TECHNICAL SCIENCES
Abbrev.: Techn. Sc., No 15(1), Y 2012

ENHANCING SEAMLESS DATA TRANSFER WITHIN COMPLEX MESH ENVIRONMENTS WITH SECURE AGENTS

***Phil Entwisle*[1], *Steve Lawrence*[1], *Ondrej Habala*[2]**

[1] Qinetiq Ltd, Portsdown Technology Park, United Kingdom
[2] Institute of Informaties Slovak Academy of Sciences, Bratislava, Slovakia

K e y w o r d s: Secure Agent Infrastructure (SAI), Multi Bearer Router (MBR), seamless data transfer, network optimization.

A b s t r a c t

QinetiQ's Multi-Bearer Router (MBR) has been designed to operate in harsh-dynamic environments and provide seamless data transfer within these situations. During certain network environments, specifically complex mesh, it has been witnessed that the MBR has not chosen the most appropriate interface quick enough to maintain seamless data transfer. This is mainly due to the MBR basing all of its decisions on local knowledge; hence a network change at two or more routers away can take too long to filter through the network for all MBRs to have up-to-date routing knowledge. Thus it was planned that a new MBR component would be created to assist in the distribution of all remote MBRs local knowledge to create a central global knowledge that can then be tailored to each MBR on the network. With updated remote knowledge the MBR can better support seamless data exchange in complex network environments without a large burden on the network or its constraints.

ZWIĘKSZENIE EFEKTYWNOŚCI „BEZSZWOWEJ" TRANSMISJI DANYCH W ZŁOŻONYM ŚRODOWISKU SIECIOWYM PRZEZ ZASTOSOWANIE SECURE AGENT

***Phil Entwisle*[1], *Steve Lawrence*[1], *Ondrej Habala*[2]**

[1] Qinetiq Ltd, Portsdown Technology Park, United Kingdom
[2] Institute of Informaties Slovak Academy of Sciences, Bratislava, Slovakia

S ł o w a k l u c z o w e: Secure Agent Infrastructure (SAI), Multi Bearer Router (MBR), „bezszwowa" transmisja danych, optymalizacja ruchu sieciowego.

Abstrakt

Opracowane przez QinetiQ rozwiązanie Multi-Bearer Router (MBR) służy do zapewniania „bezszwowej" transmisji danych w szczególnie trudnym i zmiennym otoczeniu. Zaobserwowano, że w niektórych środowiskach sieciowych (w szczególności przy złożonych sieciach kratowych) MBR nie zawsze wybiera optymalne trasowanie. Głównym powodem takiego stanu rzeczy jest fakt, że decyzje w MBR są podejmowane na podstawie wiedzy dostępnej lokalnie. Zaobserwowano, że czas propagacji informacji bywa za długi, by cała sieć MBR miała aktualne informacje o dostępnych trasach. W następnej wersji komponentu MBR przewidziano wsparcie do budowy scentralizowanej i globalnej wiedzy. Byłaby ona dostępna dla poszczególnych urządzeń sieciowych, a te mogłyby wykorzystać i dostosować tę wiedzę do własnych potrzeb. Poszczególne routery MBR – dysponujące taką wiedzą – byłyby w stanie lepiej wspierać „bezszwową" wymianę danych w złożonych środowiskach sieciowych, z jednoczesnym minimalizowaniem obciążenia sieci.

Introduction

Initially the QinetiQ Multi-Bearer Routers (MBR) make all their decisions locally, occasionally it has to make assumptions whether or not the next MBR in-line has access to the required networks. Therefore, a need arises to allow MBRs to communicate and pass their local knowledge to all other MBRs for dissemination of the network routing information, especially with regard to meshed networks. After discussions and a couple of meetings with Ustav informatiky Slovenska akademia vied (UI SAV) it was decided that the Secure Agent Infrastructure (SAI) could help solve this mesh-network situation and ensure that all MBRs have an up-to-date local knowledge.

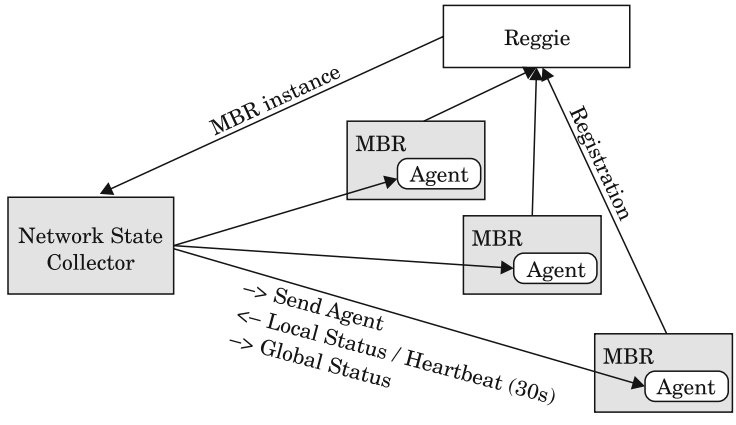

Fig. 1. Overview of SA & MBR Integration

The design was broken into three core sections:

1. MBR Interface – QinetiQ
2. MBR Secure Agent – UI SAV
3. Network State Collector (NSC) – QinetiQ & UI SAV

The MBR Interface would enable communications between the Agent and the MBR to ensure the local knowledge is passed to the NSC and the tailored remote knowledge is passed back into the MBR policy-engine for better decision making.

The MBR Secure Agent would be responsible for going to all registered MBRs on the network and passing the data back and forth between the appropriate locations.

The NSC will take all the information gathered from the MBRs in the network and build a representation of the topology so that it can be tailored to each MBR and returned via the Agent.

Components

MBR Interface

The MBR Interface to the SAI will allow the agent to request what is referred to as MBR local knowledge, i.e. the network state in which that MBR is currently in, the subscribing subnet and local state of bearers. The Agent will ensure each MBR's local knowledge is passed to the NSC to calculate the wider picture of the network. This in turn is used to create the unique remote knowledge for each MBR and the agent will ensure that each MBR receives its personal remote knowledge picture.

For example, consider the simple three node MBR network in Figure 2 below.

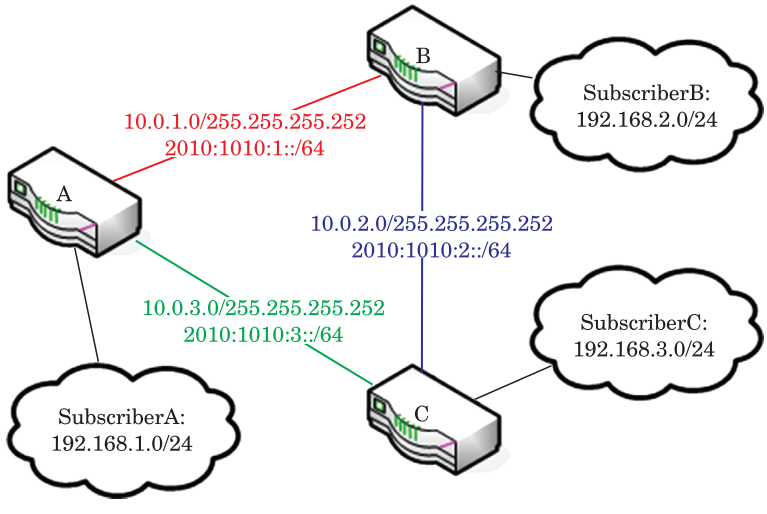

Fig. 2. Three Node MBR Network

MBR A has a direct link with MBR B (red line) and MBR C (green line), but has no knowledge about the status of the link between MBR B and C (blue line). However, if each MBR collates its own knowledge of the network and distributes this to a central server the state of all links can be disseminated to all MBRs.

Given a situation, such that the link between MBR B and C is unavailable and the link between MBR A and B is unavailable.

– MBR A senses its loss of link to MBR B
– MBR C senses its loss of link to MBR B
– MBR B senses its loss of links to MBR A and C

However, MBR A will "expect" that MBR C is still connected with MBR B but has no method of validation, however, with both MBR A and C advertising its local knowledge MBR A can now "learn" that C has not got a connection to MBR B and MBR C would "learn" that MBR A has not got a connection to MBR B. Therefore, neither MBR would allow traffic to traverse the network which is destined for MBR B, thus reducing the amount of bandwidth that is utilised and helping to ensure all accessible services are supportable through the network.

The proposed component would bridge the MBR Orchestration Layer (`orchd`) with the SAI. The orchd application is the core daemon and message bus for MBR operations, enabling components to communicate with each other along with the Linux kernel. The Orchestration Layer utilises a SHA-1 encryption and comprehensive packing functionality to allow secure and compact communications. Therefore to ensure compatibility a new component (`open-interface`) will expose certain MBR functionality to the Agent via specific messages passed over a loopback network socket. This functionality will consist of:

– Register MBR – This function would register the MBR with the "reggie" service (expected to be running with the NSC) to enable Agent communications.

– Retrieve Local Knowledge – This message type returns the specific MBRs current knowledge of the status and network configuration of each of the network interfaces and the serving Local Area Networks. This is one of the two core messages required to enhance the MBRs knowledge, the Agent will collect this information from all MBRs in the network and submit it to the NSC for processing.

– Update Remote Knowledge – This message type accepts the tailored updated remote knowledge calculated by the NSC. This is the second core message type required for the enhancements by giving the MBR an understanding of the topology of the network to know which communication bearers access which served Local Area Networks.

– Change Running Configuration Set – This function would allow the Agent to alter the current running configuration set to another one stored on the MBR. An added bonus being that an Agent could be sent out to all registered MBRs in the network and told to start running a pre-set configuration, such as an emergency configuration to disable any restrictions on network access. Currently only capable through the MBR Web Interface (web-utility) and acts as proof that other web functionality could be ported across.

– Retrieve Policy Table – This function returns a copy of the specific MBRs installed policy table, both IPv4 and IPv6. Another added bonus being that an Agent could be sent out to all registered MBRs and bring back a copy of each policy table to build an exact replica for management purposes. Currently only capable through the MBR Simple Network Management Protocol (SNMP) Interface (network-agent) and acts as proof that other SNMP functionality could also be ported across.

The local knowledge is passed back to the NSC for processing into global knowledge (topology) of the entire network. The NSC then tailors the remote knowledge for each MBR, giving it a personal view of the entire network accessible to them. Taking the previous simple network example (Fig. 2) in a full mesh:

– MBR A can be told:
  – MBR B is reachable via both MBR B and C
  – MBR C is reachable via both MBR B and C
  – MBR B is directly connected via interface one
  – MBR C is directly connected via interface two
– MBR B can be told:
  – MBR A is reachable via both MBR A and C
  – MBR C is reachable via both MBR A and C
  – MBR A is directly connected via interface one
  – MBR C is directly connected via interface two
– MBR C can be told:
  – MBR A is reachable via both MBR A and B
  – MBR B is reachable via both MBR A and B
  – MBR A is directly connected via interface one
  – MBR B is directly connected via interface two

Therefore, if any interface changes state while parts of the mobile mesh network move around the personalised remote knowledge sent to each MBR can be updated. Such as if the link between MBR B and C is unavailable, due to if either or both MBRs disconnected from a shared/multi-nodal network such as Wi-Fi or Internet, the tailored remote knowledge for each MBR would change to become:

- MBR A would be told:
  - MBR B is reachable via only MBR B
  - MBR C is reachable via only MBR C
  - MBR B is directly connected via interface one
  - MBR C is directly connected via interface two
- MBR B would be told:
  - MBR A is reachable via only MBR A
  - MBR C is reachable via only MBR A
  - MBR A is directly connected via interface one
- MBR C would be told:
  - MBR A is reachable via only MBR A
  - MBR B is reachable via only MBR A
  - MBR A is directly connected via interface one

If and when MBR network interfaces change state they immediately inform the NSC via the Agent a quicker response to network changes should be achieved, maybe not in the region the MBR is used to (< 20 ms) but considerably better than the dynamic routing protocols available today (~ 40s) and without over flooding the network with adjacency messages along with other issues that routing protocols face in an ever changing environment (e.g. designated and backup router assignment).

Even with this regularly updated knowledge ultimately a timeout would still have to exist for assistance in capturing against ambiguous groups forming. For example taking the original setup where MBR B is completely disconnected from MBR A and C, as long as MBR A and/or C has connectivity to the wider mesh network, the NSC will process the updates that MBR A and C send when they locally sense the loss of those interfaces connection to MBR B. However, if MBR A and C have formed a separate group without access to the NSC then the original problem would still exist; this is due to the centralisation/server-style of the NSC.

Secure Agents

The MBR Agent has been developed using the framework and standards common for the whole SAI. The standard envelope of a secure agent has been used, and filled with code specific to the task of communicating with a MBR via its socket-based interface.

Figure 3 shows the class diagram of the final MBR Agent implementation. The class `MbrAgent` is derived (as said above) from the standard SAI Agent envelope (GATIAL et al. 2011). It uses a structure of several support classes, which are abstracted into several interfaces – most notably `MBRStatusInter-`

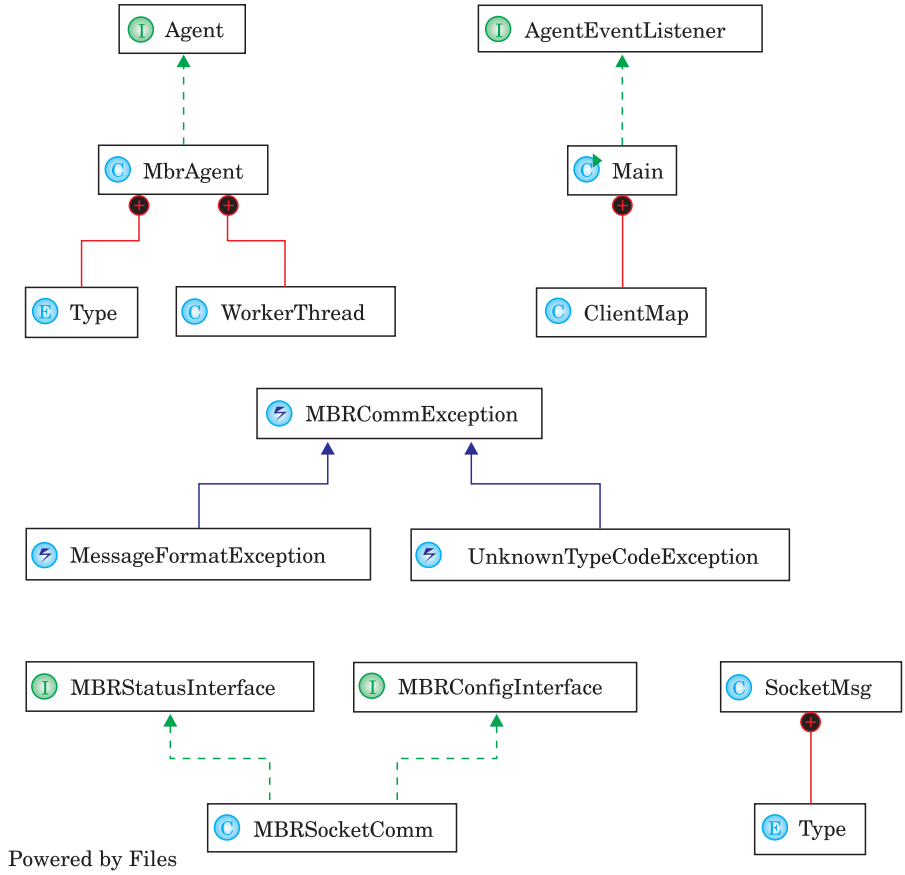

Fig. 3. The class diagram of MBR Agent implementation

`face` (methods for querying MBR status) and `MBRConfigInterface` (methods for changing MBR configuration), which are both implemented in the MBRSocketComm class, which gives these methods actual communication transport – via sockets to the MBR Interface. Exception handling is done via a small hierarchy of `MBRCommException` and its implementing classes.

Testing of a SAI agent is a complicated task, since agents are code objects which do not act on their own, but upon request. To allow for proper testing of the code which implements the agent, a separate envelope in the form of a common Java program has been developed (`eu.secricom.dsap.agent.mbr.test.MbrTests`) and this has been linked to the same classes, which implement the actual `MbrAgent` class. This program can be run using command-line arguments, and so the tests may be automated. Other methods

of automated testing have not been used, since they are not standardised in the SAI development environment, and would create too much complexity.

A placeholder implementation of the MBR interface has also been created, in order to be able to autonomously test the MBR Agent communication. This may be found in the class `eu.secricom.dsap.agent.mbr.test.DummyServer`.

Only after the tests of the MBR Agent have been done using this artificial testing grounds, a real `MbrAgent` implementation has been successfully tested with the real MBR Interface.

Network State Collector

The learning process is provided by the NSC component itself which gathers together the different fragments of local knowledge coming from the MBRs and analyses them to work out the inherent connectivity. By performing an iterative search process that "joins up" the details about active interfaces, the NSC is able to determine, for each MBR, the possible networks accessible from any one interface. Figure 4 illustrates the effects that network topology can have on the "reach" of an interface, i.e. what other network nodes can be accessed from a particular interface.

Network A in the example is a fully "looped" arrangement of connections that allows any interface on any node to route its traffic to all other nodes. Network B illustrates that interfaces marked as C1 and C2 may only broadcast to nodes A and B respectively; likewise for interfaces D1 and D2 on node D which can only reach nodes E and F.

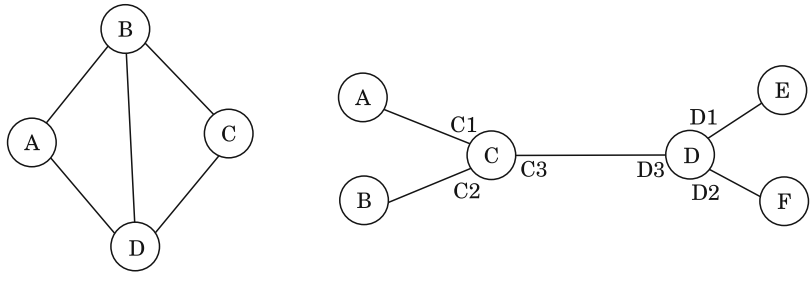

Fig. 4. Examples of interface "reach"

The NSC needs to inspect the network as a whole to draw up a picture of the possible routing choices that will be used to build the global knowledge passed to still-alive and reachable MBR's.

The development of the Network Monitoring system for the MBR was implemented in parallel. QinetiQ took the lead on developing the MBR Interface along with the NSC. UI SAV took the lead on developing the SA for this purpose. Further use-cases were discussed around the MBR Interface component, such as the ability to get and set certain MBR properties.

The development of the NSC took place in two phases. During the initial phase, a bespoke tool (see Figure 5) was created to stand in place of the "live" MBR agent, which was still being developed in parallel. This tool was capable of being configured to show a variety of network forms and also the connectivity required between the MBRs.

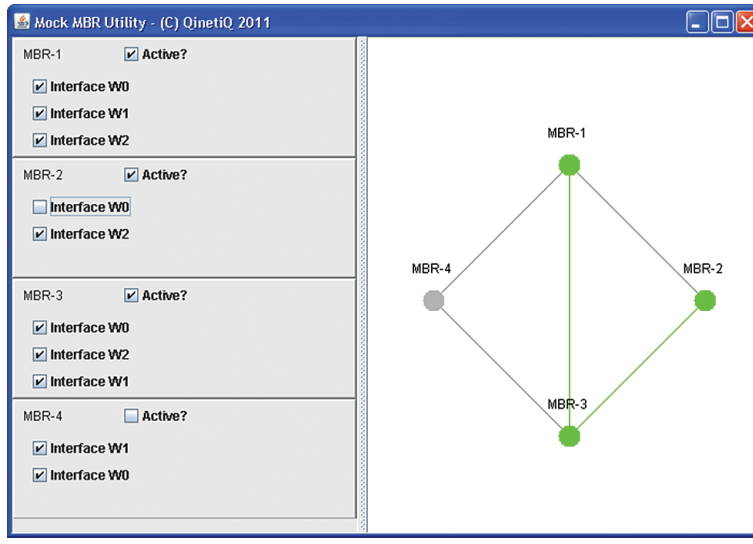

Fig. 5. Bespoke "mock" MBR tool

Communication between the mock MBR tool and the NSC was achieved using network socket architecture. The tools interface allowed for the "state" of an illustrated MBR to be altered by turning on and off different aspects of its functionality. Whenever this would happen, a new set of local knowledge was sent to the NSC which would re-assess the situation and define a fresh set of global knowledge data. This ability to precisely choose the network state meant that it was possible to very easily validate the functionality of the NSC component.

It was decided to replicate the network graph drawing within the NSC also in the first phase of development. This provided the enormous benefit of being able to perform a comparison between the diagrams shown in the two different

tools. As the network state in the mock MBR window is modified, the NSC was shown to "catch up" with its own view of the network;s revised infrastructure based on a change in broadcast local knowledge.

The NSC can use its complete knowledge of the network to build tailored remote knowledge unique to each MBR registered in the network. Ideally it will do this quickly enough that the instance an updated local knowledge message comes from an MBR the NSC will be able to recalculate the topology of the network and send updated tailored remote knowledge back to each MBR.

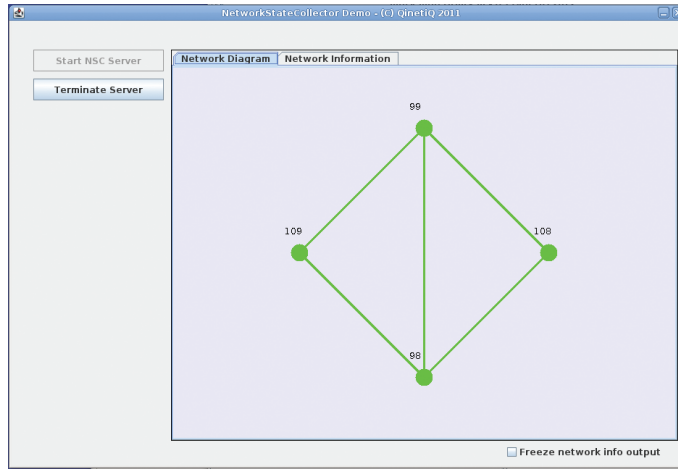

Fig. 6. Network state collector application

Having satisfactorily completed the logic and implementation of the NSC, the second phase of its development involved replacing the socket communication mechanism with the SAI toolkit functionality.

Currently, the NSC is designed to work with a "mock" or "fake" MBR service that can supply details about MBR devices on the network. This is handled by a `ServerProcess` class and it is this class which would need to undergo most of the changes to make use of the SAI toolkit as required.

First of all however, the actual agent "definition" would need to be added into the NSC project. An agent item is a component that can be dispatched to a location (an MBR in this case) that makes use of the Distributed Secure Agent Platform (DSAP) service. This service would receive the agent and allow it to execute in its "address" space. For this work package, UI SAV are creating a "layer" that would exist on an MBR and would act as the go-between for the MBR and the NSC. Note that the agent definition (also being created by the UI SAV) is based on a subclass of Agent which is part of the toolkit and would

enter into a permanent loop (via the toolkit `AgentThread` class) polling the MBR device to acquire local knowledge.

This agent component has to be deployed initially by the NSC itself when the NSC first initialises and detects the presence of running MBRs. This would be done by adding a small element of start-up code to the NSC which would look for the "reggie" service that should already be active on the network. This service can be used to obtain the details about the registered MBRs (when an MBR device starts up, it would need to locate the "reggie" service and register its existence).

For each "discovered" MBR, the lookup-client `upload` function would need to be called to send an instance of the agent to the MBR where it would start executing. The agent would presumably call its `fireEvent` method to send data to the NSC about local knowledge and heart-beat messages.

Within the `ServerProcess` of the NSC, the `notify` method is used to receive the messages coming in from the deployed agents. Once the data from an agent is received, it is handled in exactly the same way as now. If global knowledge needs to be created by the NSC, it can be passed back to the agent using the appropriate SAI toolkit message.

Conclusion

One immediate issue that sprang from the use of the "mock" MBR system to mimic the required behaviour and infrastructure was the impact of the routing protocol. During tests of the system working with real MBR's, when an MBR link was disabled, it broke the communication with the MockMBR component (and ultimately with the NSC). Once the routing protocol had resolved an alternative route, the NSC was able to once again determine the change in local knowledge and so compute new global knowledge. However, it takes quite some time for this process to be performed so that the NSC can resume its full function. It is hoped therefore to demonstrate with the use of the SAI toolkit, this problem is alleviated. It would be interesting to see if this large amount of latency is eliminated by using another layer of communication which is presumably fundamental to the toolkit and sits below the routing protocol.

Functionality that exposes the information for remote processing helps further on the reduction of hardware requirements for the MBR enabling the software to easily operate on hardware constraint mobile devices.

The ability to combine the exposing functionality from the Web and SNMP Interfaces in a unified secure manner is a new opportunity for remote management and control of many MBR aspects. Once full integration with the

SAI has been completed it will be a simple case of adding new message type functionality to the open-interface MBR component and creating an Agent with a new `fireEvent` method.

Recommendations

The core recommendation would be to de-centralise the NSC, so that each MBR can initialise its own operating knowledge directly with other MBRs and not reliant on a single NSC. This should also help in keeping seamlessness and reducing delay in passing tailored local knowledge out to all relevant MBRs.

Further to de-centralisation of the server, by combining the MBR knowledge and routing information dissemination the system could reduce its network bandwidth footprint and ensure that no information is unnecessarily duplicated. To make this easier the QinetiQ Routing Bridge (SPENCER et al. 2009) (a patented proprietary routing protocol) could be utilised in assisting the translation between routes and remote knowledge.

Translated by AUTHORS

Accepted for print 30.06.2012

References

GATIAL E., BALOGH Z., ŠIMO B., HLUCHÝ L. 2011. *Distributed secure agent platform for crisis management*. In: SAMI 2011: 9th IEEE International Symposium on Applied Machine Intelligence and Informatics. Budapest, IEEE, p. 247-253. ISBN 978-1-4244-7428-8.

SPENCER J., KENDRICK G., MILLINGTON P., STEPHENSON I., ENTWISLE P. 2009. *Communications System QinetiQ Limited*, December 2008, WO 2009/001041 A1.