

Rainbow: An Approach to Facilitate Restorative Functionality Within Distributed Autonomic Systems

P. Miseldine, A. Taleb-Bendiab, M. Randles

*School of Computing and Mathematical Science, Liverpool John Moores University, Byrom St.
Liverpool, L3 3AF, UK
{cmppmise, a.talebbendiab, cmsmrand}@livjm.ac.uk*

Abstract

Core autonomic behaviours, namely self-healing, self-tuning, and self-adaptation, are now prevalent features of modern software architectures. Current implementations have typically relied on static rule and goal oriented approaches that respond to operational triggers to provide self-adaptive behaviour. This paper builds on the introspective nature of the previous work of the authors, namely the Cloud framework and Neptune language, to produce Rainbow, a system that enables storage and restoration of configuration data from an autonomic system. It is shown that the exposition of such configuration data enables system evolution to be captured and documented for analysis, and later restoration, essential abilities for successful anticipation and handling of changes in the operational context of an executing system. A working implementation of the proposed system is introduced, with conclusions made based on the future directions the research introduced in this paper can take.

1. Introduction

Much has been written [1, 2, 3] about the production of systems that support a key set of well researched principles that collectively produce autonomous systems. As such, by implementing autonomic behaviour, it is hoped a system will be more resilient in execution by adapting its operation to its operative environment, offering users intuitive interaction without any involvement in the configuration of the executing systems. This adaptation is essential in modern distributed software architectures, where issues of complexity are significant. Thus, as the complexity of a system outstrips the power of human ability to provide management for them, the systems themselves can automatically take care of the

majority of the associated mundane management tasks. However, due to this inherent complexity, approaches tried so far have produced little in the way of scaleable autonomic systems [4].

Current implementations of self adaptation have typically relied on static rule and goal oriented approaches that respond to operational triggers. In previous work [5], the author's contended autonomic systems themselves have little knowledge of their own execution and composition at runtime. This led to the specification, design and development of an autonomic configuration language, Neptune, and associated distributed framework, Clouds, that aims to produce a scalable approach to autonomic system deployment [6].

This paper introduces a methodology to enable the storage and retrieval of the configuration that constitutes and controls the autonomic behaviour of systems, by building on the open, introspective nature of the Neptune programming language, and Cloud Framework. Whilst future work has been identified that will further refine the processes involved, Rainbow, an implementation of the concepts outlined is introduced that uses a web service architecture model to expose all configuration data to end users.

2. Autonomic System Configuration

IBM has been at the forefront of recent research into autonomic system specification [1]. This research led to the defining of core self-management capabilities including; self-configuration, self-healing, self-optimisation, and self-protection. This approach is typified by DIOS++, [7] which provides an infrastructure that enables rule-based autonomic adaptation and control of distributed scientific applications.

Other research focused on self-healing software that modify their architecture during the system's execution

commonly known as runtime dynamism [8]. Formally defined models of system architectures such as the Hirsh et al. approach [9], the Le Métayer approach [10] and the CommUnity approach [11] allow operational changes of state to be modelled and analysed in the form “*given system x, what happens when change y occurs?*” [12].

Modelling autonomic systems was undertaken using Logic or process calculi [13, 14, 15], where for example systems are modelled in the form of actions made to and by it, causing the transition between specified system states. This provides an explicit logical representation of how a system can modify its architecture to provide dynamic software evolution. Implementations exist [8, 10] that use graphical representations to process algebra and logical formalisms to provide basic support for self-managing architectures.

An assessment and comparison of 14 state-of-the-art implementations is given in [12]. This found the major problem with many of these implementations is that the management logic is currently centralised and explicitly formalised within the architecture, making scalability an issue for complex systems. In addition, the flexibility to express the types of changes that yield operation changes are limited to the configuration operations that are available at the architectural level [16].

2.1 Context Aware Systems

Such limitations make reacting to the operational context of a system difficult. The context of the operating environment in which software operates is invariably affected by its host machine and current processes being executed upon it. Detailed knowledge of the individual components that comprise the software, their current and future status, and their capacity for work is essential for software to model this environment. In addition, networked distributed software systems require knowledge of all other systems they require services from as they are as dependent on their operation as its own local process. The production and distribution of this data has yielded much research. Sensors and instrumentation provide the means to ascertain the health of a system, so that its specific operational ability can be established. Sensors [17] monitor for particular changes specified by its operational configuration, from instruments provided by the governed resource.

Autonomy, as derived from nature, can be defined as exhibiting self regenerative qualities, that being a system that implements a set of capabilities enabling it to recover from attacks and to restore functionality while removing exploited vulnerabilities [18]. Such attacks can be thought of as any behaviour that is detrimental to a system’s working operation from both malicious and benign intention. As such, self regenerative systems must be able to detect errant behaviour, establish its effect upon the system, and produce and execute an action set to adapt

safely to the situation. Data emitted from the sensors can be compared to the system operational normative intentions, to provide operable limits to safe system operation. Thus, a system can detect errant behaviour when these operational norms are breached.

As data diversity provide the means of software diversity [19] in addition to regeneration and replication of resources, re-engineering the norms governing the operation of an executable resource can constrict it to operate within adjustable safety margins and thereby avoid behaviour that might be detrimental to its operation. With these governance norms externalised from the logical construction of the executable resource, such changes can be made without adapting the code base of the resource itself thereby producing a representative model to enable safe and adaptive re-engineering.

2.2 Neptune

To facilitate such knowledge representation within an adjustable framework at runtime, the Neptune Scripting Language [20] enables axioms, norms, and governance rules to be symbolised within an introspective object framework, such that the individual constructs that comprise a logical statement or assignment can be both inspected and modified without recompilation at runtime

In addition, Neptune Scripts include an abstraction based on the separation of process flow and their underlying logical model. By linking forks expressed within the process flow with the logic determining the path through the model, the context of a decision can be provided by way of data association.

As an illustrative example, to represent the concept of responsiveness from a service on the system, a Neptune script can provide a logical statement that determines the availability of a software component by inspecting and deliberating upon metrics obtained from the component’s instrumentation:

```
function responsive as boolean
{
  if (service.roundtripTime > 50)
  {
    return true;
  }
  else
  {
    return false;
  }
}
```

Figure 1: Neptune Script Representing Concepts

In this example, the metric “roundtripTime” is compared to a static value “50”. If the metric is above this threshold, the function returns true, otherwise, the function returns false. In essence, a set of logical constructs are

encapsulated within a concept. Accordingly, alteration of the underlying logical constructs allows the behaviour of the concept to be altered without affecting the structural signature of the concept when it is referenced elsewhere within the system.

2.3 The Cloud Framework

The notion of a Cloud is a system with loose boundaries which can interact and merge with other such systems. A Cloud can be thought of as a federation of services and resources controlled by a system controller and discovered through a system space. In a distributed system, oftentimes services and dependencies can overlap with different configurations on different systems. Systems based on the Cloud framework can interact with each other, sharing and pooling resources for greater efficiency over a large deployment such as an enterprise.

Neptune objects are executed on demand through an event model exposed by the Clouds architecture yielding a powerful extensible platform that is both wholly configurable at runtime, and that can be modelled at runtime.

2.3.1 Shared Memory: SystemSpace

Persistent data storage for service registration and state data is provided within a Cloud by a System Space, a service that controls access to a distributed data store. Data is stored in a generic format through a unified data structure. This structure contains an Access Code which describes the type of data being stored, an Information Token that is unique to the service to which the data stored is related to, and the data itself, which can be stored in both binary and XML formats. Data is retrieved by providing an access code, and an information token to return the data stored. This provides a clean methodology of organising and querying data within a relational database as data is described explicitly by both its type (the access code) and its owner (the information token). The structure of storage is also well suited to storing object definitions with the XML format capable of accepting serialised objects.

2.3.2. System Controller

At the centre of a Cloud is the System Controller (SC), a distributed service that controls access to and from the individual services and resources that are within the cloud. The SC brokers requests to services based on the system status and governance rules defined in Neptune Objects

2.3.3. Service and Resource Meta Objects

Each service and resource when it first registers itself to the Cloud sends a meta object serialized from an XML

definition file. This meta object contains the properties and state data of the service it is describing and is stored within the System Space at registration. Each service maintains its own meta object and updates the System Space when changes in state occur.

Due to the generic form of these meta objects, they provide an attribute system for services that can be queried inside Neptune scripts just as an object can be queried through its published properties in traditional object orientated software. The XML definition file contains all information required for the Cloud to discover the service through registration contained in the service element, and allow querying of the service status through the published state properties contained within the state element. As the meta objects are stored within the System Space, they enjoy system wide scope, allowing any other service or resource to request read and write access to the properties of a service through the system controller.

3 Configuration Exposition

By defining and deploying a system using Neptune objects and the Cloud framework, the configuration of the autonomic policy that governs the operation of the entire system is exposed as a set of serialisable objects. Any autonomous behaviour or reconfiguration a system exhibits is a direct consequence to the rules and policy represented within these objects. Figure 1 showed the concept of responsiveness encoded as a concept. This concept can be reused within any deliberative process of the system, and is represented within the data store in XML form, a section of which is shown in Figure 2:

```
<neptuneobject id = "ABCD-E344A-BFA3-ACA1">
  <functions>
    <function name = "responsive">
      <uses>
        <external id = "ABCD-E344A-BFA3"/>
      </uses>
      <logicblock>
        <statement>
          <evaluation type =
            "Neptune.Type.Integer"
            lefthand ="ABCD-E344A-BFA3"
            righthand = "50"
            operator =
              "Neptune.Types.largerThan">
            <return value = "true"/>
          </evaluation>
        </statement>
      </logicblock>
    </function>
  </functions>
</neptuneobject>
```

Figure 2: Configuration XML of Concept

This XML serialisation shows several distinct sections, which are useful in ascertaining configuration, as described in Figure 3.

With the complete set of these Neptune objects providing the configuration of the system, at any given point in the system's execution, a snapshot of the currently configured autonomic policy governing the system can be taken and stored.

A similar model currently successfully implemented within millions of systems worldwide is Microsoft's Windows System Restore. By storing the configuration data of a Windows PC (principally the system registry via keys and values) at regular intervals, systems can be reverted to a previous stable state at will, by replacing the current configuration with a stored snapshot. In relation to the configuration proposed here, the System Space is akin to the Windows Registry service, and the Neptune objects akin to the keys and associated values therein.

Such configuration storage enables the use of techniques such as State Based Analysis and statistical behavioural modelling to analyse state information, as typified by STRIDER for Windows System Restore [21].

context will change from showing typical usage of the system to a heavy usage context, and the system will adapt appropriately, as shown in figure 4.

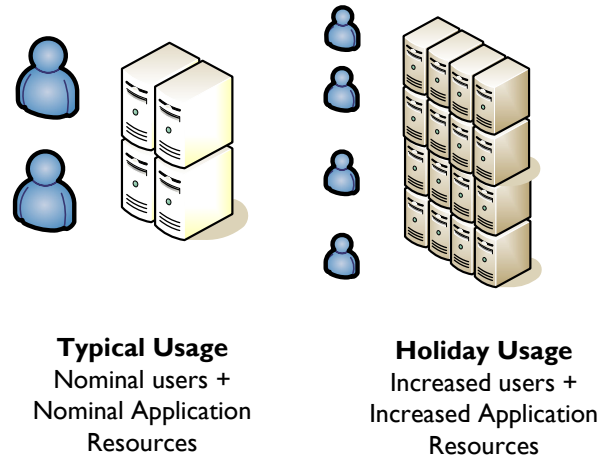


Figure 4: Context Variation

At this point, the system will be configured to operate within a heavy usage context. However, if the context reappears at a later occasion, such as at the Christmas holiday season, the system will have to adaptively move towards the new context, having no prior configuration knowledge of the context it encountered earlier. This means regeneration and reasoning upon the system, consuming resources and resulting in non-optimum resolution of the new context.

In contrast, using a restorative model, the configuration produced to deal with the heavy usage context at the summer period will have been stored, and in anticipation of a forthcoming period of heavy load, this stored configuration could have been restored, producing automatic regeneration of the system to react to the context.

XML Node	Relationship to Configuration
id attribute on neptuneobject	How the object can be reused within other configuration objects
function elements	A series of functions that collectively represent the features or methods of the object
uses element	References to the IDs of any Neptune object this object is reliant upon.
evaluation element	References any other Neptune object that is used to provide a deliberative measure

Figure 3: Configuration Entities

3.1 A Context-Aware Restorative Model

This configuration model enables the production of a restorative framework that can apply a previously stored snapshot of serialised objects to a live system. In this way, when operable system configuration is achieved, its state can be stored. This is of particular importance in an autonomic system, where regeneration often occurs through self-tuning, and self-adaptation.

As highlighted in Section 2.1, self-regenerative systems adapt to their operative environment, by sensing context. It can be foreseen that a context will regenerate at different occasions. For example, in an e-commerce enterprise selling holidays, the run up to the summer holiday period may produce an adverse number of visits to their web-based ordering system. Accordingly, the operational

3.2 Distributed State Based Analysis

Of particular relevance to a distributed system domain, is the concept of producing a state vector containing all configuration data regarding a system. This yields a model that allows comparison between these vectors, and consequently, between configurations. As such, the system can produce stable (or golden) state vectors from stored stable states, which may be available on different resources on the distributed system, providing a space domain, or "the mass", which can be compared against a known problematic state, or "the mess" as in the concept of "Attacking The Mess With the Mass" [21] allowing resolution of state. Accordingly, the analysis such a restorative model could produce is definite motivation to produce such functionality for autonomic systems.

4 Rainbow: A Restorative System Framework

What follows is a discussion of a proposed framework to facilitate the storage and retrieval of configuration data for systems based on the Cloud framework.

4.1 Storing and Restoring Configuration Snapshots

As stated in section 2.3.1, the meta models and policy definitions that adapt during the process of the system execution are stored within the System Space under an information token and access code. By mirroring this data store, the current configuration metrics can be captured, to be later restored back to the database. These loose constraints on the process of storage open the system to a wide ranging set of well-developed technologies to solve many of the issues surrounding data integrity, data backup, and data distribution. Microsoft SQL Server 2005 [22], the database management system used within the development of the Cloud and Rainbow evaluations, includes automatic backup and replication services, allowing the Cloud system to delegate these responsibilities to the DBMS used. However, a caveat is warranted here: such constraints also limit the deployment scenarios talked about within this paper to a centralised data store, whilst the Cloud itself is capable of using both locally stored Neptune objects and objects globally stored within a System Space. As such, discussion regarding the issues surrounding distributed replication and copy are outside the scope of this paper, but are discussed briefly in the Further Work section of this paper.

Whilst the system configuration is stored within Neptune objects, this is not sufficient to provide a notion of context upon which the type of deliberation discussed in Section 3 can occur. Operating system instruments, due to their very nature provide specific metrics regarding the operating ability of the resource being inspected. Such metrics could include CPU usage, memory allocation, and number of executing concurrent threads. Such data is essential for ascertaining the environmental context of a component of the overall system, in the same way as temperature and humidity provide part of the operational context of a marathon runner: such metrics will affect performance. Accordingly, the instrumentation and sensor information being provided throughout the system can be used to construct a measure of context. Indeed, an episodic model of context can be included allowing instrumentation measurements to be taken over a time frame, or for a period of activity, anticipated to be unique, critical or important to capture.

In the same manner in which data can be retrieved from the System Space, equally the System Space can be restored from a set of serialised Neptune objects. As objects are all referenced throughout the framework as

abstract IDs, changes to the object that represent the abstract IDs are propagated out throughout the system.

Indeed, whilst this paper focuses on the notion of a centralised storage mechanism, if Neptune objects were widely distributed and dispersed, all the requisite information regarding the relationships and links within the objects is available for inspection, and application to techniques commonly used within distributed data systems [23]

4.2 Analysis of Configuration Snapshots

With the configuration of an autonomic system available in XML form, considerable analysis and adaptation can be made offline, through either a system of simulation or comparative techniques. Rainbow can reproduce any objects within the configuration as well as chart their dependencies through the Neptune object relations described in section 3. As such, Rainbow can produce adaptive filters (or *beams*) that allow the reproduction of only sections of the configuration related to a particular concept or analysis to be undertaken.

One approach would be to consider the snapshots as an action history that could be applied to the principals of situation calculus as described in [24]. Such an approach would allow verification of the intentions made upon the system, for example, whether the system adapted as predicted or as required.

In addition, any alteration to the configuration made through such analysis can be re-implemented through the XML representation, providing both. As discussed in Section 3.2, comparisons with well known, proven configurations can be made, and adoption of the proven constructs made accordingly.

4.3 Implementation Specifics

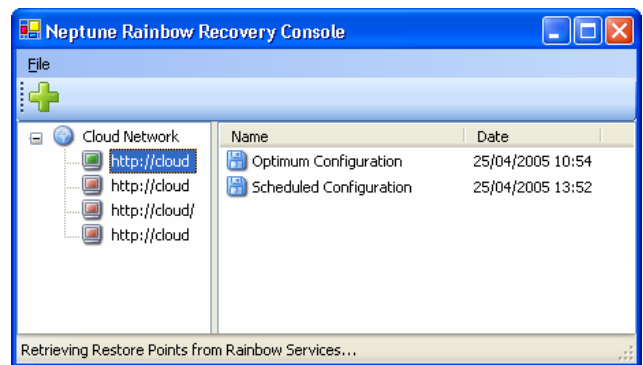


Figure 5: Rainbow Recovery Console

A set of web services have been developed to implement that allow external applications to request configuration snapshots to take place, transfer the configuration snapshot (or use the notion of a *beam* to transfer a discrete subset of

the configuration) and request a restoration of state to occur. As such, a host application that provides a graphical user interface to these services, called the Rainbow Recovery Console (see figure 5) has been developed that shows all available restorative points for a particular Cloud, and serves as a platform upon which to build an intuitive utility to provide the features discussed in this paper to the end-users of autonomic systems.

5. Conclusions and Future Work

The techniques introduced in this paper yield much further research, and other interesting propositions. If a set of systems all have autonomous system executions, it would be useful to compare their operating configurations to ascertain a base configuration which contains all configurable details common to these implementations. As such, a configuration consensus could be reached across a varied set of systems. This in itself would allow further benefits for system design, in as much as the following could be effectively answered: If a new system is introduced, what would its optimum configuration be when compared to existing systems in its environment?

The reverse is also true. By analysing what is *common* in configuration, when a configuration failure occurs, it should be feasible to pinpoint what is *different*, and therefore ascertain the erroneous configuration given a set of working configurations. The deployment scenario outlined in this paper includes the notion that all data will be centrally stored within the System Space (see Section 4). The Cloud however, allows distributed storage of the Neptune objects that collectively describe current system configuration. The future direction of the implementation outlined in this paper will include appreciation for distributed data replication.

References

- [1] IBM Research. Autonomic Computing. <http://www.research.ibm.com/autonomic>. Accessed March 2005
- [2] A.G. Ganek, T.A. Corbi, "The Dawning of the Autonomic Computing Era" IBM Systems Journal, 42(1) pp5-18, 2003
- [3] Cohn, D. L. (2003). Autonomic Computing Proceedings of the The Sixth International Symposium on Autonomous Decentralized Systems (ISADS'03) IEEE Computer Society
- [4] M Shackleton, F Saffre, R Tateson, E Bonsma, C Roadknight "Autonomic Computing for Pervasive ICT — A Whole-System Perspective", *BT Technology Journal*, Vol.22 No.3 pp 191-199, July 2004
- [5] M. Randles, A. Taleb-Bendiab, P. Miseldine, A. Laws "Adjustable Deliberation for Self-Managing Systems", *12th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2005)*, Washington MD, April 2005
- [6] P. Miseldine, A. Taleb-Bendiab "A Programmatic Approach to Applying Sympathetic and Parasympathetic Autonomic Systems to Software Design". *To appear in the 3rd International Workshop on Self-Adaptive and Autonomic Computing Systems*.
- [7] H. Liu and M. Parashar. "DIOS++: A Framework for Rule-Based Autonomic Management of Distributed Scientific Applications," Proceedings of the 9th International Euro-Par Conference (Euro-Par 2003), Lecture Notes in Computer Science. August 2003.
- [8] Li, D. and Muntz, R, Runtime Dynamics in Collaborative Systems. Department of Computer Science University of California, Los Angeles. 1999.
- [9] D. Hirsch, P. Inverardi, and U. Montanari. Graph grammars and constraint solving for software architecture styles. In Proc. of the Int. Software Architecture Workshop, pages 69–72, Nov. 1998.
- [10] D. L. M'etayer. Software architecture styles as graph grammars. In Proc. of the ACM SIGSOFT Symp. On Foundations of Software Engineering, pages 15–23. ACM Press, 1996.
- [11] M. Wermelinger, A. Lopes, and J. L. Fiadeiro. A graph based architectural (re)configuration language. In Proc. of the Joint European Software Engineering Conference and Symp. On the Foundations of Software Engineering, pages 21–32.
- [12] J.S. Bradbury, J.R. Cordy, J. Dingel and M. Wermelinger, "Supporting Self-Management in Dynamic Software Architecture Specifications", submitted to WOSS'04 - ACM SIGSOFT 2004 Workshop on Self-Managed Systems, submitted August 2004, 10 pp.
- [13] H. J. Levesque, F. Pirri and R. Reiter (1998) 'Foundations for the situation calculus'. Linköping Electronic Articles in Computer and Information Science, Vol. 3(1998): nr 18.
- [14] Allen J. F. 'Towards a general theory of action and time'. *Artificial Intelligence*, 23, pp 123-154. 1984.
- [15] Thielscher M. 'Introduction to the fluent calculus' Linköping Electronic Articles in Computer and Information Science, Vol. 3(1998): nr 14.
- [16] M. Wermelinger. Towards a chemical model for software architecture reconfiguration. *IEE Proceedings - Software*, 145(5):130–136, October 1998.
- [17] J. Hill, "A software architecture supporting networked sensors," Master's thesis, U.C. Berkeley Dept. of Electrical Engineering and Computer Science, 2000/
- [18] Low PA. Autonomic nervous system function. *J Clin Neurophysiol*. 1993;10:14-27.
- [19] Hiller M. Software Fault Tolerance Techniques from a Real-Time Systems Point of View: An Overview Technical Report No. 98-16, Dept. of Computer Engineering, Chalmers University of Technology, Sweden, 1998
- [20] Miseldine P., Taleb-Bendiab A. Neptune: The Specification of Adjustable Runtime Introspection. Pending Acceptance.
- [21] Wang Y, Verbowski C/, Dunagan J., Chen Y., Wang H., Yuan C., and Zhang Z. Microsoft Research In *Proceedings of LISA* (2003).
- [22] Microsoft Corp. *Introducing SQL Server 2005*. <http://www.microsoft.com/sql/2005/default.asp>. [Last accessed 28/04/2005]
- [23] M. T. Ozsu and P. Valduriez. Principles of Distributed Database Systems (2nd Edition). Prentice Hall, 1999.
- [24] M. Randles, P. Miseldine, A. Taleb-Bendiab. A Stochastic Situation Calculus Modelling Approach for Autonomic Middleware, Proceedings of PGNet 2004, Liverpool John Moores University, 2004