Autonomic Computing for B2C E-Commerce Applications

Devasia Kurian
Christ University, Hosur Road,
Bangalore 560029,
Karnataka, India

Pethuru Raj Chelliah, PhD.
Wipro Consulting Services Division
Wipro Technologies, Sarjapur, Bangalore 560035
Karnataka, India

ABSTRACT
Ever increasing complexity, higher demand for pro-activeness, and high speeds of innovation resulting from heavy competition, demand the adoption of more intelligent systems, which are capable of producing optimal results, in all fields. B2C E-Commerce applications are no different. The major challenge in transitioning a brick and mortar business to an online environment is to provide the same user experience as that of a wayside store, like the consultation, up selling, pro-activeness, negotiation, delivery, etc. This requirement has created a lot of intelligent tools and necessitates further evolution of more intelligent tools. Autonomic computing provides the framework for design of independent intelligent self-managing components, and is thereby optimally suited to assist E-Commerce in this journey. This framework can be extensively used to upgrade existing E-Commerce systems with autonomic features. This paper introduces the concept of autonomic computing in e-commerce applications, and provides a generic architecture, with specific focus on self-optimizing characteristics of autonomic computing. Details of concrete implementation of autonomic components in an e-commerce environment are provided.

This paper is the specific application scenario of the generic autonomic concept presented in "Autonomic Computing Architecture for Business Applications" [1].

General Terms
Autonomic computing, E-Commerce

Keywords
autonomic computing; e-commerce; associative display; differential evolution; automatic discount; online shop

1. INTRODUCTION
The efficient transition of a brick and mortar business to online business requires much more than a website to display products, select the products and pay for them. E-Commerce in today's stage needs more sophisticated, proactive, and intelligent systems to achieve even a reasonable conversion. The advances made in various areas such as analytics, data mining, usability, artificial intelligence accelerate the move towards intelligent systems. Autonomic computing is well suited to become an underlying design principle for the creation of such systems. In fact, the conscious introduction of autonomic computing design in projects can be considered as a disruptive movement, which will help us create a world of intelligent equipments and systems.

If we happen to find ourselves in an adverse situation, like in front of a lion, our body immediately increases the heart rate, pumps adrenalin into blood, and alerts all the reflexes. Thanks to millions of years of human evolution, nature has built in quite a few mechanisms into our autonomous nervous system, which help us to sleep soundly, as well as take care of critical situations. Compared to the thousands of years of human evolution, computer systems have not really crossed the first hundred years. These self-managing characteristics exhibited in human beings need to be built into appliances, servers, networks, and applications to create a world of intelligent components. This world of autonomic components should form the basics of a new generation of intelligent independent E-Commerce systems.

2. AUTONOMIC JOURNEY
IBM, in its seminal paper [2], introduced the vision, characteristics, and design principles of a new era of Autonomic computing, with specific emphasis in the networking and server domain. A pre-runner to the IBM initiative were multiple developments in the areas of robotics and some control systems.

Most of the initial research drew control systems theory to form the theoretical background for defining the system behavior and its convergence[3]. A recent trend is to have Differential Evolution[4] as the algorithmic base for autonomic implementations focusing on self-optimization.

As an output of intensive research, multiple models, architectures and algorithms were proposed[5]. Concepts of policy framework and subsequent refinement of these frameworks were attempted using Ontologies[6] and advanced Organic Computing[7].

A very early stage adopter of autonomic principle was robotics. Networking and communication fields are the two further-most prominent areas, where real application of autonomic principles happened. Such adaptive behavior was also designed into solutions such as switches, routers, and gateways with the help of autonomic principles.

A few of the famous implementations are Unity, AUTONOMIA[8], FOCALE[9], SASSY[10], PAWS[11], ANTS, Rudder[12], CHASE[13] etc. IPAutomota[14] is a well-known commercial product from IPCenter, which claims a 10-times efficiency increase in terms of human resource utilization. Autonomic components [15] have also been deployed in E-Commerce environment. This is mainly on the server side for load balancing, server resource optimization and so on, and not on the application side to optimize user interactions.

3. AUTONOMIC APPLICATION CHALLENGES
Although quite a lot of progress has been made theoretically in the research above, the practical deployment of these findings are not very remarkable. The progress made by other computing models like J2EE, Agile Programming, .Net etc. are far more noticeable than what has been achieved in the

field of autonomic computing. Researchers world-wide are unanimously convinced about the far reaching potential of this approach, and even advocate that autonomic computing should become that ubiquitous, that it is not a mere after implementation, but a design aspect during the conception of a system. An in-depth analysis of the possible reasons could help us in working towards a brighter future for autonomic computing.

Industry Support: IBM had identified difficulties in managing large network systems, as early as 2002, and advocated the deployment of autonomic systems to resolve this. The industry failed to catch up on this initiative and to take it forward as a daily discipline. Therefore, autonomous computing got a niche technology status, identified more in line with sophisticated technology such as artificial intelligence, and therefore not becoming a common technology.

Clarity of definition: Autonomous systems were defined with four self-managing characteristics (configuration, healing, optimization, protection) initially. Researchers kept on adding new characteristics, which lead to a dilution of focus on implementations. This ambiguity unclarity of definition resulted in diluted focus.

Generic Architecture: Autonomic system definitions evolved from multiple perspectives such as control systems, agent based systems, component based systems etc, resulting in multiple architecture definitions. These approaches hindered, to a large extent, the evolution of a generic architecture.

Positioning Conundrum: Complexities involved in IT systems in a daily manner can be solved at three different levels of sophistication:

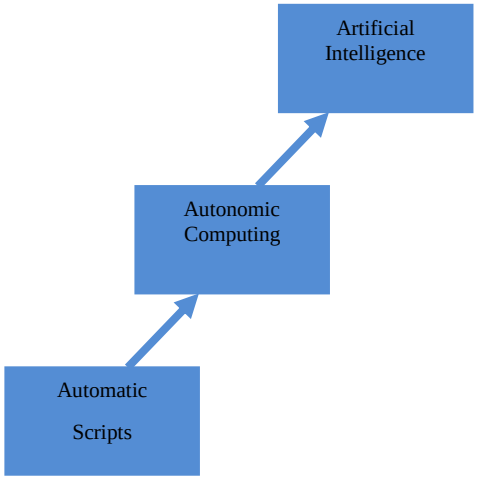

Figure 1: Autonomic Computing - Positioning

Scripts are used to carry out daily tasks, whereas autonomous systems make a regular monitoring and control in regular intervals possible. Artificial intelligence incorporates multiple levels of complex intelligence for managing regular tasks. Implementers tend to either choose the easier way scripts, or immerse in the myriads of artificial intelligence. The fact that autonomic system can produce a near to effect like artificial intelligence without getting into complex implementation is not ommon knowledge, and therefore gets ignored.

Standards, Toolkits: The success of any new paradigm is very much dependent on its adoption by industry leaders in their products. To enable this, standards have to be defined, and toolkits need to be provided. In case of SNMP (simple network management protocol), standards for MIBs (Management Information Base) came into place. Similarly, SOAP (Simple Object Access Protocol) defined standards for description and definition of the involved services. Autonomic computing failed to follow these trends.

In short, the vision of autonomic computing can be achieved only through a huge awareness campaign and the provision of simple architecture and standards.

4. GENERIC ARCHITECTURE – AUTONOMIC COMPONENT

The generic architecture of an autonomic agent constitutes a simple cycle of Plan, Execute, Monitor, and Analyze:

Figure 2: Generic Architecture -Autonomic Component

This simple cycle is quite powerful in exhibiting strong autonomic self-optimizing characteristics, with the basis of differential evolution detailed in the next section. The individual modules have clearly defined tasks. The 'Plan' module plans the variable values based on the method administered by the policy like default values, random values, or analysis. The 'Execute' module administers the behavior determined by the values decided by the plan module. The 'Monitor' module keeps a tab on the results of the behavior administered by the execute module. The 'Analyze' module helps to analyze the monitored data using various analytical methods and derive intelligent results, upon which the plan module can further act upon.

The complete control of the cycle depends on the policies defined in the policy module. The scheduler helps to schedule activities as defined in the policy module.

5. GENERIC ARCHITECTURE – AUTONOMIC SYSTEM FOR E-COMMERCE

The generic architecture for an autonomic system is very similar across various application scenarios. Autonomic characteristics can be injected into a system during its conception or as an add-on to an existing system. This paper mainly handles the latter case for Online Retail systems. Existing online systems, for example, Open source systems such as Magento, Zencart, and OSCommerce, have some basic modules for product categorization and definition, display processing, shopping cart with ordering mechanism, and order processing. They are also equipped with additional

mechanisms for discounting, association display, bundling and so on.

Multiple autonomic components can be introduced into the system like AdManager – for managing marketing communications, DiscountManager – for managing discounts, AssociativeManager – for managing product associations. Each of them run as independent agents, executing the complete autonomic cycle of Planning, Executing, Monitoring and Analysing. At the core, they use a Differential Evolution Algorithm chasing a moving optimum. An Enterprise Service Bus (ESB) provides a middle layer for the components to access the required services. The Reusable Assets Repository (RAR) contains the services used by the components. The ESB and RAR may be eliminated, if a very compact system is envisaged, sacrificing certain flexibilities.

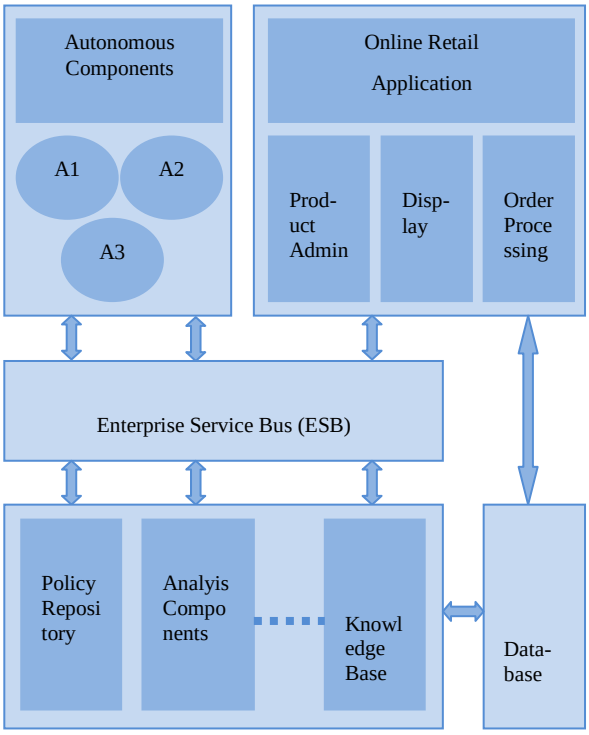

Figure 3: Autonomic components for Online Retail

The customer effects are created by manipulating the values related to the database. The Online Retail database is accessed through a set of Repository components. The monitoring components capture the feedback, which is evaluated by the analysis components.

6. DIFFERENTIAL EVOLUTION

The self-optimization characteristic of autonomous systems is normally a complex and non-differentiable problem. For example, in case of determination of optimal association strength between two products, we encounter a huge number of influencing factors such as nature of products, climatic temperature, running advertisement campaigns, political situations (war, flood, etc) and so on. Modeling of all the factors into a mathematical equation and arriving at an optimal solution is near to non-achievable task. Differential Evolution[4] (DE) is suitable for search of such optimization parameters, which are very complex and non-differentiable. It tries to make no assumptions of the influencing parameters, and focuses on the final utility. The basic approach is to optimize the problem focusing on the utility function, and iterate the influencing variables to maximize the utility function.

The problem complexity gets compounded by the fact that the optimum keeps changing over time. Many of the factors listed above will have a profound influence on the optimal value, for example, the association strength of bread and beer will be stronger in summer, than in winter. So, the autonomous component keeps chasing a moving optimum constantly.

The basic DE algorithm is designed to find an optimum parameter combination to minimize the utility value by applying techniques of mutation, combination and selection in an iterative manner.

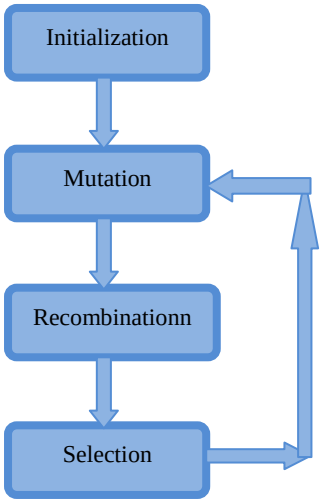

Figure 4: Classical DE Algorithm

In autonomic self-optimization, only an iterative mutation aiming at an optimal result is logical. The moving optimum makes a recombination and selection of not much use, as the optimization goal itself has changed. So, these steps are being avoided in this context. A simulation with practical values for associative manager later, shows that this method is almost as good as an intelligent human selecting the values.

7. E-COMMERCE IMPLEMENTATION

The emergence of applications, such as sale of jewelry, grocery etc. online, is transforming the e-commerce landscape in terms of customer experience. Following modules were identified as autonomic agents:

AdManager: Online shops resort to a lot of online as well as offline advertisements, as the concept of footfall is almost zero (Organic Search is considered to be equivalent to footfall). Compared to minimal rent, maintenance, etc, Ad spend forms the main expense. The AdManager, autonomic manager for optimizing advertisement, distributes the budget provided by the administrator to various advertisement media, monitors the effectiveness based on utility functions like clicks, sale etc., and re-plans the spend to optimize the effectiveness of the whole budget.

AssociativeMgr: Customer experience is improved by suggestions of associated products generated by the system. Data mining algorithms can be used to mine associative data from purchase data of other customers. AssociativeMgr optimizes the associations between the products in a continuous manner.

DiscountMgr: Like any other retail shop, online shops also need to provide discounts based on policies administered. The

discounts need to be changed periodically, and based on sales data to enhance sales. DiscountMgr takes care of these activities.

DemographicMgr: Product displays, discounts etc can be suited to the demographic profile of the customer.

We will examine each of the managers in detail with simulation of results.

8. GENERIC CLASS STRUCTURE

A generic class structure is explained in brief here for the benefit of real implementers to bring in uniformity and easy understanding. The implementation constitutes of six main components – PC (PlanClass), EC (ExecutionClass), MC (MonitorClass), AC (AnalyzeClass), PoC (Policy Class), and SC (Scheduler Class).

PC – Plan components decide the new values of parameters based on the results of analysis. The results of a plan need to be checked for boundary conditions to ensure stability. In case the boundary conditions are violated, the plan module may request for a human intervention. The plan module may also reset the value to maximum, minimum, or default based on the module policy.

EC – Execution is the actual deployment of planned parameters in the real system. This might involve changing values in database, reprogramming components to evaluate the new parameters, etc.

MC – Monitoring is a constant activity and normally data is saved, by the system itself, in selected repositories. MC will provide a snapshot of data from time to time to the AC.

AC – Analysis class encompasses algorithms for analysis of the data and extraction of new information. This could involve various domain specific algorithms or generic algorithms from the field of data mining, etc.

PoC – The policy information forms the main knowledge base for the complete module. Each manager is equipped with a set of policies, which determine the actual behavior. The policy module is responsible for the definition of the policies, setting the parameter values for each policy and also fine tuning the policy.

SC – A practical scheduling of all the above activities are required for the smooth running of the system. A system for online retail may run the analysis, and planning during midnight, and execution and monitoring may happen throughout the day.

9. ASSOCIATIVE MANAGER

AssociativeMgr is well suited for a detailed description of the implementation details of the components. Product associations are a common feature in today's online retail web-sites. Product pairs, which are displayed under the category "*people who bought this, also bought…*" are such products. Data mining algorithms help to mine huge amount of sales data and extract product combinations.

EC: Execution component takes care of setting the associative values in the database and reprogramming the interface modules to display the products in descending order according to the strength of association.

MC: Design of monitoring component depends on the activity to be measured such as navigation behavior, buying patterns, etc. In case of navigation behavior, the sequence of navigation from the customer needs to be recorded. E-commerce systems mandatorily store purchase data and therefore, buying pattern can be directly examined.

AC: Analysis is done by standard algorithms data mining algorithms such as a priori.

PC: Planning module consists of an initial planning or a default value, and subsequent values. The products enter into an associative mode when the support value exceeds a certain threshold, which is specified by a policy parameter. The initial value of strength of association is assigned in 2 ways: values coming out of a priori and random associations. Random associations are assigned to identify new associations.

Planning component operates on a theoretical basis of Differential Evolution as described earlier. An algorithmic representation of the core logic is as follows:

```
Calculate associations using a priori;
For all (associations > MinimumAssociativeValue) {
        If (new association > old association) {
                new association = old association +
                IncrementValue;
        } else {
                new association = old association -
                DecrementValue;
        }
}
```

The algorithm uses a fixed increment/decrement to ensure a smooth transition avoiding spikes. MinimumAssociative Value, IncrementValue, DecrementValue, etc are policy parameters.

PoC – This component stores mainly the policy parameters for the individual managers. No attempt is made for a perfect separation of the policy and the individual managers. The individual managers implement the policy and the policy component provides the policy parameters, which gives specific shape to the policy. To illustrate this, some main policy parameters of Associative Manager are listed:

DMAlgo: {a priori} – Select the algorithms used.

Following parameters depend on the algorithm used:

- **MinimumAssociativeValue:** Specifies the minimum support at which two products will be considered to have an association.
- **IncrementValue:** The delta to be incremented in case the support moves in a positive direction.
- **DecrementValue:** The delta to be incremented in case the support moves in a negative direction.

10. AssociativeMgr-SIMULATION

In a real world scenario, the strength of association between two products is dependent on multiple parameters such as price, season, climate, festivals, economy, etc. Adding the transient nature of each of these parameters makes it impossible to have a perfect solution for each day. The differential evolution techniques provide a continuous search towards an optimal/sub-optimal solution.

A simulation was carried out to determine how the solution compares to a completely human determined system. The efficiency of the system was measured in terms of the difference between a set value and an actual value. Actual value is considered to be a better approximation towards the optimal value and therefore the difference is taken to be a better measure of distance from the optimal value.

The value set by human was simulated in multiple modes such as fixed value, equivalent to yesterday's actual value. Also, the new actual values were calculated in random or as random variation from yesterday's value. The results were:

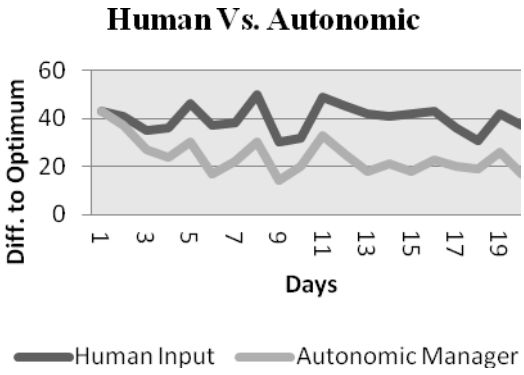

Figure 5: Case 1: Random actual value, Fixed Human Value

Case 1: Human input is a fixed associative value, which is given at the start of the simulation, and it remains unchanged. The daily associative value is assumed to be a random value within a range. The associative algorithm determines the difference between the actual value and the daily associative value, and assigns a new associative value. The difference from human value and new associative value to the actual value is determined and plotted for few days. The darker line depicts the performance of the autonomous system, and it is very evident that this value is closer to the optimum.

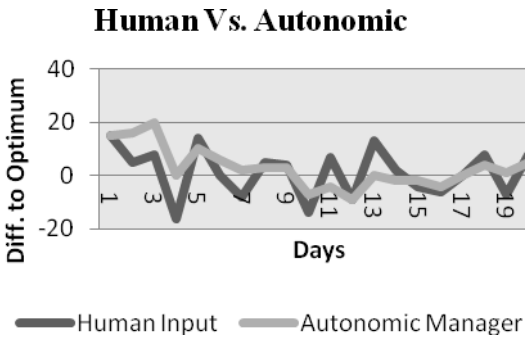

Figure 6: Case 2: Random actual value, Optimized Human Value

Case 2: Human is assumed to change the associative value applying his mind. The logic assumed in the simulation is that the new value will be equal to or nearer to the value from the day before. The simulation shows that on some days, human is more accurate. Overall, the performance is very similar to the human with intelligence.

Case 3: The actual values are not just random values, but follow a smoother curve, which is closer to a real life situation. In this case also the performance of the autonomous system is very similar to the human mind.

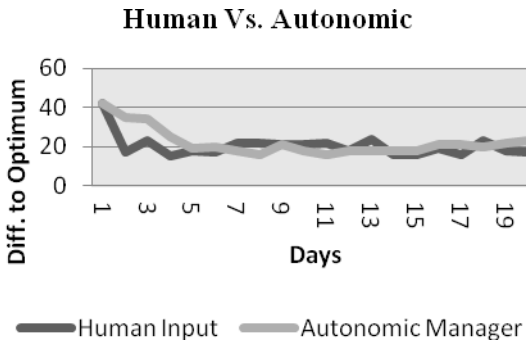

Figure 7: Case 3: Smoothened Random actual value, Optimised human value

11. AdMgr

Effective control and optimization of the advertisement budget remains the biggest challenge for a Marketing Manager. The AdManager implements this function using autonomous techniques to achieve an optimal distribution of the individual budgets among various media.

EC: Execution component takes care of setting the budget values in the respective media. Many popular online advertisement web-sites like Google, Bing provide online interfaces to set the daily budget. In such cases, the budget provisioning can happen in a fully autonomic manner. Print media, Outdoor advertisement, etc. mostly do not have such automatic provisions. In this case, the system outputs values for the budget every day and the manual operator needs to issue further instructions for the changes and confirm to the system that the change has been effectuated.

MC: The effectiveness of the advertisement can be measured in terms of customers visiting the web-site, leads generated by those clients, etc. Google, for example, provides metrics for cost per click, number of clicks, etc. The monitoring component may extract the data from servers such as Google analytics[16], or from deployed open source analytic servers like piwik[17]. Print media data collection should happen by offering the readers some incentives to visit the web-site and fill in some data.

AC: Analysis is done by calculating the utility function value like cost per click for each media.

PC: Planning module consists of an initial planning or a default value, and subsequent values. Initially, each media is assigned a default value, say 10% of the budget. This value is then increased or decreased based on the effectiveness of the utility function.

Planning component operates on a theoretical basis of Differential Evolution as described earlier. An algorithmic representation of the core logic is as follows:

```
Calculate CostPerClik for each Media;
If (new CostPerClick < old CostPerClick) {
        new Budget = old Budget + IncrementValue;
} else {
        new Budget = old Budget - DecrementValue;
}
Adjust Total NewBudget to be equal to AllowedBudget;
```

The algorithm uses a fixed increment/decrement to ensure a smooth transition avoiding spikes. The IncrementValue,

DecrementValue, etc. are policy parameters, which are self-explanatory.

12. DiscountMgr

Efficient management of discounts is another area of concern for an online retail store. Effective control and optimization of the discount remains the biggest challenge of a Site Manager. The DiscountMgr implements this function using autonomous techniques to achieve an optimal determination of the discount percentage.

EC: Execution component takes care of setting the discount values for various products. The discounts are displayed alongside the products during the navigation of the customer through the products.

MC: The Monitoring component collects the daily sales data.

AC: The best way to measure the effectiveness of the discount is to calculate the utility function, Daily Profit Sum.

Daily Profit Sum = $\sum(SalePrice - ProductCost)$

As the discounts increase, the volume of sale increases, which again leads to higher total profits.

PC: Planning module consists of an initial discount value and subsequent values. Initially, each product, that is eligible for discount is assigned a default value, say 10% discount. Further, this value is increased or decreased based on the effectiveness of the utility function, Daily Profit Sum.

Planning component operates on a theoretical basis of Differential Evolution, as described earlier. An algorithmic representation of the core logic is as follows:

```
Calculate DailyProfitSum for Product;
If (new DailyProfitSum  < old DailyProfitSum) {
        new Discount = old Discount - DecrementValue;
} else {
        new Discount = old Discount + IncrementValue;
}
if (Discount>UpperBoundary) then
        Discount = UpperBoundary;
if (Discount<LowerBoundary) then
        Discount = LowerBoundary;
```

13. DemographicMgr

DemographicMgr controls the influence of navigation control based on the demographic of the logged in person like age, geography, gender, etc. Data mining algorithms for clustering like regression analysis are used to identify trends.

EC: Execution component takes care of setting the values for selected demographies. Initial values may be set up in a random style, which also helps to unearth certain clusterings. That young people like pink clothes maybe not a known fact to the system. Only when the random module sets 'pink' for 'young people', people start buying pink clothes, and then the cluster is formed. If the random module is not used, then this trend might go unidentified.

MC: The effectiveness of the demographic placement can be measured in terms of increase in targeted sales of the particular product. It might be difficult to segregate and quantify the increase, and therefore the sales itself can be considered as a measure.

AC: Analysis is done by carrying out a regression analysis and identifying the clusters.

PC: Planning module consists of an initial planning or a default value, and subsequent values. Initially, each media is assigned a random value, indicating that 'young people' like pink. Further, this value is increased or decreased based on the effectiveness of the sales values.

An algorithmic representation of the core logic is as follows:

```
Calculate Sales for Demographic Association;
If (new Sales > old Sales) {
        new Association = old Association +
        IncrementValue;
} else {
        new Association = old Association -
        DecrementValue;
}
```

14. IMPLEMENATION LESSONS

Certain practical aspects of the implementation are listed below for the benefit of implementers:

Design Level: Modularization of a new system to autonomic components during its conception will serve as a powerful tool for simplification of complex systems.

AddOns: Autonomic components can be planned as AddOns for existing system, even for proprietary implementations. In such cases, it might be easiest to manipulate the database values directly, to produce the desired effect. A wrapper for the existing tables, with the required boundary checks, would be a neater implementation.

Limiting values: The optimizing values need to be limited at the upper and lower boundaries to avoid unexpected accidents. For example, a percentage discount value above 50 % might not make sense for a particular product. The differential evolution method might search for the optimum even at 90% discounts, if the utility function is just based on sales.

Trend following: The variation from the current value might be implemented as a percentage of the current value. This is to avoid a high volatility in the values and to implement the behavior of identifying a trend and following the trend. For example, a single day increase in milk consumption, due to a festival, should not drastically influence the regular discount levels available for this product.

Fixed Steps: Fixed increment steps in the direction of the trend would help to keep the volatility further under control.

SOA: A service oriented implementation with SOAP implementation helps in providing a neat implementation base. This also supports deployment in cloud environments in a future scenario.

The above are a few guidelines from our experience in implementation and not a comprehensive list.

15. METRICS – VOLATILITY, STABILITY

Autonomicity of a system, means how close to the ideal system does the current implementation manages itself, cannot be directly measured, as in most cases the definition of the ideal system itself is vague. For example, if we take the case of discount manager, the normal system would be an administrator deciding the discounts like a fixed discount, varying discounts based on last day's sale, varying discount based on predictive sale, etc. As we see, even a human being cannot be similar to the ideal system. It is also impossible to calculate the ideal discount. Therefore, an objective metrics is

impossible for this overruling attribute, and mostly one needs to resort to a subjective measure. Some attributes, which can be measured for autonomic components are:

Level of Autonomicity: LoA - [18] is an attempt to bring in some objectivity into the overall autonomic behavior of a system, by summing up the various self-CHOP attributes levels using a scale based approach.

Volatility: The difference between the actual values and the trend curve is a measure of volatility experienced during the search.

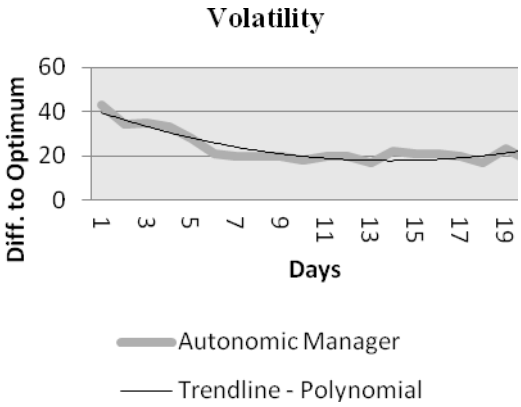

Figure 8: Volatility Graph

The Volatility is calculated as Root Mean Square Deviation:

$$\text{Volatility} = \sqrt{\sum[[TrendValue - ActualValue]^2]}$$

Volatility gives an indication of how close to the optimal value does the algorithm calculate the actual value. The measure only helps to compare two algorithms for the same environment, and not two entirely different algorithms, as the scales vary. The lesser the volatility value, the less volatile the system is and therefore better performance.

Stability: The optimum searches in the above examples are controlled by certain boundary parameters. For example, we can easily say that percentage values should never go beyond 100 %. The number of times the calculated value goes beyond 100 % is a measure of stability of the system. Stability is measured as a percentage of the cases which crosses the boundary to the total number of cases.

$$\text{Stability} = \frac{\text{Total Number of Cases} - \text{Number of Boundary cases}}{\text{Total Number of cases}}$$

A higher value indicates higher stability.

16. FUTURE WORK

This research work has outlined the framework for autonomous applications in an online retail environment. Practical results from deployment of these modules in very large environments like Amazon, eBay, etc. should give more insights about scalability, stability, etc. The framework, as well as the sample implementation will be published as open source.

The development of similar components for other application areas like ERP, CRM, etc. would bring out more intelligent applications. Establishment of certain standards for each domain will help vendors to interoperate. Development and publishing of toolkits would help many developers to start working in this area.

17. CONCLUSIONS

The article gives a very brief overview of the modern autonomic computing journey after IBM has taken the initiative to introduce this technology to the IT segment. Further, an analysis of the possible reasons for the non-penetration of this technology is made. The simple generic architecture is described and its deployment as various classes is detailed. Also, differential evolution as a mathematical base is being examined.

Specific implementation of autonomic components for an online retail system is being examined. A simulation for human associative manager is carried out with the outcome that the manager is as good as or sometimes even better than the human manager itself. Sample implementation for AdMgr, DemographicMgr, and DiscountMgr is provided. A metrics for measuring autonomicity, volatility, stability of autonomous system has been discussed.

Development of autonomous components for more applications, establishments for domain specific as well as interoperability standards, availability of toolkits, etc. are some requirements to take autonomous computing to a ubiquitous technology.

18. REFERENCES

[1] D. Kurian and P. R. Chelliah, "An Autonomic Computing Architecture for Business Applications," in *IEEExplore Digital Library, WICT 2012*, Trivandrum, 2012.

[2] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *IEEE Computer Society*, pp. 41-50, January 2003.

[3] Y. Diao, L. J. Hellerstein, S. Parekh and Griffith, "A Control Theory Foundation for Self-Managing Computing Systems," *IEEE Journal on Selected Areas in Communication*, vol. 23, pp. 2213--2221, 2003.

[4] R. Storn and K. Price, "Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continous Spaces," *Journal of Global Optimization*, vol. 11, pp. 341-359, 1997.

[5] M. Parashar and S. Hariri, "Autonomous Computing: An Overview," pp. 247-259, 2005.

[6] L. Stojanovic, J. Schneider, A. Maedche and S. Libischer, "The role of ontologies in autonomic computing systems," *IBM Systems Journal*, vol. 43, pp. 598-616, 2004.

[7] H. Schmeck, C. Müller-Schloer, E. Cakar, M. Mnif and U. Richter, "Adaptivity and Self-Organisation in Organic Computing Systems," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 5, no. 10, pp. 1-32, September 2010.

[8] X. Dong, S. Hariri and L. Xue, "AUTONOMIA: An Autonomic Computing Environment," 2003.

[9] J. C. Strassner, N. Agoulmine and E. Lehtihet, "FOCALE – A Novel Autonomic Networking Architecture," 2006.

[10] A. D. Menascé, H. Gomaa, S. Malek and P. J. Sousa, "SASSY: A Framework for Self-Architecting," *IEEE*

Software, pp. 78-85, November 2011.

[11] D. Ardagna, M. Comuzzi and E. Mussi, "PAWS: A Framework for Executing Adaptive Web-Service Processes," 2007.

[12] L. Zhen and M. Parashar, "Rudder: An agent-based infrastructure for Autonomic Composition of Grid Applications," 2005.

[13] M. Rak and A. Cuomo, "CHASE: an Autonomic Service Engine for Cloud Environments," in *20th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2011.

[14] February 2012. [Online]. Available: http://www.ipsoft.com/.

[15] H. S. Venkatarama and C. Kandasamy, "An approach in using differentiated services to Maximise Profit in an autonomic Computing System," *International Journal of Computer Applications*, vol. 5, no. 8, pp. 22-26, 2010.

[16] "Google Analytics," google, 15 December 2012. [Online]. Available: http://www.google.co.in/analytics/. [Accessed 15 December 2012].

[17] "Piwik Analytics," Piwik, [Online]. Available: http://piwik.org/. [Accessed 15 December 2012].

[18] R. A. C. W. A. S. Thaddeus Eze, "A Technique for Measuring the Level of Autonomicity (LoA) of Self-managing Systems," in *ICAS2012*, Netherlands, 2012.