Testing Tools: Landscape and Overview

CML00088-01

Code Magus Limited (England reg. no. 4024745)
Number 6, 69 Woodstock Road
Oxford, OX2 6EY, United Kingdom
www.codemagus.com

December 15, 2020

Contents

List of Figures

List of Tables

1 Introduction

This document sketches out areas in which the Code Magus test tools operate, and describes the landscape of the tools. The discussion covers the requirements of functional and non-functional testing, including the components that deal with data and preconditions around this style of testing. Functional testing includes coverage of tools that perform both on-line (as in Eresia) and offline (as in *Verify*) testing.

2 Modular Design

The Code Magus tools are modular in architecture and construction. As a result of this there is a considerable degree of orthogonality in the design which leads to significant reuse in components. There are a number of components which are common across all types of testing and data tools. When a tool requires the services of one of these components it gains access to that function through an API. There could be many implementations of these API's each providing the required function in a manner dictated by the circumstances, but largely independent of the details within the caller of the API. For example, there is an API designed to give the functional and non-functional tools access to a communications transport layer. In most cases, the specific transport layer has no bearing on the user of the API. Another example is reading and writing data by the scripts and tools. In this case, the actual access method has no bearing on the test script details except at the point of binding to the access method through the API.

The success of these API's and this design is highlighted by the fact that the bindings can be done at execution time (of the tools and scripts), and that from the point of view of the tool or script using the interface, the behaviour is independent of the specific bindings (provided that they have been specified correctly and appropriately).

There is an API which is responsible for presenting a generic network interface to the tools and scripts, and details of this API are not dependant on the exact transport layer made available through an electable module, thus the means by which, for example, MQ messages are sent and received or general TCP/IP messages are sent and received is achieved through the same API. In the case of the API used to send and receive messages, the implementation modules of the API are called *Network Control Programs (NCP)*.

The interface through which all the tools and scripts read and write record streams to files and read the result sets of queries is also all done through an API. The binding to the actual module used for reading and writing files, and any parameters the module might require are specified to the API by means of an open specification string. The module that implements the API for a specific set of circumstances is called an *Access Method*.

There are other elements of the design which portray a modular design. The script engine we employ for functional testing allows solutions to be built from various modules and components. Typically, execution starts by driving a `package` script, responsible for orchestrating the execution of various `usecase` scripts in order to complete a facet of functional testing. These script artefacts can reside in different structures, are bound to upon start of execution and are located by means of an internal machine independent naming scheme, together with an external machine dependant binding scheme. This allows scripts that are developed on one platform within one context to be used on another platform in a different context.

In addition, there are API's which allow custom code or third party code to be bound to and called from within the scripts, and the resultant components form part of the same naming and binding scheme that the other script components use.

In all cases, where it might be beneficial for the customer or a third party to supply implementations of these interfaces, the API details and sample code is supplied.

3 Functional Testing

To set the scene in which the Code Magus functional tools landscape can be described, it is useful to consider the nature of the activities required to accomplish this testing.

3.1 Preconditions

For each test scenario, there is a requirement on the conditions of (a portion of) the state data prior to the execution of a test scenario. This requirement could be, for example, that an account within a specific product system must reflect a specific product code, must reflect a specific status code and the accounts need to have a balance in excess of some amount. When required, the precondition is expressed as an expression over the meta-data symbols of the data of the system.

3.2 Test Execution

Execution of the test scenario involves the composition of a transaction, or series of transactions, that reference the elements of the entity, or entities, satisfying the preconditions for that scenario, together with scenario specific data elements. For us, a scenario is a test unit designed to demonstrate, possibly with other scenarios, that one, or some, of the requirements of the system under test has been met.

3.3 Validation and Post-Conditions

The final point of our generic test is the establishment of whether the post-condition is true or not. The post-condition is expressed as an expression over the state of the system post-execution of the list and over the symbols of the meta-data over any channel that responded to the transaction request. When the expression is evaluated, these symbols then refer to the actual data of the elements of the entities in the system, transacted against during the execution of the scenario, as well as the actual data elements that the system responded with. Examples of channels could be a web browser interface, web services, MQ messages, with content implemented in various formats, X.25 links, TCP/IP, UDP/IP, SNA, etc. The format of the content of the messages, which is independent of the transport mechanism, could be expressed in any number of formats including SOAP, XML in general, HTTP; all with or without authentication schemes (such as Kerberos), with or without encryption schemes at the element level, with or without element signatures, etc. Outputs are not restricted to network based channels, but include REPORTS and files.

Whether or not a test scenario is executed as an atomic unit or not depends on various factors.

A test scenario can be atomically executed under the following conditions:

- That the precondition is known to be true of the entities that the test will transact against. Knowing that the precondition is true, could be as a result of a separate exercise which sources elements satisfying the preconditions, or it could be as a result of an interaction with the system by the test pack prior to the test pack executing the test scenario (for example, topping up an account to make sure that the balance satisfies a subsequent test transaction).
- That the effect of the transactions on the system synchronously executes all processing required by that scenario, and updates all state data prior to the system responding to the scenario transactions request, or at least within a small and bounded amount of time following the transaction (the actual bound would need to be established by the requirements).
- That there is a mechanism, for the sake of test validation, for the test scenario to access those elements of the application data required to evaluate the post-condition expression.

The execution of a test scenario cannot be executed atomically in certain conditions. Under these conditions, the full execution of the test scenario is performed in various stages. Test scenario executions cannot be performed atomically when, amongst others, the following conditions arise:

- If the establishment of the preconditions cannot immediately be done or determined as part of the execution of the test scenario.
- When the processing triggered by the test scenario transaction does not finish within a small, bounded amount of time (for example, when it is recognised that some deferred or batch processing is required).
- When certain updates, effects, or outputs, are made or sent on a channel that are not immediately available to the test scenario execution, and these outputs, updates, effects etc, contain elements recognised for the evaluation of the post-condition.
- When the channel that one needs to interact with the system is a one-way channel without immediate feedback. For example, a gateway when the state being updated cannot immediately be seen by the test script executing the scenario. Alternatively, when the input channel is a file, and subsequent processing is required to process the file, defining a response file or access to the updated state of the system.

Without indicating whether or not an interaction with the system is synchronous or not, the manner in which preconditions are determined or arranged, and the manner in which post-conditions are determined or established; or whether a transaction scenario is executed in an atomic step or over various stages; or whether transaction scenarios are grouped together or executed individually. Figure 1 represents the process sketched out here generically.

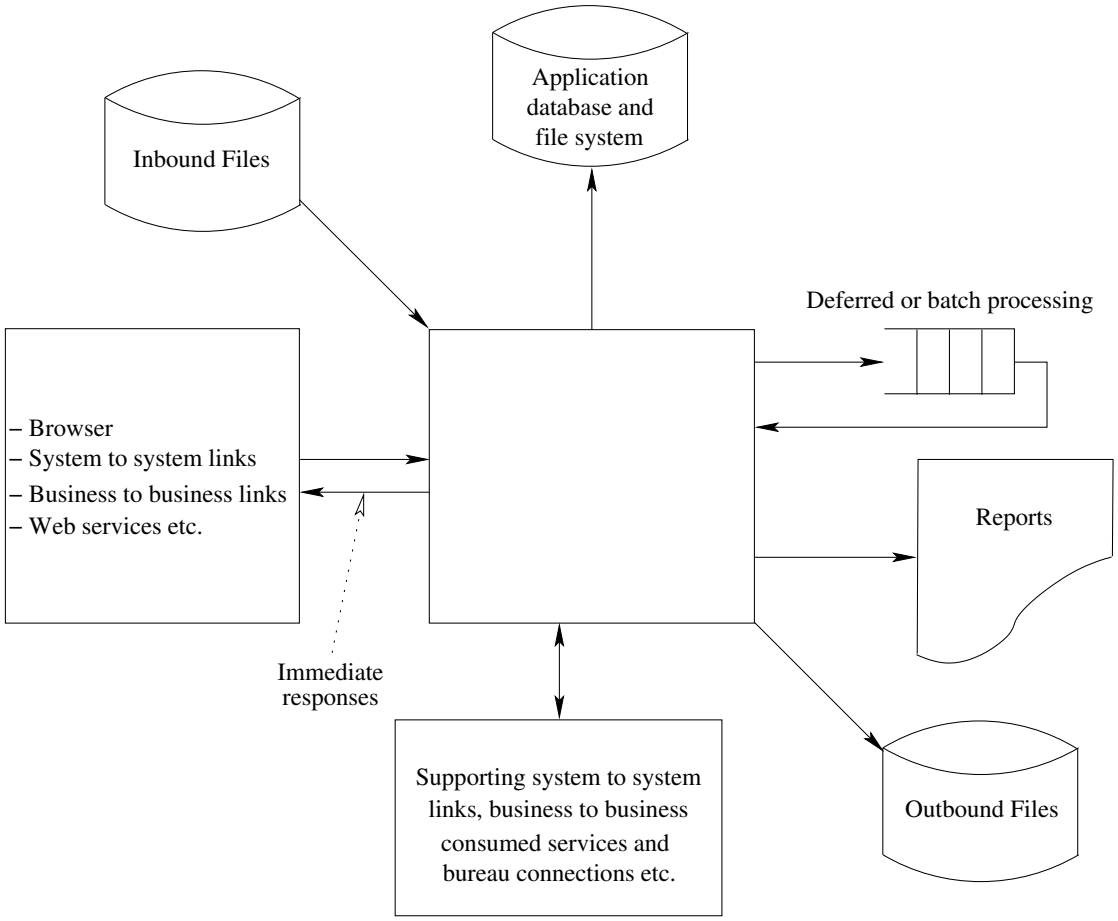

Figure 1: Schematic of Generic System Under Test Detailing Points of Interaction

While the complexity of all the available options could be quite daunting, even in the generic cases, we can abstract the system under test and represent it schematically as depicted in Figure 2.

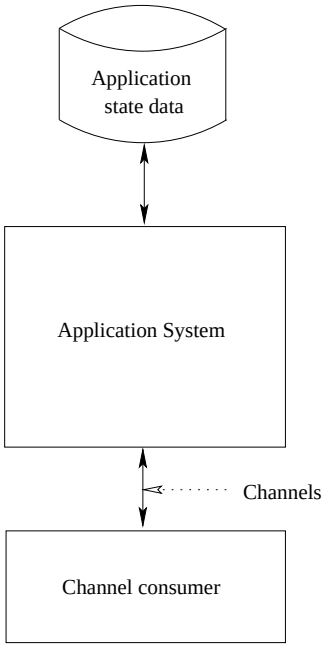

Figure 2: Schematic of Abstract System Under Test

And when viewed within the time-line of the phases of test execution, that is, precondition establishment, execution and verification, the schematic in Figure 3 is the result.

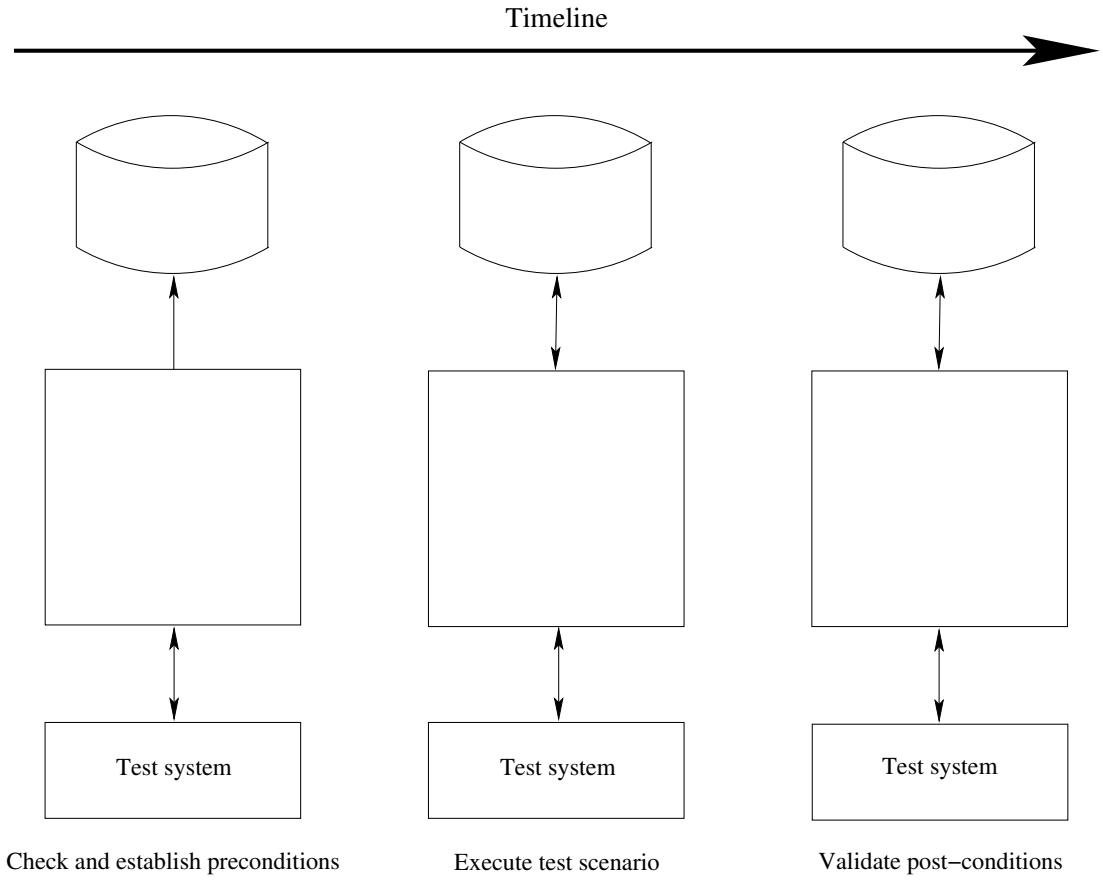

Figure 3: Schematic of Time Line of System Under Test

Our tools support the automation test effect, both within our own technology and in cases where other tools are used. The component-interface we have followed in our own technology allows us to extend the spread of situations covered, and allows us to apply our tools to the optimisation of test automated testing effects by complementing technology from third parties.

The test framework is shown in position below over the schematic of Figure 3 and shown as such in Figure 4.

The preparation, execution, and validation phases of the test scenario executed have been discussed previously.

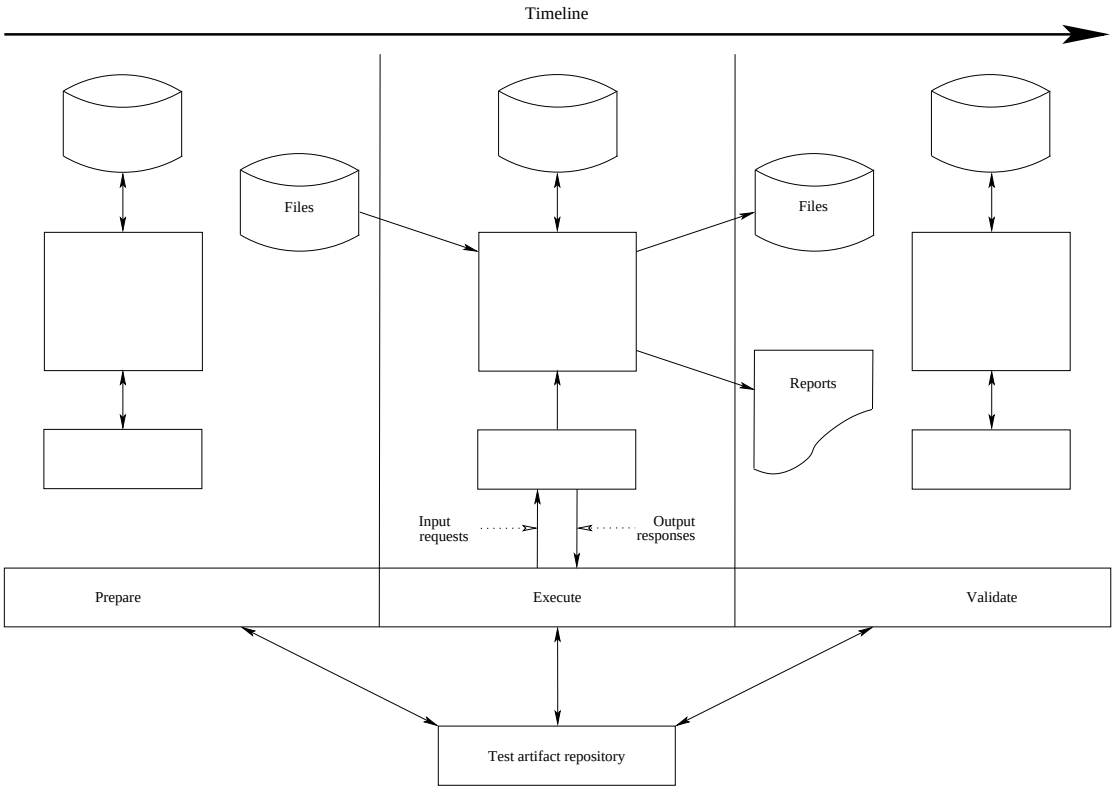

Figure 4: Schematic of Time Line of System Under Test

3.4 Functional Testing Tools

The following table describes the functional testing tools.

Tool	Description
Eresia	Eresia is a framework comprising of the core components which, together with portals that manipulate the interfacing technology of the systems under test, achieves automated software testing. These core components include a graphical user interface, editor, script compiler, debugger, interface definitions and run time. Implementations of the portal interface type are responsible for describing (recording), manipulating and feedback of the system under test by binding to that system's technology. The Eresia framework: • Enables rapid repeatable testing of application software. • Enables thorough testing to be done with minimal effort in a fraction of the time that would otherwise be the case. On regression testing, this can realise a significant productivity benefit. • Is a generic mechanism to script tests. • Eases error detection by highlighting variations from expected results. Eresia runs on a workstation and has the means to connect and interact with technology that is hosted on a wide range of platforms such as z/OS, Unix, fault tolerant machines (e.g. Stratus) and Windows.
	continued on next page

continued from previous page

Tool	Description
File Injection Portal	Eresia File Injection Portal (FIP) is a software testing tool that together with Eresia enables test data to be generated from meta data. As part of the Eresia framework, the FIP: • Enables rapid generation of test data. • Enables significant productivity gains relative to software testing data setup; On regression testing, this can realise a significant productivity benefit.
Network Injection Portal	Eresia Network Injection Portal (NIP) is an automated testing tool that together with Eresia enables messages from any network input point to be scripted and fed to the application systems. These scripts are reusable for future testing. As part of the Eresia framework, the NIP: • Enables rapid repeatable testing of application software. • Enables thorough testing to be done with minimal effort in a fraction of the time that would otherwise be the case. On regression testing, this can realise a significant productivity benefit. • Is a generic mechanism to script tests over virtually any network message interfaces (i.e. transport and presentation layers). • Eases error detection by highlighting variations from expected results.

continued on next page

continued from previous page	
Tool	Description
XML Injection Portal	The XML Interface Portal (XIP) is a software testing tool which allows a user to perform tasks on XML documents and interact with various other components, such as XML Schemata, Candidate Data Files and Network Control Programs. XML documents can be viewed, edited, created, deleted and sent over a network to a server from which the response document can be captured and presented. All documents, settings and state information for a workspace can be saved in a user named workspace file.
MAP3270 Portal	The Eresia Map 3270 Portal is a tool used in the scripting of Mapped 3270 Screen applications. The tool uses the map information as meta-data and scripts are generated using this meta-data. During test execution the map definition is used to rebind the inputs to the screen layout and hence to the 3270 data stream. Screen outputs are interpreted for both the Thistle scripting language using the map definitions and portraying the data using the meta-data therein. The portal includes a special purpose 3270 emulator and a component on z/OS which assists in determining which map describes a data stream.

Table 1: Functional Testing Tools

4 Test Artefact Repository

The test artefact repository contains all the test automation scripts, all the pre-validation data selection artefacts, all the configured data scenario scripts, etc. The repository also contains all the artefacts required for non-functional testing, as will be discussed later.

4.1 Requirements

The test artefact repository needs to be a full function version management system, suitable for managing and versioning both binary objects as well as text objects, such as scripts and configuration files.

The test artefact repository must be such that it encourages the use, sharing, and distribution of information amongst those involved in the testing process. It is also the tool responsible for collecting, maintaining and sharing intellectual property relating to the testing of systems. Our methodology regarding the effects around test automation is designed to encourage the creating of intellectual property in the testing space. This is necessary given the cost of development, and testing larger systems, as well as their complexity. The means by which this investment pays off is delivered through the repository and the subsequent re-use of the artefacts in down stream regression and enhanced functionality of test cycles.

Code Magus does not supply a test artefact repository, but we do support the usage of a few third party and open source repositories that do meet the requirements; examples being CVS, subversion and perforce.

- The version control system must also allow concurrent use and concurrent maintenance of the artefacts.
- The repository must allow test aligned structures, as well as application system aligned structures.
- The repository, or the repository hosting system, must allow individual user management, possibly with module level access controls.
- The repository, or the repository hosting system, must provide a search facility for searching available artefacts by name and content.
- The repository must facilitate the simple addition of test results by testers and test analysis.

Figure 5 shows an overlap of the test scenario execution as depicted in Figure 1. The following sections cover the various functional areas.

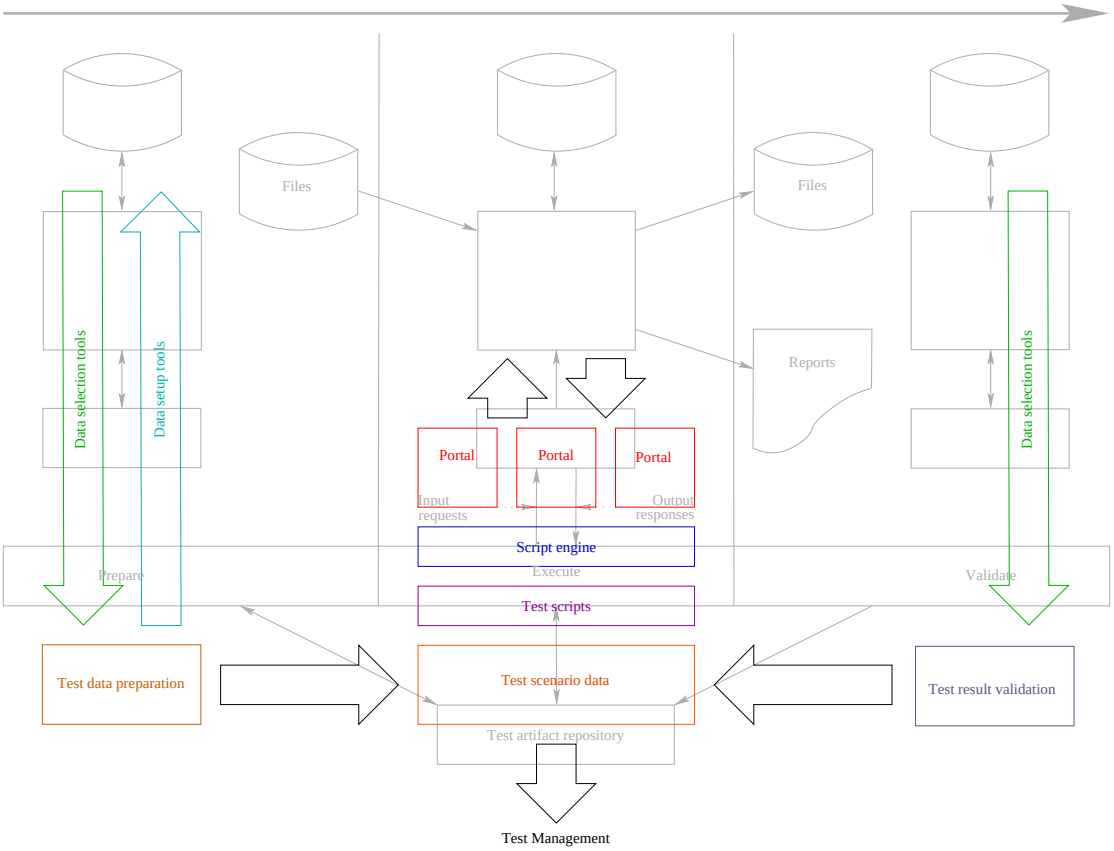

Figure 5: Tools and Components in the Execution of Functional Testing

4.2 Data Preparation and Preconditions

It is possible with small stand-alone and isolated systems, to have on hand a database containing the entities which match all the required test scenario preconditions. Additionally, with relatively small isolated systems, the logistics of restoring the database back to this initial state in preparation of a test cycle is a relatively painless exercise.

In these situations, the maintenance of the entities matching these preconditions could amount to the maintenance of a small template database over time. The tooling requirement for this would be minimal.

For larger, more complicated and inter dependent systems, the ability to maintain state data in this fashion, matching the test scenario preconditions, is a little more difficult. On top of this, where different projects in the same application space have staggered and overlapping execution times, the ability to synchronise all projects in order to restore the data to some initial known state is very difficult if it has to be done without impacting those projects.

To circumvent these situations projects typically find themselves doing one of the following:

- Use the system to create new entities with the derived given state. This now allows different projects to use the system to add entities (e.g. accounts) at will, and with no or minimal impact to other users of the system. Our tools support this approach which can be effected simply by configuring and writing the required scripts to be executed when preparing for a test execution cycle.

  There are, however, some downsides to this approach and it may not always be possible.

  - Certain scenarios may have a precondition that the entities have a history. When historical content or significant lapsed times are required, some other approach to the preparation of the data is required.
  - Certain scenarios may have a precondition that requires that the state of the entities against which the tests will subsequently be executed can only be assigned by the system over a period of time.
  - Systems that are maintained over many years, may have elements of state data that were established under conditions which are no longer supported by the system. For example missing information may have once been indicated in one way (e.g. NULLs) and are not represented by default values (e.g. spaces or zeros). The line system may still have data entered one way, but is expected to continue working properly under the new code. It is the responsibility of the regression pack to establish this fact.

In these cases it is sometimes desirable to prepare a test system once-off (and occasionally) from a production system (possibly with suitable downsizing and obfuscating of

personal data), leaving the variety of logic driving elements intact.

To support this, our tools allow for the configuration and labelling of preconditions required by the test scenarios. In this case, the test cycle preparation would include processing of extract data from the system, making sure that that data matched the required preconditions and placed the required elements of test data into the actual test scenario data (usually by populating the required elements of a driving spreadsheet).

With this tool and configuration of test scenarios, it is possible to know the coverage of the required test scenario preconditions that are satisfied by the system. Additionally, no laborious manual fishing exercise is required as part of the preparations of the test cycle.

4.3 Coverage

Knowing the variety of scenarios that a system ought to support, based either on the system specification or on detailed knowledge of the functional behaviour of the system, is one thing, but this may not be available or may be outdated. This makes it difficult to understand the bounds on the required coverage. Additionally, the data in a particular instance of the system may reflect more variety than the current specification allows, or as understood by the parties available (as previously mentioned, this situation could, for example, have arisen by removal of inconsistencies, addressing faults and adding better data validation over time).

It is always useful when starting an exercise of test automation to understand this upper bound, to keep track of how the data profile in the system changes over time, and to continuously measure the coverage of the test automation effect against this bound.

To assist in this process, we have a tool which identifies with an application system's data tables or files, the unique values of fields in that table file (or join up various tables and/or files), or groups of combinations of fields. This process also reports on the cardinality of each of these fields or cross product of fields when they are grouped, showing how many rows, records, etc, have that value or combination of values.

This tool, also, optionally produces a file containing images of the input records which, modulo a given factor, have unique field values, or unique groups of field values, thus producing a minimal source with the same data value coverage.

4.4 Securing Data

Where data access is required to be read and produced in the form of records of a file or rows of a table query result, the access to this data is described through a generic API which uses one of a number of access methods responsible for the actual data or media access. Such media access methods support local and remote execution of queries,

reading and writing of files, etc. In addition to the pre-existing access methods, there is an SDK for developing additional access methods, dealing with proprietary, name, or legacy situations.

This immediately makes any data servers or media accessible through this new access method available to all the Code Magus tools. In addition, the library is separately available, allowing proprietary tools and third party tools to benefit from data access using any of the access methods.

5 Non-Functional Testing Tools

Orkhestra, the primary Code Magus non-functional testing tool, drives various configured channels into a system under test. Observations made by Orkhestra are reported to a real time data collection and rendering agent; or alternatively recorded for later analysis and reporting. In addition to these external observations of the behaviour of the system, as is evident by the activity on the channels into the system, probes deployed onto the hosting elements of the system under test observe, record, and report resource utilisation and performance metrics to the same data collection and rendering agent.

There are probes that extract resource utilisation (CPU, memory, paging, network activity, disc activity, etc.) and performance metrics from machines such as Windows servers, AIX, Solaris, Linux, devices that support a Simple Network Management Protocol (SNMP) using the Management Interface Base (MIB), etc. Should a customer have a requirement for a probe not supported, we will assess the feasibility of providing one.

It is the function of the probe to invoke a local machine's API in order to extract resource usage and performance data and communicate this data in a common format for reconciling, analysis, reporting and rendering. There is a kit available for customers to develop their own probes.

As with functional testing, non-functional testing requires access to specific elements of the application data of the system under test. These might be function, or state specific criteria that these elements are expected to match. We use the same access method mechanism that is across the entire tool suite. In particular, one of these access methods lends itself to the non-functional testing space, by providing the non-functional scripts access to streams of application data, delivered in a pre-read, or read-ahead scheme, over the network, thus sharing data between all the instances driving the system under test, while minimising the possibility of starving the test system of data.

Figure 6 illustrates the components of the driving system for non-functional testing.

To describe the behaviour of the element driving the system under test, an input-output finite state machine (IO FSM) is used. For the purposes of observing the behaviour of the system under test, for analysis and reporting purposes, a certain amount of high-level or abstract state is required. This is the level at which state information is known in the IO FSM. Additionally, for the purpose of controlling the instances of the uses of the channels of the system under test, a certain amount of direction of the instance is required. This control takes the form of setting and cancelling times, delaying progress based on configured times drawn from configured distributions and allowing, for example, think times to be executed.

In order to maintain an actual conversation with the system under test, the IO FSM uses a control program which maintains the detail state required to look after the instances of one of more instances of the controlling IO FSM. Because the communication be-

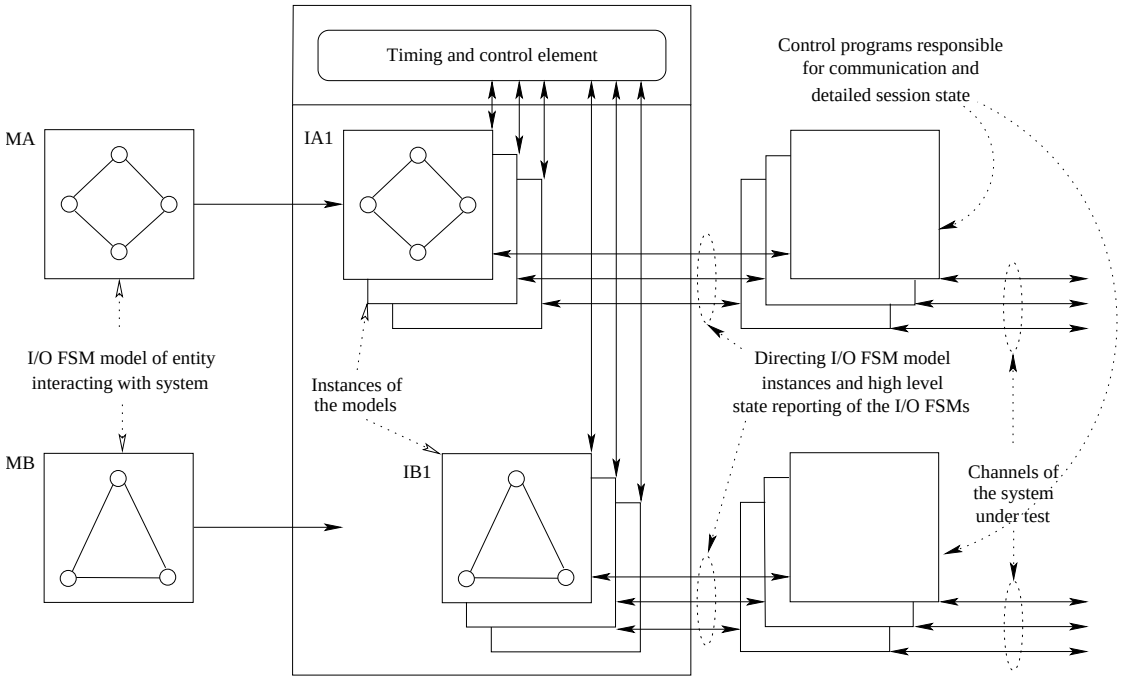

Figure 6: Orkhestra Components for Driving Non-Functional Testing

tween the controlling element and control programs is minimal, and because the control programs report events to the controlling element, time stamped at the time of observation, and the fact that the software internally synchronises all clocks, the architecture lends itself to a scalable solution in which the control programs are distributed to allow greater loads on the system under test by using the resources of more machines, network adaptors, network segments, etc.

The elements of the test system in the Figure 7 are represented by the Block TS in Figure 8.

The Code Magus tool Serfboard is responsible for the collecting, recording and on-line rendering of external observations (such as throughput, response times, breakdowns of response times and time-outs by response types), machines resource usage run times (such as CPU usage, memory usage, paging activity, network bandwidth usage, files, system bandwidth usage) of the hosting components of the system under test, and of the performance metrics (such as content switches, ready queue or load average values, or any of typically hundreds of metrics available through machine specific performance API's).

The data sent to Serfboard originates from the Orkhestra test system as well as the machine and components of the system under test through the various probes deployed on and adjacent to these components.

In addition to the network driven functional and non-functional testing scenarios described above, the Code Magus tool suite includes a component called Verify. Verify

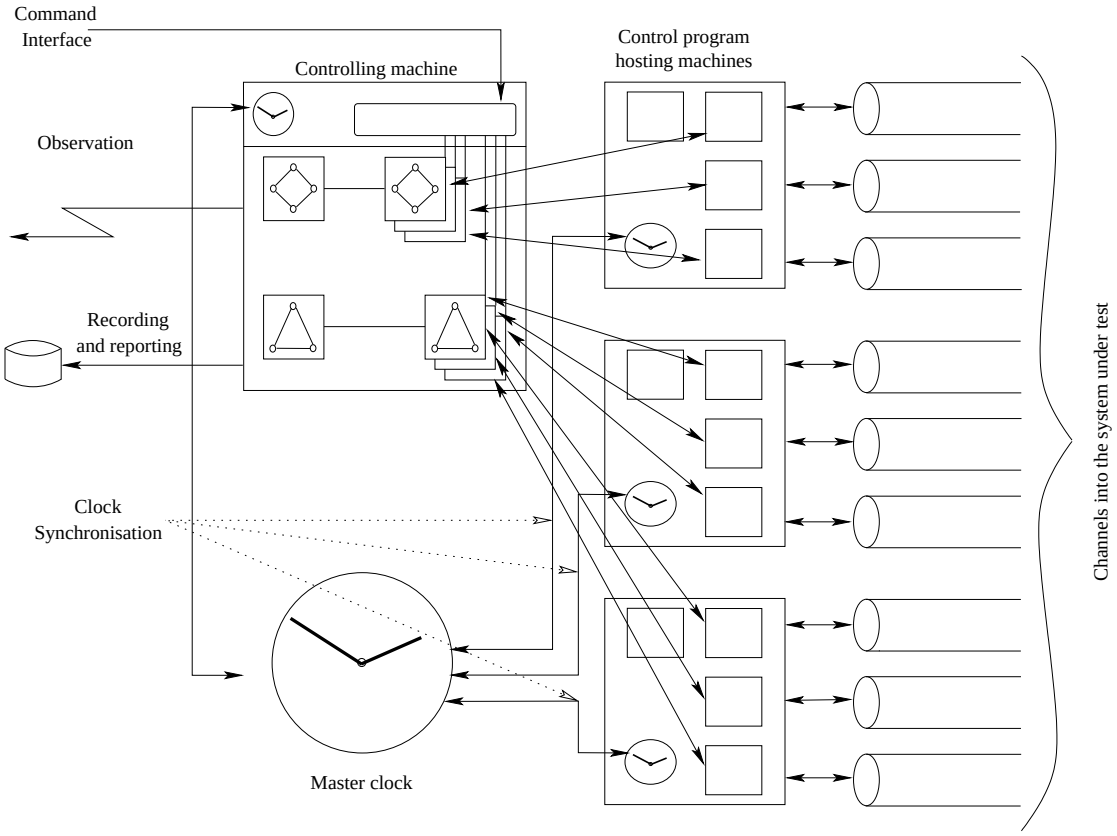

Figure 7: Orkhestra Configuration for Non-Functional Testing

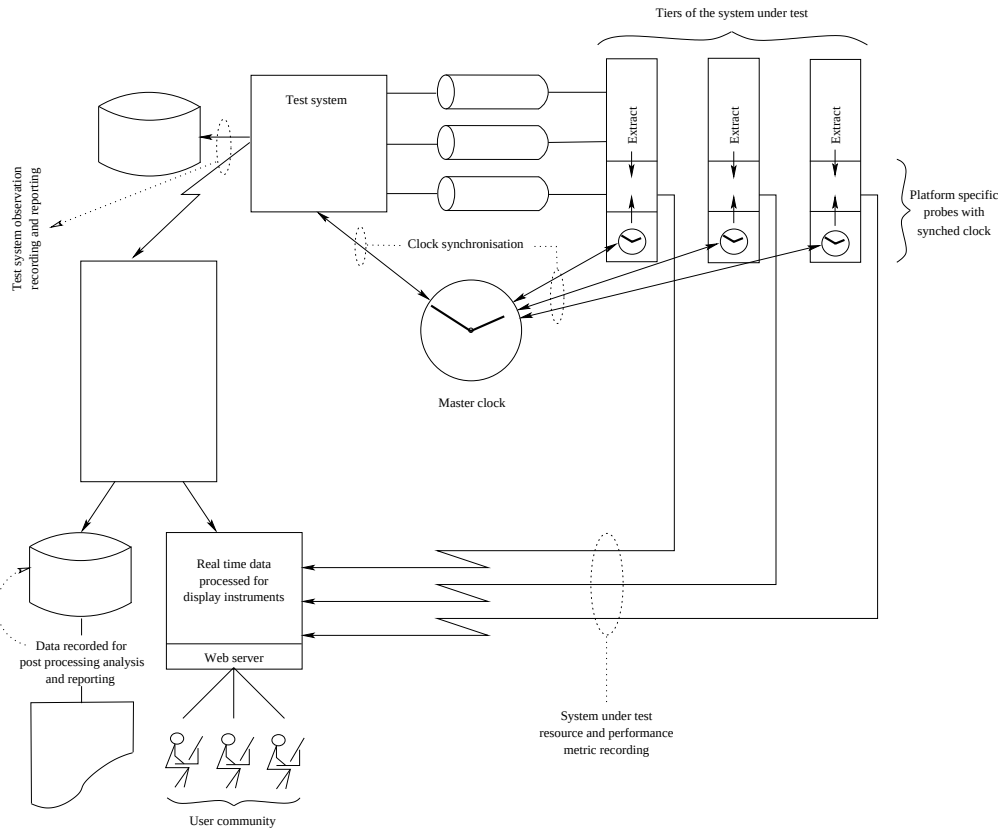

Figure 8: Orkhestra Configuration in Relation to Components of the System Under Test

is used for the bulk testing and verification of prepared cases. This is accomplished by a classification and rules system that processes prepared cases in bulk and reports on the servers on failure of these cases, including the variance where connectiveness is quantitative rather than qualitative.

As part of the reporting of failed cases, the tool also reports on the process of deciding rules, by the process of classifying cases according to the configured conditions, and presents this trace for consideration, including the reporting of the contents of the case.

Figure 9 illustrates the process of Verify.

Verify is particularly useful in test situations when applied pricing or fee generation requires testing.

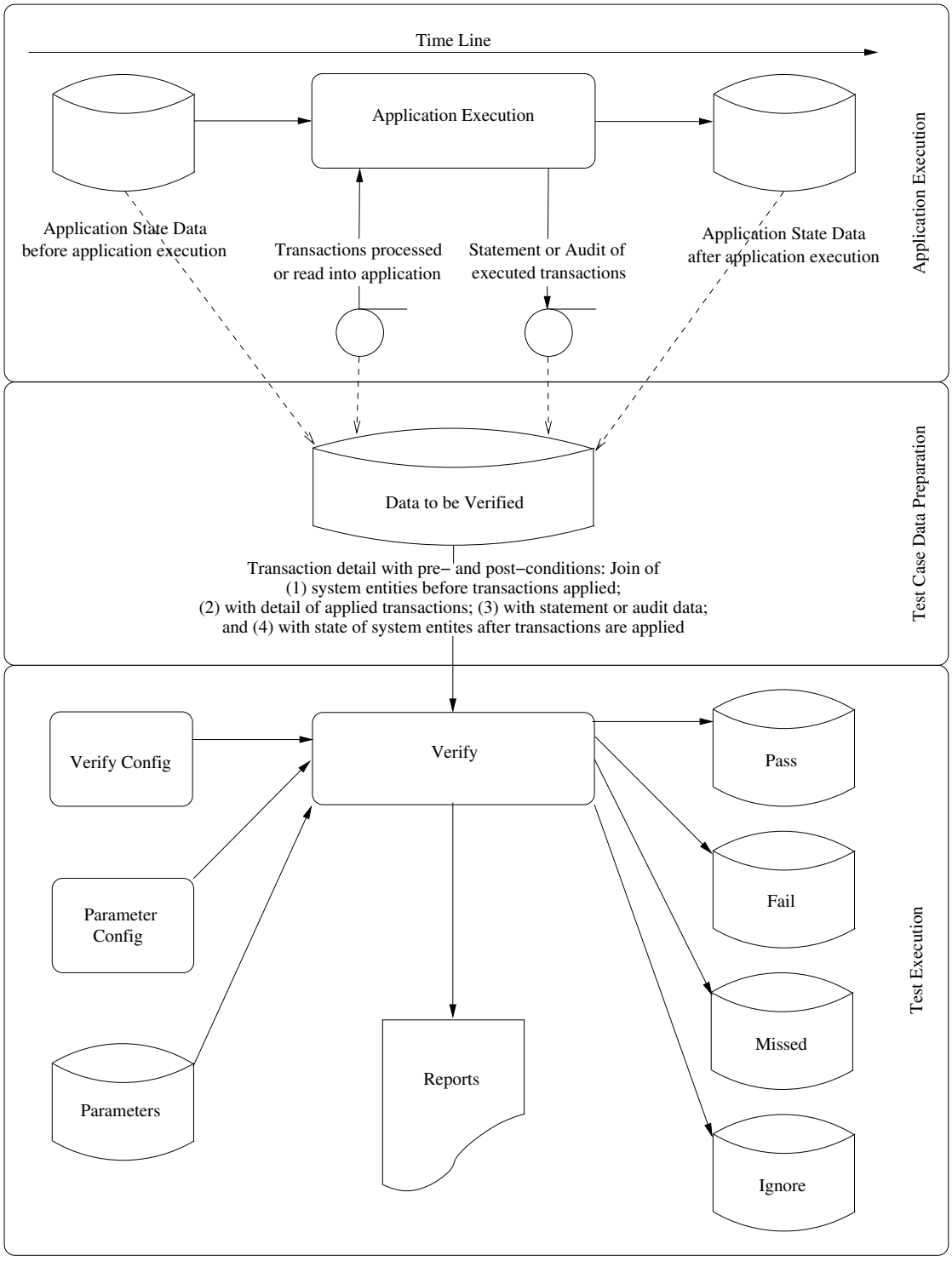

Figure 9: Schematic of Generic System Under Test Detailing Points of Interaction

6 Access Methods

We have already mentioned the tools usage of access methods. Each tool that requires access to a record or message stream uses the Code Magus Recio library. This library allows the tool to connect to an access method by means of an open specification string (an open spec) which details the access method, the 'object' being accessed, and, optionally, any access method specific options.

Code Magus supplies access methods for reading and writing files locally or remotely, in local or remote file formats, for reading result sets of queries executed locally or remotely (from a DB2 instance running under z/OS) or remotely via ODBC.

There is also a toolkit which describes the interface and the requirements for the customer or third party to develop their own access methods.

Access methods can also be stacked, allowing processing of data to be injected into the record or message reading and writing process. This is useful for aligning or adjusting the serialisation and de-serialisation of objects suitable for processing in one form, so that they are presented in another form suitable for some other type of processing.

6.1 Supplied Access Methods

Each access method is available for use by, and therefore extends, any application enabled by the use of the recio[1] library functionality.

Access Method	Description
Binary	The binary access method[3] supports fixed and variable length record files implemented on byte-stream file systems. This includes VOS stream files, UNIX binary files, Windows/DOS binary files and z/OS classic MVS and HFS files. Classic MVS files can be supported either in their underlying record mode or by treating the files as binary stream files. On byte-stream based file system files, variable length record files are implemented by the inclusion of a record descriptor word in front of each record. The format of this RDW is the same as the format on Classic MVS RECFM=V files. And is the same format that is included when these files are binary copied with FTP from MVS systems using the '`quote site rdw`' option.

continued on next page

continued from previous page

Tool	Description
Text	The Text access method[9] supports delimited record text files implemented on byte-stream file systems and on MVS for record based files. This includes VOS stream files, UNIX text files, Windows/DOS text files and z/OS classic MVS and HFS files. With the exception of MVS classic files (which are record based files) all the other types (i.e. stream based files) may be read or written on any of the supported platforms.
ODBC	The odbc access method[20] supports reading data from an ODBC data source on Linux, Unix and Windows platforms.
MVS	The MVS access method[6] supports access to fixed and variable length record files implemented on MVS systems. JCL format syntax is used to specify files allowing correct and efficient file allocation.
Remote	The remote access method[7] and server support any other Recio access method to be used remotely across a TCP/IP network.
DB2query	The db2query access method[17] supports reading data using a DB2 SQL query. This includes UDB on Linux, UNIX and Windows, and DB2 for z/OS. Using the associated access method DB2dclgn the meta data for the query can be obtained.
Dataset	The dataset access method[4] supports named and cataloged configurations of access method open specification strings. These open specifications can be generic, arranged in a hierarchy, and can differ for the same cataloged name depending on the mode that the Recio record stream is opened in. This is then a convenient method of abstracting away from the details of the access method specification string; especially where it may be lengthy or complex.
Edit	The edit access method[14] supports editing of input or output records after reading or before writing them using an underlying access method. An edit process is often used to expand compressed records on reading or to contract records prior to writing them. For example, self defining compressed records are easily processed in this way allowing the application the ability to only process complete uncompressed records.

continued on next page

continued from previous page

Tool	Description
Source	The source access method[19] supports reading from a source server on a remote platform. The source access method and the source server enable the ability to read a remote recio data stream in an efficient manner. This is most relevant when the remote data stream is required by a program or programs (the requester) within a system under test (SUT) where the effect of network latency needs to be minimal. Reading the data does not cause any delay or slowdown of the network traffic, as the data is received immediately it arrives thus clearing the system wide buffers within the network layer. The source access method and source server also offer the ability for a source to be shared amongst multiple requesters within the SUT, circumventing multiple streams to the same recio data stream. In this instance the multiple streams are served from one source definition and each stream is processed in a round robin manner. Finally the source access method and source server allow a stream to be repeated continuously. This allows a test to be set up based on time limits and a constant flow of data; in other words an end of file condition would not prematurely end a defined test cycle
Directory	The directory access method[5] supports reading directory entries on Unix, Linux and Windows based systems and Catalog entries on MVS.
Image	The image access method[10] supports reading a record stream from a DB2 Image Copy dataset, where both compressed and non-compressed backups of table spaces are supported. One instance of an image access method recio stream allows a program to read the data from a given object (determined by OBID) row by row.
Concatenate and Split	The concat and split access methods[22] support concatenating recio input streams and splitting recio output streams, using an underlying access method for the actual data IO. A concatenation of input streams reads the streams in the sequence they are named in the list of objects in the recio open specification. When an output stream is split into more than one stream the object name is modified as each stream is opened for writing.

continued on next page

continued from previous page	
Tool	Description
Null	The null access method[18] supports writing to an access method that discards all records. This is similar to writing to the file /dev/null on Unix platforms and is useful when the a vast amount of data is produced all of which is not required to be kept in any manner. When reading from this access method an immediate end of file is returned. This access method is very useful in stress testing or non functional testing.
Standard	The standard access method[8] supports reading from standard input and writing to standard output and error on byte-stream file systems. This includes under the operating systems VOS, UNIX, Windows/DOS and z/OS (both classic MVS and HFS), all using the Standard C I/O Functions of streams.

Table 2: Access Methods

7 Network Control Programs

Where ever network access is required to send or receive messages over a transport layer, the Code Magus toolkit uses an interface to an object called a network control program (NCP) which has the responsibility of maintaining the state of that transport layer connection, as well as the responsibility for transmitting and receiving messages over the particular transport layer.

The Code Magus suite of tools includes a number of network control programs. Table 3 on page 29 describes the available network control programs.

In addition to the supplied network control programs, a kit is provided for customer or third party developed network control programs.

NCP	Description
ncpcrdon	ncpcrdon sends and receives messages to and from processing systems that consume on-line transactions based on credit card association specifications such as VISA, MasterCard and American Express.
ncphttp	ncphttp sends and receives messages using the HTTP protocol.
ncpkhttp	ncpkhttp sends and receives messages using the HTTP protocol and provides the security features required by Kerberos version 5 published by the Massachusetts Institute of Technology (MIT).
ncpmq	ncpmq sends and receives messages using Websphere MQ Queues.
ncpsoap	ncpsoap sends and receives messages using HTTP/SOAP messages.
tcpframe	tcpframe is a generic NCP that sends and receives messages over TCP/IP to and from a remote host. It offers various different options for framing the message for correct delivery to the receiving server. It also allows output messages to be serialised for efficient transport (for example compressed) and input messages to be de-serialised for consumption by the test tool or application (for example expanded).

Table 3: Supplied Network Control Programs

8 Meta-Data

Access methods and meta data control programs send/write and receive/read serialised, binary, or document data in various forms. For the sake of robustness and convenience of referencing to the data, by humans and machines (for example, in scripts), the Code Magus test scripts seeks to bind to applications meta data whenever possible. For XML XSD described data, this is typically bound into an XML library to de-serialise the document for consumption by the test tool, and then serialised by the library for transmission or storage.

For mapped messaged and records, as is typical of rows of query result sets, formatted records in the files of an application system, standards based files such as IATA DISH, card association and interchange file and message layouts (for example, ISO8583 based formats) and the endless possibility of customer and customer application system based formats, we use a scheme of abstracting the meta data scheme, and where required, use an access method injected serialisation and de-serialisation mechanism (once configured this stacking is not visible to the casual user). In particular, the abstraction layer is capable of consuming customer application specific meta data schemes (in, for example, COBOL copybook formats), or DBMS column meta data describing the columns of a query result set.

The artefact configuration of this scheme binds the underlying schemes (for example, copybook or copybook elements) to the various record or message types, and exposes, through the API, the elements of those schemes appropriate for that record or message type.

All Code Magus tools requiring the binding of application specific meta data of this type access application data elements through this scheme.

9 Data Manipulation/Management Tools

9.1 Overview

In addition to the test driving functional and non-functional tools described here, the Code Magus tool suite includes data query, investigation and test preparation tools. An example of such a tool is the tool Unique, which was mentioned earlier.

The Eresia portals NIP, FIP and XIP process test content for driving and interpreting reports in the form of messages, records and documents of a system under test. These tools do not only operate as a test system but also allow the user to inspect, perform ad hoc queries and ad hoc preparation of file records, messages and documents (for example, SOAP messages for a web service hosted application). When used in this manner, they present a graphical interface for the direction of the tool. When these GUI fronted components read and write records, they are directed to do so by prompting the user for a recio open specification string, and access the specified object using the indicated access method and supplied options. And when these GUI fronted tools require network access for the ad hoc interaction with the system under test, they do so by prompting the user for a network control program. The network control program interface includes a mechanism of discovery of network control program specific attributes, some of which are mandatory and some of which are optional. The tool then has the opportunity to prompt the tool user for values for these options.

To prevent the process of using these tools on an ad hoc basis being a burden for the user, these choices and configurations are saved in work spaces as files on the local machine. These work spaces have associations with the tools, thus making the sharing and returning to the same configuration a simple matter of keeping track of the corresponding work space files.

When the GUI fronted components interpret log messages, file records, and messages off the wire, they do so using the same meta data scheme of object types mentioned earlier.

This scheme, together with the access method scheme, allows the user to query local or remote files, tables or result sets of remote or ODBC queries, in a manner which assists the user of the tool to construct the query by pull-down presentation and choice of the available meta data. These queries are also saved in the tool work space, so can be canned and shared between the users.

9.2 File Tools Suite

The File tools[11] suite of tools offer the ability to consistently access and prepare data, especially for stress or non functional testing. All these tools use the recio functional-

ity when accessing data which offers the ability to process many different sources and source types, which may be hosted on remote platforms.

Tool	Description
Print/Unpack	The Print tool reads an input file mapped by an object types definition[24] and produces one of a number of different formatted outputs.
Copy	The Copy tool reads an input file optionally mapped by an object types definition[24] and copies it to the output file.
Compare	The Compare tool reads two input files mapped by an object types definition[24], compares them and produces a report of the result.
Pack	The Pack tool reads a comma separated (CSV) file of data and writes an output file mapped by an object types definition[24].
Unique	The Unique tool does simple domain frequency analyses and performs a type of downsizing based on making certain fields, or combinations of fields unique in an output file by filtering the records from the given input file. The tool works by supplying a copybook and a record in that copybook which is assumed to map the file. There can be any number of fields or groups of fields. Ideally a field or group of fields would be chosen because the cardinality of the column or column group is relatively small and the field or group of fields forms some sort of path or conditional indicator to the processing programs or system.
Shuffle	The Shuffle tool will read in a file described by a Recio open string specification and write out the file described by the output Recio open string specification by shuffling the sequence of the records from the input file in a random manner. This means that although the output file is the same size and holds the same records as the input file repeated runs using the same input file will not produce the same sequence of records in the output file from one run to the next.

Table 4: File Tools Suite Components

9.3 Cross Platform Copy

Cross Platform Copy[23] copies files with multiple fixed format records suitable for processing on one platform, so that they are suitable for processing on other platforms. The files can contain records with multiple record formats, but the content of any particular record must be able to distinguish the layout of that record. In order for Cross Platform Copy to be able to copy a file and convert its records, certain configuration data is required. This configuration data is kept in a separate file and contains logic and references to copybooks. Within the configuration file there is a section for each record type. These sections, referred to as layouts, assign a mapping to the record by way of selecting 01-level items in a copybook together with an indication of which fields in the copybooks are actually present in each case. In order to know whether or not a partic-

ular layout applies to a record, a predicate is included in each layout. This predicate is restricted in that it only comprises of a conjunction of equalities and in-equalities in which fields are compared to literals.

Together with the layouts in the configuration file, there are input and output options which pertain to the source and target systems on which the file originates (input) and on which the the file will be processed (output). These platforms may be different to the platform on which the Cross Platform Copy tool ultimately executes.

9.4 Report Compare Utility

The Code Magus CSV File and Report Compare Utility compares cell contents of a subset of report files under the specification of a row key, a column delimiter and a list of all or some of the columns to compare and reports on any differences found. At the end of the difference report a summary of the differences is printed.

By specifying the column delimiter the utility can compare comma separated variable (CSV) files or fixed column reports.

The utility can also

- Skip rows at the start of the report so that report heading lines may be ignored.
- Set a numeric tolerance so that two compared numeric cells are considered equal if they are within a certain tolerance of each other either specified as a relative or absolute tolerance range.
- Generate column headings if they are not evident on the first row of the reports to be compared. The headings generated are similar to those of popular spreadsheet applications. They range from A to Z, AA to ZZ and BA to BZ onward.

The Code Magus CSV File and Report Compare Utility runs on Unix and Unix-based platforms, Windows platforms, z/OS USS and Classic MVS environments.

9.5 Open System Sort

Open System Sort[2] is a Unix styled sort utility which can be configured to use the resources of an available machine in order to improve the sort times and throughput. Open System Sort sorts files which are expected to be made up of records with fixed or variable length format, but which adhere to a fixed layout typical of files described and used in legacy systems such as PL/I and COBOL.

Open System Sort uses a configuration file in which the bounds on the resources that an instance of Open System Sort may use are stated.

From an application point of view, the format of the key specification follows the same syntax and semantics as is used on traditional host systems such as MVS. In addition it is possible to use the application meta-data to specify fields and hence to describe the sort requirements in a symbolic manner.

10 Services

In addition to the tools and components, Code Magus offer a number of support services within the automated testing space. These services range from basic training on the tools, to training in methodologies using the tools in a robust and flexible manner. The methodology that Code Magus has adopted has proven itself and has been documented in training material that we make generally available.

In addition to supporting the scripting by Code Magus clients, Code Magus also have a number of domain specific solutions scripted. We also provide bespoke solutions on behalf of our clients.

The following table shows domain specific solutions that Code Magus offers.

Solution	Description
AMEXCardOnline	The American Express on-line transaction script suite is a solution aimed at providing users with a fully functional set of American Express on-line transactions derived from and in compliance with American Express that will perform chosen transactions against the desired target system.
AMEXSettlement	The American Express settlement script suite facilitates the creation of settlement files based on the American Express Programmer Specifications. The settlement files will be generated in a format suitable for submission to the desired target system.
MCICISCardOnline	The MasterCard Customer Interface Specification on-line transaction script suite is a solution aimed at providing users with a fully functional set of MasterCard on-line transactions derived from and in compliance with Mastercard that will perform chosen transactions against the desired target system.
MCISettlement	The MasterCard settlement script suite facilitates the creation of settlement files based on the IPM Clearing Format specification. The settlement files will be generated in a format suitable for submission to the desired target system.
VISACardOnline	The VISA on-line transaction script suite is a solution aimed at providing users with a fully functional set of VISA BASE I online transactions derived from and in compliance with VISA that will perform chosen transactions against the desired target system.
VISASettlement	The VISA settlement script suite facilitates the creation of settlement files based on the BASE II Clearing Interchange Formats. The settlement files will be generated in a format suitable for submission to the desired target system.

Table 5: Bespoke solutions

11 Parameterisation Schemes

11.1 Application Parameters

Parameters that control a particular run whether automatically set or supplied on demand from the user are part of every tool within the Code Magus software portfolio. Application parameters, together with the defined user interface, enable an application to define, manipulate and store the current state of a parameter set. During an application or tool start-up the user, if foreground processing is chosen, has the ability to set and change the values of any of the parameters in the parameter cache. Whether foreground or background processing is selected the application will not be allowed to start until all the parameters have a valid value.

The configuration file defines the processing mode, user interface, title and description, sets environment variables for use in the file, names the parameters and sets the attributes for each parameter.

The parameter attributes define what a valid parameter is. To this extent the set of parameters defined by a particular configuration are bound to that application; meaning that each application would have its own set of parameters.

The user interface can be set for the platform on which the user is logged on to. For example on Windows there is a tabbed or list form GUI available. On all platforms the default user interface is command line based.

11.2 Structured Environment Variables

The Code Magus software also provides an interface for any tool, program or customer application to retrieve the value of a structured environment variable (SEV). A SEV is an environment variable that resolves to a value obtained from a specific cache or store of values and is defined by a configuration file. The structure of the name of the SEV identifies it as a SEV, names the store and lastly names the actual variable. Using this interface, caches of variables can be set up; For example there may be a cache for the current machine or the running application.

Structured environment variables are an extension of system environment variables. They are specified in the same way but the structure of the name allows for qualifying the variable within a particular cache.

12 Monitoring Non-Functional testing

Code Magus SerfBoard is a system that enables a user to monitor, and possibly interact with, a running system or systems via a web based graphical dashboard. Dashboards of this nature that show a unified view of key performance indicators of different systems are extremely effective in the management of networked systems testing or systems stress testing. In stress testing, being interactive, the user can change various thresholds and key values in order to watch the effect of different loads on the system under stress. The SerfBoard system comprises the following parts:

- The core server (SerfBoard) which is a daemon process (executes in the background) and receives metric data, stores it and supplies it to the web application on request.
- The display instrument plug in programs and web application which are responsible for rendering the final dashboard as viewed by the user. The web application continuously requests updates of data from the server.
- Metric feed probes and feed library. The metric feed probes are responsible for querying a specific system or interface that supplies metrics (for example SNMP), extracting the relevant data, converting it to the SerfBoard format and forwarding the metric data on the the server. There is also a C programming library interface available for development of user written metric feeder programs.

Metrics are supplied to Serfboard from the following probes:

Probe	Description
cmlxsnmp[12]	Serfboard SNMP Metric Probe. This probe can extract metrics for by a system or device that exposes an SNMP MIB interface.
cmlxaixp[13]	Serfboard AIX Performance Metric Probe. This probe runs on AIX and uses the AIX perfstat programming interface to extract AIX specific performance metrics.
cmlxwinp[15]	Serfboard Windows Performance Metric Probe. This probe runs on Windows platforms and uses the Performance Data Helper application programming interface to extract Windows specific performance metrics.
cmlxwasp[16]	Serfboard Websphere Application Server Performance Metric Probe. This probe uses the WAS perfServletApp application to extract Websphere performance metrics.
cmlxsolp[21]	Serfboard Solaris Performance Metric Probe. This probe runs on Solaris and uses the AIX kstat programming interface to extract Solaris specific performance metrics.

Table 6: Metric Probes

References

[1] recio: Record Stream I/O Library Version 1. CML Document CML00001-01, Code Magus Limited, July 2008. PDF.

[2] PPMSORT: Parallel Open Systems Sort Fixed Format Record Based File Sorting User Guide and Reference Version 1. CML Document CML00003-01, Code Magus Limited, July 2008. PDF.

[3] binary: Fixed and Variable Length Record Stream Access Method Version 1. CML Document CML00005-01, Code Magus Limited, July 2008. PDF.

[4] dataset: Catalog Access Method Definitions Version 1. CML Document CML00013-01, Code Magus Limited, July 2008. PDF.

[5] directory: Directory Record Stream Access Method Version 1. CML Document CML00014-01, Code Magus Limited, July 2008. PDF.

[6] MVS: MVS Record Stream Access Method Version 1. CML Document CML00016-01, Code Magus Limited, July 2008. PDF.

[7] remote: Remote Record Stream Access Method Version 1. CML Document CML00022-01, Code Magus Limited, July 2008. PDF.

[8] standard: Standard Input And Output Using Recio Version 1. CML Document CML00030-01, Code Magus Limited, July 2008. PDF.

[9] text: File Access Method Using POSIX Streams Version 1. CML Document CML00031-01, Code Magus Limited, July 2008. PDF.

[10] image: DB2 Image Copy Reader Access Method Version 1. CML Document CML00036-01, Code Magus Limited, July 2008. PDF.

[11] **File Tools**: Reference and Guide Version 2. CML Document CML00043-02, Code Magus Limited, June 2009. PDF.

[12] cmlxsnmp: SNMP Metric Probe. CML Document CML00044-01, Code Magus Limited, June 2009. PDF.

[13] cmlxaixp: AIX Performance Metric Probe. CML Document CML00045-01, Code Magus Limited, June 2009. PDF.

[14] edit: Recio Edit Access Method Version 1. CML Document CML00047-01, Code Magus Limited, June 2009. PDF.

[15] cmlxwinp: Windows Performance Metric Probe. CML Document CML00048-01, Code Magus Limited, June 2009. PDF.

[16] cmlxwasp: Websphere Application Server Performance Metric Probe. CML Document CML00049-01, Code Magus Limited, June 2009. PDF.

[17] db2query: Recio DB2 Query Access Method Version 1. CML Document CML00050-01, Code Magus Limited, November 2009. PDF.

[18] null: Null Access Method Using Recio Version 1. CML Document CML00055-01, Code Magus Limited, January 2009. PDF.

[19] source: Source Access Method Using Recio Version 1. CML Document CML00056-01, Code Magus Limited, January 2009. PDF.

[20] odbcdcl: Recio ODBC DCL Generator Access Method Version 1. CML Document CML00064-01, Code Magus Limited, February 2010. PDF.

[21] cmlxsolp: Solaris Performance Metric Probe. CML Document CML00065-01, Code Magus Limited, June 2009. PDF.

[22] concat: Recio Concatenate and Split Access Method Version 1. CML Document CML00067-01, Code Magus Limited, May 2010. PDF.

[23] cmlxpcpy: Cross Platform File Copy Version 2. CML Document CML00029-02, Code Magus Limited, July 2008. PDF.

[24] Code Magus Limited. objtypes: Configuring for Object Recognition, Generation and Manipulation. CML Document CML00018-01, Code Magus Limited, July 2008. PDF.