CHAPTER 8

SUMMARY, CONCLUSIONS & DIRECTIONS FOR FUTURE WORK

In this final chapter we bring together the work described in the previous chapters of this thesis. Section 8.1 explains the significance of the thesis in terms of its main contributions and summarises the main findings. Section 8.2 presents the conclusions drawn through the research experience gained, regarding network monitoring applications in which agent mobility can be effectively used. Section 8.3 identifies areas in which this work could be developed further.

8.1. SUMMARY OF MAIN CONTRIBUTIONS

The main objective of this thesis has been to assess the values and weaknesses of Mobile Agent (MA) technology form the management viewpoint and identify ways for the effective use of this technology in distributed management applications. That involved the design and implementation of MA platform (MAP) expressly oriented to Network & Systems Management (NSM), and the development of three network monitoring and performance management applications implemented on the top of our framework. The research contributions of this thesis have been outlined at the end of Chapters 4-7. We re-iterate through the main findings below.

The contributions and findings related to the design and implementation of our MA-based management framework are summarised in the following:

- The MA framework introduced in Chapter 4, is characterised by a *lightweight* design that makes it particularly suitable for developing MA-based applications with minimal impact on network and system resources. That has mainly been achieved through designing a code distribution scheme that involves the transfer of MA bytecode to managed devices at the MAs' creation time, with only MAs state being transferred following that. This is in contrast with the majority of MAPs, which enforce the transfer of both the MA code and state in every agent migration. This code distribution model has been refined in Chapter 6,

by taking advantage of remotely located Mobile Distributed Managers (MDM) to implement a *tree multicasting* code distribution scheme. Mobile Agent Servers (MAS) have also been designed so as to have a lightweight footprint on hosting devices.

- Being implemented in Java, the framework is *portable* across all platforms supporting the Java Virtual Machine (JVM), addressing the requirement for integrated management.
- *Integration with legacy systems* is achieved through an MA-to-SNMP gateway integrated within the MAS, which allows incoming MA objects to locally interact with SNMP agents.
- A set of *security features* has been incorporated, shielding network devices against malicious MAs attacks and protecting management data from eavesdropping. In particular the Security Component (SC), an integral component of the MAS, provides authentication and access control (authorisation) services, whilst the data carried by MAs can optionally be encrypted.
- *Fault-tolerance* issues have also been addressed, covering scenarios where a host included in the MAs' itinerary (or the manager) fails or an interconnecting link breaks.
- The MA class versioning problem has been addressed through a *customised MA ClassLoader* (MACL), which allows distinguishing between different versions of the same MA class, thereby enabling modifications of MA-based NSM tasks at runtime.
- A Mobile Agent Generator (MAG) tool has been implemented to *automate the introduction/customisation of service-oriented MAs* through a user-friendly Graphical User Interface (GUI).
- Additional features include: (a) design of GUIs offering Management Information Base (MIB) browsing, data visualisation, management tasks customisation, etc.; (b) support for Remote Method Invocation (RMI) interactions between the manager and the managed devices; (c) visual profiling and active control over remotely executing MA threads; (d) support for MA transfers over TCP or UDP protocols, with the choice made by the administrator depending on the application; (e) management data compression.
- Experimental results revealed that our framework marginally outperforms Java RMI both in terms of network overhead and response time when considering simple manager-managed systems interactions. However, a detailed investigation of the factors affecting MA-based NSM scalability indicated an *exponential growth of management cost with the managed network size when using a single multi-hop MA* to collect data from a number of managed devices ('flat' MA-based management); that represents a non-scalable solution, especially when MAs are not highly selective. This conclusion dictates that *MA itineraries length should be limited* and led to the optimisations described in Chapter 5.

- In particular, Chapter 5 introduced *two complementary polling schemes* that aim at answering the scalability problems of flat MA-based management. In particular, *Get 'n' Go* (GnG) polling is suitable for collecting real-time data and performing short-term control and configuration tasks on multiple network elements (NE). Response time is minimised by relying on a number of MAs to carry out distributed management tasks in a parallel fashion. In addition, when considering networks with large numbers of managed hosts, the network overhead of flat management is also reduced due to the limited number of hops corresponding to each MA (the overall amount of data accumulated within MAs state is reduced). Regarding the off-line analysis of management data, we proposed the *Go 'n' Stay* (GnS) polling scheme. In this approach, single-hop agents are dispatched to managed devices and collect a large number of data samples before delivering performance reports to the manager, thereby leading to drastic reduction of MA transfers. Collected data can be delivered either by the originally deployed MAs, clones of these MAs or RMI invocations; the second approach has been found to be the most cost-effective solution. In the GnS approach, MAs-manager communication can also be event-driven, with the manager notified only when specific performance thresholds are crossed. Concluding, the selection of the appropriate polling scheme is application-dependent, with the management task execution period and the type of management data to be collected being the main factors. Experimental analysis of GnG scheme has shown that there is an optimal number of MAs that minimises the overall latency and network overhead, depending on the managed network size; GnS polling overhead is reduced for lower data delivery frequencies.
- In Chapter 6, we introduced further extensions, devising a *flexible and adaptive hierarchical MA-based management model*, which meets the requirements for the management of dynamic, large-scale enterprise networks. That model incorporates mobile mid-level manager entities (MDMs), transparently deployed to remote network domains to take over their management responsibility and localise the associated traffic. Our infrastructure offers adaptability to changing networking environments and defines concrete policies regarding network segmentation in management domains and MDMs deployment, explicit determination of domain boundaries, etc.
- To minimise network overhead on remote domains, *MDMs rely on MAs for the data collection process*. Depending on the application, these can either be single-hop or multi-hop agents. MDMs may also move within their managed domain to ensure *balanced distribution of processing and memory load* over managed systems. In particular, MDMs periodically inspect the resources availability on managed nodes and choose to move and resume execution to the least loaded host. Finally, *fault tolerance* issues are also addressed,

securing that distributed MDM objects continue to perform their decentralised tasks even when the communication with the manager platform is not feasible. When considering the management of remote domains, empirical results demonstrated that the proposed architecture outperforms alternative solutions, both in terms of the overall management cost and the bandwidth usage of low-bandwidth WAN links.

- It is emphasised that the management models proposed throughout Chapters 4-6, are supplemented by analytical *quantitative evaluations* and *experimental results*, which quantify their performance in terms of the latency and network overhead incurred when used for realistic management operations.

The contributions and findings related to the network monitoring and performance management applications described in Chapter 7, are listed below:

- The first application involved the *computation of health functions* (HF) by MAs, enabling the semantic compression of several MIB variables into a single system indicator. That allows the direct observation of managed systems health and obviates the need for transferring vast amounts of data over the network. HF configurations may be dynamically updated to reflect management needs at different systems or times, while HF values can optionally be delivered to the manager only when pre-determined thresholds are crossed. Either single-hop or multi-hop agents can be used to compute HFs. In LAN environments, the latter perform worse than SNMP, while single-hop agents (GnS polling) offer the most scalable solution. In LAN-WAN management scenarios, multi-hop MAs outperform SNMP as they reduce the usage of expensive WAN links. However, in that scenario, the MA-based hierarchical model improves management scalability even further.
- The second application involved *efficient SNMP table retrievals*. MAs have been used to obtain table snapshots through local interactions with SNMP agents; table snapshots are then delivered to the manager through single transfers. That scheme improves management scalability as it reduces the number of network interactions and applies data compression, whilst ensuring improved consistency of retrieved table contents. A variety of applications can benefit from table snapshots to investigate transient problems of short duration or identify trends on changing networking conditions. In cases that views of SNMP tables including large amounts of data are requested, single multi-hop agents represent an non-scalable approach as the state of these MAs would rapidly grow; in that case, using multiple MAs (GnG polling) can offer performance benefits, yet, GnS polling provides the most scalable solution.
- The third, and most complex, application exploits the ability of MAs to perform intelligent *filtering of SNMP tables* and enable delivery of high-level, pre-processed information to

the manager platform. Filtering patterns can be defined/reconfigured at runtime, using a variety of textual and arithmetic operators, ensuring the transfer of only those table values that satisfy specified filtering criteria. Table filtering operations may lead to significant cost savings, especially when highly selective filtering patterns are involved. Furthermore, multi-hop agents have been used to *correlate* the data collected during their itinerary and offer domain/global views of managed devices. Empirical results have shown that MA-based approaches outperform SNMP with sufficient distinct, especially when using our flexible hierarchical model for table filtering applications in a LAN-WAN environment.

- An important contribution of the introduced applications relates to a factor that currently prevents the acceptance of MAs, whose nature is less technological. MAs will be adopted only when a sufficient body of literature will provide incontrovertible evidence about *when*, *where* and *how* they are useful. The proposed applications serve as case studies in which the implications of using MAs in a real application domain can be *thoroughly analysed on a quantitative and experimental basis*; such case studies are still very rare. The MA research community must put more effort into validating its own outcomes, in order to gain credibility outside.

8.2. Applicability of Agent Mobility in Monitoring Applications

One of the main objectives of the research work described in this thesis has been to identify efficient mobility patterns for management applications. That involved the investigation of single-hop and multi-hop mobility, with the latter representing the most common use of MAs [KU97, SAH98, CHI99, ZAP99, ELD99, PUL00b]. Generally, multi-hop agents can offer flexibility and performance benefits over single-hop agents in the scenarios described in Section 3.4. In the management context and particularly on network monitoring and performance management, we have identified the following areas in which multi-hop agents can enhance flexibility and scalability:

- *Collection of real-time data*: Multi-hop agents represent an efficient approach for real-time monitoring operations, i.e. when response time restrictions apply. Specifically, when the latency involved in MA migrations is relatively small (i.e. lightweight, efficient agent transfer protocols are used), multi-hop mobility may significantly improve the efficiency of management data collection operations, compared to the single-hop mobility approach, in which an MA object is deployed to every device. However, this performance gain is conditional to the following: (a) short-term monitoring tasks performed over large sets of hosts should be involved, especially when these hosts reside in remote subnets (in such case, each MA would typically execute for a time negligible with respect to its deployment

time); (b) MAs should be able to perform semantic compression of collected data (otherwise their state size will rapidly grow, affecting both the network overhead and response time of the monitoring task). In addition, the latency of time-sensitive management operations can be further reduced using the GnG polling scheme.

- *Data correlation*: The ability of MAs to realise multi-hop itineraries can be exploited to perform data correlation, bringing forth the concept of global filtering. In particular, MAs deliver to the manager a domain/global-level view of managed devices and obviate the need for further processing/correlation of data either by the manager or a mid-level entity. That saves the manager from considerable processing burden and provides a more even distribution of computational load. To the best of our knowledge, this is the only scenario in which MAs employed in monitoring applications use the information already collected to perform a superjacent level of data filtering. Hence, the ability to perform data correlation can be used as a solid argument in favour of multi-hop mobility in the management domain.
- *Flexible and adaptive hierarchical management*: MAs have been shown particularly suitable to implement flexible mid-level managers (MDMs), able to perform decentralised management tasks. In this context, agent mobility can be effectively exploited to localise management traffic, reduce the dependency on inter-connecting links and perform distributed management tasks without the manager's intervention. Although, in theory, this can be achieved through simply uploading the mid-level manager code to a remote host (i.e. using single-hop agents), multi-hop agents add a new dimension in management flexibility and autonomy. For instance, they can dynamically migrate to another host to optimise the average resource usage on managed systems. Furthermore, they can directly observe changes in topology or traffic characteristics, re-define the boundaries of their assigned domain and create clones to share its management responsibility. Remote cloning offers the advantage of reducing the latency and overhead associated with the deployment of a new mid-level manager entity from a central location [LIO99].

It should be emphasised though that *multi-hop agents do not represent a one-way approach* on MA-based management applications. As explained in Section 3.2.3., multi-hop MAs can be regarded as a 'superset' of single-hop mobility, as they can offer all the functionality provided by the latter, with the additional ability of autonomous migration. As a result, single-hop agents can complement their multi-hop counterparts and provide an effective management delegation mechanism in management scenarios where the latter fail to offer performance gain. In particular, single-hop agents are suitable for dynamically augmenting the management capabilities of NEs, monitoring the performance and health of managed systems over long

periods and minimising data transfers by returning QoS alarms and periodic summarisation reports after processing raw management information.

8.3. DIRECTIONS FOR FUTURE WORK

The current prototype of our core MA framework presented in Chapter 4 can serve for structuring scalable, flexible and dynamically customised distributed management operations. The following optimisations could be considered in future extensions:

- The framework can be extended to enable web-based management operations. To achieve that, the front-end of the framework should be designed as an *applet* rather than an *application*, allowing the administrator to remotely control its managed network through loading the management applet on standard web browsers.
- MA transfers can be implemented using Java RMI as 'transport protocol' and compared, in terms of latency and network overhead, against MA transfers over TCP and UDP.
- To provide a more complete performance evaluation, the performance of our framework should be compared against that of the Common Object Request Broker Architecture (CORBA). Although in theory CORBA is more heavyweight and slow than Java RMI, the choice of an appropriate implementation may lead to marginal performance gain compared to RMI [BOH00a].
- The integration of a resource accounting tool, such as *Jres* [CZA98] can be considered as a means of controlling the occupation of system resources (CPU, memory, etc) within the Java runtime system and potentially killing mis-behaving MA processes.
- The integration of the SC with the Java Cryptography Architecture (JCA) can also be considered to allow compliance and interoperability with other cryptography implementations. Such integration will incorporate the design of a *provider* [Providers], encapsulating the implementations of the SC component's authentication and encryption algorithms.
- The framework's fault tolerance could be improved so as to cover a broader range of fault scenarios in addition to the ones mentioned in Section 4.4.2. For instance, the case where the hosting device of an MA fails while the MA is still executing its task should also be taken into account. Therefore, MAs should transparently resume computations that are affected by system or network interruptions or failures. That can be achieved by periodically saving the MA's persistent state information, as proposed in [BRU01]. This feature should be optionally used depending on hosting devices storage capacity.

- The functionality of the MACL (described in Section 4.4.3.) can be extended so as to maintain information about the last time each of the existing MA classes has been loaded. For those classes not being used for time exceeding a certain limit, their bytecode will be removed from the MACL's hashtable and their associated class files deleted. That way, the use of local memory and disk space would be minimised. In the case that one of the MAs whose bytecode has been removed visits the same device, MACL would download its code from the manager's Mobile Code Repository (MCR) component. An alternative way for optimising the usage of local memory and disk resources would be to enforce the MAs themselves to inform the managed devices whether they execute a long or short-term management task. In the latter case, the Migration Facility Component (MFC) would request MACL to remove the MA code from the hashtable, as soon as the MA migrates to another host.
- Additional code distribution models need to be investigated and compared against the solution adopted in our prototype. For example, the *code pre-fetching* proposed in [SOA99], whereby the bytecode of an MA object is transferred to the hosts included into the MA's itinerary right before the MA's travel begins.
- Finally, several optimisations could also be performed to maximise the framework's performance with respect to the network overhead and response time associated with MA transfers. In particular, the optimisations suggested in Appendix B considerably reduce the MAs state size and therefore moderate the impact of MA migrations upon network resources. In addition, response time could be further reduced by saving the time needed to create MA objects at the beginning of each Polling Interval (PI) and also the time needed to translate host names to their respective IP addresses. This issue is also discussed in Appendix B.

Regarding the two complementary polling schemes described in Chapter 5, the following optimisations could be considered in future extensions, to further improve their efficiency:

- GnG scheme performance could possibly benefit by applying a variation of the technique described in [BAR99]. Namely, enforce individual MAs to download their state (i.e. their collected data) to the manager station after visiting a fixed number of NEs. That would represent significant gain in network overhead, especially when large amounts of data are collected from each host, as it would prevent MAs state from growing over a certain limit; yet, response time penalties associated with the employment of such scheme should be investigated in detail.
- Optimal itineraries can be designed by implementing the Optimal Itinerary Planning (OIP) algorithm described in Section 5.2.1.2.

- In GnG polling, network segmentation could be automated and dynamically adapted to the managed network, aiming at minimising the response time and/or network overhead depending on the number and physical distribution of NEs.
- The performance of GnS polling can be optimised by eliminating information carried within the MAs' state that is only useful for multi-hop agents, e.g. itinerary.
- Additional experiments should be conducted in large-scale networking environments comprising several remote management domains. That would allow to evaluate the effect of network partitioning in reducing the overall response time over flat MA-based model. Should experiments on such environments are not feasible, 'virtual' remote segments could be created, introducing some fixed time penalty to MA transfers whenever an agent traverses a simulated 'WAN link'.

The functionality of the hierarchical model proposed in Chapter 6 can be further improved through incorporating the following optimisations:

- The topology map GUI can be designed as an expandable graphical component: The map will initially display a representation of the managed network where individual subnets will be illustrated as icons that will expand to provide a detailed view of their included managed devices.
- MDMs deployment Policy 2, described in Section 6.3.3., can be implemented to enable the dynamic placement of MDMs depending on the management cost associated with the management of a remote domain rather than the number of devices included therein.
- The Resources Monitoring Tool (RMT) tool (see Section 6.3.7) can be extended so as to support Windows 95/98 and 2000 operating systems.
- The configuration of the hierarchical model can be saved through serialising the `HierarchicalSettings` object (see Section 6.3.3.) whenever the settings are modified. The serialised information would be stored in a file, used to restore the settings at manager startup. That would obviate the need for the administrator to re-define the hierarchical system's configuration.
- MDMs autonomous decision making can be further enriched: in the current version of our prototype, the decision regarding the segmentation of a management domain (already assigned to an MDM) in two parts, is made by the manager. For instance, should a new device is connected to a segment within the boundaries of the examined domain, its local MAS server will publicise its initialisation through communicating directly with the manager. It is then left to the manager to decide whether the remote MDM can cope with the increased number of managed devices or its domain should be divided in two. As a

future extension, we intend to enforce the MDMs to make such decisions without requiring any communication with the manager. In particular, at the time that an MAS server starts, it will first get (through a network 'map') a list of the hosts residing on the same subnet and then broadcast an "I'm-alive" message to them. Should an MDM executes on one of these devices, it will instantly acknowledge the receipt of the message and append the 'discovered' host to its list of managed devices, otherwise, after a pre-determined time interval elapses, the MAS will send the message to the manager platform. In the former case, if the number of NEs managed by the MDM exceeds the specified limit, the MDM will divide its domain in two parts, create a clone of itself and send its clone to take over the management of the second domain. That approach will increase the ability of the MDMs to make autonomous decisions, while minimising the dependency on links connecting remote subnets to the manager site and also the overhead of MDM deployment.

- The current three-level hierarchical model can be extended to a multi-level hierarchy, whereby MDMs could be supervised by other MDMs residing on higher levels rather than directly by the manager platform.
- The existing prototype allows the manager to download on-the-fly new monitoring task definitions to distributed MDMs. However, the functionality of the MDMs themselves cannot be modified at runtime. Therefore, rather than deploying MDMs with fixed capabilities able to perform only dynamically customisable monitoring tasks, the MDM should also provide an interface allowing the manager to plug-in software components implementing new functionality. That interface could be implemented through the `MdmRmiServer` component thereby enabling the design of MDMs as extensible engines.

The functionality of the monitoring applications described in Chapter 7 can be further enriched through applying the following extensions:

- Management applications often need to retrieve information scattered among several MIB tables, e.g., routing information specific to certain type of interfaces and their current utilisation. For instance, the MIB-II `ipRouteTable` [McC91] keeps track of IP routes, with interfaces information found in `ifTable`. An application may need to correlate routes with interface utilisation for capacity planning purposes. That correlation could be achieved by using the `ipRouteIfIndex` column of `ipRouteTable` as index for the corresponding retrievals from the `ifTable`. Such operations imply *joining* two MIB tables, in a fashion similar to database table joins. Currently the table filtering operations described in Section 7.5 do not support table joins, which could be considered in future extensions.

- The applications described in this chapter considered exclusively filtering of MIB-II objects/tables. Hence, additional applications and filtering patterns should be envisaged and applied to other MIBs. For instance, MAs may work in conjunction with Remote Monitoring (RMON) probes to provide, on demand basis, high-level network-oriented statistics, e.g. an MA could visit all the devices with installed RMON probes and return to the manager information about the least loaded Ethernet segment. This information, retrieved from the RMON MIB [WAD95], could be used for network planning, for example to help the administrator decide where to connect a new managed device so as to achieve even distribution of traffic.

  Another scenario could be to return a number of *N* host pairs that communicate in more frequent basis and therefore generate a substantial amount of traffic (this information can be extracted from the RMON *matrix* group). The administrator might then consider connecting these hosts in the same segment so that the generated traffic will only affect that segment. In the case that no RMON probes are installed, MAs could extract similar information from packet sniffers, like the C-based sniffer tool used in [FER01].

- The latency associated with table retrievals can be reduced by implementing the 'get-table' operation through successive `get-bulk` (instead of `get-next`) requests; that would also improve table values consistency. Since the get-bulk operation is only available in SNMP v2c and v3 frameworks, MAs could be instructed to first check the SNMP version of the local agent and then accordingly invoke an appropriate 'get-table' method implementation, issuing either `get-next` or `get-bulk` requests.

- Current filtering patterns allow Table Filterer (TF) agents to either obtain *single* column values or the whole table *row*. These patterns should be extended so as to allow encapsulation of an *arbitrary* number of table column values.

- The MAG tool's functionality can be extended so as to provide the user a 'library' of actions to be triggered at the event of specific threshold crossings. Such an extension would enable MAs to perform simple configuration or fault management tasks.