# Table of Contents

Table of Contents ........................................................................................................... 2  
List of Figures ................................................................................................................ 3  
List of Tables .................................................................................................................. 4  
List of Listings ............................................................................................................... 5  
1.0 Introduction to Autonomic Computing .............................................................. 6  
1.1 Autonomic Computing Concepts ....................................................................... 6  
1.2 Autonomic Computing Model ........................................................................... 6  
1.2.1 Autonomic Manager .......................................................................................... 7  
1.2.2 Managed Resource ............................................................................................. 8  
1.2.3 Managed Resource Touchpoint .......................................................................... 8  
1.3 Levels of Autonomic Maturity ........................................................................... 8  
2.0 Autonomic Computing Toolkit .......................................................................... 8  
3.0 Resource Models ................................................................................................ 9  
3.1 Decision Algorithm Flow ................................................................................. 10  
3.2 Autonomic Management Engine Hosting Environment .................................. 11  
3.3 The CIM-M12 Information Model .................................................................. 12  
4.0 Common Base Events ...................................................................................... 13  
4.1 Managed Resource and Manager Touchpoints ................................................ 14  
5.0 Generic Log Adapter ........................................................................................ 15  
6.0 Log Trace Analyzer .......................................................................................... 16  
7.0 Samples ............................................................................................................ 16  
7.1 Level 2 Maturity .............................................................................................. 18  
7.2 Level 4 Maturity .............................................................................................. 18  
7.3 Building a Resource Model ............................................................................. 19  
7.3.1 Deploying a Resource Model ........................................................................... 23  
8.0 Problem Determination Scenario ..................................................................... 25  
9.0 Solution Installation and Deployment Scenario .............................................. 27  
10.0 On-Demand ...................................................................................................... 29  
10.1 Provisioning ..................................................................................................... 30  
10.2 Policy-Based Orchestration ............................................................................. 30  
11.0 ITITO and ITPM ............................................................................................... 31  
11.1 ITITO Architecture .......................................................................................... 31  
11.1.1 Data Acquisition Engine .................................................................................. 31  
11.1.2 Application Controller ..................................................................................... 32  
11.1.3 Deployment Engine ........................................................................................ 34  
11.1.4 Global Resource Manager ........................................................................... 34  
11.2 Workflows ........................................................................................................ 35  
12.0 Orchestration and Provisioning Library ........................................................... 35  
13.0 Success Stories ................................................................................................. 36  
13.1 STAR Technologies, UK ................................................................................. 36  
13.2 Galeries Lafayette Group ................................................................................. 37  
13.3 eBay ................................................................................................................. 37  
14.0 Similar Products ............................................................................................... 37  
14.1 HP Adaptive Enterprise ................................................................................... 38  
14.2 Microsoft Dynamic Systems Initiative (DSI) .................................................. 38  
15.0 IBM’s Strengths and Challenges ..................................................................... 38  
References .................................................................................................................... 40
List of Figures

Figure 1 Four Attributes of Autonomic Computing [6] .................................................6
Figure 2 The Autonomic Computing Model [6] ............................................................7
Figure 3 Generic Decision Algorithm Flow [4] ............................................................11
Figure 4 Autonomic Management Engine Hosting Environment [2] ..........................12
Figure 5 CIM-M12 Information Model [2] .................................................................13
Figure 6 Interaction Between the two Touchpoint Interfaces [6] .................................14
Figure 7 Data Flow of an Application Using Generic Log Adapter [6] .......................15
Figure 8 Screenshot of Generic Log Adapter Configuration File.................................16
Figure 9 Screenshot of Events view the Log Trace Analyzer .....................................17
Figure 10 Screenshot of Correlation view in Log Trace Analyzer ...............................17
Figure 11 Components for Maturity Level 2 [6] ..........................................................18
Figure 12 Components for Maturity Level 4 [6] ..........................................................19
Figure 13 An Event and its Corresponding Action in Resource Model Builder ...............21
Figure 14 Interactions of the Problem Determination Scenario [8] .........................25
Figure 15 Details of each Components of the Problem Determination Scenario [8] .26
Figure 16 Self-Healing Event Flow in the Problem Determination Scenario [8] ..........26
Figure 17 Features of an On-Demand Operating Environment [7] .............................29
Figure 18 Detailed Breakdown of the Features of On-Demand [7] ...............................29
Figure 19 Architecture Overview of ITITO [7] .........................................................32
Figure 20 Architecture Overview of Data Acquisition Engine [7] ..............................33
Figure 21 Architecture Overview of Application Controller [7] .................................34
Figure 22 Architecture Overview of Deployment Engine [7] .....................................34
Figure 23 Architecture Overview of Global Resource Manager [7] ............................35
Figure 24 Examples of Workflows [7] .......................................................................36
List of Tables

Table 1 Autonomic Computing Concepts ................................................................. 7
Table 2 Four Parts of the Intelligent Control Loop ....................................................... 8
Table 3 Four Interaction Styles Between Sensors and Effectors ................................. 8
Table 4 Description of Levels of Autonomic Maturity ............................................... 9
Table 5 Technologies and Tools Provided in Autonomic Computing Toolkit .......... 9
Table 6 Attributes of a Resource Model ................................................................. 10
Table 7 Data Sources to Decision Algorithm ........................................................... 12
Table 8 Operations Supported by CIM-M12 .......................................................... 13
Table 9 Description of Fields and SubFields in Common Base Event ................. 14
Table 10 Description of Fields in Generic Log Adapter Configuration File ............ 15
Table 11 Steps in Filling in Parameters of Resource Model .................................... 20
Table 12 SARA Commands .................................................................................... 23
Table 13 Self-Healing Steps of Problem Determination Scenario ....................... 27
Table 14 Common Services in Solution Installation and Deployment Technology ... 28
Table 15 Components of Solution Installation and Deployment Technology .......... 28
Table 16 Three Features of an On-Demand Operating Environment ...................... 30
List of Listings

Listing 1 FolderMonitor MOF File ................................................................. 20
Listing 2 Decision Tree Script Init Function ............................................... 22
Listing 3 Decision Tree Script VisitTree Function ...................................... 22
Listing 4 Install Resource Model Java Code ............................................. 23
Listing 5 Create Resource Model Instance Java Code ............................. 24
Listing 6 Start Resource Model Instance Java Code ............................... 24
Listing 7 Stop Resource Model Instance Java Code ............................... 25
Listing 8 Decision Tree Script of Problem Determination Scenario .......... 27
1.0 Introduction to Autonomic Computing

There is a growing concern in the IT industry regarding a looming software complexity crisis [5]. The overwhelming complexity and sophistication of software applications and environments are requiring more skilled IT professionals to install, configure, tune and maintain them. For example, the need to integrate several heterogeneous environments into corporate-wide computing systems, and to extend that beyond the company boundaries into the Internet, introduces several new levels of complexity to the system. Computing systems’ complexity would soon be pushing beyond the limits of human capabilities to install, configure, tune, and maintain them.

The motivation for companies to adopt an alternative is clear. One such alternative is to create a computing environment with the ability to manage itself and dynamically adapt to changes in accordance with business policies and objectives. In other words, the goal of the system is to be able to dynamically deploy, monitor, manage, and protect IT infrastructure to meet business needs with little or no human interventions. This concept is called autonomic computing, and it will shift the burden of managing IT systems from IT professionals to the systems themselves. With autonomic computing, the cost and complexity of owning and operating an IT infrastructure would be much reduced. More details are discussed in [5].

1.1 Autonomic Computing Concepts

The four attributes involved in a self-managing autonomic computing system are shown pictorially in Figure 1. This set of attributes is commonly coined by the term self-CHOP. A description of each of these concepts is shown in Table 1. Please refer to [6] for more details.

![Figure 1 Four Attributes of Autonomic Computing](image)

1.2 Autonomic Computing Model

The autonomic computing model that is used by IBM is shown in Figure 2. There are three components in this architecture, namely the autonomic manager, the managed resource touchpoint, and the managed resource. In essence, the autonomic manager facilitates an intelligent control loop which delivers the self-CHOP attributes to the managed resource.
Table 1 Autonomic Computing Concepts

<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-Configuration</td>
<td>The ability of the system to automatically adapt to dynamically changing environment, based on policies provided by IT professionals. An example of such a change is to configure a new component properly so that it can be used in a cluster with high workload</td>
</tr>
<tr>
<td>Self-Healing</td>
<td>The ability to discover, diagnose, and react to service disruptions. Components in a self-healing system can detect system malfunctions and initiate corrective actions involving state changes in the own component or other components based on a set of provided policies</td>
</tr>
<tr>
<td>Self-Optimization</td>
<td>The ability of the component to monitor and tune resources automatically to meet end-user or business needs. One such example is the reallocation of resources to improve overall server utilization and to ensure that business transactions can be completed in a timely manner</td>
</tr>
<tr>
<td>Self-Protection</td>
<td>The ability to anticipate, detect, identify, and protect against attacks and malicious users. Examples of hostile actions which could undermine the system include unauthorized access, virus infections, and denial of services attacks. Corrective measures can be carried out to make the system less vulnerable</td>
</tr>
</tbody>
</table>

Figure 2 The Autonomic Computing Model [6]

1.2.1 Autonomic Manager

The autonomic manager is where the intelligent control loop is implemented and executed. Based on a set of known information about the managed resource, the autonomic manager carries out the control loop operations. At the same time, new information is gathered and put into the knowledge base as the autonomic manager learns more about the managed resource. The four parts of the intelligent control loop is described in Table 2.
Table 2 Four Parts of the Intelligent Control Loop

<table>
<thead>
<tr>
<th>Part of Control Loop</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor</td>
<td>Mechanisms for collecting, filtering, managing, and reporting details (e.g. metrics) from a managed resource</td>
</tr>
<tr>
<td>Analyze</td>
<td>Mechanisms for correlating and model complex situations to learn about IT environment and predict future situations</td>
</tr>
<tr>
<td>Plan</td>
<td>Mechanisms for constructing recommendations and actions to achieve provided goals and objectives</td>
</tr>
<tr>
<td>Execute</td>
<td>Mechanisms for controlling the execution of the plan with possible dynamic updates</td>
</tr>
</tbody>
</table>

1.2.2 Managed Resource

A managed resource is any system component that is controlled by the autonomic manager for the purpose of self-management. It can be a single resource (e.g. a server or a router) or a set of resources (e.g. a pool of servers).

1.2.3 Managed Resource Touchpoint

The managed resource touchpoint is an interface which facilitates communication between the autonomic manager and the managed resource. Sensors are used to transmit events or properties from a managed resource to an autonomic manager, while effectors are by the autonomic manager to incur some change in the managed resource. There are two interaction styles on both the effector and the sensor. These interactions styles are described in Table 3.

Table 3 Four Interaction Styles Between Sensors and Effectors

<table>
<thead>
<tr>
<th>Touchpoint</th>
<th>Interaction Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor</td>
<td>Retrieve-State</td>
<td>The autonomic manager polls the managed resource for some details</td>
</tr>
<tr>
<td></td>
<td>Receive-Notification</td>
<td>The managed resource sends spontaneous messages to the autonomic manager</td>
</tr>
<tr>
<td>Effector</td>
<td>Perform-Operation</td>
<td>The autonomic manager issues a command to against the managed resource</td>
</tr>
<tr>
<td></td>
<td>Call-Out-Request</td>
<td>The managed resource consults autonomic manager for some details before taking action</td>
</tr>
</tbody>
</table>

1.3 Levels of Autonomic Maturity

IT infrastructures are classified into five levels of maturity that show how the autonomic capabilities and the supporting processes and skills in the system are evolving. A description of each level is shown in Table 4.

2.0 Autonomic Computing Toolkit

IBM provides a free set of tools in its Autonomic Computing Toolkit to help product developers to begin integrating autonomic capabilities into their products. The entire technology and tool set provided in the Autonomic Computing Toolkit is summarized
in Table 5. Some of these technologies and tools will be discussed in more details in subsequent sections. More information can also be found in [1], [2], [3], [6], [8], [9], and [10].

Table 4 Description of Levels of Autonomic Maturity

<table>
<thead>
<tr>
<th>Level</th>
<th>Maturity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Basic</td>
<td>IT professionals manage each infrastructure element independently with manual setup, monitor, deployment and replacement operations</td>
</tr>
<tr>
<td>2</td>
<td>Managed</td>
<td>System management technologies are in place to collect information from disparate systems and to display the information in a meaningful and user-friendly manner. The monitor portion of the intelligent control loop is facilitated in the level</td>
</tr>
<tr>
<td>3</td>
<td>Predictive</td>
<td>On top of monitoring of situations that arise in the environment, analysis capabilities are built into the system to analyze the situations and provide possible courses of actions. The monitor, analyze, and plan portions of the control loop are facilitated</td>
</tr>
<tr>
<td>4</td>
<td>Adaptive</td>
<td>IT infrastructure can automatically take actions based on the analysis from the available information and knowledge gathered from the environment. The entire control loop of monitor, analyze, plan, and execute is beginning to take form</td>
</tr>
<tr>
<td>5</td>
<td>Autonomic</td>
<td>Business policies and objectives automatically take full control of the IT infrastructure operations. IT professionals only need to interact with the autonomic technology to monitor business processes and alter objectives</td>
</tr>
</tbody>
</table>

3.0 Resource Models

A resource model represents any resources that one wish to monitor. Examples of resource models include hard drive, CPU usage, memory, application and network load. The attributes that comprises a resource model are listed in Table 6. Please refer to [2], [4] and [6] for added information.

Table 5 Technologies and Tools Provided in Autonomic Computing Toolkit

<table>
<thead>
<tr>
<th>Type</th>
<th>Tools Available</th>
</tr>
</thead>
</table>
| Autonomic Managers          | ➢ Log and Trace Analyzer  
                               ➢ Autonomic Management Engine  
                               ➢ Solution Installation and Deployment  
                               Dependency Checker and Change Manager |
| Managed Resource Touchpoint | ➢ Generic Log Adapter Touchpoints  
                               ➢ Solution Installation and Deployment Touchpoints |
| Customization Tools         | ➢ Resource Model Builder  
                               ➢ Adapter Configuration Editor  
                               ➢ Integrated Solution Console Toolkit |
| User Access                 | ➢ Integrated Solution Console |


Table 6 Attributes of a Resource Model

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Collection</td>
<td>Specifies how data is to be collected and fed to the autonomic manager engine to be analyzed. Data is collected synchronously and cyclically.</td>
</tr>
<tr>
<td>Cycles</td>
<td>Specifies how often data is to be collected. This interval of time must be passed before new data is collected and analyzed.</td>
</tr>
<tr>
<td>Events and Indications</td>
<td>An indication occurs when a resource states meet a certain criteria as specified by the user (e.g. CPU usage above 80%). An event is a notification to the system that the status of the resource has changed, as indicated by the indicator.</td>
</tr>
<tr>
<td>Attributes</td>
<td>Strings or numeric values associated with events according to the information collected by the resource model. Can be used to identify and describe the event that occurred.</td>
</tr>
<tr>
<td>Actions</td>
<td>The corrective measures that are automatically triggered when a specific event occurs.</td>
</tr>
<tr>
<td>Thresholds</td>
<td>A limit on the resource’s acceptable level of performance. An indication is generated if the performance reaches above the threshold.</td>
</tr>
<tr>
<td>Logging</td>
<td>Specifies how information regarding the attributes of the resource model is to be logged.</td>
</tr>
<tr>
<td>Decision Tree Script</td>
<td>Defines the initialization steps of a resource, and the actions to be carried out at each interval when data is collected. More discussion on this in the next and subsequent sections.</td>
</tr>
<tr>
<td>MOF Files</td>
<td>Managed Object Format file defining resource and specifies the providers used to perform tasks. Defines the methods and operators available as well as the attributes that can be extracted from the resource.</td>
</tr>
<tr>
<td>Dependencies</td>
<td>Additional files (e.g. scripts, executable files) that are needed to run the resource (e.g. as corrective measures).</td>
</tr>
</tbody>
</table>

3.1 Decision Algorithm Flow

The decision algorithm is specified in the VisitTree() function of the decision tree script. The decision algorithm is typically written in JavaScript or VBScript. The autonomic management engine periodically runs the decision algorithm at each interval as specified in the cycle attribute.

The decision algorithm should check and analyze each resource’s attributes, such as resource availability and performance status, against any quality parameters specified in the resource model. If the measurement of an attribute passes the threshold value, an event is triggered to signal the problem. A corrective action can be optionally carried out in response to this event. Performance data can be logged for historical analysis purpose, such as autocorrelation and time variance analyses.

The generic flow of the decision algorithm is shown in Figure 3. More details can be found in [4].
3.2 Autonomic Management Engine Hosting Environment

The AME hosting environment structure is shown in Figure 4. The three main sources of data to be provided to the decision script are summarized in Table 7. Multiple instances of the resource model can be run simultaneously, each with a different configuration or context.
Table 7 Data Sources to Decision Algorithm

<table>
<thead>
<tr>
<th>Data Source</th>
<th>Description</th>
</tr>
</thead>
</table>
| RM Descriptor    | An XML document describing the resource model. It contains the following pieces of information:  
  ✓ Thresholds  
  ✓ Parameters to customize the resource model  
  ✓ Event information, including event name, clearing policy, notification policy, messages, key fields, and consolidation rules  
  ✓ Data logging information |
| RM Context       | Context of the resource model described in key=value pair                   |
| Information Model| Provides resource metrics to the decision algorithm. As of the time of this writing, only CIM-M12 is supported |

### 3.3 The CIM-M12 Information Model

CIM stands for Common Information model, and the CIM-M12 information model is currently the only supported class that model resources to be monitored by the autonomic management engine. CIM classes and properties can fully describe system and application resources and CIM methods can be used to specify actions that can be performed on the modeled resource.
The description of the resource is specified in the MOF file, which contains information about which provider needs to be invoked to perform particular operations. Supported operations are specified in Table 8.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Obtain an resource attribute</td>
</tr>
<tr>
<td>ENUM</td>
<td>Enumerate resource instances</td>
</tr>
<tr>
<td>INVOKE</td>
<td>Invoke an instance method or class method on a resource</td>
</tr>
</tbody>
</table>

One of the components of the autonomic management engine is the CIM Object Manager (CIMOM). MOF files, which are automatically compiled, are registered with CIMOM and are used by CIMOM when needed. CIMOM is then responsible for loading the providers which will perform any of the defined GET, ENUM, and INVOKE operations. The interaction between these components is shown in Figure 5.

![Figure 5 CIM-M12 Information Model](image)

### 4.0 Common Base Events

Common Base Events are used to provide events in a structured and standardized manner. It is being used as the standard for reporting events in many enterprise applications. The main application of Common Base Events is to standardize events generated from different software applications from different vendors and to convert these events to a common format which can be understood by the autonomic management engine.

Different kinds of events, including logging, tracking, management, can be mapped to Common Base Events, which are expressed in the XML format. A description of the fields in a Common Base Event is shown in Table 9, with the first three entries as mandatory fields.
Table 9 Description of Fields and SubFields in Common Base Event

<table>
<thead>
<tr>
<th>Field</th>
<th>SubFields</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reportComponentID</td>
<td>location</td>
<td>The identification of the component that is reporting the situation</td>
</tr>
<tr>
<td></td>
<td>locationType</td>
<td></td>
</tr>
<tr>
<td></td>
<td>component</td>
<td></td>
</tr>
<tr>
<td></td>
<td>subcomponent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>componentIDType</td>
<td></td>
</tr>
<tr>
<td></td>
<td>componentType</td>
<td></td>
</tr>
<tr>
<td>sourceComponentID</td>
<td>location</td>
<td>The identification of the component that is affected by the situation</td>
</tr>
<tr>
<td></td>
<td>locationType</td>
<td></td>
</tr>
<tr>
<td></td>
<td>component</td>
<td></td>
</tr>
<tr>
<td></td>
<td>subcomponent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>componentIDType</td>
<td></td>
</tr>
<tr>
<td></td>
<td>componentType</td>
<td></td>
</tr>
<tr>
<td>Situation</td>
<td>situationType</td>
<td>The specifics of the situation itself</td>
</tr>
<tr>
<td></td>
<td>reasoningScope</td>
<td></td>
</tr>
<tr>
<td></td>
<td>categoryName</td>
<td></td>
</tr>
<tr>
<td>msgDataElement</td>
<td>Description</td>
<td>Description of the event</td>
</tr>
<tr>
<td>creationTime</td>
<td>Description</td>
<td>The time of the event generation</td>
</tr>
</tbody>
</table>

4.1 Managed Resource and Manager Touchpoints

The communication between autonomic manager and the managed resources is done through a pair of API – the API of the managed resource is defined by IManagedResourceTouchpoint and the API of the autonomic manager is defined by IAutonomicManagerTouchpointSupport. The interaction of these two interfaces is shown in Figure 6.

![Figure 6 Interaction Between the two Touchpoint Interfaces][6]

Currently touchpoints only support communications on the same machine. In the future, the touchpoints can support remote communications, where they can use Remote Method Invocation (RMI) to register themselves, discover resources and managers, and to invoke remote methods on the discovered touchpoints.
Currently, the only method defined in the `IManagedResourceTouchpoint` interface is `assignManager`, which assigns a manager to the managed resource for it to send events to. The `SendEvent` method is defined in the `IAutonomicManagerTouchpointSupport` interface to handle the incoming Common Base Events. More information can be found in [1] and [6].

### 5.0 Generic Log Adapter

The Generic Log Adapter is a tool provided in the Autonomic Computing Toolkit [3] to convert log messages which may be in text format to Common Base Events format. The generated Common Base Events can then be consumed by the autonomic manager. The Generic Log Adapter is generally used with legacy applications, where it may be more straightforward to convert text messages in log files to Common Base Events than to modify the application to generate log messages directly. Moreover, the Generic Log Adapter can be used in situations where a third-party application is used, where it is not possible to modify the application to generate Common Base Events as the source code is not available. The data flow of an application using the Generic Log Adapter is shown in Figure 7. The conversion rules for each context, which are saved in a configuration file, in the Generic Log Adapter application is shown in Table 10. The configuration file can be created from Eclipse using the Autonomic Computing Toolkit plug-in. A screenshot is provided in Figure 8.

![Figure 7 Data Flow of an Application Using Generic Log Adapter](image)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor</td>
<td>Defines the mechanism in which the log content is to be processed</td>
</tr>
<tr>
<td>Extractor</td>
<td>Provides mechanism to receive lines from sensor and separate the event messages (i.e. defines the rules to recognize message boundaries)</td>
</tr>
<tr>
<td>Parser</td>
<td>Defines a set of string mappings to convert messages received from the extractor to Common Base Events entries</td>
</tr>
<tr>
<td>Formatter</td>
<td>Takes attributes and values from the parser and creates the Common Base Event Java object instance</td>
</tr>
<tr>
<td>Outputter</td>
<td>Wraps the formatted Java object in a suitable format (e.g. XML)</td>
</tr>
</tbody>
</table>
6.0 Log Trace Analyzer

The Log Trace Analyzer is a tool in the Autonomic Computing Toolkit [3] that facilitates the viewing, analysis, and correlation of log files. It is generally used in level 2 autonomic manager applications. With this tool, administrators can analyze correlation between events created by different products, link them and look for possible solutions.

A symptom database, which describes different events and suggested solution to problems, can be used to more effectively deduce a potential mean to solve a problem. The Log Trace Analyzer can be invoked through the Autonomic Computing Toolkit plug-in of Eclipse.

Figure 9 and Figure 10 show two screenshots of the Log Trace Analyzer in Eclipse.

7.0 Samples

This section describes some examples of how different levels of autonomic computing can be achieved on managed resources.
Figure 9 Screenshot of Events view in the Log Trace Analyzer

Figure 10 Screenshot of Correlation view in Log Trace Analyzer
7.1 Level 2 Maturity

Maturity level 2 of autonomic computing can be achieved by using the Generic Log Adapter and Log and Trace Analyzer. The overall structure of this scenario is shown in Figure 11. First, the Generic Log Adapter converts messages in a legacy log file to Common Base Events as per the rules defined in the adapter configuration file. Then, the generated Common Base Events can be analyzed using the Log and Trace Analyzer to determine the characteristics of the events. An optional symptom database can be used to suggest actions for these events.

![Figure 11 Components for Maturity Level 2](image)

7.2 Level 4 Maturity

Level 4 maturity of autonomic computing can be achieved by using the Generic Log Adapter and the Autonomic Management Engine. The Generic Log Adapter converts the log messages in a legacy log file to Common Base Events, which are delivered to the AME hosting environment. The Autonomic Management Engine will analyze the incoming events and perform needed actions as defined in the resource model. The overall structure of such a system is shown in Figure 12. Alternatively, Common Base Events, if possible, could be generated directly by the application, which will bypass the step where the Generic Log Adapter converts log messages to Common Base Events. Moreover, rather than storing the Common Base Events in a log file of its own, it is possible for these events to be transmitted over a network or be stored in a database, if the autonomic manager is not located on the same machine as where the events are generated.
7.3 Building a Resource Model

A plug-in for Eclipse called Resource Model Builder is one of the easiest ways a developer can use to quickly create a resource model.

One of the prerequisite for building a resource model is the MOF file, which specifies the operations that are available to the resource model as well as the attributes that can be extracted from it. It also specifies the classes that are responsible for facilitating the touchpoint and provider functionalities. An example MOF file for the FolderMonitor sample application is shown in Listing 1. For complete information on the FolderMonitor resource model, please refer to [4].
Listing 1 FolderMonitor MOF File

```plaintext
[  
  Description ("FileSystem Folder info"),  
  provider("com.ibm.amw.touchpoint.cimom.ifc.M12JavaProvider"),  
  M12_Instrumentation  
  "Java.com.ibm.amw.sample.ilt.foldermonitor.MyFolder || INVOKE")  
]

class Folder  
  {
    [Description("Path of the parent folder"), key]
      string parentFolder;

    [Description("The name of this folder"), key]
      string name;

    [Description("The number of files/folders this folder contains.")]
      sint32 numberOfObjects;

    sint32 ZipOldestNFiles([IN] sint32 N);
  }
```

This resource model MOF file specifies the provider and instrumentation classes. The Folder class also contains three attributes and one method, which takes in one parameter.

The Basic Resource Model Wizard will guide the user through a series of step to specify the necessary parameters of the resource model. The steps for filling in the parameters of the resource model can be summarized in Table 11.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a triggering condition based on one or more of the attributes of the resource model. The threshold value and the severity level for the event triggering are specified here as well.</td>
</tr>
<tr>
<td>2</td>
<td>Select properties to log. Choose which of the attributes of the resource model to log in the log file.</td>
</tr>
<tr>
<td>3</td>
<td>Add a response action to the events. Specify the corrective measure that is to be carried out when an event is triggered. This action can be the CIM method that is specified in the resource model definition.</td>
</tr>
<tr>
<td>4</td>
<td>Add any parameters associated with the resource model. For example, the parameters that are required for the example FolderMonitor resource model are the folders that it is going to monitor.</td>
</tr>
<tr>
<td>5</td>
<td>Add any dependencies. Some dependencies include the MOF files, the associated JAR or class files, and any external scripts or executable programs.</td>
</tr>
</tbody>
</table>

A screenshot of the Events Editor and the Actions dialog box is shown in Figure 13.
After specifying all the parameters of the resource model, the next step is to add the brain of the resource model. The control of the resource model is specified in a script file called decision tree script, which is written in JavaScript or VBScript. The decision tree script contains the necessary steps to initialize the resource model, and the actions that are to be carried out at each interval.

The initialization script is shown in Listing 2. It simply checks the number of folders to be monitored and then associate the parameter “Folders” to “Folder”.

The `VisitTree` function contains the script that is run at every cycle. An excerpt of the `VisitTree` function from the FolderMonitor class is shown in Listing 3. In essence, the `VisitTree` function carries out the control loop as specified in Figure 3. It enumerates through all the folders and obtains the number of files in that directory. If the number of files exceeds upper bound threshold specified in the resource model, the event `Ev_Folder_numberOfObjects_too_high` is thrown, and it will be up to the autonomic management engine to execute the correction action associated with this event.
Listing 2 Decision Tree Script Init Function

```javascript
function Init(Svc)
{
    Svc.Trace(TRACE_FINEST, TRACE_SOURCE + "Start Init");
    var dimension = Svc.GetStrParameterCount("Folders");
    if (dimension > 0 )
    {
        Svc.AssociateParameterToClass("Folders", "Folder");
    } else
    {
        Svc.Trace (0, "No Folders to be monitored specified here");
        return(1);
    }
    Svc.Trace(TRACE_FINEST, TRACE_SOURCE + "EXIT: Init");
    return (0);
}
```

Listing 3 Decision Tree Script VisitTree Function

```javascript
function VisitTree(Svc)
{
    for(idx = 0; idx < Svc.GetNumOfInst("Folder"); idx++)
    {
        Svc.RemoveMapAll(hPropTable);
        curFoldernumberOfObjects = Svc.GetNumProperty("Folder", idx, "numberOfObjects");
        Svc.SetMapNumElement(hPropTable, "numberOfObjects", curFoldernumberOfObjects);
        curFoldername = Svc.GetStrProperty("Folder", idx, "name");
        Svc.SetMapStrElement(hPropTable, "name", curFoldername);
        curFolderparentFolder = Svc.GetStrProperty("Folder", idx, "parentFolder");
        Svc.SetMapStrElement(hPropTable, "parentFolder", curFolderparentFolder);
        if (curFoldernumberOfObjects > Svc.GetThreshold("Thr_Folder_numberOfObjects_gt"))
        {
            Svc.SetMapNumElement(hPropTable, "UpperBound", Svc.GetThreshold("Thr_Folder_numberOfObjects_gt");
            Svc.SendEventEx("Ev_Folder_numberOfObjects_too_high", hPropTable);
        }
        Svc.LogInstEx ("Folder_Availability","Folder", hPropTable);
    }
    return (0);
}
```
7.3.1 Deploying a Resource Model

After creating the resource model, it can be deployed using either one of two methods: through a command-line front end application called Simple Agent Reference Application (SARA), or through Java classes which facilitates the same functionality as provided in SARA.

The sequence of commands that are generally executed using SARA is listed in Table 12. After installing a previously created resource model, an instance of that model must be created, and this instance will carry out the functionalities as specified in the resource model. For complete information, please refer to “Create a Resource Model”.

Table 12 SARA Commands

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startrme</td>
<td>Starts the resource model engine</td>
</tr>
<tr>
<td>instrmtype</td>
<td>Install a resource model</td>
</tr>
<tr>
<td>mkrminstance</td>
<td>Create an instance of a previously installed resource model</td>
</tr>
<tr>
<td>startrminstance</td>
<td>Start a previously created resource instance</td>
</tr>
<tr>
<td>lsrminstances</td>
<td>List the conditions of a resource instance</td>
</tr>
<tr>
<td>stoprminstance</td>
<td>Terminate a previously running resource instance</td>
</tr>
<tr>
<td>rmrminstance</td>
<td>Remove a previously created resource instance</td>
</tr>
<tr>
<td>uninstrmtype</td>
<td>Uninstall a previously installed resource model</td>
</tr>
</tbody>
</table>

Alternatively, it is possible to write Java code to carry out the above operations automatically. The `com.ibm.amw.rme` package provides a number of classes that can be used to programmatically interact with resource models. Listings 4 to 7 show snippets of Java code that carry out the respective functionalities provided in SARA.

Listing 4 Install Resource Model Java Code

```java
public void installRMType(String rmLocation, boolean replace_existing) throws MalformedRMPackageException, RMTypeAlreadyPresentException, java.io.IOException, SetPermissionsOnFileException, FileExistsException, InvalidRMDescriptorException, InvalidRMBundleException {
    checkStarted();
    ZipInputStream z = null;

    try {
        TutorialRME rme = TutorialRME.getInstance();
        z = new ZipInputStream(new FileInputStream(rmLocation));
        rmBundleManager.installRMType(z, replace_existing);

        try{z.close(); } catch(Exception e){}
    }
    catch (Exception e) {
        if (z!=null){try{z.close();}catch(Exception e){}}
        e.printStackTrace();
    }
}
```
Listing 5 Create Resource Model Instance Java Code

```java
public void createRMInstance(String rmName, String rmType, String rmCtx, String rmDescriptor)
    throws RMAlreadyPresentException, RMOperationFailureException, RMTypeNotFoundException, ContextNotFoundException, RMNotFoundException, InvalidRMDescriptorException
{
    checkStarted();
    RMIdentifierImpl id = new RMIdentifierImpl(rmName, rmType, rmCtx);
    rmManager.create(id, rmCtx);
    if (rmDescriptor==null)
    {
        rmDescriptor = rmPackageManager.getRMDescriptor(rmType);
        if (rmDescriptor == null)
        {
            rmManager.delete(rmid);
            throw new RMTypeNotFoundException("Resource model type '" + rmType + '" not found.");
        }
    }
    boolean done = false;
    try
    {
        rmManager.load(id, rmDescriptor);
        done = true;
    }
    finally
    {
        if (!done)
        {
            rmManager.delete(id);
        }
    }
}
```

Listing 6 Start Resource Model Instance Java Code

```java
public void startRMInstance(String rmName)
    throws RMNotFoundException, RMOperationFailureException
{
    checkStarted();
    RMIdentifierPatternImpl thisrm = new RMIdentifierPatternImpl(rmName, "+", "+");
    RMIdentifier[] rmids = rmManager.getIdentifiers(thisrm);
    // There must be only one rminstance
    if (rmids!=null && rmids.length == 1)
    {
        rmManager.start(rmids[0]);
    }
    else
    {
        throw new RMNotFoundException(new RMIdentifierImpl(rmName, "-", "-"));
    }
}
**Listing 7 Stop Resource Model Instance Java Code**

```java
public void stopRMInstance(String rmName)
    throws RMNotFoundException, RMOperationFailureException
{
    checkStarted();
    RMIdentifierPatternImpl thisrm = new
            RMIdentifierPatternImpl(rmName, "\*", "\*"); 

    RMIdentifier[] rmids = rmManager.getIdentifiers(thisrm);
    // There must be only one rminstance
    if ( rmids!=null && rmids.length == 1 )
    {
        rmManager.stop(rmids[0]);
    }
    else
    {
        throw new RMNotFoundException(new RMIdentifierImpl(rmName,
            "-", ";-"));
    }
}
```

### 8.0 Problem Determination Scenario

The Problem Determination Scenario is a sample in the Autonomic Computing Toolkit that represents a simple self-managing system that uses an intelligent control loop to collect system information, from which the autonomic manager analyzes and plans the appropriate response actions, and subsequently makes necessary adjustments to resolve the problems. It facilitates the self-healing property of autonomic computing, and is an example of an autonomic system of maturity level 4. For complete information, please refer to [8].

The interaction between each component of the scenario is shown in Figure 14, with a more detailed picture of each component shown in Figure 15.

**Figure 14 Interactions of the Problem Determination Scenario [8]**

The problem determination scenario works as follow: a web application is setup to run on WebSphere Application Server and continuously read the sample data from the Cloudscape database. As the web application queries for database information, the products output event information to their logs. Since the products used do not output log file data in that format, the scenario utilizes the Generic Log Adapter to provide the conversion. The autonomic manager consumes the Common Base Event data and analyzes the log data using a dedicated resource model, looking for specific events. After an event has been detected, the autonomic manager issues corrective action on the appropriate managed resource.
The self-healing actions of the system are shown in Figure 16, with the description of each step summarized in Table 13. A partial decision tree script of the scenario is shown in Listing 8.
### Table 13 Self-Healing Steps of Problem Determination Scenario

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The Cloudscape database is shutdown, and the communication link between the web application and the database fails</td>
</tr>
<tr>
<td>1</td>
<td>The web application logs the error in its log file</td>
</tr>
<tr>
<td>2</td>
<td>Generic Log Adapter picks up the log message</td>
</tr>
<tr>
<td>3</td>
<td>Generic Log Adapter converts the log message to Common Base Event</td>
</tr>
<tr>
<td>4</td>
<td>Autonomic Management Engine picks up the Common Base Event and detects the condition</td>
</tr>
<tr>
<td>5</td>
<td>Autonomic Management Engine restarts Cloudscape database which fixes the error condition</td>
</tr>
</tbody>
</table>

### Listing 8 Decision Tree Script of Problem Determination Scenario

```javascript
var str_situation = Svc.GetStrProperty("CSF", idx, "str_CBERecord");
var str_situationcategoryName = getParmValue(str_situation, "categoryName");
var str_situationDisposition = getParmValue(str_situation, "situationDisposition");
if ((str_situationcategoryName == str_CONNECT) &&
    (str_situationDisposition == str_CLOSED))
{
    if (bool_connect && (str_situationcategoryName == str_CONNECT) &&
        (str_msgId == str_CONN_ERR_1))
    {
        if (Svc.ExistsMapElement(tmpMap, str_conn_errorRecovery))
        {
            continue;
        }
        Svc.SetMapStrElement(tmpMap, str_conn_errorRecovery, str_dlr_ON);
        shellRc = Svc.ShellCmd(stopScript);
        Svc.DetachedShellCmd(StartNetworkServer);
        shellRc = Svc.DetachedShellCmd(startScript);
    }
}
```

### 9.0 Solution Installation and Deployment Scenario

The Solution Installation and Deployment Scenario is a sample in the Autonomic Computing Toolkit that demonstrates the self-configuring aspect of Autonomic Computing. The goal of the solution installation and deployment technology is to enable software solutions to be deployed and supported with reduced human interaction. Such goal is carried out by creating a set of common services, as summarized in Table 14. The components of the Solution Installation and Deployment Technology are described in Table 15. For complete information, please refer to [9].
<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infrastructure</td>
<td>Common infrastructure to ensure a simpler and more consistent deployment experience</td>
</tr>
<tr>
<td>Packaging</td>
<td>Common packaging to minimize the number of software instances installed within an enterprise</td>
</tr>
<tr>
<td>Descriptors</td>
<td>Common descriptors to describe the installation capabilities and dependency requirements for a given software package</td>
</tr>
<tr>
<td>Tools</td>
<td>Common tools to reduce the cost and complexity of building, deploying, and maintaining software solutions</td>
</tr>
<tr>
<td>Dependency Checking</td>
<td>Common dependency-checking technologies address the major configuration and change management concerns</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installable Unit (IU)</td>
<td>Any software entity that can be deployed to an IT system. Depending on the architecture of the IU, it contains one deployment descriptor and one or more artifacts to be deployed. The deployment descriptor is an XML file that describes the content of an IU, its checks, and its dependencies. An artifact is a deployable file that defines the required actions to install an IU to a managed resource</td>
</tr>
<tr>
<td>Root IU</td>
<td>The smallest unit of packaging</td>
</tr>
<tr>
<td>Features</td>
<td>Provides the mechanism that enables a user to select IUs to deploy from within the root IU. Each feature corresponds to an IU, which includes an artifact and descriptor.</td>
</tr>
</tbody>
</table>
| Change Manager          | Taking the deployment descriptor as input, the change manager coordinates change management operations across the appropriate hosting environments, utilizing other run-time components. The change manager coordinates change requests and tracks their progress and success. The change manager also provides the following functionalities:  
  ➢ Determines which IU must have change management applied, based on descriptor  
  ➢ Validate dependencies of target environment  
  ➢ Determine order IUs should be applied  
  ➢ Execute the change through touchpoints  
  ➢ Register change with installation DB (registry)  
  ➢ Track progress. Revert to stable state if aborted |
| Dependency Checker      | Before installing any artifact, the dependency checker uses this information to determine whether the dependencies are met by the hosting environment. Dependencies include the version of the software required on the local machine, the capacity of cards, the amount of available disk space, and the hardware configuration and operating system required on the local machine. |
10.0 On-Demand

IBM’s definition of an on-demand environment is as follows: “An on demand operating environment unlocks the value within the IT infrastructure to be applied to solving business problems. It is an integrated platform, based on open standards, that enables rapid deployment and integration of business applications and processes. Combined with an environment that allows true virtualization and automation of the infrastructure, it enables delivery of IT capability on demand.” [7]

The three features that represent an on-demand operating environment are shown in Table 16 and pictorially in Figure 17. A more detailed breakdown of each of the three features is shown in Figure 18.
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integration</td>
<td>Provides the facilities to gain a unified view of processes, people, information, and systems</td>
</tr>
<tr>
<td>Virtualization</td>
<td>Simplifies deployment and improves use of computing resources by hiding the details of the underlying hardware and system software, allowing for consolidation and the ability to adapt to changing demand</td>
</tr>
<tr>
<td>Automation</td>
<td>Overcomes the complexity of systems management to enable better use of assets, improved availability and resiliency, and reduce costs based on business policy and objectives.</td>
</tr>
</tbody>
</table>

### 10.1 Provisioning

IBM’s vision of provisioning is as follows: “Provisioning is the end-to-end capability to automatically deploy and dynamically optimize resources in response to business objectives in heterogeneous environments. Provisioning helps to respond to changing business conditions by enabling the ability to dynamically allocate resources to the processes that most need them, as driven by business policies. Provisioning of individual elements, such as identities, storage, servers, applications, operating systems, and middleware, is a critical step to being able to then orchestrate the entire environment to respond to business needs on demand.” [7]

### 10.2 Policy-Based Orchestration

IBM’s definition of Policy-Based Orchestration: “Policy-based orchestration is about providing an end-to-end IT service that is dynamically linked to business policies, allowing the ability to adapt to changing business conditions.” [7]

According to IBM [7], orchestration should be able to provide:

- Orchestration allows the manipulation of the IT environment in real time, according to defined business policies, to achieve desired business goals.
- Orchestration senses changes in demand for resources and automatically triggers action to re-allocate resources (such as hardware, software, and applications) accordingly throughout the entire system.
- Orchestration introduces a level of automation that can evolve to act as the intelligence across security, availability, provisioning, and optimization, ensuring that the entire IT environment is aligned to business goals.

And some key advantages of policy-based orchestration include:

- Software product that adapts to existing hardware, software, architecture and processes
- Packaged Intelligence using well-defined policies, workflows, and scripts that capture best practices for execution of repeatable tasks
- Autonomic technology that senses and responds to changes
- Flexibility to adapt at a customer’s own pace
11.0 ITITO and ITPM

The IBM Tivoli Intelligent ThinkDynamic Orchestrator (ITITO) and the IBM Tivoli Provisioning Manager (ITPM) are two IBM products that are designed to be automated resource management solutions for data centers.

IBM Tivoli Intelligent ThinkDynamic Orchestrator proactively configures resources among applications in a multi-application environment to balance end-user traffic demands, excess capacity, and the obligations of service level objectives. The IBM Tivoli Intelligent ThinkDynamic Orchestrator automates three key data center processes: infrastructure provisioning, capacity management, and service level management. A comparison between the two products is shown in Table 17.

<table>
<thead>
<tr>
<th></th>
<th>ITITO</th>
<th>ITPM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orchestrates automation</td>
<td>Coordinates provisioning</td>
<td></td>
</tr>
<tr>
<td>Determines why, when and where</td>
<td>Executes what and how</td>
<td></td>
</tr>
<tr>
<td>Senses why to take action, anticipates when to start provisioning, and prioritizes where to put resources.</td>
<td>Coordinates what data center resources are provisioned and executes “personalized” how.</td>
<td></td>
</tr>
<tr>
<td>Benefits:</td>
<td>Reduced hardware and software capital costs</td>
<td>Controlled labor costs by allowing execution of IT tasks in a repetitive, error-free manner</td>
</tr>
<tr>
<td></td>
<td>Improved alignment between business priorities and IT resources</td>
<td>Automated best practices of the company</td>
</tr>
<tr>
<td></td>
<td>Increased resource utilization</td>
<td>Investment protection by working with your existing hardware, software, and network devices</td>
</tr>
<tr>
<td></td>
<td>Improved speed of deployment</td>
<td>Improved server-to-administrator ratio</td>
</tr>
<tr>
<td></td>
<td>Heightened service level delivery</td>
<td></td>
</tr>
</tbody>
</table>

11.1 ITITO Architecture

The general architecture of IBM Tivoli Intelligent ThinkDynamic Orchestrator is shown in Figure 19. Each of the components in the architecture overview will be further discussed in this section. For even more details, please refer to [7].

11.1.1 Data Acquisition Engine

The Data Acquisition Engine is responsible for acquiring and preprocessing performance metric data from each managed application environment. Data can be captured and filtered from the operating system and infrastructure layers. The Data acquisition Engine can be composed of the components shown Figure 20 and described in Table 18.
Table 18 Components of Data Acquisition Engine

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Layer</td>
<td>Gathers information from the application layer through the application layer interface. (e.g. SQL transaction metrics)</td>
</tr>
<tr>
<td>Operating System</td>
<td>Gathers information relating to the operating system layer through the server infrastructure interface. (e.g. I/O utilization)</td>
</tr>
<tr>
<td>Server Infrastructure</td>
<td>Gathers information related to the server infrastructure layer through the server infrastructure interface. (e.g. Resource metrics)</td>
</tr>
<tr>
<td>Network Infrastructure</td>
<td>Gathers information relating to networking infrastructure within the data center. (e.g. Load Balancer metrics)</td>
</tr>
</tbody>
</table>

11.1.2 Application Controller

Based on the application’s workload model and predictions, as well as on real-time performance data acquired from the Data Acquisition Engine, this component determines the resource requirements of the application. This component provides information to the decision-making task. It consists of three components shown in Figure 21 and described in Table 19.
Table 19 Components of Application Controller

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prediction</td>
<td>The prediction component receives demand information from the Data Acquisition Engine through the monitoring data interface and predicts the future demand for system resources. The Autocorrelation Subsystem detects non-randomness in the demand information received. The Time-varying trends detection subsystem receives the demand information and the autocorrelation function from the autocorrelation subsystem. The Time-varying trends removal subsystem removes the time-varying components from the demand information received from the time-varying trends detection subsystem.</td>
</tr>
<tr>
<td>Workload Modeling</td>
<td>Estimates each application environment’s response to incoming traffic.</td>
</tr>
<tr>
<td>Classification</td>
<td>Determine how each application environment meets the predetermined level of service and what changes should be made to maintain it, by uses the information obtained from the Workload modeling component.</td>
</tr>
</tbody>
</table>
11.1.3 Deployment Engine

The Deployment Engine is responsible for the creation, storage, and execution of repeatable workflows that automate the configuration and allocation of assets. It processes requests by sequentially executing workflow transitions or steps. The components of Deployment Engine are shown in Figure 22 and described in Table 20.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workflow Assembly</td>
<td>Receives a request or command and coordinates the translation of the deployment request into an executable workflow.</td>
</tr>
<tr>
<td>Workflow Execution</td>
<td>Receives the workflow for execution and determines which asset is pertinent to each transition.</td>
</tr>
</tbody>
</table>

11.1.4 Global Resource Manager

The Global Resource Manager receives requirements for servers or network devices from all Application Controllers and manages the overall optimization of data center assets. It is the job of the Global Resource Manager to make optimal resource allocation decisions and to ensure a stable control over the application infrastructure. Its components are shown in Figure 23 and described in Table 21.
Table 21 Components of Global Resource Manager

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Broker</td>
<td>Manages the resource pools, and attempts to service requests made from the Application Controllers.</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Uses component information compiled by the Application Controllers to assist in determining a balance between separate application environments that are competing for the same resources.</td>
</tr>
<tr>
<td>Stabilizer</td>
<td>Analyzes and filters the server change requests from the Optimization component to determine whether similar changes were recently implemented.</td>
</tr>
<tr>
<td>Resource Pool Manager</td>
<td>Builds a server according to the specification of the resource pool definition to hasten the deployment of a resource into active duty.</td>
</tr>
</tbody>
</table>

11.2 Workflows

A workflow is a representation of the set of steps that must be followed in a data center environment in order to carry out a provisioning operation. It is an automation of an existing manual process or a step of a process. An important point to note is that if there is no manual process, there is no automated process. If the manual process is poorly designed, automated process will be flawed, and it essentially defeats the purpose of having the autonomic computing system.

Individual steps of a workflow are called transitions. They can take on either one of four forms: logical operations, simple command, Java plug-in, or nested workflows. Example workflows are shown in Figure 24. For more information, please refer to [7] and [11].

12.0 Orchestration and Provisioning Library

The Orchestration and Provisioning Library (OAPL) is an open-source initiative that attempts to pull together tools, best practices, education, validation testing and promotional activities to help third parties exploit the workflow engine within ITITO and ITPM. It is also an excellent directive to help other vendors to develop workflows for their products to be included in the package as well. Several IBM contributions to the OAPL include IBM Global Services, pSeries, iSeries, zSeries, xSeries, DB2, Lotus, Rational and WebSphere, and third party contributions include Cisco Systems, Citrix, DataSynapse, Inkra Networks, PeopleSoft, Nortel, Network Associates, Siebel Systems, VMware, Cap Gemini Ernst & Young, and many more. For more information, please refer to [13].
### 13.0 Success Stories

This section discusses some success stories where companies have benefited from adopting autonomic computing and either ITITO or ITPM. Detailed descriptions of each case as well as other success stories not mentioned in this document can be found in [12].

#### 13.1 STAR Technologies, UK

STAR Technologies is a provider of managed technology services to mid-sized businesses and public sector organizations in the UK. Its services needed to be highly responsive to changing customer demand, and it anticipated that customers would want a high degree of flexibility and the ability to sense and respond to changing business needs.

STAR Technologies was facing the inflexibility of a typical mid-size organization. Its IT infrastructure consists of a number of stand-alone servers, each with its own direct-attached storage volumes, where their utilization is low, and spare capacity cannot easily be shared between different systems. However, by adopting the on-demand environment, resources can be shared flexibly and automatically between different users and systems, enabling the business to respond rapidly to new demands.

As a result, STAR Technologies were able to achieve lower administrative and infrastructure costs in storage, and 85% efficiency increase in server provisioning; thus enabling more competitive pricing, and be able to offer multiple price-performance options to customers. Furthermore, they were able to manage complex storage environment from a single point of control as well as to assign automatically new storage capacity and processing power in response to increased demand, enabling customers to respond more rapidly to new business opportunities.
13.2 Galeries Lafayette Group

The Galeries Lafayette Group is a French department store company. Their IT Systems, incorporating mainframe, UNIX and Intel technologies, had been acquired as applications were needed, with little thought about interconnectivity. As a result, it hindered application deployment, performance optimization, backup and recovery and system security. Moreover, systems management tasks took time and personnel away from efforts to support Galeries Lafayette Group’s initiatives.

They wanted to support a strategy to meet the changing requirements of Galeries Lafayette Group more efficiently and cost-effectively, one which can incorporate the group’s highly variable sales cycle, which demanded greater system resources during holidays and clearance sales, as well as to accommodate the existing infrastructure while providing short-term enhancements and long-term improvements.

With the move to autonomic computing, it helped Galeries Lafayette Group to respond more quickly to changing industry conditions, to boost operational efficiency and to become more resilient by preventing disruptions. It identified both short-term enhancements and long-term directions, including an anticipated move to on-demand business. As a result, they achieved a 15 percent in savings within two years based on savings from increased efficiencies.

13.3 eBay

Current data shows that eBay serves more than 104 million registered members from around the world with one new member joining nearly every second. More than US $23 billion in goods and services changed hands through eBay in 2003. Never knowing what a seller may list and how much interest there may be, eBay needs to dynamically adjust its services to support rapidly changing customer demand.

With such a high performance demand, manual process is simply not feasible. On the other hand, autonomic concepts can automate monitoring of the operating environment, identifies issues that could cause service problems and leverages autonomic, self-healing capabilities to recover from service issues automatically. They can enable staff to identify spikes in interest for a specific item or category and dynamically adjust its system resources to support the increased usage, and to replicate changes across its website quickly and consistently so that new services and features are available faster.

IBM services and Tivoli best practices helped eBay deploy its new business service management solution in under 6 months. Moreover, they have automated system-level monitoring across all its platforms - Microsoft Windows, Sun Solaris and Linux - in just 60 days, and reduced application downtime, cutting the time it takes to remove and reactivate a system from 20 minutes to just 4 minutes.

14.0 Similar Products

Products that serve similar functionalities of ITITO and ITPM will be considered in this section.
14.1 HP Adaptive Enterprise

Like ITITO and ITPM, the Adaptive Enterprise from Hewlett-Packard focus on aligning IT resource allocation with changing business needs. However, it places a bigger emphasis on the consultative component. Its aim is to create a more service-oriented architecture. One of the biggest differences between HP Adaptive Enterprise and ITITO and ITPM is that the former has put in a richer investment in network management, where two of the Enterprise’s components, OpenView and Internet Usage Manager, are designed to serve those purposes.

More information can be found at [14].

14.2 Microsoft Dynamic Systems Initiative (DSI)

Microsoft declares its Dynamic Systems Initiative as an application life-cycle methodology “moving towards autonomic Computing”. It attempts to “look across the life cycle of an application and capture knowledge of what it takes to run a complex system”. Its design is such that humans manage by exception, while the system does all the day-to-day work. At the time of this writing, DSI is for Windows only, and as a result, it cannot be totally considered as an enterprise solution.

For more information, please refer to [15].

15.0 IBM’s Strengths and Challenges

One of the biggest strengths of IBM in the field of autonomic computing is that its self-managing concept has been around the longest. It has foreseen the need for what is now coined autonomic computing the earliest, and thus has a decent head start on the research in this area over many of its competitors. With the increasing complexity of software, the growing variety in hardware, and the limited number of IT professionals in a company, the need for such autonomic systems are becoming more pressing. And a lot of such demand is going back to IBM.

Secondly, IBM has the luxury that it is capable of building the technology right down to the chip and the fabrication level. For example, IBM has more than 400 features in more than 50 of its hardware and software products that provide autonomic capabilities. With full control over the entire pipeline from hardware and software, IBM has to power to change anything in any stage as they see fit. This flexibility turns out to be a huge advantage for IBM.

Moreover, with the open approach, IBM can actively participate in setting the industry standards. It can potentially steer the standard settings effort to more suit their directive. Through the Orchestration and Provisioning Library, not only does it encourage third parties to contribute in their initiative and to make more products from different vendors interoperate, it also helps to promote its autonomic computing technology to the world. More people can be using it, since many vendors do support such initiative, thus making the product more well-known.
Nonetheless, there are also several challenges that IBM faces. Autonomic computing is not yet for everyone. One of the biggest challenges is that for autonomic computing to work, it must have a solid manual process that it can automate. It must have a founded business policy that it can orchestrate. There is no guarantee that business policies at one company would apply to another company, or that one business policy might be the right one for a company. Such processes and policies are sometimes difficult to facilitate, thus, there are many cases where autonomic computing is not fully ripe.

Finally, one of the issues with the field of autonomic computing is how much it can be trusted. The basic model behind autonomic computing is not difficult to understand, but can system administrators trust that the system can handle situations properly, especially in exceptional situations. This is an issue even beyond the field of autonomic computing, as sometimes it may not seem that the technology is at the stage where system administrators can leave machines making decisions on its own. Nonetheless, this technology would only improve over time, and IBM has certainly been contributing a huge part in this technological push.
References