Provided for non-commercial research and education use.
Not for reproduction, distribution or commercial use.

(This is a sample cover image for this issue. The actual cover is not yet available at this time.)

This article appeared in a journal published by Elsevier. The attached copy is furnished to the author for internal non-commercial research and education use, including for instruction at the authors institution and sharing with colleagues.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier's archiving and manuscript policies are encouraged to visit:

http://www.elsevier.com/copyright

Author's personal copy

Simulation Modelling Practice and Theory 19 (2011) 2256–2275

Contents lists available at SciVerse ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

An empirical study of modeling self-management capabilities in autonomic systems using case-based reasoning

Malik Jahan Khan [a,b,*], Mian Muhammad Awais [a], Shafay Shamail [a], Irfan Awan [b,c,d]

[a] Department of Computer Science, Lahore University of Management Sciences (LUMS), Lahore, Pakistan
[b] Namal College, Mianwali, Pakistan
[c] School of Computing, Informatics and Media, University of Bradford, Bradford, UK
[d] Computer Science Department, King Saud University, Saudi Arabia

ARTICLE INFO

Article history:
Received 1 July 2011
Received in revised form 12 August 2011
Accepted 13 August 2011

Keywords:
Autonomic computing
Case-based reasoning
CBR based modeling
Simulating self-management
Simulating problem diagnosis

ABSTRACT

Autonomic systems promise to inject self-managing capabilities in software systems. The major objectives of autonomic computing are to minimize human intervention and to enable a seamless self-adaptive behavior in the software systems. To achieve self-managing behavior, various methods have been exploited in past. Case-based reasoning (CBR) is a problem solving paradigm of artificial intelligence which exploits past experience, stored in the form of problem–solution pairs. We have applied CBR based modeling approach to achieve autonomicity in software systems. The proposed algorithms have been described and CBR implementation on externalization and internalization architectures of autonomic systems using two case studies RUBiS and Autonomic Forest Fire Application (AFFA) have been shown. The study highlights the effect of 10 different similarity measures, the role of adaptation and the effect of changing nearest neighborhood cardinality for a CBR solution cycle in autonomic managers. The results presented in this paper show that the proposed CBR based autonomic model exhibits 90–98% accuracy in diagnosing the problem and planning the solution.

© 2011 Elsevier B.V. All rights reserved.

1. Introduction

With the passage of time, the reliance and dependence on computers has increased tremendously due to automation and computerization of manual processes. This has led to increased process speed and higher accuracies, but has added to the complexity. The software systems have grown in size, functionality and complexity. As a result, it has become difficult to solve an unexpected problem. Solving a problem manually after its diagnosis in large scale systems is not a trivial job. Immediate solution strategies are needed that not just minimize the downtime but also the involvement of the human administrator.

Inspired from the human nervous system, in which most of the activities are performed without explicit permission of human itself, autonomic computing [8,23,28,36,56] is a concept which injects some self-managing capabilities in the large systems. The complexity of the systems increases by adding autonomic capabilities but in return the human intervention for managing such systems can be minimized. The additional layer of complexity thus ensures a lot of problem solving responsibilities to be performed automatically and autonomously.

* Corresponding author at: Department of Computer Science, Lahore University of Management Sciences (LUMS), Lahore, Pakistan.
E-mail addresses: jahan@lums.edu.pk (M.J. Khan), awais@lums.edu.pk (M.M. Awais), sshamail@lums.edu.pk (S. Shamail), i.u.awan@bradford.ac.uk (I. Awan).

1569-190X/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.simpat.2011.08.005

Author's personal copy

Two architectures are available that incorporate autonomic behavior in a computing system: externalization and internalization. In externalization architecture, autonomic manager which is responsible for injecting the self-managing behavior, resides outside the self-managed component. In internalization architecture, autonomic manager is built in within the self-managed component. The properties of autonomic systems are collectively known as self-* properties. These include: self-configuration, self-optimization, self-protection, self-healing, self-awareness, context-awareness, open, and anticipatory [28,36].

Self-configuration capability enables seamless adaptation of a new component or a new execution environment without significant human intervention. Self-configuring systems also need to configure themselves according to user-defined high level goals. User only specifies what is needed and not how it should be achieved. Autonomic manager prepares a configuration plan to seamlessly adapt changes in environment according to user-defined specifications [6,33,71]. A self-optimizing system is supposed to continuously seek for the possibilities of optimization in the computing environment.

Self-optimization capability optimizes various parameters at run time according to environmental conditions [19,50]. It enables monitoring, experimenting and tuning various parameters to make best use of the existing resources [36].

Systems having the self-protection capability are expected to defend against various malicious attacks and risks causing large-scale system-wide problems or localized problems within the components [16,72].

Self-healing property detects various system failures and recovers from failures to enable minimum downtime of the system [4,52,57,58]. A self-healing system is supposed to remain conscious of the failures or abnormal behaviors of services and should diagnose and plan recovery action [36]. Autonomic systems should be self-aware of their state, contextually aware of environmental changes, open for portability and common standards, and anticipate their needs, context and behavior [28,56].

In the literature, various intelligent techniques like rule-based systems [14,33,34], artificial intelligence planning [6], hot swapping [5], PeerPressure [67], process query systems based on Hidden Markov Model (HMM) and state machines [60], active probing [59], and control theoretic approach [22,25], etc. have been used to enable various self-management capabilities in autonomic systems. Conversational case-based reasoning [29] has been applied for diagnosis of system failures and recovery which exploits the knowledge gained from recent successes and failures but has been applied at a very limited level in context of autonomic systems. Rule-based systems required rich well-formalized domain knowledge to construct the rule-base. Planning-based techniques require complete state space before searching the goal state. Hot swapping has been applied for self-configuration and adaptation of various code snippets dynamically. Hence, its application scope is limited. Control theory is model-based approach which uses mathematical models that are developed based on certain assumptions, thus reducing their applicability. Other probabilistic approaches like PeerPressure, HMM, state machines, active probing, etc. also rely on the availability of rich prior knowledge for learning the models. These techniques get benefit from the past experience to some extent but recent experiences are not considered while devising the solution. So, experience base is not continuously updated and utilized. These approaches limit the learning growth with passage of time and having new experiences. They do not learn from the recent failures or successes.

Case-based reasoning (CBR) [1,9,17,49,70] is a lazy learning paradigm which is an attractive option to be used in the domain of autonomic computing because of the re-occurring nature of the autonomic computing problems in particular and computing systems in general. CBR is a reasoning methodology which uses past experience to find solution of a new problem. It is a good candidate technique to be adapted for autonomic systems due to various reasons. The detailed discussion on CBR cycle follows in Section 4.

CBR works well where well-structured knowledge is not available before hand because it offers a flexible case structure. Its proposed solution is not rigid and can be revised using adaptation algorithms or human intervention. Revised solution is then finally adapted as the solution of the current problem. Current problem–solution pair may be retained in the case-base to add new knowledge for future use. Past experience is maintained in a repository called case-base.

Though some work on CBR has been performed in autonomic computing earlier [38,40], the focus of this research work is to evaluate the impact of CBR based modeling approach in autonomic systems for the externalization and internalization architectures. The details follow in Section 5. Their focus was on improving the effectiveness of CBR cycle by presenting a clustered CBR approach that resulted in the reduced computational cost and case-base size. However the focus of the present work is to apply CBR in both of the autonomic architectures exploiting its different similarity measures and adaptation algorithms.

In this paper, we present CBR-based autonomic computing models and self-management algorithms for externalization and internalization options of self-management. We propose to apply CBR model within autonomic manager as an analysis and planning tool. Our proposed approach promises to change and improve the way autonomic managers analyze the monitored problem and plan the remedial action. We have designed and implemented a tool called *CaseBasedAutonome* (*CBA*) which can be tailored to be used for both externalization and internalization options of self-management. We have investigated the efficiency and effectiveness of our proposed approach empirically. The empirical study is based on a multi-dimensional research methodology. These dimensions include similarity measure for retrieval of nearest neighbors, cardinality of set of nearest neighbors, solution algorithm, adaptation algorithm and two different case studies pertaining to externalization and internalization respectively.

We selected Rice University Bidding System (RUBiS) [65] for our experiments based on externalization architecture. RUBiS is used as a benchmark for performance and optimization of distributed systems. It is a Java based three tier application which provides basic functionality of an auction site like eBay. It uses MySQL as the database management system at

Author's personal copy

back-end. Inherently, RUBiS is a non-autonomic system. We implemented emulator of an external autonomic manager based on *CBA* for self-configuration, self-tuning and self-healing capabilities of RUBiS.

The second case study selected for our experiments is Autonomic Forest Fire Application (AFFA) [46,47] based on internalization architecture. This is a simulation of forest fire and has built-in autonomic capabilities of self-optimization using a rule agent which uses rule-based system. The limitations of rule-based system include its dependence on complete prior domain knowledge and its rigidness in terms of adaptability and revision of a proposed solution. We have replaced the rule-agent with *CBA* and studied its behavior for self-protection and self-configuration.

Rest of the paper is organized as: Section 2 gives an overview of autonomic computing, Section 3 presents an overview of existing self-management techniques in context of externalization and internalization and Section 4 gives an overview of CBR. Section 5 discusses the proposed CBR-based approach for externalization and internalization and Section 6 presents the research methodology adopted in this paper. Section 7 presents the empirical results and analysis of the proposed approach on RUBiS case study and Section 8 presents the empirical results and analysis of the proposed approach on AFFA case study. Finally, Section 9 concludes the paper and discusses some future work.

2. Architectures of autonomic systems

As mentioned in the previous section, there are two major architectural options available in the literature for building an autonomic system: externalization and internalization [28,36,56]. In externalization approach, autonomic manager resides outside the managed element and is built as a well-separated implementation layer as shown in Fig. 1. Usually, this approach is adapted when an existing non-autonomic system has to be made autonomic. This architecture is quite useful in applications where autonomic behavior is enabled at some later stage of the application life cycle or internal implementation details of the application are not that much clear. Legacy systems are a good candidate for externalization option. In internalization approach, there is no clear separation between the autonomic manager and the managed element as shown in Fig. 2. This approach makes more sense when we have to implement a system with built-in self-management capabilities. For the internalization architecture, various programming frameworks have been proposed in the literature like Accord [47], Rudder [45], vGrid [41], etc. Accord programming framework enables internalization using three kinds of ports: control port, operational port and functional port. Control port is a shared interface between element manager and the computational component. Operational port contains rules to be used by the rule agent for self-management of computational component. Functional port is used to expose the functionality to other autonomic components.

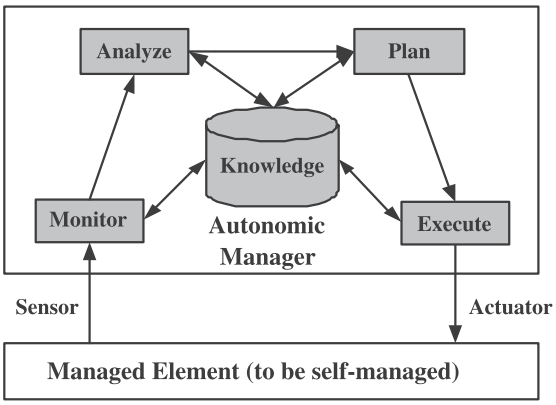

Fig. 1. Externalization architecture of autonomic systems [36].

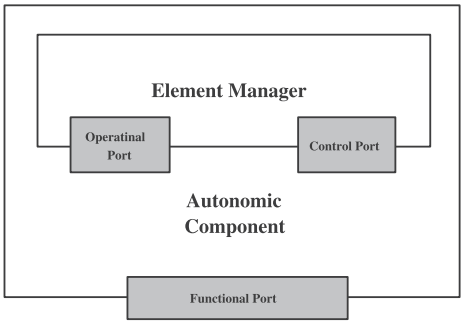

Fig. 2. Internalization architecture of autonomic systems [47].

Author's personal copy

To implement or enable any of the self-* properties, an autonomic computing system has to perform many different steps which collectively constitute autonomic computing cycle as shown in Fig. 1 [28,36,56]. The four main steps are:

2.1. Monitor

The state of the managed resource is periodically checked via a sensor and is tested for stability. If it is found stable then no further action is taken and next sensed state is monitored. If it is found unstable then its data is handed over to the diagnose phase.

2.2. Diagnose

This phase receives data of an unstable state and analyzes it to diagnose the cause of unstability. It may be an internal cause or external cause. This diagnostic step needs to be intelligent enough to correctly diagnose the problem. This step may be carried out using any pre-determined diagnostic algorithm.

2.3. Plan

Diagnosed problem is reported to the plan phase which searches for a suitable solution of the reported problem. Searched solution is prepared in the form of a solution plan.

2.4. Execute

Solution plan is handed over to the execute phase which prepares an execution plan and hands over the execution plan to the actuator of the controller interface.

3. Existing self-management approaches: A review

Various approaches exist in the literature to self-manage a system with respect to externalization and internalization. A brief overview is provided below.

3.1. Self-management approaches for externalization

In order to autonomize an existing application or system, autonomic managers are built as a separate layer and interact with the managed resource through sensors and actuators as discussed above. Various techniques and infrastructures have been proposed in the literature for enabling autonomic behavior in existing systems, an overview is presented below.

A decentralized approach of enabling autonomic computing has been presented in an agent-based architecture known as Unity in [64]. This architecture supports optimum autonomic resource allocation based on computing many local resource-level utility functions and then aggregating them into a global utility function. Similarly, policy-based access control frameworks have also been proposed to devise a global solution after analyzing the local states [44]. In Ref. [58], Pip, an architecture for detecting the unexpected behavior in a distributed system, has been proposed which compares the expected behavior with the current behavior. Expected behavior can be declared by the users using a declarative interface. Jiang et al. [34] propose an alerts based problem determination mechanism in large scale systems. One problem may trigger many alerts and then alerts are ranked using a ranking criteria. Finally, a rule-based approach is used to determine the exact problem. Control theory has been exploited in the literature for monitoring and fine-tuning various attributes in an autonomic environment [2,18,22,35,62]. Controller with the feedback loop acts as an autonomic manager. Service-oriented architectures have been proposed in the literature to be used for enabling autonomic behavior and fault detection in distributed systems [7,11,15,66]. [59] have used a statistical approach for problem determination in distributed systems using active probing. A probe is a an end-to-end test transaction which outputs based on the system's components. Appropriate probes are selected to diagnose the problem and their results are analyzed. PeerPressure is a statistical solution suggested to diagnose the problems resulted from misconfigurations [67]. A lot of snapshots of machine configuration are stored in a database called GeneBank database. PeerPressure uses a statistical analyzer that calculates the probability of each suspected component to be sick using Bayesian estimation. In Ref. [3], an adaptive action selection technique has been proposed for autonomic software systems. It selects an appropriate action from a finite pool of actions using reinforcement learning. In Ref. [13], decision tree learning approach has been presented to diagnose the failure in large distributed systems. At nodes of the tree, various parameters are ranked according to their information gain value. In [4,53,54], CBR has been suggested to diagnose the failure in service delivery system and plan remedial actions. Each problem scenario and corresponding remedial action is represented as a case and a collection of such scenarios constitutes a case-base. Simple CBR cycle is followed to diagnose a new problem and resolve it. In Ref. [29], environment knowledge is represented in the form of a case-base and a rule-base. Conversational CBR is used to detect the failure and diagnose the problem. Rule-base system is used to

Author's personal copy

resolve the detected problem. Conversational CBR keeps the user in continuous loop to ask questions for case generation and most of the time, it needs user's feedback to diagnose the problem.

3.2. Self-management approaches for internalization

Various approaches for built-in mechanisms of autonomic computing have also been proposed in the literature.

In Ref. [51], an autonomic failure detection algorithm has been proposed which suggests a self-regulating mechanism for failure detection by setting bounds on resource usage and failure detection latency. In Ref. [46,47], Accord, a component based framework for building autonomic applications, has been proposed. It extends some capabilities of existing conventional programming frameworks and introduces the formulation of autonomic components, their composition and interaction. Dynamic reconfiguration has been suggested in [33] as a building block for workload managers in IBM pSeries servers which provides self-optimization, self-healing and self-configuration capabilities. To achieve dynamic reconfiguration for distributed systems, a component framework has been presented in Ref. [14] to manage the interactions between components during dynamic reconfiguration. In Ref. [50], an autonomic query optimizer known as LEO has been presented which validates the mathematical model of query execution without human intervention and is capable to correct the wrong optimization estimates. In Ref. [12], a technique for cheap recovery known as Micro-reboot has been presented which starts the recovery process by rebooting components from a granular level, and incrementally grows the reboot scope if recovery is not achieved. Rx is another technique to recover from failures presented in Ref. [57]. It treats bugs as allergies by rolling back the program to previous safe checkpoint and re-executing in a modified environment. Hot swapping [5] is another built-in approach of autonomic computing which enables dynamic adaptation of the execution environment by interpositioning the code or replacing the code. Interpositioning enables insertion of a new component between two existing components and replacement enables one passive component to take over an executing component. In Ref. [37], an artificial intelligence approach has been proposed for autonomic computing policies which is basically built on the state search model of artificial intelligence. In [6], dynamic reconfiguration planning has been suggested. The planner is given the description of the initial state, final state and domain of possible transitions. Then using various search mechanisms like breadth first search, depth first search, best first search, etc., goal state is reached. An agent-based architecture has been proposed in Ref. [10] to build autonomic systems. In this architecture, various functionalities of an autonomic system like controller, sensor, actuator, etc. are delegated to individual agents which collaborate in a multi-agent environment. In [48], a framework for development of autonomic web services has been proposed which enables self-configuring capability in web services and is based on the adoption of a simulation engine. The literature survey reveals that CBR has not been used in internalization architecture of autonomic systems so far. [30] proposed a method to design and develop an autonomic system which explicitly contains the control loop as its crucial part. Clustered CBR based approach has been used for internalization architectures to improve the efficiency of the CBR cycle with the tradeoff of performance in Ref. [39,40].

3.3. Limitations of the existing self-management approaches

Autonomic computing is a nature-inspired phenomenon. In nature, environment is dynamic and humans have to observe diverse scenarios with passage of time. So, there is a demand of continuous learning. As human administrators are replaced by autonomic managers, so their learning curve should not be static. Autonomic systems may observe new experiences as the time passes on. Most of the self-management techniques discussed above enforce learning on the current dataset and do not have the cushion for dynamic learning with passage of time. Therefore, such techniques will not be able to handle unseen scenarios, whereas human administrators learn dynamically and are capable to handle such scenarios. They also learn from their mistakes. So, their learning curve is continuously improved. Another limitation of most of these artificial intelligence and probabilistic approaches is that they need well-formalized and structured knowledge for training purposes which is rarely available in case of autonomic systems.

CBR is an effective candidate technique to be adapted for enabling self-management capabilities in autonomic systems where rich and well-formalized background knowledge is not easily available. CBR has been applied for self-healing and failure diagnosis in the literature as discussed earlier. The self-healing approach suggested in [4,53,54] is based on strong assumptions that limits its applicability. First assumption is that application of the proposed solution of CBR cycle will not cause another problem in the managed element. This is a strict assumption which is not applicable in real world scenarios. Second assumption is that a solution algorithm can take indefinite amount of time. This is also not a practical assumption. Another restriction of this approach is that it is only applicable to self-healing capability and cannot be generalized for self-management of autonomic systems in general. This approach is also strictly applicable in externalization architecture only. In [29], CBR has been applied for failure diagnosis. This approach has two major limitations. First limitation is that it is specific to failure detection like the other proposed approaches in Refs. [4,53,54] and cannot be generalized to detect optimization, protection or configuration problems in autonomic systems. Second limitation is that it uses CBR only to diagnose the problem but does not exploit it for remedial actions. Instead, a static rule-base has been used for failure recovery, which limits the continuous learning of the autonomic manager.

In this paper, we intend to exploit CBR in its full capacity for diagnosing problems and planning solutions of autonomic systems. Before the details of the proposed architecture are discussed, a brief overview of CBR is presented in the next section.

Author's personal copy

4. Case-based reasoning

CBR is a problem solving methodology which utilizes the past experience of similar kind of problems to solve the current problem in an elegant fashion. CBR is a lazy learning paradigm which is very effective in scenarios where well-formalized knowledge is not available and classical learning models do not work efficiently. This technique reduces the knowledge acquisition bottleneck and enables learning in a problem domain where rich and complete background knowledge is not easily available [54]. Past experience is maintained in a repository in the form of problem–solution pairs. Each pair is known as case and the whole repository is referred to as case-base.

CBR has been applied in the literature in a wide variety of application domains. An interesting application of textual CBR is intelligent FAQ system [9]. Against user query, more closely relevant documents are retrieved and by applying various layers of knowledge extraction, answer is found. Nature and depth of knowledge layers vary from domain to domain. CBR has been applied in product selection tasks in E-commerce in Ref. [69]. Help desk applications are a popular application area of conversational CBR [68]. It has been realized in [9,61] that medical diagnosis is a reasoning process which extensively uses experience. CBR has been applied in this area as well for diagnostic purposes. Planning and design is another application of CBR [9]. CBR has been effectively used in various sub-domains of software engineering as well like software quality prediction [42] and software reuse [20,21,24,63]. As discussed in Section 3.1, CBR has also been exploited in self-healing and failure diagnosis of software systems at a limited scale [4,29,53,54].

CBR systems work in five major phases which are also collectively known as 5R's of CBR as shown in Fig. 3 [17], and are explained below.

4.1. Retrieve

Current case is compared with the existing cases in case base and the cases that match the most are retrieved. To accomplish this task, various similarity metrics are used. Some of the commonly used similarity metrics for this purpose include City block distance [27], Euclidean distance [27], Mahalanobis distance [42], geometric similarity metrics [26], probabilistic similarity measures [55] and similarity measures based on fuzzy inner and outer products [38]. This gives us a set of nearest neighbors of the current case. Selection of similarity measure depends upon problem domain, level of accuracy needed, time complexity, etc. Inverse distance functions can also be used as measure of similarity between two cases.

4.2. Reuse

Solution of the nearest neighbors is used to devise solution of the current case. There exist various solution algorithms which can be applied on solutions of the nearest neighbors to find out the solution of case at hand. These algorithms include arithmetic average, weighted average [27], case-based inferencing using fuzzy rules [32], probabilistic methods [31], etc. Selection of a solution algorithm depends on the priorities of neighbors, problem domain, nature of data and level of target accuracy.

4.3. Revise

For better adaptation, we may need to revise the solution of the current problem. However to make CBR more sensible, adaptation phase should be kept minimal. Substantial case adaptation or revision may harm the knowledge engineering advantages to be obtained from CBR [17].

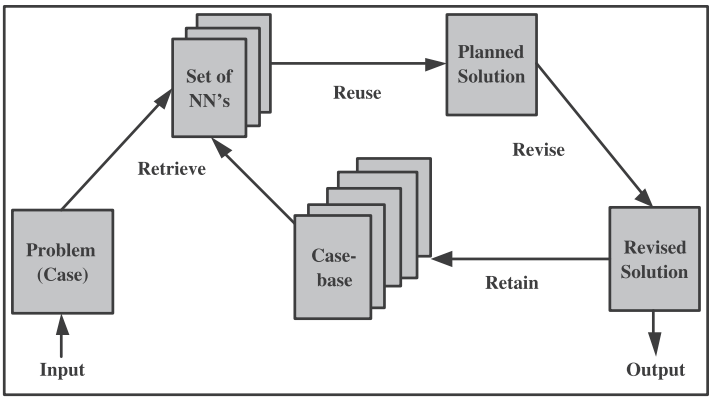

Fig. 3. CBR cycle [1].

Author's personal copy

4.4. Retain

Once we have completed the job of devising solution of the current case, we may retain this new problem–solution pair in the case-base. Decision about retention of the current case depends on the magnitude of newly acquired knowledge from the current case. This is the beauty of the CBR approach that a recently solved problem is immediately available in the case-base for future use [17,70].

4.5. Repair

Case-base growth is a continuous process due to the retain phase of CBR cycle. As the case-base size grows, it causes computational bottlenecks. So, case-base needs to be repaired periodically so that efficiency of CBR cycle remains unaffected. The repair phase is not a regular phase of the CBR cycle. It can be executed using different schemes given in literature [40].

Using CBR for externalization and internalization options in autonomic systems would yield benefits such as:

1. Flexibility in case representation due to non-structural approach as opposed to AI planning and control theory.
2. Simple and natural self-managing behavior due to nature inspired solution strategy.
3. Comparatively lesser input pre-processing overhead due to the flexible problem representation format.
4. The capability of exploiting past and the most recent experiences to suggest a solution of the current problem.
5. Flexible revision mechanism for update of a suggested solution.
6. Minimum off-line learning requirement as compared to most of the probabilistic and statistical learning techniques.

5. Self-management in autonomic systems: A CBR based modeling approach

This section presents CBR-based autonomic models for externalization and internalization, with their scope, proposed CBR-based algorithms and CBR-based autonomic solution finding framework *CBA*.

5.1. CBR-based model for externalization

For the externalization architecture, a CBR-based autonomic manager is proposed as shown in Fig. 4. In this approach, *Monitor* phase of autonomic manager detects the new problem using a dedicated sensor *S* which periodically executes a diagnostic test on the managed system *R* after every *T* time units. Current state $state_c$ of *R* is captured through *S*, is compared to the known target state $state_t$ and deviation *dev* of $state_c$ from $state_t$ is computed. If *dev* exceeds the deviation tolerance limit ε, then $state_c$ is passed to *Analyze* phase of the autonomic manager which transforms it into a case C_p. C_p is a vector of significant parameters representing the current problem. *Analyze* and *Plan* phases of autonomic manager utilize the CBR-based autonomic solution finder *CaseBasedAutonome* (*CBA*) which retrieves the nearest neighbors of C_p from the case-base. Detailed discussion of *CBA* follows in Section 5.3. The case-base resides inside the knowledge repository of autonomic

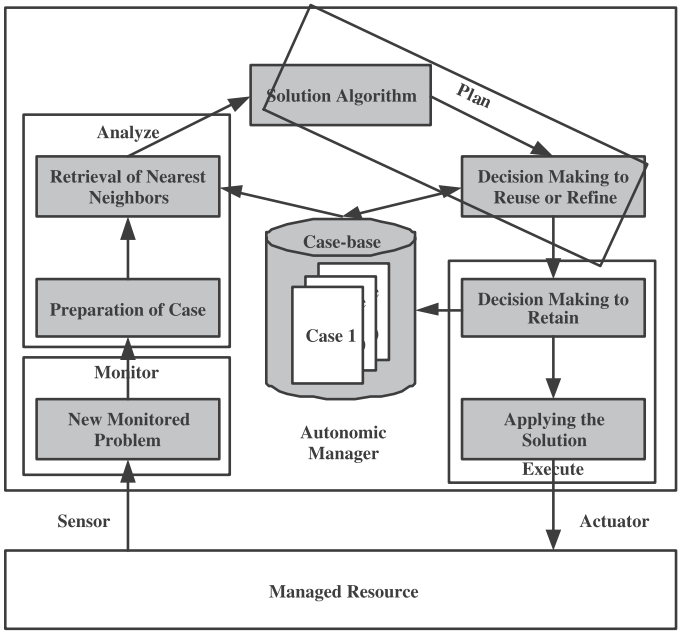

Fig. 4. Proposed architecture for externalization using CBR.

Author's personal copy

Input: Monitored resource R, sensor S, executor E, time delay T, tolerance limit ε
Output: Autonomic behavior in R
Method:

1. Repeat step 2 to 5 after T time units
2. $state_c$ = readInterface(S, R)
3. dev = compareTo($state_c$, $state_t$)
4. If($dev > \varepsilon$)
    a. c_p = prepareCase($state_c$)
    b. sol_p = CBA(C_p)
    c. executeSolver(sol_p, E, R)
    d. $state_n$ = readInterface(S, R)
    e. If(needUndo($state_n$))
        i. rollBack($state_c$)
    f. End If
5. End If
6. End Repeat

Fig. 5. Algorithm 1 – self-management algorithm using CBR.

manager. The solutions of the nearest neighbors are aggregated using a solution algorithm *SA* in the *Plan* phase of autonomic manager to compute the solution sol_p. This solution is executed by the *Execute* phase of the autonomic manager onto *R* using the actuator *E*. Cushion for the rollback option is also incorporated in case of any worse undesired new state. In such case,

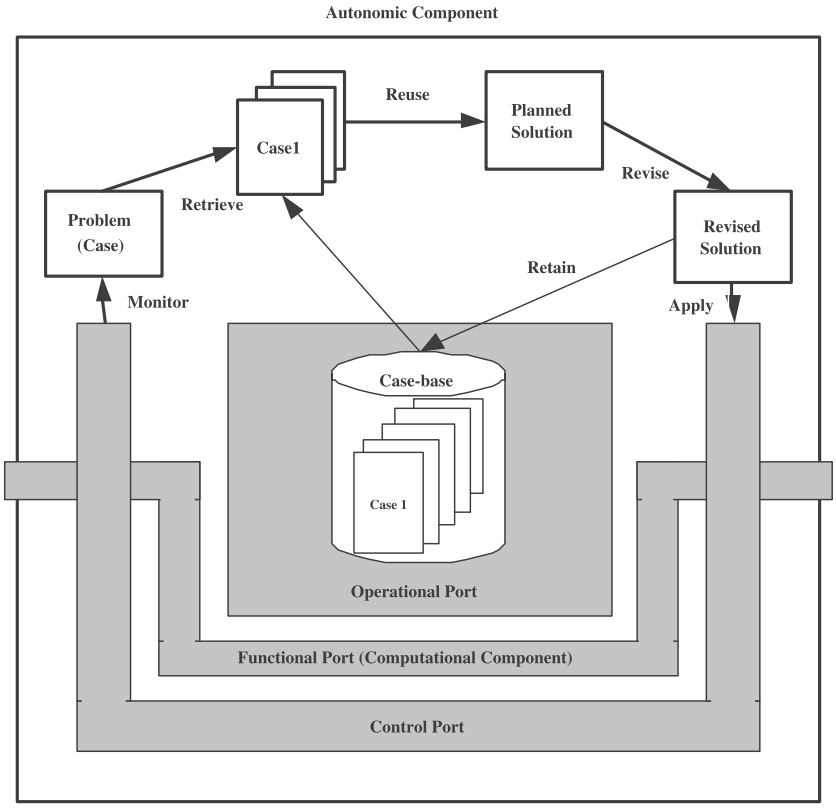

Fig. 6. Proposed architecture for internalization using CBR.

Author's personal copy

undo option may be availed. To implement the CBR-base externalization architecture, we propose Algorithm 1 outlined in Fig. 5.

5.2. CBR-based model for internalization

In internalization architecture, autonomic manager is implemented as a built-in component of managed element. We propose a CBR-based extension of Accord [47] programming framework for autonomic applications. Accord uses a rule-agent as part of the autonomic manager for analysis and planning purposes which limits the dynamic learning capabilities of the autonomic manager. In this architecture, autonomic manager and managed element are not well-separated layers in implementation as such applications inherently exhibit autonomic behavior. We propose to integrate our CBR-based autonomic solution finder *CaseBasedAutonome* (*CBA*) with control and operational ports of this programming framework using the strategy shown in Fig. 6.

In our proposed approach, control and operational ports of the Accord [47] programming framework play vital role. The control port is responsible for detecting the problem and presenting it to the CBR cycle as a case. The operational port contains current knowledge in the form of a case-base. The Analyze and Plan functionalities of autonomic manager are performed by the *CBA* using the similar steps as in Algorithm 1 but implementation process, architecture and scope of both approaches are different. As the *CBA* is part of the autonomic component and not well separated from the computational component, we do not need dedicated interfaces to act as sensors or actuators. Functional port is responsible to expose the core functionality of the autonomic component to other components and has no direct concern within the proposed architecture. In internalization architecture, operational and control ports play their role in computational component as well as autonomic manager, whereas computational component and autonomic manager in the externalization architecture are two separate layers and can only interact through dedicated sensors and actuators.

5.3. CaseBasedAutonome (CBA): CBR-based autonomic solution finder

CaseBasedAutonome (*CBA*) is a CBR-based tool and used as a basic building block in Algorithm 1. This algorithm is the crux of the CBR based self-management approach and is implemented within the autonomic manager for the *Diagnose* and *Plan* phases. The idea is to maintain a case-base *CB* containing *n* cases. Each case represents a configuration, failure, protection or optimization problem of the managed element. A case is a vector of various parameters which are extracted from the captured state of the monitored resource. Solution of each case is maintained in the form of a code snippet which is an effective way to dynamically solve problem in autonomic computing [36].

Before executing *CBA*, we have to select an appropriate similarity measure *SM*, a solution algorithm *SA* and cardinality N_n of the set of nearest neighbors *NN* which contribute to find the final solution. Current case C_p is compared with every case C_j in *CB* using *SM* and similarity sim_j is computed which may be treated as weight w_j of C_j to find solution sol_p of C_p. We may also keep same weight for all compared cases. Such decisions may vary from domain to domain. Set of nearest neighbors *NN* is extracted based on the computed similarities and *NN* is aggregated using *SA* and *w* to find sol_p. An automatic adaptation

Input: Given a case-base CB containing n cases, case representing new problem C_p, similarity measure SM, solution algorithm SA, cardinality of nearest neighborhood $|NN|=N_n$

Output: Planned solution sol_p of C_p

Method:

1. For each $C_j \in CB$
   a. sim_j = findSimilarity(C_p, C_j, SM)
   b. $w_j = sim_j$
2. End For
3. NN = extractNeighbors(N_n)
4. sol_p = aggregate(NN, SA, w)
5. If (needRevision())
   a. sol_p = revise(sol_p)
6. End If
7. If (needRetention())
   a. $CB = CB \cup \{<C_p, sol_p>\}$
   b. Increment n
8. End If

Fig. 7. Algorithm 2 – *CaseBasedAutonome* (*CBA*).

Author's personal copy

mechanism can be used to revise the solution, if needed. It is needed when C_p was an entirely new experience and its closest matches did not exist in *CB*. Again, an automatic mechanism may decide whether to retain the vector $\langle C_p, sol_p \rangle$ as new experience in the *CB*. This decision depends on the magnitude of new knowledge contributed by this new experience. Implementation level details are given in Algorithm 2 in Fig. 7.

6. Research methodology

In order to effectively analyze the performance of proposed approach, a multi-dimensional research methodology has been adopted. The performance of proposed algorithms is tested and empirically validated along each dimension. We used accuracy as root mean squared error (RMSE) as the evaluation criteria in our empirical study for analyzing various dimensions. In this section, we discuss these dimensions.

6.1. Nature of the case-base

We constructed two case-bases representing two different case studies, RUBiS and AFFA. Each case in RUBiS case-base is represented in the form of nominal and binary attributes whereas each case in AFFA case-base is represented in the form of numeric attributes scaled between 0 and 1. First case-base has been generated using an emulator for RUBiS whereas second case-base has been synthetically generated using various rules extracted from the description of AFFA ports in [47,46]. Based on the case-base nature, we used majority voting as the solution algorithm for RUBiS case-base and weighted average as the solution algorithm for AFFA case-base. We used simple hold-out validation technique for RUBiS case-base and leave-one-out (LOO) cross validation technique for AFFA case-base.

6.2. Similarity functions

Ten commonly used similarity functions for retrieval of the set of nearest neighbors of the current case have been used in this paper. The analysis presents a comparison of these similarity measures on the basis of accuracy and RMSE against varying number of nearest neighbors. The selected similarity functions used in this research are Hamming distance, Manhattan distance, Euclidean distance, Sim_1 [38], Sim_2 [38], Canberra distance, Bray–Curtis distance, Squared Chord distance, Squared Chi-Squared distance and Jaccard similarity function. These similarity functions are given in Table 1.

6.3. Cardinality of nearest neighborhood

Set of nearest neighbors (*NN*) is used as the input to solution algorithm to devise the solution. We vary cardinality of *NN* from 1 to size of the case-base. The purpose of this analysis is to find a similarity measure that requires least cardinality of *NN* to give the best results.

Table 1
Similarity functions used.

Similarity measure name	Similarity measure definition
Hamming similarity	$sim_{ij} = \frac{matches_{k=1}^{m}(P_{ik},P_{jk})}{m}$
Manhattan distance	$d_{ij} = \sum_{k=1}^{m} w_k \lvert P_{ik} - P_{jk} \rvert$
Euclidean distance	$d_{ij} = \sqrt{\sum_{k=1}^{m}(w_k(P_{ik} - P_{jk}))^2}$
Sim_1	$sim_{ij} = 1 - max(IP_{ij},OP_{ij})$ $IP_{ij} = max(min(P_{ik},P_{jk}))$ $OP_{ij} = min(max(P_{ik},P_{jk}))$
Sim_2	$sim_{ij} = 1 - t_1 * t_2$ $t_1 = min(IP_{ij},1 - OP_{ij})$ $t_2 = \frac{IP_{ij}+(1-OP_{ij})}{2}$
Canberra distance	$d_{ij} = \sum_{k=1}^{m} \frac{\lvert P_{ik}-P_{jk} \rvert}{P_{ik}+P_{jk}}$
Bray–Curtis distance	$d_{ij} = \frac{\sum_{k=1}^{m} \lvert P_{ik}-P_{jk} \rvert}{\sum_{k=1}^{m} P_{ik}+P_{jk}}$
Squared Chord distance	$d_{ij} = \sum_{k=1}^{m} \left(\sqrt{P_{ik}} - \sqrt{P_{jk}}\right)^2$
Squared Chi-Squared distance	$d_{ij} = \sum_{k=1}^{m} \frac{(\lvert P_{ik}-P_{jk} \rvert)^2}{P_{ik}+P_{jk}}$
Jaccard similarity coefficient	$sim_{ij} = \frac{C_i \cap C_j}{C_i \cup C_j}$

Author's personal copy

6.4. Selection of model

We implemented and evaluated our proposed approach using various parameters mentioned above and compared their accuracy and *RMSE*. The experimental setup giving maximum accuracy and minimum *RMSE* using the smallest possible size of nearest neighborhood is recommended to be selected as the effective model for continuous use.

7. Case Study 1: RUBiS

We selected Rice University Bidding System (RUBiS) [65] as the case study for our CBR-based externalization architecture of autonomic computing. RUBiS is an auction site prototype equipped with all core functionalities of an auction site. It is an open-source initiative, used as a benchmark in research for evaluation of various scalability and performance concerns of application servers. The core functionalities of this bidding prototype include registration, selling, bidding and browsing. There are three versions of RUBiS: PHP based, Java Servlets based and EJB based. We used the Java Servlets version as our testbed. Servlets are used at the presentation tier and generate the HTML response after retrieving data information from beans. MySQL is used as the database management system at the data tier. We implemented an emulator for RUBiS, a service monitor and a problem resolution manager. Emulator emulates user actions which characterize the system state. Service monitor monitors the current state of the system periodically. Problem Resolution Manager executes the planned remedial action. We generated its case-base by executing this emulator. This case-base represents various configuration and healing problems in the deployed RUBiS testbed. If a problem is reported then its solution is suggested by *CBA* as depicted in Fig. 4. Finally, planned solution is executed on the managed resource (RUBiS) through Problem Resolution Manager. The implementation details of *CBA* given in Algorithm 2 with RUBiS are discussed below and shown in Fig. 8.

7.1. Problem injector

The problem injector has a pool of possible problems of diverse natures which can occur in the RUBiS testbed. It randomly picks a problem from the pool after random interval and invokes a script which causes that problem to occur in the managed

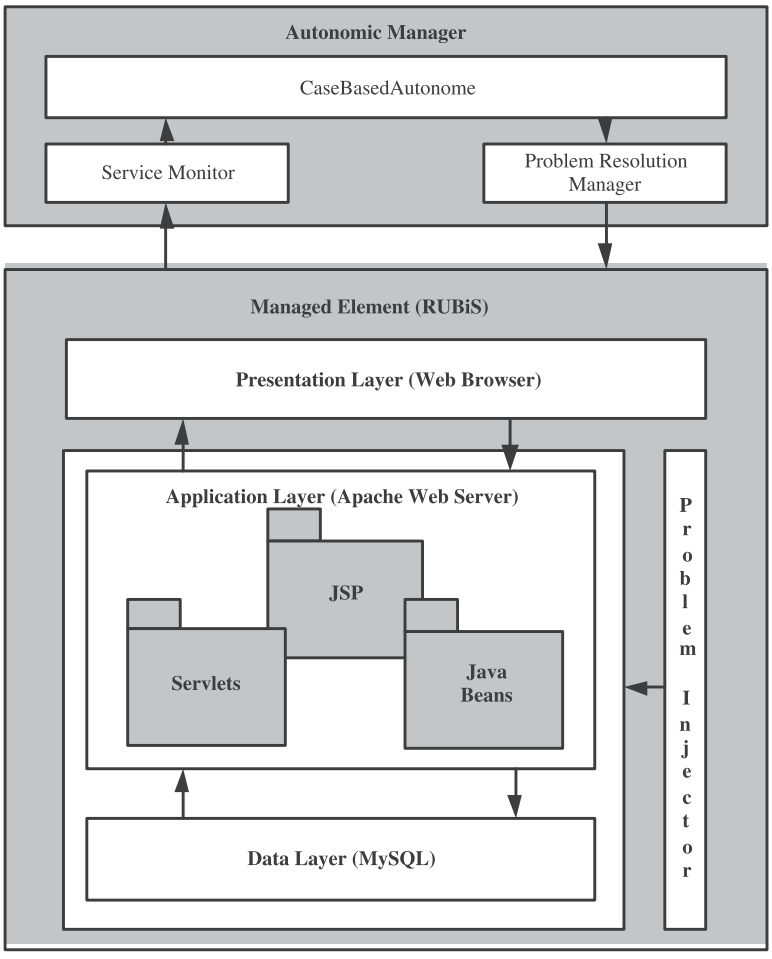

Fig. 8. RUBiS architecture with CBA.

Author's personal copy

resource. The emulator injects configuration problems and service failures in Apache web server and MySQL. These problems include corrupting the configuration file of the web server, shutting down the servers randomly, dynamically changing user permissions over resources like database table, etc.

7.2. Autonomic manager

Autonomic manager interacts with RUBiS through dedicated sensor and actuator. The autonomic manager contains following three modules:

7.2.1. Service monitor

Service Monitor periodically captures state of the RUBiS testbed through sensor by executing various health tests. Each health test returns a boolean value. 1 indicates that the tested resource is healthy and 0 indicates that the tested resource is abnormal. Captured state is compared with the healthy state and if some deviation is observed then it is handed over to the *CBA*.

7.2.2. CBA implementation for RUBiS

This section discusses the implementation details of *CBA* module for RUBiS:

i. *Case representation*: In general, a case C_i can be represented as a set of n parameters:

$$C_i = \{P_{i1}, P_{i2}, P_{i3}, \ldots, P_{in}\} \tag{1}$$

The complete case-base B is represented as a set of m cases:

$$B = \{C_1, C_2, C_3, \ldots, C_m\} \tag{2}$$

In the testbed deployment, seven parameters were used to describe a RUBiS problem along with one solution parameter. These variables were extracted from the documentation of [65]. These variables include Service Name to be monitored (*SN*), Permissions on the User table of the database (*PU*), Permissions on the Item table of the database (*PI*), Permissions on the Bid table of the database (*PB*), Apache Server Configuration (*SC*), Apache Server Health (*AH*) and MySQL Server Health (*MH*). The solution parameter is Configuration Script (*CS*) that represents name of a code snippet to be executed by the Problem Resolution Manager. These parameters are the control parameters in the autonomic context. *PU*, *PI*, *PB* and *SC* are the control parameters for self-configuration capability in the RUBiS testbed. *AH* and *MG* are the control parameters for self-healing capability in the RUBiS testbed.
Each case of RUBiS is represented as a set of the above mentioned parameters:

$$C_i = \{SN, PU, PI, PB, SC, AH, MH, CS\} \tag{3}$$

ii. *Case retrieval*: The holdout validation technique [43] was applied for the empirical evaluation of our proposed approach on RUBiS. A bootstrap case-base was generated using the experience of human experts and was used as the training case-base. A sample bootstrap case-base has been shown in Table 2. The problem injector on RUBiS was executed to generate the test cases. Appropriate similarity measures were selected from Table 1 for the retrieval of varying number of nearest neighbors from the training case-base. As *SN* is a nominal attribute (*NA*), we used a heterogeneous version of all the similarity measures by assuming equal weights of all attributes, as given in Eq. (4).

$$f(C_p, C_q) = \frac{f'(NA_p, NA_q) + (\gamma - 1)(f''(D_p, D_q))}{\gamma} \tag{4}$$

where

$$D_p = C_p - \{NA_p\} \tag{5}$$
$$D_q = C_q - \{NA_q\} \tag{6}$$
$$f'(NA_p, NA_q) = 1, \quad \text{if } NA_p = NA_q \tag{7}$$
$$f'(NA_p, NA_q) = 0, \quad \text{if } NA_p \neq NA_q \tag{8}$$

Here f is a similarity function, $NA = SN$, $\gamma = 7$ and $sim(C_p - \{NA_p\}, C_q - \{NA_q\})$ is the similarity between two cases using one of the selected similarity measures.

iii. *Case reuse*: The solution parameter in this case study is a nominal variable. Therefore, majority voting algorithm given in Fig. 9, was used as the solution aggregation algorithm during the reuse phase of CBR cycle. The solution is represented as a configuration script (*CS*). Each *CS* is a code snippet to resolve the reported problem. For example, *CS*1 is a code snippet to restart the Apache webserver and is executed when *AH* is reported as 0 by the sensor.

iv. *Case revision*: In this case study, a simple revision algorithm shown in Fig. 10 was implemented which yielded inspiring results. The solutions of the nearest neighbors were applied iteratively until a specific neighbor rectified the problem.

Author's personal copy

Table 2
Sample bootstrap case-base for RUBiS.

S. no.	*SN*	*PU*	*PI*	*PB*	*SC*	*AH*	*MH*	Solution
1	Sell	1	1	1	1	0	1	CS1
2	Sell	0	1	1	1	1	1	CS2
3	AboutMe	1	0	1	1	1	1	CS3
4	Home	1	1	1	1	1	0	CS4
5	Browse	1	1	0	1	1	1	CS5

Input: Solution set *SS*, nearest neighbors *NN*
Output: Solution with majority vote
Method:
1. For each $i \in SS$
   a. $count[i] = 0$
2. End For
3. For each $j \in NN$
   a. $count[sol_i] = count[sol_i] + 1$
4. End For
5. return $\max_{k \in |SS|}(count[k])$

Fig. 9. Majority voting algorithm.

Input: $NN(C_p)$, error threshold ε
Output: Adapted solution sol_p
Method:
1. Sort $NN(C_p)$ in descending order with respect to similarity
2. For each $C_i \in NN(C_p)$
   a. If $(|sol_i - sol_{actual}| \leq \varepsilon)$
      i. return sol_i
   b. End If
3. End For

Fig. 10. Solution adaptation algorithm.

v. *Case retention*: Two cases are treated different if they vary by at least one attribute in this case-base. If the current case is different from all of the existing cases in the case-base, it is retained in the case-base as new knowledge. In this way, our experience base gets richer incrementally and causes the improved efficiency of the proposed approach.

7.2.3. Problem Resolution Manager

Problem Resolution Manager acts as the tool for applying the proposed and revised solution of CBA. It invokes an appropriate solution script against the suggested plan through actuator, which is executed on the deployed testbed.

7.3. Results and discussion for RUBiS testbed

CBR-based externalization approach was applied in the above experimental setup and results were analyzed. Two performance measures have been used in this case study namely accuracy and root mean squared error (RMSE). Results are discussed with reference to the following dimensions:

7.3.1. Role of N_n

If N_n is increased and more neighbors are allowed to contribute in the solution of C_p, then accuracy decreases, and RMSE increases, as shown in Tables 3 and 4. Maximum accuracy obtained using varying values of N_n is shown in Fig. 11. The deceasing trend is due to nature of RUBiS case-base which contains nominal and binary attributes. In these data types, every mismatch of two values of an attribute is equally treated and is equidistant.

Author's personal copy

Table 3
Accuracy of selected similarity functions for RUBiS.

Similarity measure	$N_n = 1$	$N_n = 2$	$N_n = 3$	$N_n = 4$	$N_n = 5$	$N_n = 6$
Hamming distance	**0.76**	0.56	0.5	0.45	0.3	0.29
Manhattan distance	**0.8**	0.5	0.37	0.36	0.3	0.29
Euclidean distance	**0.8**	0.5	0.37	0.36	0.3	0.29
Sim 1	**0.34**	0.28	0.23	0.28	0.29	0.29
Sim 2	0.16	0.1	0.1	0.22	**0.29**	0.29
Bray–Curtis distance	0.17	0.11	0.11	0.23	**0.29**	0.29
Squared Chord distance	**0.8**	0.5	0.37	0.36	0.3	0.29
Jaccard distance	**0.29**	0.29	0.29	0.29	0.29	0.29

Bold values are the optimal points.

Table 4
RMSE of selected similarity functions for RUBiS.

Similarity measure	$N_n = 1$	$N_n = 2$	$N_n = 3$	$N_n = 4$	$N_n = 5$	$N_n = 6$
Hamming distance	**0.489**	0.663	0.707	0.742	0.837	0.843
Manhattan distance	**0.447**	0.707	0.794	0.799	0.837	0.843
Euclidean distance	**0.447**	0.707	0.794	0.799	0.837	0.843
Sim 1	**0.812**	0.848	0.877	0.848	0.843	0.843
Sim 2	0.916	0.948	0.948	0.883	**0.843**	0.843
Bray–Curtis distance	0.911	0.943	0.943	0.877	**0.843**	0.843
Squared Chord distance	**0.447**	0.707	0.794	0.799	0.837	0.843
Jaccard distance	**0.843**	0.843	0.843	0.843	0.843	0.843

Bold values are the optimal points.

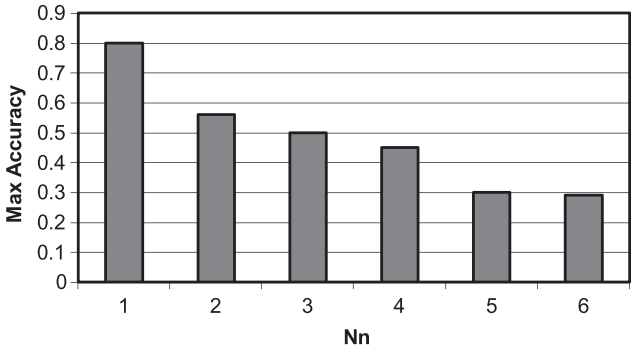

Fig. 11. Effect of N_n on maximum accuracy in RUBiS testbed.

7.3.2. Selection of similarity measure

This empirical investigation shows that three similarity measures among the selected ones perform equally good as shown in Tables 3 and 4. They are Manhattan distance, Euclidean distance and Squared Chord distance. Hamming distance also performed closer to these similarity measures. In this investigation, no adaptation algorithm was applied, and comparison was made on the basis of first suggested solution by *CBA*.

7.3.3. Role of adaptation algorithm

The iterative adaptation strategy adopted in this case study yielded inspiring results. Experiment was repeated three times and accuracy was achieved upto 98%. Results are shown in Tables 5 and 6. With the increased number of iterations

Table 5
Accuracy of adaptation algorithm for RUBiS.

Similarity measure	No adaptation	One iteration	Two iterations
Hamming distance	0.76	0.92	**0.98**
Manhattan distance	0.8	0.93	**0.98**
Euclidean distance	0.8	0.93	**0.98**
Sim 1	0.34	0.51	0.69
Sim 2	0.16	0.32	0.51
Bray–Curtis distance	0.17	0.33	0.52
Squared Chord distance	0.8	0.93	**0.98**
Jaccard distance	0.29	0.45	0.72

Bold values are the optimal points.

Author's personal copy

Table 6
RMSE of adaptation algorithm for RUBiS.

Similarity measure	No adaptation	One iteration	Two iterations
Hamming distance	0.489	0.283	**0.141**
Manhattan distance	0.447	0.245	**0.141**
Euclidean distance	0.447	0.245	**0.141**
Sim 1	0.812	0.7	0.557
Sim 2	0.916	0.824	0.7
Bray–Curtis distance	0.911	0.818	0.693
Squared Chord distance	0.447	0.245	**0.141**
Jaccard distance	0.843	0.742	0.529

Bold values are the optimal points.

in the adaptation strategy, accuracy increases, and RMSE decreases. Iterations 1, 2 and 3 yielded 80%, 93% and 98% accuracy respectively.

7.3.4. Lower bound in CBA performance

An interesting observation in this study is that performance of *CBA* becomes constant after $N_n = 6$. Another observation is that all similarity measures used in this case study perform equally bad at the maximum value of N_n.

7.3.5. Recommended model for CBA of RUBiS

On the basis of this empirical study, we selected *CBA* model $N_n = 1$, Manhattan distance, Euclidean distance or Squared Chord distance as similarity measure and the iterative adaptation strategy upto three iterations.

8. Case Study 2: Autonomic Forest Fire Application

Autonomic Forest Fire Application (AFFA) [46,47] was selected for the experiments based on the proposed CBR-based internalization architecture of autonomic computing. The application forecasts the strength, speed and direction of the forest fire as it propagates under various conditions. This prediction process is based on various dynamically changing environmental conditions. The application consists of four components as shown in Fig. 12: Data Space Manager (DSM), Computational Resource Manager (CRM), Rothermel, and Wind Model. DSM represents the forest as a two-dimensional data space of cells. CRM keeps records of the available resources and keeps DSM informed about them. Each cell may have unburned, burning, or burnt status. Rothermel computes the value and direction of fire spread based on wind direction and intensity provided by Wind Model. We have customized this application to predict the probability of a particular cell to be burnt. On the basis of this probability, CRM may run a configuration script to update its configurations and take further steps to adapt precautionary protective measures and reconfigure the forecasted parameters. AFFA is a rule-based self-configuration application. These rules have been defined in the operational port of the autonomic component. The case-base *CB* was created by recording the $\langle \alpha, \beta, \omega \rangle$ triplet by executing the rule-base, where α is the attribute, β is the corresponding value of α and γ is the

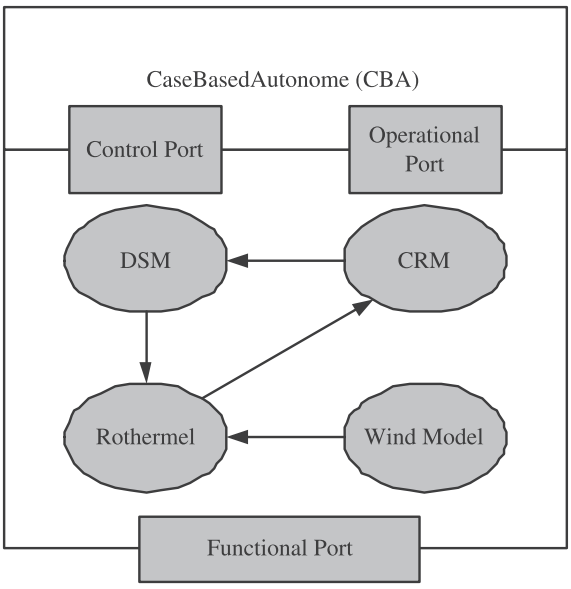

Fig. 12. AFFA architecture with CBA.

Author's personal copy

outcome of the rule. The direct and indirect attributes were extracted from the description of the rules and ports of the autonomic component given in Refs. [46,47].

AFFA uses the Accord programming framework [47] and constitutes two major parts [46,47], computational component and element manager, discussed below.

8.1. Computational component in AFFA

Computational part is the managed resource and constitutes the core business functionality of the whole autonomic application. In conventional architecture of the autonomic element, this is also known as managed resource. In our simulation, computational component represents the dynamic fire growth against different environmental conditions in the forest. Forest is represented by a two-dimensional grid. In our simulation, it is a 6×6 grid. Following the Accord framework [47], it defines the following three ports:

8.1.1. Functional port

This part of the autonomic application represents and manages the interactions of an autonomic component with other components and is represented as Π. It defines an input–output set of functionalities $\eta \in \Omega \times \Lambda$ where Ω represents the functionalities being used by the autonomic component and Λ represents the functionalities being exposed by this component to other components. This port has no contribution in this case-study because handling the functionality exposed by an autonomic component to the environment is out of scope of this research work.

8.1.2. Control port

Control port (Φ) of the autonomic component defines the relation between the managed resource and element manager for observing the current state and actuating self-management plan. As autonomic manager is *CBA* in this testbed, therefore *CBA* interacts with the computational component using Φ.

8.1.3. Operational port

This is another core part of the autonomic component which maintains rules and constraints used by the element manager and is represented as Γ. In this testbed, it maintains the case-base *CB* to plan the remedial action of the current case C_p. *CBA* interacts with the knowledge-base through Γ during various steps of CBR-based problem solving process.

8.2. CBA implementation for element manager of AFFA

Element manager is an embedded autonomic manager which takes the responsibilities of invoking appropriate rules from operational port. Element manager is implemented as a rule agent in the Accord framework [47]. The rule-agent has been replaced by *CBA* as shown in Fig. 12.

As AFFA is an internalization based application so autonomic manager, which is essentially *CBA* with the application ports, is not implemented as an isolated layer. A synthetic case-base of *500* cases of AFFA simulation was prepared using the rules and information of various ports given in Refs. [46,47]. Details of *CBA* implementation in this testbed are discussed as follows:

8.2.1. Case representation

For testing purposes, five parameters were used to describe the problem along with two dependent parameters to represent the solution. Selected parameters include Number of Burning Cells in the Grid (*N*), Wind Direction with reference to *X* (*WD*), Fire Speed (*FS*), Wind Intensity (*WI*) and Minimum Distance from the Burning Cell (*MD*). Two dependent attributes to be predicted are Predicted Probability (*PP*) that cell *X* will be exposed to fire and Configuration Script (*CS*) based on the *PP* value. *PP* values are distributed in four equidistant bins where each bin represents a *CS*.

Each case of Autonomic Forest Fire Application is represented as a set of the above mentioned parameters:

$$C_i = \{N, WD, FS, WI, MD, PP, CS\} \tag{9}$$

8.2.2. Case retrieval

The leave-one-out (LOO) cross-validation technique was used for the experiments in this case-study. Only those similarity measures were selected from Table 1 which are applicable on numeric datasets. We tested these similarity measures using different number of nearest neighbors as discussed in Section 6.

8.2.3. Case reuse

Case reuse phase is used to devise the solution of current problem using *NN*. We applied weighted arithmetic average (*WAA*) as the solution algorithm to devise the solution of C_p. *WAA* is given in Eq. (10).

$$sol_p = \sum_{q \in NN} w_{pq} sol_q \tag{10}$$

Author's personal copy

Table 7
Accuracy of selected similarity functions for AFFA.

Similarity measure	$N_n = 1$	$N_n = 2$	$N_n = 3$	$N_n = 4$	$N_n = 5$	$N_n = 6$	$N_n = 7$	$N_n = 8$
Manhattan distance	0.84	0.84	0.88	0.87	0.89	0.88	0.89	**0.9**
Euclidean distance	0.88	0.88	0.89	0.89	0.89	0.89	**0.9**	0.9
Sim 1	0.58	0.57	0.65	0.64	0.69	0.7	0.71	**0.72**
Sim 2	0.58	0.57	0.61	0.61	0.61	**0.62**	0.61	0.6
Canberra distance	0.79	0.79	0.82	0.82	0.82	**0.83**	0.83	0.83
Bray–Curtis distance	0.83	0.83	0.84	**0.86**	0.84	0.85	0.84	0.85
Squared Chord distance	0.85	0.85	0.87	0.86	0.87	0.87	0.87	**0.88**
Squared Chi-Squared distance	0.85	0.85	0.87	0.86	0.87	**0.88**	0.87	0.88
Jaccard distance	0.6	0.59	0.67	**0.68**	0.66	0.67	0.68	0.68

Bold values are the optimal points.

Table 8
RMSE of selected similarity functions for AFFA.

Similarity measure	$N_n = 1$	$N_n = 2$	$N_n = 3$	$N_n = 4$	$N_n = 5$	$N_n = 6$	$N_n = 7$	$N_n = 8$
Manhattan distance	0.101	0.08	0.078	0.078	0.077	**0.076**	0.077	0.077
Euclidean distance	0.09	0.08	0.076	0.076	0.076	0.076	**0.075**	0.075
Sim 1	0.181	0.145	0.141	0.137	0.134	0.131	0.13	**0.128**
Sim 2	0.173	0.148	0.137	**0.134**	0.134	0.135	0.135	0.136
Canberra distance	0.117	0.098	0.092	0.092	0.092	0.092	**0.091**	0.091
Bray–Curtis distance	0.1	0.09	0.085	**0.084**	0.084	0.085	0.085	0.085
Squared Chord distance	0.1	0.082	**0.078**	0.078	0.079	0.078	0.078	0.079
Squared Chi-Squared distance	0.099	0.083	0.078	0.078	0.078	**0.077**	0.078	0.078
Jaccard distance	0.169	0.144	0.135	0.135	0.131	0.129	0.129	**0.128**

Bold values are the optimal points.

where,

$$w_{pq} = \frac{sim_{pq}}{\sum_{q \in NN} sim_{pq}} \tag{11}$$

8.2.4. Case revision

Case revision strategy adapted in this case study involves variation of N_n. If current solution is not fitting in the current scenario then we re-iterate Algorithm 2 by changing N_n until an optimal value of N_n is found. It is evident from the empirical results that this approach yields up to 90% accuracy as shown in Table 7. This revision strategy is different from that of RUBiS due to the difference of datasets' nature.

8.2.5. Case retention

Once a solution of a new problem has been suggested then this new problem–solution pair is retained in the case-base as a new case. In this way, experience-base keeps on growing as well as getting richer. We retain a new problem–solution pair, if it does not match exactly with any of the existing cases. This affects retrieval efficiency when case-base gets too large. We intend to investigate this problem in the future work.

8.3. Results and discussion for AFFA

Accuracy and root mean squared error (RMSE) have been selected as the performance measures of the proposed methodology for this case study. The results of this empirical investigation have been analyzed across following dimensions.

8.3.1. Role of N_n

In this experimental setup, value of N_n was varied from 1 to the size of the case-base to investigate a value of N_n for which optimal accuracy is obtained for each similarity measure. Optimal values for all similarity functions were found for N_n in the range of 1–8. As depicted in the results shown in Tables 7 and 8, variation in accuracy is observed upto 2% which is not that much significant with the increasing value of N_n. This behavior with maximum possible accuracy is shown in Fig. 13. This behavior is due to the numeric nature of AFFA case-base in which all numeric attributes are scaled between 0 and 1.

8.3.2. Selection of similarity measure

Euclidean distance can be inferred as the best similarity measure for this case-study as it gives maximum accuracy and minimum RMSE among all similarity measures as shown in the second columns of Tables 3 and 4. Other closer similarity

Author's personal copy

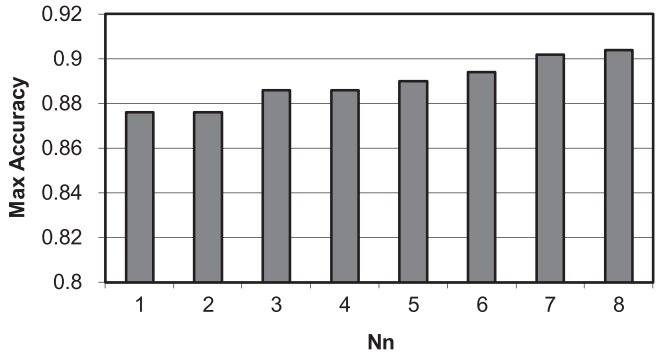

Fig. 13. Effect of N_n on maximum accuracy in AFFA testbed.

measures are Manhattan distance, Bray–Curtis distance, Squared Chord distance and Squared Chi-Squared distance. In this investigation, no adaptation algorithm was applied, and comparison was made on the basis of first suggested solution using N_n by *CBA*.

8.3.3. Role of adaptation algorithm

The adaptation strategy adopted in this case study was varying values of N_n. This adaptation strategy yielded accuracy upto 90% at $N_n = 7$ for Euclidean distance and $N_n = 8$ for Manhattan distance. Results shown in Tables 7 and 8 also indicate that this adaptation algorithm yielded upto 88% accuracy in case of Squared Chord distance and Squared Chi-Squared distance. With the increase of N_n in the adaptation strategy, no more optimal point beyond $N_n = 8$ was observed.

8.3.4. Recommended model for CBA of AFFA

On the basis of this empirical study, we selected *CBA* model with $N_n = 1$ and Manhattan distance, Euclidean distance, Squared Chord distance or Squared Chi-Squared distance as similarity measure. Adaptation strategy may give better performance but improvement observed in this adaptation strategy is upto only 2%. We recommend $N_n = 1$ because it yields upto 88% accurate results and offers minimum computational overhead. We recommend all these four similarity measures as the good candidate for the selected model because there is no abrupt major change in their behavior.

9. Conclusion and future work

In this paper, a CBR-based modeling approach has been presented for the externalization and internalization architectural options of autonomic systems. A CBR-based solution finder for autonomic systems *CaseBasedAutonome* (*CBA*) has been proposed and discussed in implementation level details. An empirical investigation has been conducted for exploring effect of different CBR parameters in externalization and internalization domains. Two case studies namely RUBiS and AFFA have been used in this empirical study. The results show that the proposed framework promises upto 98% accuracy in externalization architecture and 90% accuracy in internalization architecture. The study also reveals that Manhattan distance and Squared Chord distance perform as good as Euclidean distance in the externalization case study. But Euclidean distance is the best similarity measure for the internalization architecture across all cardinalities of nearest neighborhood and different adaptation schemes. This variation in performance is due to the nature of the datasets. The study also discusses the effect of cardinality (N_n) of nearest neighborhood and adaptation algorithms. No significant improvement in the performance of *CBA* has been observed for the larger values of N_n. Smaller values of N_n show adequate performance and have been used for confining the solution space of the monitored problem. Adaptation mechanisms of CBR presented in this paper have resulted in 18% and 6% performance improvement in externalization and internalization architectures respectively.

This study considers only discrete cases. In many scenarios, it is not possible to fetch the precise information from the managed element. Incorporating the vague and imprecise knowledge in autonomic manager by applying fuzzy theory along with CBR is the intended future direction of this work.

References

[1] A. Aamodt, E. Plaza, Case-based reasoning: foundational issues, methodological variations, and system approaches, AI Communications, vol. 7, IOS Press, 1994, pp. 39–59. 1.
[2] S. Abdelwahed, N. Kandasamy, S. Neema, A control-based framework for self-managing distributed computing systems, in: Proceedings of Workshop on Self-Healing Systems, ACM Press, 2004, pp. 3–7.
[3] M. Amoui, M. Salehie, S. Miararab, L. Tahvildari, Adaptive action selection in autonomic software using reinforcement learning, in: Proceedings of Fourth International Conference on Autonomic and Autonomous Systems (ICAS'08), IEEE Computer Society, 2008, pp. 175–181.
[4] C. Anglano, S. Montani, Achieving self-healing in autonomic software systems: a case-based reasoning approach, in: Proceedings of International Conference on Self-organization and Adaptation of Multi-agent and Grid Systems, IOS Press, 2005, pp. 267–281.

Author's personal copy

[5] J. Appavoo, K. Hui, C.A.N. Soules, R.W. Wisniewski, D.M.D. Silva, O. Krieger, M.A. Auslander, D.J. Edelsohn, B. Gamsa, G.R. Ganger, P. McKenney, M. Ostrowski, B. Rosenburg, M. Stumm, J. Xenidis, Enabling autonomic behavior in systems software with hot swapping, IBM Systems Journal 42 (1) (2003) 60–76.
[6] N. Arshad, D. Heimbigner, A.L. Wolf, Deployment and dynamic reconfiguration planning for distributed software systems, in: Proceedings of 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'03), 2003.
[7] X. Bai, Y. Liu, L. Wang, P. Zhong, Model-based monitoring and policy enforcement of services, Simulation Modelling Practice and Theory 17 (8) (2009) 1399–1412.
[8] R. Barrett, P.P. Maglio, E. Kandogan, J. Bailey, Usable autonomic computing systems: the administrator's perspective, in: Proceedings of 1st International Conference on Autonomic Computing, IEEE Press, 2004.
[9] B. Bartsch-Spörl, M. Lenz, A. Hübner, Case-Based Reasoning: Survey and Future Directions, Lecture Notes in Computer Science, vol. 1570, Springer, 1999.
[10] J.P. Bigus, D.A. Schlosnagle, J.R. Pilgrim, W.N. Mills, Y. Diao, Able: a toolkit for building multiagent autonomic systems, IBM Systems Journal 41 (3) (2002) 350–371.
[11] R. Calinescu, Implementation of a generic autonomic framework, in: Proceedings of International Conference on Autonomic and Autonomous Systems (ICAS'08), IEEE Computer Society, 2008, pp. 124–129.
[12] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, A. Fox, Microreboot – a technique for cheap recovery, Proceedings of 6th Symposium on Operating Systems Desing and Implementation (OSDI), USENIX Association, 2004.
[13] M. Chen, A.X. Zheng, J. Lloyd, M.I. Jordan, E. Brewer, Failure diagnosis using decision trees, in: Proceedings of International Conference on Autonomic Computing (ICAC'04), IEEE Computer Society, 2004, pp. 36–43.
[14] X. Chen, M. Simons, A Component Framework for Dynamic Reconfiguration of Distributed Systems, Lecture Notes in Computer Science, Springer-Verlag, 2002. pp. 82–96.
[15] Y. Chen, W.T. Tsai, Towards dependable service-orientated computing systems, Simulation Modelling Practice and Theory 17 (8) (2009) 1361–1366.
[16] D.M. Chess, C.C. Palmer, S.R. White, Security in autonomic computing environment, IBM Systems Journal 42 (1) (2003) 107–118.
[17] P. Cunningham, CBR: strengths and weaknesses, Proceedings of 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Springer-Verlag, 1998, pp. 517–523.
[18] Y. Diao, J.L. Hellerstein, S. Parekh, R. Griffith, G. Kaiser, D. Phung, Self-managing systems: a control theory foundation, in: Proceedings of 12th International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'05), IEEE Computer Society, 2005.
[19] K. Dias, M. Ramacher, U. Shaft, V. Venkataramani, G. Wood, Automatic performance diagnosis and tuning in oracle, in: Proceedings of the 2005 CIDR Conference, 2005.
[20] G. Fouqué, S. Matwin, CAESAR: a system for case based software reuse, in: Proceedings of 7th Knowledge-Based Software Engineering Conference, IEEE Press, 1992.
[21] G. Fouqué, S. Matwin, Compositional software reuse with case-based reasoning, in: Proceedings of the 9th International Conference on Artificial Intelligence for Applications, IEEE Press, 1993, pp. 128–134.
[22] N. Gandhi, J.L. Hellerstien, S. Parekh, D.M. Tilbury, Managing the performance of lotus notes: a control theoretic approach, in: Proceedings of the Computer Measurement Group, 2001.
[23] A.G. Ganek, T.A. Corbi, The dawning of the autonomic computing era, IBM Systems Journal 42 (1) (2003) 5–17.
[24] P.A. González, Applying knowledge modelling and case-based reasoning to software reuse, IEE Proceedings Software 147 (5) (2000) 169–177.
[25] A. Gounaris, C. Yfoulis, R. Sakellariou, M.D. Dikaiakos, A control theoretical approach to self-optimizing block transfer in web service grids, ACM Transactions on Autonomous and Adaptive Systems 3 (2) (2008).
[26] K.Z. Haigh, J.R. Shewchuk, Geometric similarity metrics for case-based reasoning, in: Case-Based Reasoning: Working Notes from the AAAI-94 Workshop, AAAI Press, 1994, pp. 182–187.
[27] J. Han, M. Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann Publishers, 2002.
[28] S. Hariri, B. Khargharia, H. Chen, J. Yang, Y. Zhang, M. Parashar, H. Liu, The autonomic computing paradigm, Cluster Computing 9 (2006) 5–17.
[29] S. Hassan, D. McSherry, D. Bustard, Autonomic self healing and recovery informed by environment knowledge, Artificial Intelligence Review (2007) 89–101.
[30] R. Hebig, H. Giese, B. Becker, Making control loops explicit when architecting self-adaptive systems, in: Proceedings of 2nd International Workshop on Self-Organizing Architectures (SOAR'10), ACM Press, 2010.
[31] E. Hullermeier, Case-Based Approximate Reasoning, B. Springer, 2007.
[32] E. Hüllermeier, D. Dubois, H. Prade, Formalizing case-based inference using fuzzy rules, Soft Computing in Case-Based Reasoning, Springer, 2000. pp. 47–72.
[33] J. Jann, L.M. Browning, R.S. Burugula, Dynamic reconfiguration: basic building blocks for autonomic computing on IBM pSeries servers, IBM Systems Journal 42 (1) (2003) 29–37.
[34] G. Jiang, H. Chen, K. Yoshihira, A. Saxena, Ranking the importance of alerts for problem determination in large computer systems, in: Proceedings of 6th International Conference on Autonomic Computing (ICAC'09), IEEE Computer Society, 2009.
[35] N. Kandasamy, S. Abdelwahed, J.P. Hayes, Self-optimization in computer systems via online control: application to power management, in: Proceedings of the International Conference on Autonomic Computing, IEEE Press, 2004.
[36] J.O. Kephart, D.M. Chess, The vision of autonomic computing, IEEE Computer Society, 2003. pp. 41–50.
[37] J.O. Kephart, W.E. Walsh, An artificial intelligence perspective on autonomic computing policies, in: Proceedings of the 5th International Workshop on Policies for Distributed Systems and Networks, IEEE ComputerSociety, 2004.
[38] M.J. Khan, M.M. Awais, S. Shamail, Achieving self-configuration capability in autonomic systems using case-based reasoning with a new similarity measure, Communications in Computer and Information Science, vol. 2, Springer, Berlin, Heidelberg, 2007. pp. 97–106.
[39] M.J. Khan, M.M. Awais, S. Shamail, Self-configuration in autonomic systems using clustered CBR, in: Proceedings of 5th International Conference on Autonomic Computing (ICAC'08), IEEE Computer Society, 2008.
[40] M.J. Khan, M.M. Awais, S. Shamail, Improving efficiency of self-configurable autonomic systems using clustered CBR approach, IEICE Transactions on Information and Systems E93-D (2010) 3005–3016.
[41] B. Khargharia, S. Hariri, M. Parashar, L. Ntaimo, B. Uk Kim, vGrid: a framework for building autonomic applications, in: Proceedings of the International Workshop on Challenges of Large Applications in Distributed Environments, IEEE Computer Society, 2003.
[42] T.M. Khoshgoftaar, N. Seliya, N. Sundaresh, An empirical study of predicting software faults with case-based reasoning, Software Quality Journal (2006) 85–111.
[43] R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), Morgan Kaufmann Publishers, 1995.
[44] H. Koshutanski, F. Massacci, Interactive access control for autonomic systems: from theory to implementation, ACM Transactions on Autonomous and Adaptive Systems 3 (3) (2008).
[45] Z. Li, M. Parashar, Rudder: an agent-based infrastructure for autonomic composition of grid applications, Multiagent and Grid System: An International Journal 1 (4) (2005) 183–195.
[46] H. Liu, M. Parashar, A component based programming framework for autonomic applications, in: Proceedings of the International Conference on Autonomic Computing, 2004.
[47] H. Liu, M. Parashar, Accord: a programming framework for autonomic applications, IEEE Transactions on Systems, Man and Cybernetics – Part C: Applications and Reviews 36 (3) (2006) 341–352.

Author's personal copy

[48] E. Mancini, U. Villano, M. Rak, R. Torella, A simulation-based framework for autonomic web services, in: Proceedings of the 11th International Conference on Parallel and Distributed Systems, IEEE Computer Society, 2005.
[49] R.L.D. Mántaras, D. Mcsherry, D. Bridge, D. Leake, B. Smith, S. Craw, B. Faltings, M.L. Maher, M.T. Cox, K. Forbus, M. Keane, A. Aamodt, I. Watson, Retrieval, reuse, revision, and retention in case-based reasoning, The Knowledge Engineering Review 20 (2005) 3.
[50] V. Markl, G.M. Lohman, V. Raman, LEO: an autonomic query optimizer for DB2, IBM Systems Journal 42 (1) (2003) 98–106.
[51] K. Mills, S. Rose, S. Quirolgico, M. Britton, C. Tan, An autonomic failure-detection algorithm, ACM SIGSOFT Software Engineering Notes 29 (1) (2004) 79–83.
[52] N.H. Minsky, On conditions for self-healing in distributed software systems, in: Proceedings of The International Autonomic Computing Workshop, IEEE Press, 2003.
[53] S. Montani, C. Anglano, Case-based reasoning for autonomous service failure diagnosis and remediation in software systems, Lecture Notes in Artificial Intelligence, 4106, Springer, 2006. pp. 489–503.
[54] S. Montani, C. Anglano, Achieving self-healing in service delivery software systems by means of case-based reasoning, Applied Intelligence 28 (2) (2007) 139–152.
[55] P. Myllymaki, H. Tirri, Massively parallel case-based reasoning with probabilistic similarity metrics, Lectures Notes in Artificial Intelligence, 837, Springer-Verlag, 1994. pp. 144–154.
[56] M. Parashar, S. Hariri, Autonomic computing: an overview, Lecture Notes in Computer Science, 3566, Springer, 2005. pp. 247–259.
[57] F. Qin, J. Tucek, J. Sundaresan, Y. Zhou, Rx: treating bugs as allergies – a safe method to survive software failures, in: Proceedings of 20th ACM Symposium on Operating Systems Principles, ACM Press, 2005.
[58] P. Reynolds, C. Killian, J.L. Wiener, Pip: detecting the unexpected in distributed systems, in: Proceedings of 3rd Symposium on Network Systems Design and Implementation, 2006.
[59] I. Rish, M. Brodie, N. Odintsova, Real time problem determination in distributed systems using active probing, in: Network Operations and Management Symposium, IEEE Press, 2004, pp. 133–146.
[60] C. Roblee, V. Berk, G. Cybenko, Implementing large-scale autonomic server monitoring using process query systems, in: Proceedings of 2nd International Conference on Autonomic Computing, IEEE Press, 2005.
[61] L. Spalazzi, A survey on case-based planning, Artificial Intelligence Review 16 (2001) 3–36.
[62] Tarek Abdelzaher, G. Kang, N.B. Shin, Performance guarantees for web server end-systems: a control-theoretical approach, IEEE Transactions on Parallel and Distributed Systems 13 (1) (2001) 80–96.
[63] C. Tautz, K.-D. Althoff, Using case-based reasoning for reusing software knowledge, Lecture Notes in Computer Science 1266 (1997) 156–165.
[64] G. Tesauro, D.M. Chess, W.E. Walsh, R. Das, A. Segal, I. Whalley, J.O. Kephart, S.R. White, A multi-agent systems approach to autonomic computing, in: Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems, IEEE Press, 2004.
[65] RUBiS: Rice University Bidding System, 2004. <http://rubis.objectweb.org/>.
[66] W. Tsaia, C. Fana, Y. Chena, R. Paul, A service-oriented modeling and simulation framework for rapid development of distributed applications, Simulation Modelling Practice and Theory 14 (6) (2006) 725–739.
[67] H.J. Wang, J.C. Platt, Y. Chen, R. Zhang, Y.-M. Wang, Automatic misconfiguration troubleshooting with peerpressure, in: Proceedings of the 6th Symposium on Operating Systems Design and Implementation, IEEE Press, 2004.
[68] I. Watson, Applying Case-Based Reasoning: Techniques for Enterprise Systems, Morgan Kaufmann Publishers, 1997.
[69] I. Watson, A case-based reasoning application for engineering sales support using introspective reasoning, AAAI/IAAI (2000) 1054–1059.
[70] I. Watson, F. Marir, Case-based reasoning: a review, The Knowledge Engineering Review 9 (4) (1994) 327–354.
[71] J. Wildstorm, P. Stone, E. Witchel, R.J. Mooney, M. Dahlin, Towards self-configuring hardware for distributed computer systems, in: Proceedings of International Conference on Autonomic Computing, IEEE Press, 2005, pp. 241–249.
[72] Z. Yu, J.J.P. Tsai, T. Weigert, An adaptive automatically tuning intrusion detection system, ACM Transactions on Autonomous and Adaptive Systems 3 (2008) 3.