page (SEBASE.org) provides a comprehensive up-to-date account of the various applications of optimization techniques to software engineering problems. To sum up, the majority of applications addressed over the years are of static nature or ‘offline’ optimization problems [9]. In “static” problems, the algorithm is executed off line in order to find a solution to the problem in hand [9]. Our work, in contrast, is concerned with “dynamic” or “on line” SBSE, where the solutions are repeatedly generated in real time and applied during the lifetime of the execution of the software architecture [9]. Furthermore, we have motivated the use of market-based theory as a simple, intuitive, and inexpensive dynamic search technique. Up to the author’s knowledge, none of the software architectures contributions over the years have addressed the tradeoff problem of scalability and power from a search-based software engineering perspective.

The use of market-based approach for self-managing

Quality Attribute in the cloud has been reported in [21][22]. The approach, however, does not consider power as an attribute in the runtime optimization process.

Rainbow framework from [4] is in the spirit of our layering approach to self-management. However, their self-management process is not aimed at energy management concerns. The Rainbow framework does not exploit the market-based theory, which we propose. The work from [6] is closely related to our work in terms of architectural level energy management but, their each self-adaptation process has to wait for a specific time interval till the previously executed adaptation change is stabilized. However, our approach demonstrated its effectiveness in cases of higher inflation and lower recession scenarios by handling more than one adaptation process simultaneously. Furthermore, the economic theory and the two levels of control on the underlying system (supply and demand) makes our approach stand out from theirs. [2] worked on a similar approach with resource prediction strategy for efficient energy management, but we argue resource prediction may sometimes be inaccurate and thus incurs additional energy costs due to unanticipated situations. [1] have focused on distributed decentralized self adaptation. But, a decentralized approach to power management incurs additional energy costs for different number of the components to be installed. We compliment the work of Chiyoung et. al. [7] for estimating the energy consumption of our architecture layer itself. There are few efforts made in applying market-based economy in system implementation level. [8] Compiles some of the approaches.

6. CONCLUSION AND FUTURE RESEARCH DIRECTION

Green-aware constraints such as power bring new challenges to the way we systematically evolve and scale software architectures. The contribution has extended the current "static" SBSE work by introducing an economics inspired approach, using market-based theory, to address online dynamic optimization of software engineering problems. Self-adaptive energy consumption monitoring and management in software in relation to scalability requirements has been used as a case to illustrate how market based approaches can be effective in dynamic SBSE.

Different power un-aware architectures can benefit from our approach with minimal modifications. Many modern architectural paradigms, such as cloud, which has clear separation of supply and demand, can also benefit from our approach. As a part of our future research, we are looking at relating other architectural dependability requirements to power. We are also working on extending the functionality of DM to raise inflation or recession scenarios to improve the overall self-adaptation process.

The research will raise the understanding of evolution trends in dynamic systems, and improve their quality and robustness through dependability and power measurement and control. More widely, we hope the research results will feed into long-term vision of helping in reducing power consumption and CO2 emissions in ICT infrastructures.

7. REFERENCES

[1] L. Baresi, S. Guinea, and G. Tamburrelli. Towards decentralized self-adaptive component-based systems. In SEAMS '08: Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems, pages 57–64, 2008.

[2] S.-W. Cheng, V. V. Poladian, D. Garlan, and B. Schmerl. Improving architecture-based self-adaptation through resource prediction. In Software for Self-Adaptive Systems, pages 71–88, 2009.

[3] D. Garlan. Software architecture: a roadmap. In ICSE'00: Proceedings of the Conference on The Future of Software Engineering, pages 91–101, 2000.

[4] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. Computer, 37(10):46–54 October 2004.

[5] J. Kramer and J. Magee. Self-managed systems: an architectural challenge. In FOSE '07: 2007 Future of Software Engineering, pages 259–268, 2007.

[6] V. Petrucci, O. Loques, and D. Moss´e. A framework for dynamic adaptation of power-aware server clusters. In SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing, pages 1034–1039, 2009.

[7] C. Seo, G. Edwards, S. Malek, and N. Medvidovic. Framework for estimating the impact of a distributed software system's architectural style on its energy consumption. In WICSA '08: Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008), pages 277 280, 2008.

[8] C. S. Yeo and R. Buyya. A taxonomy of market-based resource management systems for utility-driven cluster computing. Softw. Pract. Exper., 36(13):1381–1419, 2006.

[9] M. Harman: The Current State and Future of Search Based Software Engineering. FOSE 2007: 342-357, 2007.

[10] S. Yang, Y.-S. Ong, and Y. Jin (editors): Evolutionary Computation in Dynamic and Uncertain Environments, in the book series Studies in Computational Intelligence,vol. 51, Springer-Verlag Berlin Heidelberg, 2007.

[11] Y. Jin and J. Branke (guest-editors), Special Issue on Evolutionary Optimization in the Presence of Uncertainties, IEEE Transactions on Evolutionary Computation, vol. 10, no. 4, August 2006.

[12] R. Bahsoon and W. Emmerich, W: "An Economics-Driven Approach for Valuing Scalability in Distributed Architectures". In Proc. of

the 7th Working IEEE/IFIP Conference on Software Architecture (WICSA 2008), Vancouver, Canada. IEEE Computer Society, 2008.

[13] J. Kramer and J. Magee: Analyzing Dynamic Change in Distributed Software Architectures. In: IEE Proceedings Software, 145. 146-154, 1998.

[14] J. Magee and J. Kramer: Dynamic Structure in Software Architectures. In: Proceedings of the 4th ACM SIGSOFT Symposium on Foundations of Software Engineering, ACM Press, San Francisco, California, United States, 1996.

[15] N. Medvidovic, D. Rosenblum, D. and R. Taylor: A Language and Environment for Architecture-based Software Development and Evolution. In: Proc. of the 21st Int. Conf on Software Egineering. IEEE CS Press, 1999.

[16] P. Oreizy, M. Gorlick, R. Taylor, D. Heimhigner, GJohnson, N. Medvidovic, N, Quilici, D. Rosenblum, A. Wolf: An Architecture-based Approach to Self-adaptive Software, Intelligent Systems and Their Applications, IEEE, 1999.

[17] EDSER 1-8: Proceedings of the Workshops on Economics-Driven Software Engineering Research: In conj. with the 21st through 28th International Conference on Software Engineering 1999 – 2006.

[18] M. Carzaniga, A., P. Inverardi, P. and A. Wolf Light-weight Infrastructure for Reconfiguring Applications: In: Proc. of 11th Software Configuration Management Workshop (SCM03), LNCS, 2003.

[19] E. Dashofy, E., A. van der Hoek, and R. Taylor: Towards architecture-based self-healing systems. In: Proc. of the first workshop on Self-healing systems, ACM Press, Charleston, South Carolina, 2002.

[20] D. Garlan D. and B. Schmerl: Model-based Adaptation for Self-healing Systems. In: Proc of the first workshop on Self- healing systems, ACM Press, Charleston, South Carolina, 2002.

[21] V. Nallur, R. Bahsoon, and X. Yao. Self-Optimizing Architecture for Ensuring Quality Attributes in the Cloud. In the Proc. of the 8th Working Conference on Software Architecture (WICSA 2009), Cambridge, UK, 2009.

[22] V. Nallur, R. Bahsoon. Design of a Market-Based Mechanism for Quality Attributes Tradeoffs of Services in the Cloud, To appear, in the Proceedings of the 25th ACM Symposium of Applied Computing (ACM SAC 2010), 2010.