The Remote Messaging System: An Overview

1 Introduction
This document is intended to be a fairly high level introduction to RMS: what it is and how it works from the perspective of a user wanting to send and receive messages. The actual design and implementation is not covered: for that, see the “Documentation” folder in the “Messaging System” project in StarTeam.
Note that sometimes RMS stands for “Remote Messaging System”, sometimes for “Remote Management System”, but it’s always the same thing.

2 Contents
1 Introduction ................................................................................................................. 1
2 Contents ......................................................................................................................1
3 Messages ..................................................................................................................... 2
  3.1 Message Type ..................................................................................................... 2
  3.2 Message Data ...................................................................................................... 2
  3.3 Destination .......................................................................................................... 2
  3.4 Originator ............................................................................................................ 2
  3.5 Time to Live ........................................................................................................ 3
4 Components ................................................................................................................ 4
  4.1 Message Router ................................................................................................. 4
  4.2 Certification Manager ......................................................................................... 4
  4.3 Management Agent ............................................................................................. 4
  4.4 AutoUpdate Agent ............................................................................................... 5
  4.5 EM Library Agent ............................................................................................... 5
  4.6 Other Clients ....................................................................................................... 5
5 Component Interaction .............................................................................................. 6
  5.1 Finding Components ........................................................................................... 6
  5.2 CORBA ............................................................................................................... 7
  5.3 Logging On ......................................................................................................... 8
  5.4 Message Routing ................................................................................................. 9
  5.5 Message Persistence .......................................................................................... 11
6 RMS in Use ............................................................................................................... 12
  6.1 Installation ......................................................................................................... 12
  6.2 Registry Settings ............................................................................................... 12
  6.3 DNS and IP Addresses ...................................................................................... 13
  6.4 Firewalls ............................................................................................................ 14
  6.5 Alternate Topologies ......................................................................................... 15
  6.6 Resolving Problems .......................................................................................... 15
7 Further Reading ........................................................................................................ 16
3 Messages

The fundamental purpose of RMS is to securely transmit messages between computers. In RMS terms, a message consists of a collection of data elements bundled together. What follows is a description of the more important elements of a message: those elements not covered can be assumed to be related to the internal workings of RMS and not directly relevant to programs just wanting to send and receive messages.

3.1 Message Type

This is a string that identifies the contents of the message. For example, when requesting the status of an agent, a message of type “EM-GetStatus” is sent to the agent; in return the agent sends a message of type “EM-GetStatus-Reply”.

3.2 Message Data

The actual contents of the message. This is a block of data in some particular format. The end result is that it is a block of structured data (for example, a string followed by an integer followed by a boolean).

Usually messages of a given message type all have message data in the same format, but this is not required. Message routers do not need to be able to understand the format of the message data of all messages: all that is required is that the initial sender and the final destination of the message know how to interpret it.

3.3 Destination

This is a string describing where the message is to be sent. Message routing is covered further on in this document, but as a simple example, suppose the destination of a message is “Router1.Agent” (note the full stop separating the elements of the destination). This means that the router processing this message should send it to “Router1”, which should then send it to “Agent”.

3.4 Originator

This field acts as the inverse of the destination. When the message is received at its final destination, the originator field contains the names of the components through which the message has passed on its way from start to finish. This allows components to reply to messages: if the destination field of a reply message is set to the originator field of a message that is interpreted as a request for information, then the reply message will be delivered to the sender of the request message.

---

1 The format is specified by a CORBA typecode structure, which is explained in excruciating detail in the CORBA references given at the end of this document.
3.5 Time to Live

This is a number that gives an indication of how long the message can be considered valid for. If a router cannot deliver a message to another router or a client of RMS, it will keep the message until the recipient is available. However, if the message has been held for the router for longer than the “time to live” value, then the message is discarded, helping prevent large numbers of messages building up in queues, waiting for a currently unavailable computer.

If this value is zero, the message does not expire.
4 Components
What is referred to as RMS is really a set of interacting components. These components can be divided up as follows:

- The message router
- The certification manager
- The agents (such as the management agent and the AutoUpdate agent)

In addition, there are other users of RMS that are not themselves part of RMS, such as the ES and SBS management services.

4.1 Message Router
This is the component that appears on computers as the service “Sophos Message Router”. The message router is the fundamental unit of the messaging system. The router has two principal functions:

1) To send and receive messages between different computers.
2) To store messages if the recipient of the message is not available.

Client programs (such as agents) connect to a specific router (usually on the same machine as the client program, but not necessarily) and use that router to send and receive messages.

Components using RMS communicate with each other by using message routers to pass messages back and forth. The routers do not interpret or understand the format of these messages, just merely pass them on to their destination.

4.2 Certification Manager
This is the component that appears on the server computer as the service “Sophos Certification Manager”. It is responsible for issuing security certificates. Any entity attempting to use the messaging system must present a valid security certificate to the router it is attempting to log on to.

4.3 Management Agent
This is the component that appears on managed computers as the service “Sophos Agent”. It is responsible for reporting the state of the managed computer to the server. To do this, it communicates via RMS with other, subsidiary agent components on the same computer.

The management agent also has another mechanism for communicating with other components by way of “adapter” DLLs that are loaded by the management agent. This provides a way for (say) SAV to report its status to the agent without requiring the overhead of a separate subsidiary agent process. In time all the subsidiary agent components will be converted to be adapters.
4.4 AutoUpdate Agent
This is the component that appears on managed computers as the service “Sophos AutoUpdate Agent”. It is responsible for reporting the state of AutoUpdate to the management agent, and passing on messages to change the configuration and run an update to AutoUpdate.

4.5 EM Library Agent
This is the component that appears on computers running the Enterprise Suite version of EM Library as the service “Sophos EMLibUpdate Agent”. It is responsible for reporting details of the CIDs available from that machine to the server.

4.6 Other Clients
The above list of components is what composes (more or less) what is referred to as “RMS”. However, components outside of RMS can make use of it. The most significant of these is the management service, which runs on the server. The management service communicates with RMS, the database back end and the GUI front end, using incoming RMS messages to update the database and converting user input in the front end to RMS messages sent to managed computers.
5 Component Interaction

As stated previously, all components usually talk to a router on the same computer, while only routers communicate between computers. In a typical installation, this results in connections between RMS components as shown in figure 1:

![Schematic representation of communication between RMS components]

**Figure 1:** Schematic representation of communication between RMS components

5.1 Finding Components

To understand how RMS operates, it is important to understand how the connections between components (as shown in figure 1) are made. These connections are transient: that is, they are recreated every time the components involved are started (usually when the computer is rebooted).

Connections are established in a fixed order. This order is represented in figure 1 by the vertical position of the component. Following figure 1, connections are established from the component at the bottom end of the line (the child) to the component at the top end of the line (the parent, which is always a router). Thus, the agents on the client establish connections to the router on the client; the certification manager and the management service on the server establish connections to the router on the server; and the router on the client establishes a connection to the router on the server.

In order to establish a connection, a child must have some way of finding its parent router. To make this possible, each router has an open port from which information on how to connect to it can be read by anyone, which is known as the “IOR sender port”, for reasons covered below. The number of this open port is usually fixed at 8192\(^2\). You can see this information in its raw form yourself (though you won’t learn anything much!) by running the following in a command prompt on a computer with RMS installed:

\[^2\text{The port is set in the MRInit.conf file used to initialise RMS. 8192 is 2000 in hexadecimal, if you’re trying to figure out which line in the file it is.}\]
This information allows access to the router via CORBA.

5.2 CORBA

CORBA (Common Object Request Broker Architecture) is a standard designed to make it easier to write programs that interact over a network. In CORBA’s programming model, programs expose abstract entities called CORBA objects. These objects are described in a declarative (that is, only an interface, no implementation) language called IDL (Interface Definition Language), which allows the developer to specify functions that go to make up the object. When one program (the “client”) has a reference to an object exposed by another program (the “server”), then the client can call functions on the server object to make the server do work for it.

This communication between the server and client is done with a standard transport layer (usually TCP/IP, the networking protocol of the Internet). From the point of view of writing the client program calling a function in a CORBA object exposed by the server looks like calling any other function, but the CORBA libraries linked into the server and client programs result in communication over the network that leads to code running on the server and returning the result to the client.

An important point to understand from the above is that CORBA is not something that the end user need be exposed to: from their perspective, all that is going on is that programs are communicating via TCP/IP. The CORBA implementation is just a library linked into all the programs that acts as a link between a “high level” layer of the program, which operates in terms of CORBA objects; and a “low level” layer, which inter-operates with the operating system’s TCP/IP implementation:
The use of CORBA does not require anything else on the network other than TCP/IP.

One unanswered question is how does a client program obtain a reference to a CORBA object from another program in the first place. The answer is that such a reference (an “IOR”) can have several forms, which the CORBA library can translate between. In a C++ program it can be a C++ object, or it can also be a string of text. This explains the use of the information obtained from the IOR sender port, as discussed in the previous section. This information is actually a CORBA object reference in string form (a “stringified IOR”), which the client program can turn into a C++ object on which it can make calls.

### 5.3 Logging On

As covered in the first section, when a component starts up it attempts to get information from its parent router that it uses to create a CORBA object reference to a CORBA object exposed by that parent router.

This object allows access to other CORBA objects, on which the component can make calls to log itself onto the parent router. Once successfully logged on, the component can use the router to send and receive messages.

The act of logging on to a router requires that the component logging on authenticate itself. This involves presenting a security certificate to the router, which provides details of who the component is, and is used by the router to verify that the component is not attempting to pretend to be something it is not. Security is a complex and difficult subject, which is beyond the scope of this document (but see the section on further reading below). For now, all that needs to be understood is that each component that uses RMS must possess a certificate, which it acquires from the certification manager when it is installed.
Once a component has succeeded in logging on, it is given a reference to a CORBA object exposed by the router that allows it to make calls to send and receive messages. During logon the component can elect to either have messages for it pushed to it, or for it to have to periodically check (by making a CORBA call) if there are messages for it at the router.

The above discussion of logging onto a router has used the term “component” for that which is doing the logging on, rather than “agent” or “child router”. This is because the way in which components such as agents connect to their router is the same as how the router on a client computer connects to the router on a server computer: the only difference is that a component such as an agent reads the information on how to connect from the IOR sender port on the local computer, whereas the router will be reading the information from the IOR sender port on a different computer.

5.4 Message Routing

Having logged onto a router, a component can now send and receive messages. The question is then who to send messages to?

Every component using the RMS has a name, which need not be unique. This name is embedded in the component’s security certificate and cannot be changed. Generally, users of RMS such as the agents and the management service have well defined names. For example, the management agent is always named “Agent”. Routers, on the other hand, have a name derived from the name of the computer on which they are running. Figure 3 shows the same diagram as figure 1 with the addition of the RMS names of the components in bold. The names of the routers are chosen as suitable examples, while the names of all the other components are as they are in SBS and ES (some of which represent historical accidents).

![Figure 3: A simple server client setup with RMS names shown.](image-url)
When a router is given a message to deliver, it looks at the destination field of the message. This field is a list of names of RMS components, separated by full stops. If the first name in the destination matches any of the components logged onto that router, then the router removes that name from the message field and passes the message on to the appropriate component.

To give a real-world example, suppose the management agent wanted to send a message to the AutoUpdate agent running on the same computer. The destination would be “ALC”. The management agent gives this message to the router on the client, which notices that the AutoUpdate agent is logged on with the name “ALC”, so the router gives the message to the AutoUpdate agent.

A problem arises when the management agent wants to send a message to the management service on the server: It does not know the name of the server’s router, so it does not know what to put in the destination field. To solve this problem, a further rule is present in routing messages: If a router cannot find anyone to send a message to, it sends that message to its parent router. (If the router on the server, which has no parent, cannot find a suitable destination for the message, then message delivery fails.) Thus to send a message to the management service, the management agent uses a destination of “EM”. It gives the message to the client router. The client router cannot find anyone to send it to, as no-one is logged onto the client router as “EM”, so it gives the message to its parent router, the router on the server. This router does have a component logged on as “EM”, so it gives the message to that component, the management service.

The above describes what happens when an agent is started up. When the management service receives this message from the management agent, it looks at the database to see if it has been contacted by that computer before. If not, the client must be a newly bootstrapped machine: in that case an entry is added to the database, and a message sent back to the client’s agent to instruct it on the configuration to apply to AutoUpdate and SAV.

In order to send this reply back, the management service needs to put in a suitable field in the destination. To do this, it looks at the originator field of the message from the client. Every time a message passes through a router that router adds the name of the sender in the same format as used for the message destination. As the message travels from the agent to the management service via the two routers, the originator field will be as follows:

<table>
<thead>
<tr>
<th>Component Message is Sent From</th>
<th>Originator Field of Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent</td>
<td></td>
</tr>
<tr>
<td>Router$Client</td>
<td>Agent</td>
</tr>
<tr>
<td>Router$Server</td>
<td>Router$Client.Agent</td>
</tr>
</tbody>
</table>
From this, the management service knows it can send a message back to the client’s agent by putting “Router$Client.Agent” in the destination field of the message. When the management service sends the reply, it gives the message to the router on the server. The server router looks at the first part of the destination, “Router$Client”. Since there is a client router logged onto the server router with that name, the server router hands off the message to the client router, adding “Router$Server” to the originator field and removing “Router$Client” from the destination field. Thus the client router receives a message with a destination of “Agent”, which it hands to the final destination, the management agent.

5.5 Message Persistence

As soon as the router receives a message, it is written to a file on disk. In normal operation this file is not looked at again, and is deleted once the router has successfully dispatched the message towards its destination.

The purpose of writing the message to a file in this way is to provide message persistence. If a message cannot be immediately delivered to its recipient, the router must wait for that recipient to log on. If the router is restarted during that time (for example, because the computer is rebooted) then the message must not be lost. This is achieved by the router scanning a directory for message files when it is started up. If any such files are found they are taken to be messages waiting for delivery, and are read back into the router’s queue of pending messages.
6 RMS in Use

6.1 Installation

The installer for RMS requires the presence of two configuration files: “cac.pem”, which contains the certificate manager’s public key, required for validating the server router during logon, and “MRInit.conf”, which contains various registry settings discussed in more detail below. In SBS and ES these files are created during server installation and copied into the CID directories.

When the installer is run the program files are copied across and registry settings created based on the above files. Services are created for the router and the agents, and these services are started.

When the router is first started, it obtains a certificate from the certificate manager. It does this by logging onto its parent router through a special interface that allows sending messages to the certificate manager only. Once a certificate is obtained the router logs onto the parent router through the normal interface discussed above.

Agents also need certificates. When they start they wait for the router to initialise itself, then repeat the router’s procedure for obtaining a certificate from the certificate manager.

Once the management agent has a certificate it attempts to determine what RMS components are present on the computer, and then sends a status message to the server. It is this first status message that causes a newly installed computer to appear in the SBS or ES console windows.

6.2 Registry Settings

Installation of RMS sets up a number of registry keys concerned with the router which can be edited. (Agents also have a few registry keys, but these should not be changed in most circumstances.)

All such router registry keys can be found under

```
HKEY_LOCAL_MACHINE\SOFTWARE\Sophos\Messaging System\Router
```

Following is a description of some of the keys. Keys not mentioned here should not be edited in most circumstances. In all cases it is necessary to restart the router to cause edits to these keys to take effect: the easiest way to achieve this is to restart the computer.

**ConnectionCache**

This value controls the size of a cache the router maintains of network connections to other RMS components. Increasing this value allows more components to connect without having to constantly re-establish network connections, but also increases the amount of memory used by the router. The default values are 10 for client computers and 2000 for the server. When the computer is accepting router connections from other computers (as with the server) a value of about double the number of client computers is recommended.
IORSenderPort
The number of the port used by the router to broadcast its IOR, as described in the section on finding components. The usual value is 8192.

LogFileCount
The number of log files that are kept. If the number of log files is exceeded, the oldest log file is deleted. The usual value is 4.

LogFileMaxSize
The maximum size (in bytes) of a log file. Once this size is exceeded the log file is closed and a new one opened. The usual maximum size is one megabyte.

LogLevel
This value controls the amount of information written to the log file. The default value of 0 writes information, warning and error messages. A value of 1 also adds debug messages, and a value of 2 tracing messages. In general these higher values provide information helpful only to Software Engineering.

ParentAddress
The IP address or DNS name of the computer running the parent router. If the field is blank, then this computer must be the server.

Starting with ES 1.0, this field can be a comma separated list of IP addresses and DNS names, which are tried in sequence.

ParentPort
The port number on which the parent router is broadcasting its IOR. As for IORSenderPort, the default value is 8192.

6.3 DNS and IP Addresses
RMS uses TCP as its low-level networking protocol. In TCP, there are two ways of addressing a particular computer: by IP address (e.g. 10.7.104.65) or by DNS name (e.g. client.test.net). A network could use only IP addresses, or both IP and DNS. If DNS is available on the network then there should be a DNS server present that provides mapping from DNS name to IP address (forward DNS lookup) and from IP address back to DNS name (reverse DNS lookup). To further complicate matters, computers can either have fixed IP addresses that do not change when the machine is rebooted, or dynamic IP addresses that are assigned by a DHCP server.

The transient network connections between RMS components (that is, the CORBA connections that are remade each time an RMS component starts up) always use IP addresses.

The only use of DNS by RMS is when the router on a client computer needs to find its parent router, as discussed above in the section on finding components. When RMS is installed, the installer writes a registry entry containing the DNS name or IP address of the computer running the parent router. This is the ParentAddress entry discussed above.
Whether ParentAddress on clients is an IP address or DNS name is determined when the server is first installed. The server installer examines the server to see if its IP address is dynamic or not. If not, clients will use the server’s static IP address, otherwise they will use its DNS name.

Thus, the only use of DNS by RMS occurs when the server computer has a dynamic IP address. In this case, it is required that DNS be working sufficiently well that a forward DNS lookup from any client of the server’s DNS name give the correct IP address. In general, it is simplest if customers install the server on a computer with a static IP address, as this removes any need to worry about DNS.

### 6.4 Firewalls

A firewall is a piece of hardware or software that restricts network connections between a group of computers (those “inside” or “behind” the firewall) and the rest of the network (or Internet). Typically, a firewall will allow a computer behind the firewall to establish a network connection with one outside, but not vice versa. Thus, a computer behind a firewall might be able to contact a web server on the Internet to request a web page. The firewall will allow the response from the web server back through the firewall because it knows that it was requested, but will ignore any other connection attempts from that web server.

The commonest RMS problem with a firewall is when a client computer is running a firewall (such as the one built into Windows XP SP2). The router on the client computer is able to send messages to the router on the server computer, but the firewall prevents the router on the server sending messages to the client.

This situation is detected when the client router logs onto the server router. When this happens the client router switches to a new mode, in which it periodically calls the server router to see if there are any messages for it, rather than waiting for the server router to push messages to it. This allows RMS to function through a firewall, but leads to slow message delivery: typically, it can take several minutes for messages to arrive through this mechanism. Starting with ES 1.0, routers will put a message into their logs warning about the possible presence of a firewall in this case.

A better solution is to open a hole in the firewall for RMS to go through. Two TCP ports need to be opened on the firewall, which by default are 8193 and 8194. (The values are determined by the MRInit.conf file created by the server installer.)
6.5 Alternate Topologies

All the examples of RMS in this document have a very simple topology, with multiple client routers connecting to a single server router. This is the default arrangement for SBS and ES.

However, other arrangements of routers are possible. In particular, in ES it is possible to have clients that do not connect directly to the server, but instead connect to another client. This arrangement avoids problems on networks where direct connections to the server are not be possible.

![Diagram of Alternate Topology]

Figure 4: Computers with connections between routers shown.

In figure 4, the routers on client computers 1 and 2 connect directly to the server router, but on client computers 3 and 4 they connect to the router on the intermediate computer, which is itself a client of the server.

To arrive at the above topology, the intermediate computer is initially installed as an ordinary client of the server. The router configuration is then adjusted to allow other routers to connect to it, and an alternate installer set (such as a CID) is set up in which the parent router is specified to be that running on the intermediate computer. This set is then used to install RMS on client computers 3 and 4.

6.6 Resolving Problems

When problems occur with RMS, it can be difficult to determine where the fault really lies. If RMS is suspected as not functioning, it is usually best to start with the management agent logs on the client, then look at the router on the client, then the router on the server. In each case, the question is whether the component is behaving correctly, or is failing, or is failing because another RMS component it depends on is not working.
7 Further Reading

To learn more about CORBA, there are many books available, but most do not cover the subject in sufficient depth. The best book, though now slightly out of date, is “Advanced CORBA Programming with C++” by Michi Henning and Steve Vinoski, published by Addison-Wesley. As the title implies, to get the most out of the book it is necessary to have at least a reasonable understanding of C++ programming.


Security and cryptography is an even bigger topic, with even more books available. Since it is also a very complex topic, a lot of what is available is of questionable value. For a programming overview with some mathematics, the standard work is “Applied Cryptography” by Bruce Schneier, published by Wiley. For a non-mathematical overview of security and why it often fails, see the very readable “Secrets & Lies”, by the same author.