M. Hubbard (IBM)
August 1992

DCE SIG SERVICEABILITY REQUIREMENTS

1. INTRODUCTION

One of the more difficult areas within management of distributed heterogeneous computing systems is the ability to properly isolate, diagnose, and correct error conditions. This paper discusses the requirements on DCE in this area under the title of ''Serviceability''. It presents the topics in terms of requirements statements rather than suggested implementations.

The ''Distributed Management Environment (DME)'' RFT will provide the framework for the distributed, remote and object-oriented elements of the requirements described by this document, however, many requirements are still independent of DME. As well, since DME (and a integrated DCE/DME offering) is not expected to be available for some time to come, a serviceability solution based solely on DCE needs to be evolved to support the initial DCE configurations. This solution can subsequently be migrated/integrated into DME at a later point in time. This paper focuses mainly on the recording of data and relies on DME to enable the management of the recorded data. The selected solutions for the requirements described herein need to consider the DME directions, but at the same time need to provide useful interim function without requiring significant changes to the DCE base code.

Note, that SOME of the requirements described in this paper are already being addressed by SOME DCE components. As a whole, however, DCE does not address these in a consistent and comprehensive way.

The requirements are discussed in 4 general groups:

(a) Error Notification Facilities

(b) Tracing Facilities

(c) Error Data Capture Facilities

(d) Externals

2. DCE REQUIREMENTS

2.1. Error Notification Facilities

This group of requirements deals with the mechanisms for reporting error conditions in terms of error function calls used, error data reported and error message processing/logging/routing.

2.1.1. REQT: Error text isolation

All messages currently embedded within the DCE component code need to be isolated and placed in message catalogues. This not only facilitates message translation into multiple national languages (really an I18N requirement), but also optionally allows tailoring and standardization of the message text to properly reflect specific vendor hardware/operating system environments (serviceability requirement).

Specifically, the requirement is for the use within DCE of:

(a) `gencat` utility for generation of message catalogues.

(b) `catopen()`, `catgets()`, `catclose()` function calls to be used.

(c) `%digit$fmt` (rather than `%fmt`) within the message text to facilitate reordering of substitution variables (refer to X/OPEN language-independent `printf()` description).

2.1.2. REQT: Common error notification function call

Currently most textual information displayed by DCE components is handled simply by issuing `printf()` functional calls (or equivalent), regardless of the type of information being processed (e.g., prompts for input, responses to commands, error messages, trace messages, etc.). This approach does not readily allow the use of system specific error handling facilities (e.g., logging, message suppression, automatic operator facilities, ''multi-console'' support, remote notification, etc.) for serviceability messages (e.g., error, trace). A set of serviceability function calls needs to be defined and implemented to isolate the different uses of `printf()`, thus allowing vendor-specific implementations of these function calls to exploit facilities of each system environment. For error message logging, a single new error function call is required to be used by all DCE components.

2.1.3. REQT: Standardized error message format

In addition to the message text and any function-specific error substitution values, each DCE error message needs to carry additional identifying information to aid in isolation of the error condition.

Specifically the following should be included (not an all-inclusive list):

(a) Product identifier (DCE).

(b) DCE component identifier.

(c) Function (within component) identifier.

(d) Symptom identifier (within function) — Product.component.fns.symptom uniquely identifies the point within the code that generated the error message. This is important since the same error message often gets generated by different sections of the code.

(e) Error (message) code — Identifies the error, the same error may be generated in multiple places (e.g., ''unable to allocate memory''). This code is useful when generating a Problem Determination Guide which would list all error messages by their code (for quicker referencing).

(f) Severity — Indication of message severity. This should be granular enough to allow for ''reasonable'' message suppression to be supported (e.g., Informational, Warning, Intervention-Required, Error, Severe, etc.). Severity can be indicated to the administrator by either a text string (or character) or numeric indicator within the message text itself.

(g) Etc.

An alphanumeric prefix could be assigned to each message (stored as part of the message text located in the catalog) to indicate the product id, component id, function id, severity and unique error code.

Within the message string, the common identifiers above need to be ALWAYS formatted in an invariant way to provide for a language/vendor implementation independent way of identifying errors. This is key for any automatic message processing (e.g., message suppression based on severity), as well as usable diagnostic documentation.

2.1.4. REQT: Remote notification

Remote notification support will be fully addressed in the future by DME. In the interim, however, heterogeneous DCE networks of any size are not a viable proposition without some level of this function. DCE either needs to support direct generation of notification flows using an already existing mechanism (e.g., TCP traps), or needs to define some DCE specific mechanism that could be mapped to specific vendor proprietary management solutions (in the interim) and later migrated to DME mechanisms. Assistance of the DME SIG is required to recommend the appropriate approach for this area.

Given that DME is the right long term solution, the ''least disruptive'' approach to DCE is required in the interim. A DCE-specific approach may be as simple as forwarding the error message text to the remote site. In this case, in addition to the standard error identifiers, one might also have to include information such as error correlation (e.g., ''error event'' UUID), error node identifier (global name?, UUID?, principal name?), error node type (vendor/OS designation), server/process identifier (?), ''pointer'' to additional error data (e.g., log records file), etc. Note, that a DCE-specific mechanism MUST NOT rely on any DCE functions to forward the notification (e.g., RPC), but rather should only use facilities of the transport network.

2.1.5. REQT: DCE exception signals

Within the client portion of the DCE code, errors should always be reported to the calling ''application'' for handling (in order to avoid ''disrupting'' the application's end user interface). This should be either done via a return code or by raising a DCE-specific exception signal. In general, the DCE client code executing on the callers thread should not explicitly issue the common error function calls (see above for description) unless this is as part of default error handling as a result of the application caller not handling specific exception signals.

2.1.6. REQT: Multiple layer error processing

Where exceptions percolate through several layers of an application, each layer should be able to find out what action a lower layer has already taken (e.g., remote notification issued, message logged,...) to avoid redundant processing. As well, if multiple layers do need to log their own message, there must be a way for a lower layer to percolate a correlation id to be used by all upper layers when they log their messages (or error data).

2.1.7. REQT: Extended error processing

Extensions to the common error function library routine allow vendors to integrate DCE error handling with each respective vendor operating environment. The functions listed below could be provided as part of the base DCE error support from OSF (preferred) or could be left up to each vendor to provide (if required) for their environment (acceptable). The later approach would potentially result in differences in the administrative interfaces for these functions that would have to be reconciled in the DME timeframe. Extended error processing includes the following:

(a) Error suppression — Need to be able to selectively suppress error messages based on administrative criteria, typically severity level. This is especially important in large processor environments supporting large number of servers and clients. In general, only informational messages and warnings not requiring explicit action would be suppressed.

(b) Message destination selection — Need to be able to direct error messages to multiple/alternate destinations based on administrative criteria (e.g., one message to multiple consoles, messages of specific type to specific console, logging vs. display, etc.). This is especially important in large processor environments where multiple administrators take partial actions, or where administrative responsibilities have been specialized by function area (e.g., all security conditions go to the security operator)). Some environments may also require a differentiation between application errors (e.g., ‘‘object not found in database’’) and system errors (e.g., ‘‘file not found’’ or ‘‘no memory available’’).

(c) Automated operations — Need to be able to run operational ‘‘scripts’’ as a result of an error condition with the intent of automatically correcting the error situation.

(d) Etc.

2.2. Tracing Facilities

This group of requirements deals with the features available to service personnel to follow the events leading up to a failure when an error condition is being recreated.

2.2.1. REQT: Active trace

DCE components need to support an active trace function that can be used for isolation of DCE errors. Active trace implies permanent trace ‘‘hooks’’ that are part of the active DCE code, rather than a compile-time option resulting in two versions of the code ... one with trace active, the other without trace capability. Tracing should be supported in both the client-side and the server-side DCE components.

The objectives of the active trace should be to minimize pathlength when not tracing, minimize additional required storage when not tracing and minimize degradation on the rest of DCE when tracing.

2.2.2. REQT: Consistent trace invocation

Trace function activation should be allowed both at component startup time as well as during component execution. There should be a single consistent trace activation/deactivation command for all DCE components (all system components within DME timeframe). It is to be used to activate/deactivate specific component trace points based on scoping/content selection criteria (see below).

2.2.3. REQT: Common trace function call

Currently any trace information captured by DCE components is handled simply by issuing `printf()` functional calls (also refer to the discussion on error message handling in the previous section). For trace message handling a new trace function call is required.

2.2.4. REQT: Trace scoping

A selective trace can be requested based on the specification of the trace selection criteria. For example, tracing should be allowed to be scoped based on (not all-inclusive list):

(a) All trace points (global trace).

(b) Specific components (e.g., CDS).

(c) Major function within a component (e.g., clearinghouse interactions).

(d) Specific trace point(s).

(e) All components for requests from a given user.

(f) Boolean expression using the above.

(g) Etc.

2.2.5. REQT: Trace entry content

Initially it is sufficient to have the content of the individual trace entries determined by the code developers (ie. ''hardwired'' in the code). Eventually the trace specification needs to allow the selection of specific control block fields/state variables for inclusion as part of the trace data (in the DME timeframe this corresponds to specific content fields of the management objects supported by the code being traced).

2.2.6. REQT: Standardized trace data format

In addition to the variable trace data, each trace entry needs to contain some additional identifying information to aid in understanding the traced function execution. Specifically the following should be included (not all-inclusive list):

(a) Product identifier (DCE).

(b) DCE component identifier.

(c) Function (within component) identifier.

(d) Tracepoint identifier (within function) — Product.component.fns.tracept uniquely identify the point within the code that generated the trace entry.

(e) Request correlator.

(f) Thread/process correlator.

(g) Etc.

Within the trace entry the common identifiers above need to be ALWAYS formatted in an invariant way.

2.2.7. REQT: Output modes

Trace output needs to be able to be directed to a file (full entries) or a display terminal (potentially partial entries).

2.3. Error Data Capture Facilities

This group of requirements deals with the recording of information directly related to an error condition. This data should indicate the cause of the error and the necessary action(s) to be taken.

2.3.1. REQT: 'Before-the-fact' data capture

Many errors (perceived or real) that can occur in a distributed communication environment do not actually result in an error message or an exception (e.g., hung client-server pair), or they trigger secondary error conditions that do not correctly indicate the actual cause of the problem. DCE components need to capture on a continuous basis key state information for each executing thread/process (e.g., in a data area associated with each thread/process). This information is then available to aid in problem determination of these types of errors (via forced dump of memory, or interactive command to ''peek'' the state data). In the DME timeframe this information would be accessible as attributes of the managed object representing the executing process.

2.3.2. REQT: 'After-the-fact' data capture

DCE components need to capture relevant error data for all critical errors. This should include any data areas, control blocks and state information pertinent to the error. A common error logging routine should be defined to record this information using the appropriate system log facilities on each platform. Error data capture should be supported both as part of the code that detected the original error, as well as within exception handling routines.

2.3.3. REQT: First failure data capture

Every attempt must be made to capture all relevant information about a failure such that the failure does not have to be ''recreated'' in order to be diagnosed. Some indication of the necessary actions (if any) required to recover must also be recorded.

Recreating a failure is not appropriate in large systems due to:

(a) Costs incurred by the customer bringing a system offline (e.g., lost sales).

(b) Performance degradation when extensive tracing must remain active for extended time periods in order to capture an intermittent failure.

(c) The possibility that critical data or processing will be affected when the error does recur.

2.3.4. REQT: Real time monitoring

The frequency of certain key events or the state of some key algorithms/ objects within a server, node or network can be used as a way to warn the administrator of impending problems before they become catastrophic. ''Counters'' for certain DCE events should be inserted into DCE code to allow real time monitoring of event frequency (e.g., retry counts). State information should also be recorded for key DCE activity (e.g., RPC conversation state table that keeps track of idle/in use connections/sockets).

Management applications can be developed to monitor event frequency or state information and make decisions about what actions to take based on some user input ''context'' (i.e. how many occurrences of a particular event is normal/abnormal).

2.3.5. REQT: Suppression of data

System administrators should have the ability to bypass the recording of data for errors that are already identified or are considered (to them) benign. Also, since errors can recur frequently after the first occurrence, the suppression of duplicate memory dumps or error records should be supported.

2.3.6. REQT: Standardized error data format

Error data needs to be stored in a standardized form across all components of DCE, always capturing common data areas in a consistent way followed by any function-specific error data. In the DME timeframe this corresponds to the ability to capture key attributes of the managed objects representing the DCE components detecting the error(s).

Error data objects should include things such as (not an all-inclusive list):

(a) Node.

(b) Device/process id (UUID?).

(c) Time of failure.

(d) Symptom keyword (to allow service database lookup).

(e) Reason code (why it happened).

(f) Reaction code (what should be done).

(g) Who to contact (e.g., system programmer, administrator).

(h) Where in the code the failure occurred.

(i) Relevant control blocks.

(j) Memory dump data.

(k) Etc.

2.4. Externals

2.4.1. REQT: Diagnostics guide

The OSF/DCE set of publications needs to be extended to include a ‘‘Diagnostics Guide’’ (or existing OSF/DCE publications need to include a comprehensive diagnostics section). The Guide needs to include the following (not an all-inclusive list):

(a) DCE error conditions.

(b) Impact of the error on the rest of DCE (in ‘‘user’’ terms).

(c) Recovery procedures/steps to correct errors.

(d) Steps to enable/activate/deactivate/disable tracing.

(e) Steps to retrieve any error data captured.

(f) Interpretation of error data.

(g) Description of any administrative commands for serviceability.

(h) Etc.

2.4.2. REQT: Serviceability command interfaces

In general, a single set of administrative commands dealing with serviceability features is required. These commands need to function consistently across all components of DCE. The interface style (command syntax, argument conventions, etc.) needs to be consistent with the rest of the DCE administrative functions.

2.4.3. REQT: Scope

Serviceability commands need to be available not only on each local DCE system, but also be accessible remotely across a network (e.g., ability to activate a trace on a remote server system). Over time each of the supported commands needs to be migrated to its corresponding integrated DME command.

2.4.4. REQT: Output consistency

In general, the output generated by the various serviceability functions needs to be formatted in a consistent way across all DCE components (e.g., trace entries should have a common structure regardless which component generated them, only the component-specific detail differs).

2.4.5. REQT: Callable service functions

DCE needs to provide a common set of callable serviceability functions as outlined in the previous sections. These need to be used by ALL existing and future components of DCE. Where possible the definition of the callable interfaces should be consistent with any existing or emerging industry standards for such services (e.g., X/OPEN ‘‘Issue 3’’ for message isolation).

3. PROPOSED PRIORITIES/IMPLEMENTATION STAGING

The requirements described in this paper are prioritized based on the anticipated rollout of OSF/DCE future releases (based on ‘‘guidelines’’ stated by Doug Hartman at a prior DCE SIG meeting). The following approximate timeframes are assumed to be associated with each of the release identifiers:

(a) 1.0.1 — 1Q/92.

(b) 1.1 — 4Q/92.

(c) 1.1.* — 1Q/93 to 3Q/93.

(d) 2.0 — Y/E 93 (2+ years after DCE 1.0, assumed to include integrated DME support).

The serviceability requirements are prioritized based on the release timeframe. Please refer to the previous sections for detailed description of each line item.

ERROR NOTIFICATION:	1.0.1	1.1	1.1.*	2.0
Error text isolation	X			
Common error function calls		X		
Standardized error message format	X			
Remote notification		X		
DCE exception signals		X		
Multiple layer error processing		X		
Message suppression			X	
Message destination selection			X	
Automated operations ("scripts")				X

TRACING:	1.0.1	1.1	1.1.*	2.0
Active trace hooks		X		
Consistent trace invocation		X		
Common trace function call		X		
Scoping support				
- simple selection criteria		X		
- boolean expression			X	
Trace entry content				
- "hardwired"		X		
- selectable				X
Standardized trace data format		X		
Output modes		X		

ERROR DATA CAPTURE:	1.0.1	1.1	1.1.*	2.0
"Before-the-fact" (state tables)			X	
"After-the-fact"		X		
First failure data capture		X		
Real time monitoring			X	
Suppression of data			X	
Common error_log function call		X		
Standardized error data format		X		

EXTERNALS:	1.0.1	1.1	1.1.*	2.0
Diagnostics guide	X	X	X	X
Consistent command interface		X		
Local/remote command invocation		X		
Standardized serviceability output		(see above)		
Common callable system functions		(see above)		

4. RELATIONSHIP TO OTHER DCE WORKGROUPS AND OSF SIGS

4.1. I18N

Initially, the most important I18N consideration is the separation of message text into catalogs since this information appears to end users or to the customer's system operators/administrators. However, trace and error logs should also utilize I18N concepts related to language translation although the urgency is not immediately apparent. Error data in the form of memory dumps, control blocks or as keywords should not have to follow I18N conventions.

If serviceability records/objects are to be routed to a remote node then character set implications must be considered (similar to the character set problems faced by DFS and the Directory Services when distributing data).

4.2. Security

Interfaces developed to control DCE server serviceability features must use proper access control to restrict their use. Serviceability data must also be protected from general access.

4.3. DFS

Serviceability records may be stored as DFS files in order to facilitate remote access or distribution of data before DME object management is available.

4.4. DME

Currently DME migration is not sufficiently addressed in this version of the requirements paper (other than cursory references to possible DME relationships to some of the serviceability functions).

The management of DCE serviceability data is critical to easing the servicing of a distributed system. Proper integration of DCE and DME must occur. The next steps for ensuring this integration are:

(a) Joint DME/DCE SIG workgroup to define DCE serviceability objects to be managed by DME.

(b) Joint DME/DCE SIG requirements/recommendation — Date TBD.

APPENDIX A. RATIONALE

The serviceability items described in this paper can be divided into 6 ''rationale'' classifications which represent the thoughts behind why they are needed within DCE.

These classifications include:

(a) Usability (U).

It is desirable to have all software on a system (and within a network) use similar ways of activating diagnostic services and/or accessing diagnostic information in order to reduce the learning curve of service personnel when a new product is introduced. Each application (and in the case of DCE, each service) having different ways of implementing error notification, tracing and error data capture, makes the servicing/debugging of system problems unnecessarily difficult. With similar activation mechanisms, similar data formats and similar logging strategies within the code, DCE becomes easier to service. As well, general tools for formatting and analyzing DCE diagnostic data are easier to develop.

(b) Integratability (I).

Common serviceability functions and services within an operating system lead to a uniform serviceability approach (e.g., common strategies, common data formats and log files, consistent commands, reusable tools and utilities, ...) by all the applications that run on it. For those platforms that do offer functions and/or services for error notification, tracing and error data capture, the ability to exploit these services and maintain the system's approach to serviceability is considered desirable (would a common set of serviceability functions/services exist in OSF/1 or in DME?). Using specific function calls within DCE code for each of the three serviceability services (mentioned earlier) allows for the serviceability strategy within DCE to be easily integrated with that of the operating system.

(c) Customizability (C).

Allowing the customer or service organization the ability to customize diagnostic data, such as allowing messages to be easily translated to other languages, is a valuable feature. Another example of customization is where the application client or server (running on top of DCE services) can customize their serviceability strategy by intercepting messages or errors logged by DCE client code.

(d) Scaleability (S).

In order for DCE to run on larger machines, the nature of large systems has to be considered. Mainly, the fact that large systems run so much software and support so many users on a single node, means that it is very easy to overload the person responsible for managing the system with diagnostic data. To avoid this problem, features that allow different types of information to be separated (e.g tracing info, error info) are required. As well, many customers like to have larger systems (and networks) run unattended, relying on remote notification, rerouting of diagnostic data and automatic handling of problems.

(e) Increased Diagnostic Support (D).

These are features that increase the diagnostic information or capabilities available to service personnel. As well, features that would make service tools easier to create (such as standardized output formats) also qualify.

(f) Distributed Network Capabilities (N).

One of the least mature serviceability issues deals with how it should be handled within a distributed network. Distributed methods of diagnosing and handling errors will be required in the near future of DCE (addressed by DME?). This paper only goes as far as including remote notification of errors and remote command invocation as ways to expand a local serviceability strategy into a distributed network serviceability strategy. Defining DCE serviceability objects (e.g., error data objects) and how they should be managed is not addressed.

The following list places each of the serviceability features under the classifications described earlier:

```
--------------------------------------------------
ERROR NOTIFICATION:
--------------------------------------------------
Message text isolation
- message numbering                          D
- message translation                        U
- message tailoring/standardization          C
Remote notification                          S,N
DCE exception signals                        C
Multiple layer error processing              D,C
Message suppression                          S
Message destination selection                S,N
Automated operations (e.g., retries)         S
Common error function call                   I
Standardized message format                  U,D
--------------------------------------------------
TRACING:
--------------------------------------------------
Active trace hooks                           D
Consistent trace invocation                  U
Scoping support                              D,S
Selective content specification              D,S,C
Output modes                                 S,I
Common trace function call                   I
Standardized trace data format               U,D
```

ERROR DATA CAPTURE:	
"Before-the-fact" logging	D
"After-the-fact" logging	D
First failure data capture	D
Real time monitoring	D
Suppression of data	S
Common error log function call	I
Standardized error data format	U,D
EXTERNALS:	
Diagnostic guide	D
Consistent command interfaces	U
Local/remote command invocation/viewing	N
Diagnosis utilities (format and analysis)	D
Common callable system functions	I
Standardized serviceability output	U,D

APPENDIX B. COMMON CALLABLE SERVICE SUGGESTIONS

TBD

(a) Error Notification

(b) Tracing

(c) Error Data Capture

APPENDIX C. SERVICEABILITY WORKGROUP MEMBERSHIP

The Serviceability Workgroup membership is the following:

(a) Steve Akers (HP)

(b) Sandy Gregus (IBM)

(c) Dave Hinman (SNI)

(d) Mark Hubbard (IBM) — Workgroup leader

(e) Vlad Klicnik (IBM)

(f) Gerard Meyer (Bull)

(g) Frank Robbins (DEC)

The workgroup's maillist is:

```
sig-dce-service@osf.org
```

AUTHOR'S ADDRESS

Mark Hubbard
Distributed System Services
IBM Canada Laboratory
Stn. 2G, 1150 Eglinton Avenue East
Toronto, Ontario
CANADA

Internet email: hubbard@torolab5.vnet.ibm.com
Telephone: +1-416-448-3919