A Methodology for Evaluating Complex Interactions between Multiple Autonomic Managers

Thaddeus Eze, Richard Anthony, Chris Walshaw, and Alan Soper
Autonomic Computing Research Group
School of Computing & Mathematical Sciences (CMS)
University of Greenwich, London, United Kingdom
{T.O.Eze, R.J.Anthony, C.Walshaw and A.J.Soper}@gre.ac.uk

***Abstract* — the very success of autonomic systems has inevitably led to situations where multiple autonomic managers need to coexist and/or interact directly or indirectly within the same system. This is evident, for example, in the increasing availability of large datacentres with multiple [heterogeneous] managers which are independently designed. Potentially, problems can arise as a result of *conflict-of-interest* when these managers (components) coexist. There is a growing concern that the lack of support for interoperability will become a break issue for future systems. We present an architecture-based solution to interoperability. Our approach is based on a Trustworthy Autonomic Architecture (different from traditional autonomic computing architecture) that includes mechanisms and instrumentation to explicitly support interoperability and trustworthiness. We posit that interoperability support should be designed in and integral at the architectural level, and not treated as add-ons as it cannot be reliably retro-fitted to systems. In this work-in-progress paper, we analyse the issue of interoperability and present our approach using a datacentre multi-manager scenario.**

Keywords- autonomic computing; interoperability; datacentre; multi-manager

I. INTRODUCTION

Autonomic Computing has progressively grown to become a mainstream concept. Earlier efforts were fundamentally concerned with getting autonomic computing to work and establishing fundamental concepts and demonstrating viability. Many mechanisms and techniques have been explored. Now that the concept of autonomic computing is well understood and widely accepted (and almost becoming commonplace), the focus has shifted to, amongst other things, addressing issues of scale and heterogeneity [1]. The increase in scale and size (of, e.g., datacentres) coupled with heterogeneity of services and platforms means that more Autonomic Managers (AMs) could be integrated to achieve a particular goal, e.g., datacentre optimisation. This has led to the need for interoperability between AMs. Interoperability deals with how to manage multi-manager scenarios, to govern complex interactions between managers and to arbitrate when conflicts arise. On the horizon these are the kind of challenges facing the autonomic computing research community [1][3].

The challenge of multi-manager interactions can be understandably enormous. This stems from the fact that, for example, components (and indeed AMs) could be multi-vendor supplied, upgrades in one manager could trigger unfamiliar events, scalability can introduce bottlenecks, one manager may be unaware of the existence of another, and managers, though tested and perfected in isolation, may not have been wired at design to coexist with other managers. Multi-manager coexistence leads to potential conflicts. A typical example is illustrated with a multi-manager datacentre scenario: consider a datacentre with two independent AMs working together (unaware of each other) to optimise the use of the datacentre –a Performance Manager (P_eM) optimises resource provisioning to maintain service level agreement (SLA). It does this by dynamically (re)allocating resources and maintaining a pool of idle servers to ensure high responsiveness to high priority applications. A Power Manager (P_oM) seeks to optimise power usage (as power is one of the major cost overheads of datacentres [4]) by shutting down servers that have been idle for a certain length of time. Although each manager performs brilliantly in isolation but by coexisting, the success of one manager defeats the goal of another –one seeks to shutdown a server that another seeks to keep alive. The (in)activities of one manager affect the costs of provisioning (e.g., delay cost, scheduling cost, competition cost etc.) for another in one way or the other. One way of mitigating this conflict is to have an external agent that can detect and diagnose the problem. The problem with this is that it introduces more complexity (e.g., any AM addition will require rewiring of other AMs) as system is scaled up (adding complexity in the process of solving a complexity problem) which is not desirable.

We have in [5] proposed a Trustworthy Autonomic Architecture (TAA). The TAA architecture, presented in Section III, employs a nested control loop technique to explicitly support run-time validation, dependability and trustworthiness. The *DependabilityCheck* component of the TAA provides a way of logically arbitrating between coexisting AMs. We present our interoperability approach in Section IV and conclude the work in Section V.

II. BACKGROUND

Kephart *et al* [2] presents a clear demonstration of the need for interoperability mechanisms. In that work two independently-developed AMs were implemented: the first dealt with application resource management (specifically CPU usage optimisation) and the second, a power manager, dealt with modulating the operating frequency of the CPU to ensure that the power cap was not exceeded. It was shown that without a means to interact, both managers throttled and sped up the CPU without recourse to one another, thereby failing to achieve their intended optimisations and potentially destabilising the system. We envisage widespread repetition of this problem until a universally accepted approach to interoperability is implemented.

Richard *et al* [3] evaluates the nature and scope of the interoperability challenges for autonomic systems, identifies a set of requirements for a universal solution and proposes a service-based approach to interoperability to handle both direct and indirect conflicts in a multi-manager scenario. In this approach, an Interoperability Service (IS) interacts with autonomic managers through a dedicated interface and is able to detect possible conflicts of management interests. In this way the IS manages all interoperability activities by granting or withholding management rights to different autonomic managers as appropriate. [3] discusses two types of conflicts in a multi-manager scenario: Direct conflicts occur where AMs attempt to manage the same explicit resource while indirect conflicts arise when AMs control different resources, but the management effects of one have an undesirable impact on the management function of the other. This latter type of conflict, in our opinion, is the most frequent and problematic, as there are such a wide variety of unpredictable ways in which such conflicts can occur.

Other works focus on bespoke interoperability solution [6], direct AMs interactions at the level of autonomic elements to ensure that management obligations are met [7], hierarchical relationship to autonomic element interactions [8] and MAPE architecture modification [9] where it is suggested to separate out the Monitoring and Analysis stages of the MAPE loop into distinct autonomic elements, with designed-in interactions between them.

The research community has made valuable progress towards AM interoperability but this progress is yet to lead to a standardised approach. Although the current state of practice is a significant step, an equally significant issue is that they do not tackle the problem of unintended or unexpected interactions that can occur when independently developed AMs co-exist in a system [3]. Further from that, and more realistically, AMs may not need to know about the existence of other managers –they are designed in isolation (probably by different vendors) and operate differently (for different goals) without recourse to one another. So, to have close-coupled interoperability (i.e., where specific actions in one AM react to, or complement those of another), the source code and detailed functional specifications of each AM must be available to all AMs. This is near impossible and where possible, requires a rewiring of each AM whenever a new AM is added. These are why we look to the autonomic architecture to provide us a solution –hence, our architecture-based approach. We posit that to avoid introducing further complexity through solving the interoperability problem, the autonomic architecture should envision (and provide for) interoperability support from the scratch. This is to say that the autonomic architecture should be dynamic enough to accommodate expected and unexpected developments.

III. THE TRUSTWORTHY AUTONOMIC ARCHITECTURE

TAA is an autonomics architectural framework that integrates three critical engine blocks (AC – *AutonomicController*, VC –*ValidationCheck* and DC – *DependabilityCheck*) in a modular fashion to lend autonomic systems extended (and robust) behavioural scope and trustability. These building blocks are implemented as modular components which are then connected to give the required trusted and dependable structure. To summarise the workings of TAA (see Figure 1), a system performs basic functions (to achieve its fundamental objectives) without any intelligent control of its activities. An autonomic manager (AC) is introduced to add some smartness by intelligently controlling the decision-making of the system. The actions of the manager are validated (VC) for correctness before they are actuated. A longer term control (DC) considers the behaviour of the manager over a period of time (after a certain number of decisions) to determine the effect of the manager's intervention on the system and to take corrective action (arbitrate) if need be. VC and DC can inhibit the decisions or actions of AC. For complete details of TAA see [5].

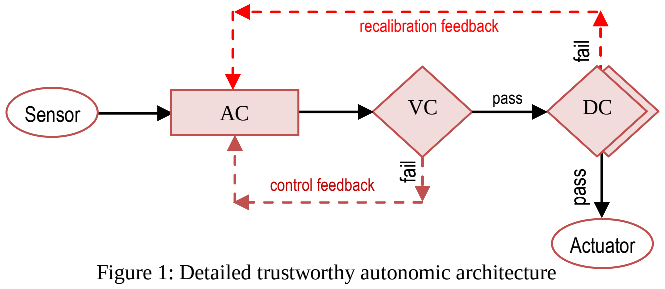


Figure 1: Detailed trustworthy autonomic architecture

In most of the autonomic systems, autonomic components are almost satisfactorily sufficient to provide required autonomic solution but in the longer term (e.g., as in multi-manager scenario), these rely on external supervision (typically by human) to extend their behavioural scope and trustability. This is resolved by the DC component. We rely on this component to address the interoperability problem as explained in Section IV. One of the powers of autonomics is its contextual generic implication and adaptation of terms and technologies. These are tailored to suit context and operational requirements. This quality allows us to adapt the TAA components (especially the DC) which can define, as necessary, stability and interoperability goals etc.

IV. THE ARCHITECTURE-BASED INTEROPERABILITY

Let us consider, in more details (Figure 2), the multi-manager datacentre example presented earlier in Section I: the datacentre comprises a pool of resources S_i (live servers), a pool of shutdown servers $\check{S}_i$ (ready to be powered and restored to S_i as need be), a list of applications A_j, a pool of services $\underset{\cdot\cdot}{U}$ (a combination of applications and their provisioning servers), and two AMs (performance manager P_eM and a power manager P_oM) that optimise the entire system. A_j and S_i are, respectively, a collection of applications supported (as services) by the datacentre and a collection of servers available to the manager for provisioning available services according to request. As service requests arrive, P_eM dynamically populates $\ddot{U}$ to service the requests. $\ddot{U}$ is defined by:

$$\ddot{U} = \begin{cases} A_1: (S_{11}, S_{12}, S_{13}, \ldots, S_{1i}) \\ A_2: (S_{21}, S_{22}, S_{23}, \ldots, S_{2i}) \\ \ldots \quad \ldots \quad \ldots \quad \ldots \\ A_n: (S_{n1}, S_{n2}, S_{n3}, \ldots, S_{ni}) \end{cases} \qquad (1)$$

Where n is the number of application entries into $\ddot{U}$. (1) indicates that a server can be (re)deployed for different applications. All the servers i in S_i are up and running (constantly available –or so desired by P_eM) waiting for (re)deployment. The primary performance goal of P_eM is to minimise oscillation and maximise stability (including just-in-time service delivery) while the secondary performance goal is to maximie throughput. The goal of P_oM, on the other hand, is to optimie power consumption. This task is simply achieved by shutting down any server that has been idle for time T_s. Figure 2 details how TAA is used to manage interoperability between P_eM and P_oM.

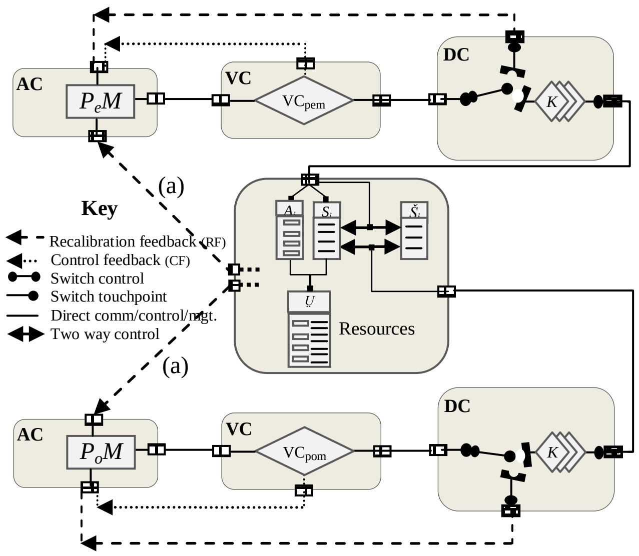


Figure 2: The DC component provides interoperability management

Figure 2 shows the communications and control within the components of the proposed architecture. The managers take performance decisions which are then validated by their respective VC (VC_{pom} and VC_{pem}) for correctness. A CF is generated if validation *fails* and with this feedback, the manager adjusts its decisions. The DC takes a longer term view of the managers' behaviour and either allows a manager to carry on with its actions (if check *passes*) or generates a RF otherwise. DC contains other subcomponents (K), e.g., *interoperability*, *stability* etc. [1] but for brevity, we will concentrate on the interoperability subcomponent here.

The *interoperability* component is implemented using knowledge-based technology. It learns and keeps track of the system's state following the *passed* decisions of the manager. If after a number of decision instances the manager senses a conflict with its decisions (based on expected versus actual system state), another RF is generated (a) to retune the manager's decisions. Take for instance, if after some time P_oM notices that the same set of servers it has shutdown have constantly come back live without it powering them, there is only one conclusion: another operation (probably a human, another manager, etc.) is not '*happy*' with P_oM's decisions. So, P_oM's DC generates a RF with an appropriate tuning parameter value (β) to throttle the size of T_s (2). By sensing the effects of its actions and dynamically throttling T_s within an acceptable boundary, P_oM is able to coexist with any other manager. Notice that the two managers do not need to know any details or even the existence of each other. In real life, this is typical of two staff that share an office space but work at different times. If both return next day and find the office rearranged, they will both adjust in their arrangement of the office until an accepted structure is reached. This can be achieved without both getting to meet. DC provides extra capacity for a manager to dynamically throttle its behaviour to suit the goal of the system.

$$T_s = (T_s \times \beta) \qquad (2)$$

There are costs associated with the operations of a datacentre. These costs are affected in one way or the other by the actions of the managers. We identify three costs (Table I) which are used in our experiment –this is not exhaustive.

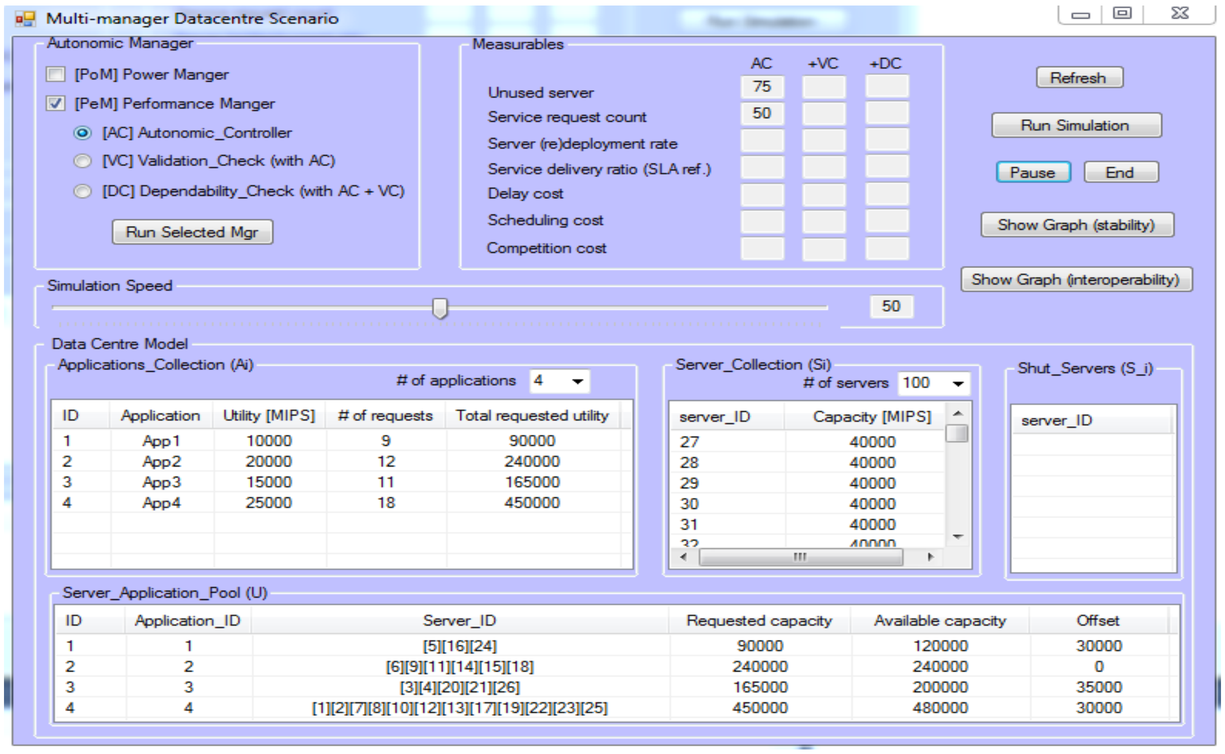
TABLE I: OPERATION COSTS

Cost	Description
Delay	Server booting and configuration time. Affects application performance
Scheduling	Reconfiguration and rescheduling time. Resource is unavailable during this time
Competition	One application has all resources and the other suffers

Apart from the costs mentioned in Table I above, other measurables from our experiment for analysing the performances of the managers include:

- Tracking SLA: service level will be measured as service delivery ratio (ratio of service delivery to service request) with an optimised value of 1.
  - Values above 1 indicate over provisioning which comes at a cost
  - Values below 1 indicate proximity to SLA
  - Server provisioning can be throttled to track SLA
- Impact of the manager on the above metrics over time

Figure 3 is a front-end snapshot of the system (still under design) which models our multi-manager datacentre case scenario example and analyses the performances of the managers. The system allows for the simulation of three different scenarios of coexisting managers. This provides for three coexisting options for P_oM and P_eM: in the first option (with AC component), the managers operate autonomously without any interoperability support; the second option (with AC and VC components) introduces local run-time validation within individual manager and without any interoperability support; the third option (with AC, VC and DC components) introduces, amongst other controls, interoperability support.


Multi-manager Datacentre Scenario

Autonomic Manager

[ ] [PoM] Power Manger
[✓] [PeM] Performance Manger
● [AC] Autonomic_Controller
○ [VC] Validation_Check (with AC)
○ [DC] Dependability_Check (with AC + VC)

Measurables

	AC	+VC	+DC
Unused server	75		
Service request count	50		
Server (re)deployment rate			
Service delivery ratio (SLA ref.)			
Delay cost			
Scheduling cost			
Competition cost			

Simulation Speed: 50

Data Centre Model

Applications_Collection (Ai) — # of applications: 4

ID	Application	Utility [MIPS]	# of requests	Total requested utility
1	App1	10000	9	90000
2	App2	20000	12	240000
3	App3	15000	11	165000
4	App4	25000	18	450000

Server_Collection (Si) — # of servers: 100

server_ID	Capacity [MIPS]
27	40000
28	40000
29	40000
30	40000
31	40000
32	40000

Shut_Servers (S_i)

server_ID

Server_Application_Pool (U)

ID	Application_ID	Server_ID	Requested capacity	Available capacity	Offset
1	1	[5][16][24]	90000	120000	30000
2	2	[6][9][11][14][15][18]	240000	240000	0
3	3	[3][4][20][21][26]	165000	200000	35000
4	4	[1][2][7][8][10][12][13][17][19][22][23][25]	450000	480000	30000

Figure 3: Multi-manager datacentre application

The third option is the main focus of this work. Other simulation options are also possible, e.g., selecting only PeM and running the above three options. On running the simulation, a script generates service requests. The service requests are measured in MIPS (million instructions per second). In the end, the performances of the managers (with and without interoperability support) are analysed against the listed measurables. This will identify, amongst other things, the effect/impact of our interoperability solution on the coexistence of the two managers.

V. CONCLUSION

We have presented, in this work-in-progress paper, an architecture-based interoperability solution. The solution is based on our earlier proposed trustworthy autonomic architecture. The architecture, which can be adapted to support several autonomic solutions, includes mechanisms and instrumentation to explicitly support run-time validation, interoperability and trustworthiness. We posit that to avoid introducing further complexity through solving the interoperability problem, the autonomic architecture should envision (and provide for) interoperability support from the scratch. This is to say that the autonomic architecture should be dynamic enough to accommodate expected and unexpected developments.

We analysed a multi-manager datacentre case example that represents a typical scenario of coexisting managers that leads to potential conflicts. This evaluates the nature and scope of the interoperability challenge and the need for a solution. We have also introduced an application that models the case example scenario. The next line of action is to run series of experiments once the case example application is fully completed. Results, analysis and further details will be published subsequently.

REFERENCES

[1] T. Eze, R. Anthony, C. Walshaw, and A. Soper, “Autonomic Computing in the First Decade: Trends and Direction,” 8th Int’l Conference on Autonomic and Autonomous Systems (ICAS, St. Maarten 2012), pp. 80-85.

[2] J. Kephart, H. Chan, R. Das, and D. Levine, “Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs,” 4th Int’l Conference on Autonomic Computing (ICAC, Florida, USA, 2007).

[3] R. Anthony, M. Pelc, and H. Shauib, “The Interoperability Challenge for Autonomic Computing,” 3rd Int’l Conference on Emerging Network Intelligence (EMERGING, Lisbon, Portugal, 2011).

[4] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, and G. Jiang, “Power and Performance Management of Virtualized Computing Environments via Lookahead Control,” 5th Int’l Conference on Autonomic Computing (ICAC, Illinois, USA, 2008).

[5] T. Eze, R. Anthony, C. Walshaw, and A. Soper, “A New Architecture for Trustworthy Autonomic Systems,” 4th Int’l Conference on Emerging Network Intelligence (EMERGING, Barcelona, Spain, 2012).

[6] M. Wang, N. Kandasamyt, A. Guezl, and M. Kam, “Adaptive performance control of computing systems via distributed cooperative control: Application to power management in computing clusters,” 3rd Intl. Conf. on Autonomic Computing (ICAC, Dublin, Ireland, 2006).

[7] M. Zhao, J. Xu, and J. Figueiredo, “Towards autonomic grid data management with virtualized distributed file systems” 3rd Int’l Conference on Autonomic Computing (ICAC, Dublin, Ireland, 2006).

[8] B. Khargharia, S. Hariri, and S. Yousif, “Autonomic power and performance management for computing systems,” 3rd Int’l Conference on Autonomic Computing (ICAC, Dublin, Ireland, 2006).

[9] M. Kutare, G. Eisenhauer, C. Wang, K. Schwan, V. Talwar, and M. Wolf, “Monalytics: Online monitoring and analytics for managing large scale data centers,” 7th Int’l Conference on Autonomic Computing (ICAC, Washington DC, USA, 2010).

[10] R. Anthony, “Policy-based autonomic computing with integral support for self-stabilisation,” Int. Journal of Autonomic Computing, Vol. 1, No. 1, 2009, pp. 1–33.