Managed Readiness Simulator (MARS) V2

Assessment of a Simulation Runtime Database Approach

Stephen Okazawa
Mike Ormrod
Chad Young

DRDC CORA TM 2009-042
September 2009

Defence R&D Canada
Centre for Operational Research & Analysis

Land Forces Operational Research Team

National Defence Défense nationale

Canada

Managed Readiness Simulator (MARS) V2

Assessment of a Simulation Runtime Database Approach

Stephen Okazawa
Mike Ormrod
Chad Young
DRDC CORA

Defence R&D Canada – CORA

Technical Memorandum

DRDC CORA TM 2009-042

Principal Author

Original signed by Stephen Okazawa

Stephen Okazawa

Defence Scientist

Approved by

Original signed by Dean Haslip

Dean Haslip

Section Head Land and Operational Commands

Approved for release by

Original signed by Dale Reding

Dale Reding

Chief Scientist DRDC CORA

Defence R&D Canada – Centre for Operational Research and Analysis (CORA)

Abstract

This paper describes technical enhancements implemented in the second version of the Managed Readiness Simulator (MARS V2). These enhancements were required to address limitations in the technology and implementation of the initial prototype (MARS V1). The limitations related to data volume, data structures, algorithms and performance, which prevented larger and more complex scenarios from being simulated. Therefore, a range of potential approaches was identified, and a novel model architecture was devised that would address these limitations. The approach consisted of exploiting database technology during simulation execution which was termed a Simulation Runtime Database (SRDB). This allowed the simulation to use the database's table data structures for runtime data storage and to invoke data processing algorithms through Structured Query Language commands. A feasibility study was conducted that proved that the SRDB could be implemented in the MARS simulation engine, Arena. This justified the implementation of the MARS V2 model using the SRDB approach. Testing of the completed MARS V2 model showed that it successfully addressed all four of the limitations identified in MARS V1. Ultimately, the SRDB approach provided a much more capable platform for large, complex simulation scenarios that will meet current and future client needs.

Résumé

Dans ce document, on décrit les améliorations techniques apportées à la deuxième version du programme de simulation de gestion de la disponibilité opérationnelle (MARS V2). Ces améliorations étaient nécessaires pour éliminer des faiblesses dans la technologie et la mise en œuvre du prototype initial (MARS V1). Les faiblesses, reliées au volume de données, aux structures de données, aux algorithmes et aux performances, empêchaient la simulation de scénarios plus grands et plus complexes. Par conséquent, on a identifié un éventail d'approches potentielles et on a créé une architecture de modèle novatrice permettant d'éliminer ces faiblesses. L'approche adoptée consistait à exploiter une technologie des bases de données pendant l'exécution des simulations, que l'on a baptisée SRDB (Simulation Runtime Database). Cela permettait à la simulation d'utiliser les structures de données des tables de la base de données pour le stockage des données pendant l'exécution et d'invoquer des algorithmes de traitement des données au moyen de commandes SQL (Structured Query Language). Une étude de faisabilité a prouvé que la technologie SRDB pouvait être mise en œuvre dans Arena, le moteur de simulation du programme MARS. Cela justifiait la mise en œuvre du modèle MARS V2 à l'aide de l'approche à SRDB. La mise à l'essai du modèle MARS V2 terminé a démontré que ce dernier éliminait avec succès les quatre faiblesses identifiées dans MARS V1. En fin de compte, l'approche SRDB a permis d'obtenir une plateforme beaucoup plus puissante pour de grands scénarios de simulation complexes qui répondra aux besoins présent et futurs des clients.

This page intentionally left blank.

Executive summary

Managed Readiness Simulator (MARS) V2: Assessment of a Simulation Runtime Database Approach

Stephen Okazawa; Mike Ormrod; Chad Young; DRDC CORA TM 2009-042; Defence R&D Canada – CORA; .

Introduction: The Managed Readiness Simulator (MARS) is a software application being developed at Defence Research & Development Canada - Center for Operational Research & Analysis as an Applied Research Project managed by the Land Force Operational Research Team. MARS is designed to quickly simulate a wide range of readiness scenarios to determine if the resources of an establishment are able to satisfy the requirements of planned operations.

The first version of MARS (termed V1) successfully conducted a preliminary analysis of the Army's plans to generate forces for Task Force Afghanistan. However, several technical limitations were encountered that related to the data volume, data structures, algorithms and performance of MARS V1. In particular, the ability to model establishment dynamics was a priority feature with respect to meeting the needs of the client , but was not considered to be feasible because of these limitations.

Therefore, a decision was made to re-examine the model architecture. This paper examines the proposed technical approaches to resolve the issues identified in MARS V1 and describes a novel solution that was devised and implemented in a new MARS V2. The solution consisted of exploiting powerful aspects of database technology during simulation execution. This approach was termed a Simulation Runtime Database (SRDB).

Results: A feasibility study and subsequent implementation and testing of the SRDB architecture demonstrated that the approach successfully addressed all four limitations identified in MARS V1. This substantially increased the ability of MARS to simulate scenarios of greater scale and complexity and increased execution speed by a factor of two.

Significance: Ultimately, the SRDB architecture allowed MARS V2 to implement important aspects of modelling readiness for the CF. In particular, it opened the door to incorporating establishment dynamics into the managed readiness model which is a high priority capability that did not exist prior to MARS V2. In general, a much higher degree of technical flexibility was achieved that will allow MARS V2 to continue to meet the increasingly complex simulation needs of CF decision makers.

Sommaire

Programme de simulation de gestion de la disponibilité opérationnelle (MARS) V2 : Évaluation d'une approche avec la technologie Simulation Runtime Database

Stephen Okazawa; Mike Ormrod; Chad Young; DRDC CORA TM 2009-042; R & D pour la défense Canada – CARO; .

Introduction: Le Programme de simulation de gestion de la disponibilité opérationnelle (MARS) est une application logicielle que l'on développe à Recherche et développement pour la défense Canada – Centre d'analyse et de recherche opérationnelle comme projet de recherche appliquée gérée par l'équipe de recherche opérationnelle de la Force terrestre. Le programme MARS a été conçu pour simuler rapidement un vaste éventail de scénarios de disponibilité opérationnelle afin de déterminer si les ressources d'un établissement peuvent répondre aux exigences des opérations prévues.

La première version du programme MARS (appelée V1) a réalisé avec succès une analyse préliminaire des plans de l'Armée visant à mettre sur pied des forces pour la Force opérationnelle interarmées Afghanistan. Cependant, on a décelé dans MARS V1 plusieurs faiblesses techniques reliées au volume des données, aux structures de données, aux algorithmes et aux performances. Tout particulièrement, la capacité de modéliser la dynamique des établissements était une fonction prioritaire pour répondre aux besoins du client, mais on a estimé que c'était impossible en raison de ces faiblesses.

Par conséquent, on a décidé de réexaminer l'architecture du modèle. Dans ce document, on décrit les approches techniques que l'on propose pour résoudre les problèmes identifiés dans MARS V1 et on décrit une solution novatrice qui a été conçue et mise en œuvre dans le nouveau MARS V2. La solution consistait à exploiter les aspects puissants de la technologie des bases de données pendant l'exécution des simulations. On a baptisé cette approche SRDB (Simulation Runtime Database).

Résultats: Une étude de faisabilité, puis la mise en œuvre et la mise à l'essai de l'architecture SRDB ont démontré que l'approche éliminait avec succès les quatre faiblesses identifiées dans MARS V1. On a pu ainsi accroître considérablement la capacité du programme MARS à simuler des scénarios plus grands et plus complexes, et on a doublé la vitesse d'exécution.

Importance: En fin de compte, l'architecture SRDB a permis au programme MARS V2 de mettre en œuvre des aspects importants de la modélisation de la disponibilité opérationnelle pour les FC. Tout particulièrement, elle a ouvert la voie à l'incorporation de la dynamique des établissements dans le modèle de disponibilité opérationnelle, ce qui est une capacité prioritaire qui n'existait pas avant MARS V2. De façon générale, on a atteint un degré nettement supérieur de souplesse technique qui permettra à MARS V2 de continuer à fournir aux décideurs des FC les simulations de plus en plus complexes dont ils ont besoin.

Table of contents

List of figures

List of tables

This page intentionally left blank.

1 Introduction

The Managed Readiness Simulator (MARS) is a versatile program that allows the user to quickly simulate a wide range of Canadian Forces (CF) readiness scenarios to determine if the resources of an establishment are able to satisfy the requirements of a set of operational tasks. The flexibility of MARS allows diverse operational tasks to be defined as processes composed of activities that place specific resource demands on the establishment. The level of fidelity of a given scenario can also be adjusted. MARS provides a graphical user interface that facilitates the creation and execution of simulation scenarios and the analysis of simulation output. The user can view aggregated simulation results and drill down to view the status of specific tasks and units over time. This provides the user with a powerful tool to anticipate problems that may arise and to identify their cause. Ultimately, MARS is intended to be used as a decision support tool for senior commanders of the CF. It provides them with forecasts of the impact of proposed changes to lines of operation, the establishment, the readiness plan, CF policy, and other factors that may affect the CF's ability to satisfy operational demands and to maintain the health of the establishment. A more detailed description of the motivation behind the development of MARS and its potential applications can be found in Ormrod *et al.* [1].

MARS is being developed by Defence Research & Development Canada - Center for Operational Research & Analysis (DRDC CORA) as one of its Applied Research Projects. DRDC CORA's Land Force Operational Research Team (LFORT) is managing the project and is using an evolutionary development approach where new capability is added incrementally to the existing model. This approach allows the model to be used in its current state with its existing capability while new capability is being added to the next version. It also allows the developers to learn from and refine the existing model while adding new capabilities to the next version.

The first version of MARS (termed MARS V1) was built as an initial prototype to demonstrate the managed readiness modeling concept. Previous publications [2, 3] document the design and implementation of MARS V1. Completed in 2007, its primary application was to assess the capability of the existing Army establishment (the current supply of qualified personnel and equipment resources) to meet the demands of planned operations. It was implemented as a Discrete Event Simulation (DES) using Rockwell's Arena software [4]. The scenario data were stored in a Microsoft Access database (DB) and were transferred to Arena's internal memory arrays when conducting a run. After completion, the output from the simulation was transferred back to the Access DB for post processing.

Using this architecture, MARS V1 successfully conducted a preliminary analysis of the Army's plans to generate the forces required for Task Force Afghanistan [5]. However, while conducting this analysis, several difficulties were encountered:

1. The large volume of data imported to Arena's arrays produced a very large runtime memory footprint. In some cases, runs had to be reconfigured because they exceeded available system memory.

2. Because Arena's arrays have fixed dimensions, the sizes of these arrays had to be very carefully managed to accommodate changes to the scenario and input data. This process was cumbersome and error-prone.

3. The Arena model was large and complicated, and lacked modularity and encapsulation. This made it difficult and time consuming to modify and debug the program, two qualities considered essential because of the evolutionary development philosophy being employed.

4. The time required to run a scenario was typically long because of a significant overhead required to initialize the model and populate the arrays at the start of each run.

The practical implication of these limitations was an inability to implement important new features in MARS. The next major development phase was to be the addition of model features and data management capabilities to realistically implement a dynamic establishment. This refers to the continuous arrival, movement and release of resources in the establishment. The implementation of this new capability was not considered feasible in MARS V1 because of its limitations. Therefore, a decision was made to re-examine the model architecture. This paper examines proposed approaches and presents a novel solution to address these limitations and allow for the continued capability enhancement of MARS.

The following two sections, 2 and 3, provide a review of MARS concepts, terminology and components. Section 4 discusses the model architecture used in MARS V1. Section 5 details the technical limitations that resulted from this architecture. Section 6 discusses possible approaches to addressing these limitations. Section 7 describes the proposed solution and how it resolves each of the limitations identified in Section 5. Section 8 details a feasibility study that demonstrated that this approach could be implemented in Arena. Section 9 provides an overview of the implementation of the MARS V2 managed readiness model using the new architecture. Finally, Section 10 discusses the performance implications of the proposed solution, and Section 11 provides concluding remarks. Annexes A and B include basic instructions and sample code for implementing this approach in Arena.

2 MARS Concepts and Terminology

The MARS program is designed to simulate a given readiness *Scenario* by forecasting the ability of an *Establishment* to generate the *Resources* required to satisfy a set of *Tasks* occurring over time under a given set of conditions. The program also records the state of every Resource throughout the simulation; therefore the results can be used to determine the utilization level of a unit within the establishment or of a specific group of Resources.

The program currently models three types of Resources: Personnel, Equipment and Facilities. Every Resource occupies a *Slot* in an *Organization* as shown in Figure 1. In general, the *Attributes* of each Slot define a particular *Resource Requirement* of the Organization and the Slot can only be occupied by a single Resource that satisfies the requirement. For example a Slot may assert that it can only be filled by a resource that satisfies the criteria: *rank = Captain AND occupation = Infantry*. However, special *Overflow* Slots can also be built into the establishment that are allowed to contain many Resources. These Slots are sometimes needed to store extra Resources that do not satisfy the Resource Requirement of any Slot or that cannot be assigned to a Slot because eligible Slots are currently occupied.

An Organization consists of a group of Slots that define the Resource Requirements of a unit. They are typically arranged in a hierarchical tree structure where only the terminal nodes of the tree contain Slots. There are two types of Organizations. Establishment Organizations define the units that contain the Resources available in the Scenario. Theatre Organizations define templates of the units required by the Tasks being modelled and do not contain any Resources. During the simulation, the program uses these templates to create a group of slots that will be filled by Establishment resources selected to perform a task.

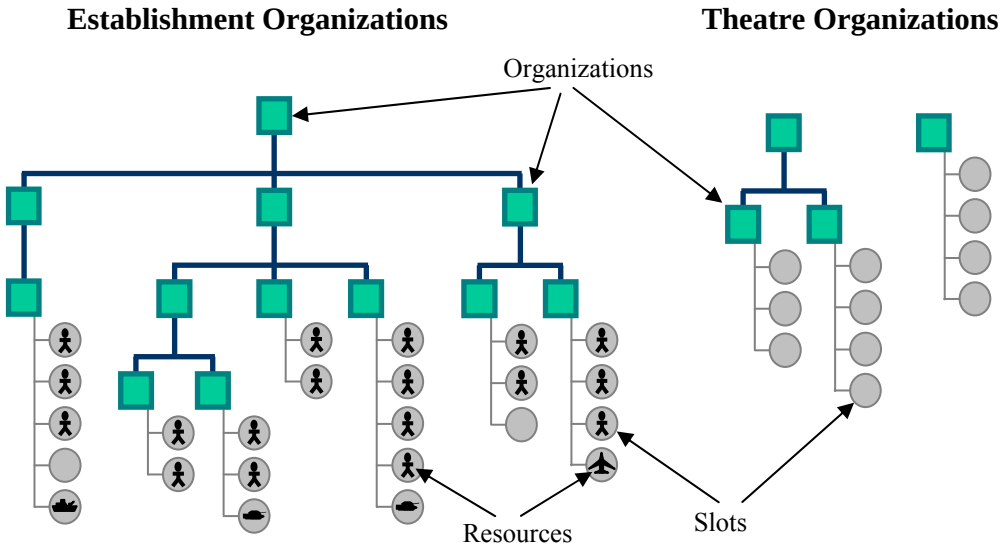

Figure 1: Establishment Organizations and Theatre Organizations showing Slots and Resources

Each Resource has Attributes that define its current state. Attributes store the information that determines whether a Resource can be chosen for a particular Task. Examples of Resource Attributes include a person's rank and qualifications. Other Attributes indicate whether the Resource is currently busy and whether there are any restrictions on what the Resource is allowed to do. In general, Attributes define the capability and availability of a Resource. These Attributes along with the Organization to which the Resource is attached are used to determine the eligibility of the Resource to be used by a given Task.

The operations and events being simulated in a MARS scenario are represented by *Tasks* within the model. Each Task is broken down into *Activities* which are scheduled within the Task so that they will occur in a specified order. Activities are responsible for assembling the Resources they require. Activities can also be linked together to pass Resources from one Activity to another or to enforce dependencies. There are two types of Activities in the model:

1. A Process Activity temporarily employs Resources for a certain period of time and may alter their state upon commencement and completion.

2. An Event Activity changes the state of the selected Resources at a single point in time.

Each Activity is triggered in the simulation according to timing and Resource constraints. Activities simulate everything from training and operational tasks to recruitment and retirement events or, if referring to equipment, acquisition and disposal events. The Activity construct is illustrated below in Figure 2.

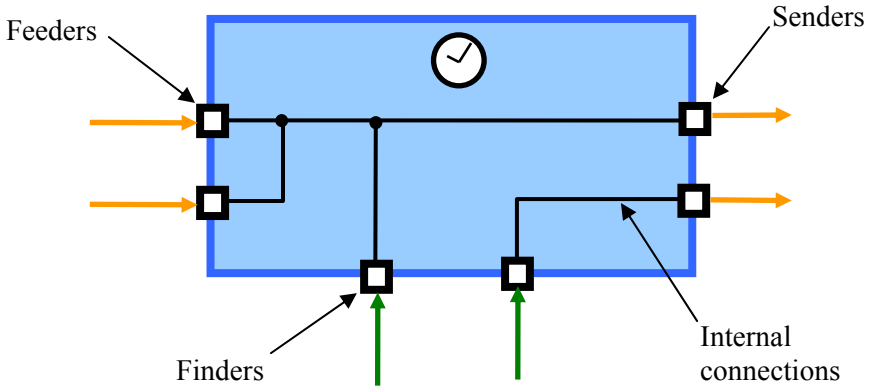

Figure 2: The Activity Construct showing Feeders, Finders and Senders and internal connections.

When an Activity is triggered and starts processing, it must acquire the Resources it needs to carry out its function. Resources enter an Activity as part of a *Resource Group* (ResGrp) through either a *Feeder* or *Finder* node and exit through a *Sender* node. An Activity may have multiple Feeders, Finders and Senders, and must have at least one Feeder or one Finder in order to act on at least a single ResGrp. A ResGrp is a set of Resources currently attached to a group of Slots. ResGrps are created by Finders which select Resources from the Establishment to participate in the Activity. Figure 3 illustrates the steps carried out by the Finders.

First, the Finder identifies the Theatre Organizations that contain the Slots that define the *Resource Requirement* for the Activity. For example, a Disaster Assistance Response Team Triage Unit might contain Slots for a medical officer, a medical technician, and a nurse.

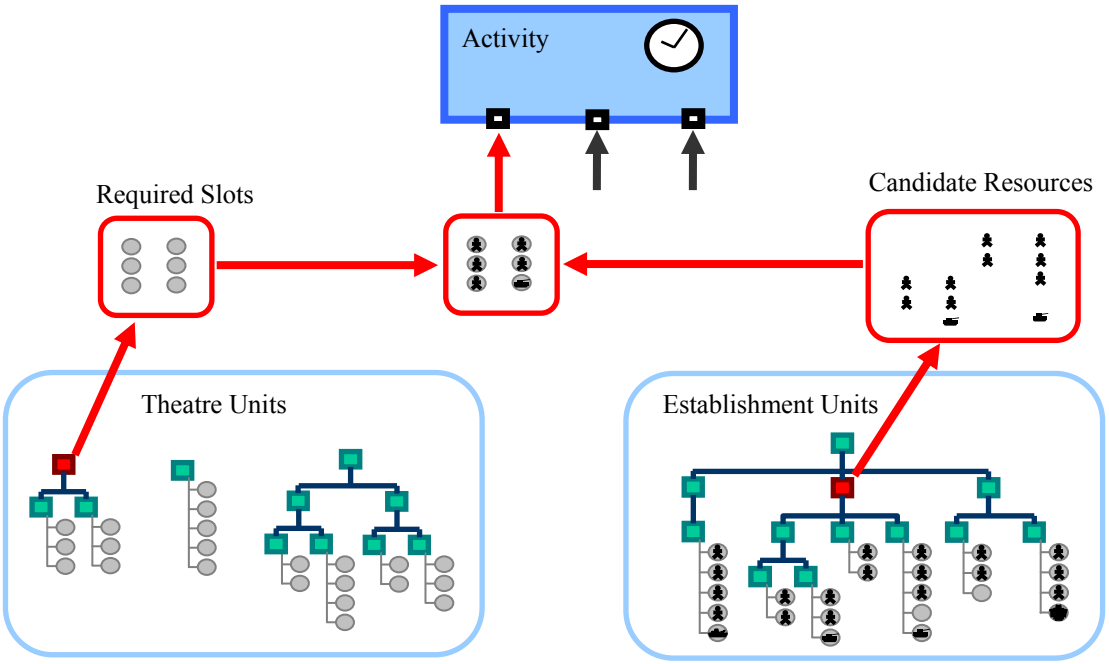

Figure 3: Resource selection process used by an Activity Finder.

Next, the Finder specifies a prioritized list of Establishment Organizations that may be searched to find Resources to participate in the Activity. This prioritized list is subject to constraints that can be used to limit the number of Resources taken from a given Organization and to filter for Resources with certain Attributes. The Finder also verifies that these Resources have not already been assigned to a conflicting Activity. This produces a list of *Candidate Resources*.

The Finder then attempts to fill each Required Slot with one of the Candidate Resources by comparing the Attributes of the Resources to the requirements of each Slot. If a suitable match is found, the Resource is assigned to the Slot and becomes part of the ResGrp being created by the Finder. To maximize the number of successful matches, the Finder attempts to assign the least qualified Candidate that meets the requirements of each Slot. For example, a Slot that can be filled by either a Major or Lieutenant Colonel will be preferentially assigned a Major because Lieutenant Colonels, being of higher rank, are in shorter supply and may be required by other Slots with more stringent requirements. This process is repeated for all Finders, with each Finder creating a ResGrp.

Activities also acquire Resources through Feeders which receive ResGrps that were created by a preceding Activity and passed on through one of that Activity's Senders. After acquiring its Resources through its Finders and Feeders, the Activity verifies that a specified minimum number of the required Slots have been filled. If this minimum requirement is not met, the Activity fails and the Resources are released. If sufficient Resources are found, the Activity takes control of the selected Resources, altering their Attributes to reflect the nature of the Activity and employing them for the duration of the Activity.

Each Feeder and Finder is connected internally to a Sender. Upon Activity completion, each ResGrp is passed to a Sender which alters the Attributes of the Resources within the ResGrp to reflect the completion of the Activity. Each Sender is then responsible for either passing the ResGrps to the Feeder node of a follow-on Activity or for releasing the Resources within the ResGrp back to their Establishment unit. Senders and Feeders are the connection nodes that allow Activities to be linked together within a larger Task. When the Activity's processing time has finished and each ResGrp has exited through a Sender, the Activity is complete. Similarly, when all of the Activities of a Task are finished, the Task is complete.

To allow Tasks to be reused within a Scenario and to control when they begin, *Task Generators* are used to assign a start time to a Task. Multiple instances of a Task can be generated on a *Rotation* schedule to model the repetition of a Task such as a cycle of deployments that make up a continuous operation. When all of the Task Generators have been processed, and all of their associated Tasks are finished, the MARS Scenario is complete and the simulation stops.

From the outputs generated by the MARS simulation, the extent to which the Establishment was able to supply the Resources required for all the Tasks being modelled can be measured. More specifically, the output is analyzed to determine how successful each Finder was in creating its ResGrp from the Establishment. By aggregating the results from the Finders, the user can determine how successful the Establishment was in generating the required Resources for each Activity, Task, Task Generator, and the entire Scenario. Similarly, the states of the Resources within the Establishment can be tracked over time. These results can be combined to plot the state over time of a selected group of Resources, a unit or group of units, or the entire Establishment.

MARS is a versatile tool with many potential applications. Its strengths are its generic constructs that allow users to quickly simulate virtually any force generation scenario and its outputs that provide users with the ability to aggregate and drill down into the simulation results to identity the causes of a particular outcome. The ability to forecast how successfully an establishment can generate the forces required to satisfy both operational and sustainment demands will provide decision makers with invaluable information that currently is unavailable.

3 MARS Application Components

The MARS application consists of three major components, shown in Figure 4: a Scenario Database, a Managed Readiness Model, and a Graphical User Interface (GUI). The Scenario Database stores all the data that define a specific scenario. The Managed Readiness Model is the discrete event representation of the process of selecting and employing resources for. The GUI allows the user to interact with the database and the model by performing three management functions. As the Input Manager, it is responsible for facilitating the transfer of scenario data into the scenario database. The input data consist of the tasks, the establishment, plans, and policies to be modelled in the proposed scenario. The data may be input directly through the GUI input screens or imported from an external source such as a Corporate DB or spreadsheet. As the Simulation Manager, the GUI controls the execution of the Simulation Scenario. Finally, as the Output Manager, it allows a user to analyze the simulation results and to generate output reports.

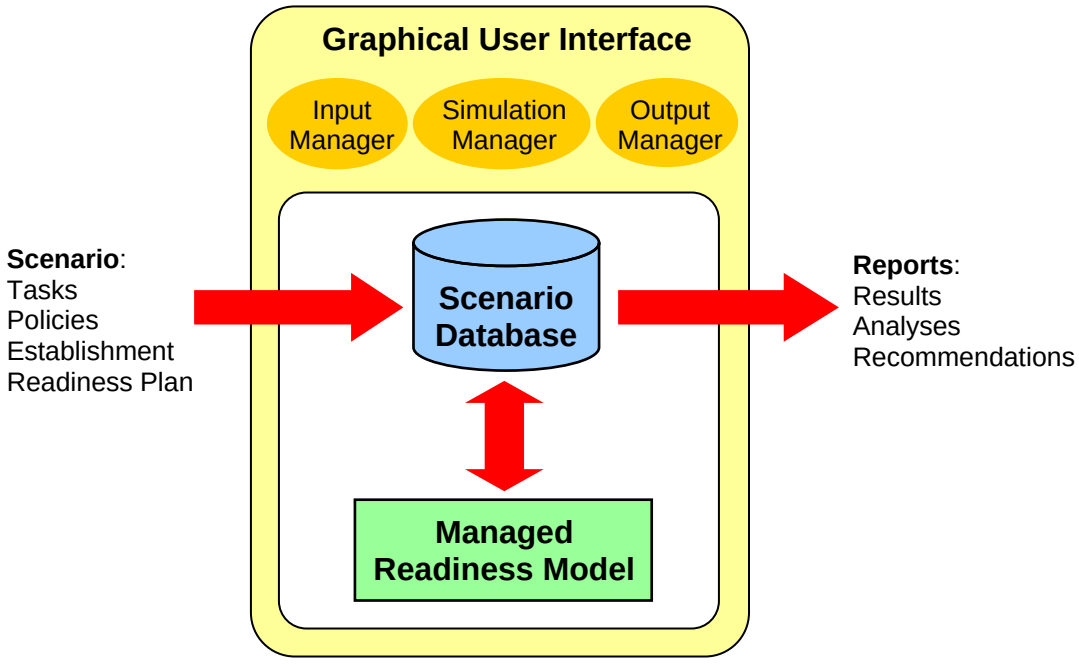

Figure 4: Components of the MARS Application with data transfer shown as red arrows.

4 MARS V1 Model Architecture

The limitations and technical enhancements described in this paper relate to the implementation of the Managed Readiness Model and its interface with the Scenario Database. The Managed Readiness Model component of MARS consists of three main parts: data, algorithms, and process logic. These parts are described below.

- The data contain the input information that specifies a simulation scenario, the state information that represents the objects and scheduled events that are active during the simulation, and the output information that records what happened during the simulation. All these data are accessed during simulation execution. For MARS, this includes the Organizations, Slots and Resources that represent the CF establishment; the Theatre Organizations and Slots; Activities, Tasks and Task Rotations; Feeders, Finders, Senders and their interconnections; the required Theatre units and candidate organizations for each Finder; definitions and descriptions of Resource Attributes; and a multitude of other tables relating to details of the simulation, intermediate processing during execution, and records of simulation events.
- Algorithms are the low-level operations that interact with the data in response to simulation events. Depending on what is involved for a given event, various algorithms may be invoked to carry out operations such as inserting and deleting data entries; searching and sorting data; updating values; and iterating through rows of data. These operations are used very heavily throughout the simulation as the bulk of the computational work in MARS consists of complex search, match, and update operations. This includes checking timing and dependency criteria prior to launching an Activity; transferring ownership of resources from one Activity to the next; searching for candidate resources and required Slots subject to constraints; matching candidate resources to required Slots based on Attributes and Attribute requirements; updating Resource Attributes; and generally keeping the state of all parts of the simulation up-to-date.
- The process logic is the high-level discrete event representation of the managed readiness process. This controls what events happen in the simulation and when they happen. When an event occurs, the process logic triggers the algorithms that do the work associated with that event to update the data. This includes carrying out the creation of Tasks and Activities; coordinating when Activities are triggered subject to Resource, timing and dependency constraints; and managing the lifecycle of each Activity from Resource acquisition through its Feeders and Finders, to Resource employment for a period of time, to the sending of Resources on to subsequent Activities or back to the Establishment.

In MARS V1, all three of the parts described above were implemented in Rockwell's Arena DES software using its native feature set. These features include a comprehensive list of DES building blocks, internal variables and arrays for data storage and computation, and Visual Basic for Applications (VBA) which provides programmatic control over the simulation. The input to the simulation was an access database that contained the scenario information and initial conditions. The simulation output was eventually written back to the database for post processing. Figure 5 shows the architecture of the MARS V1 managed readiness model and how the data, algorithms and process logic components were related. A detailed description of the model implementation was documented by Pall *et al.* [2].

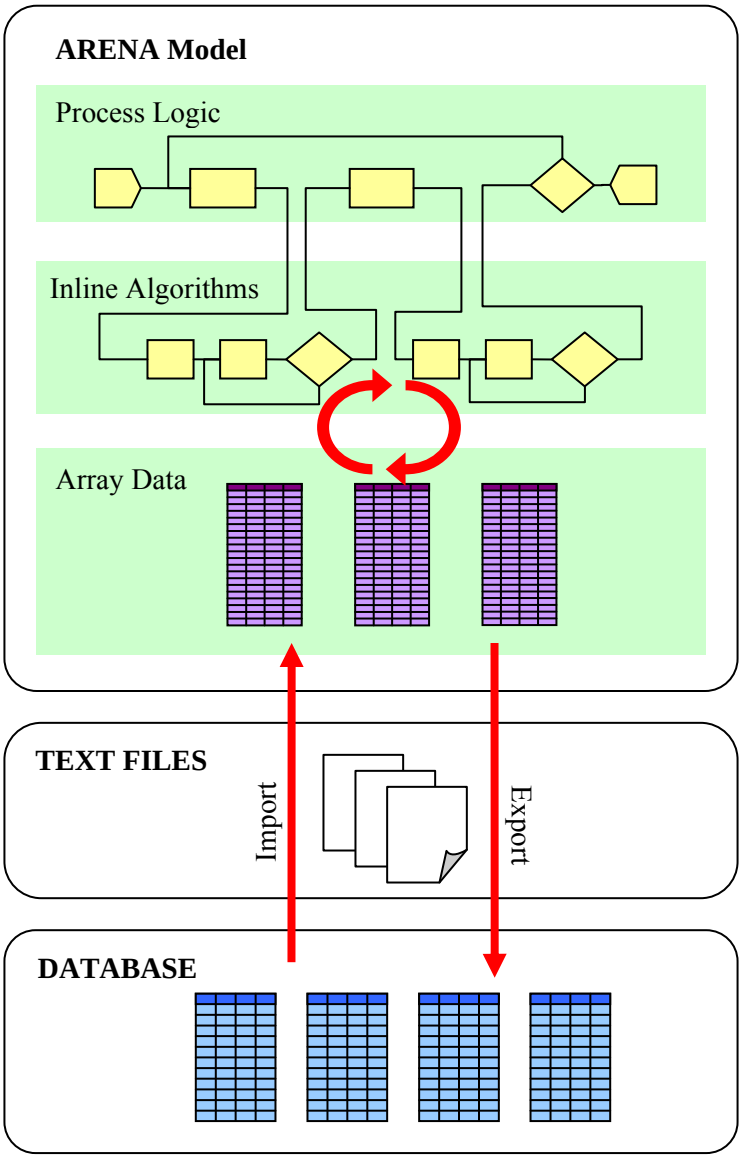

Figure 5: Architecture and data flow of the MARS V1managed readiness model.

All the data required by the model were stored in Arena's global variables and arrays. In order to initialize the data that specify the simulation scenario, a two-step data import operation from a database was automatically performed. First, the data required by the current simulation scenario were written from the Access database to a collection of comma-separated text files. Then, the text files were read into Arena's internal variables and arrays in memory. During simulation execution, the model wrote a comprehensive record of simulation events to output text files. After the simulation completed, the data in the output text files were transferred back to the Access database for post processing and analysis.

Both the algorithm and the process logic aspects of the model were implemented together as one large DES process using Arena's discrete event building blocks. At the highest level, the DES process represented the managed readiness process logic of MARS. The algorithms that act on the data stored in Arena's variables and arrays were then implemented at various points in-line with the process logic. Figure 6 shows an example of a bubble sort algorithm embedded inline with the high-level process logic and an array of rank data that it manipulates. Though this is a simple example, the algorithm implementation adds considerable bulk to the DES process. Throughout the MARS V1 model, typical algorithms involved in filtering resources and matching resources to slots based on their attributes are much more complicated.

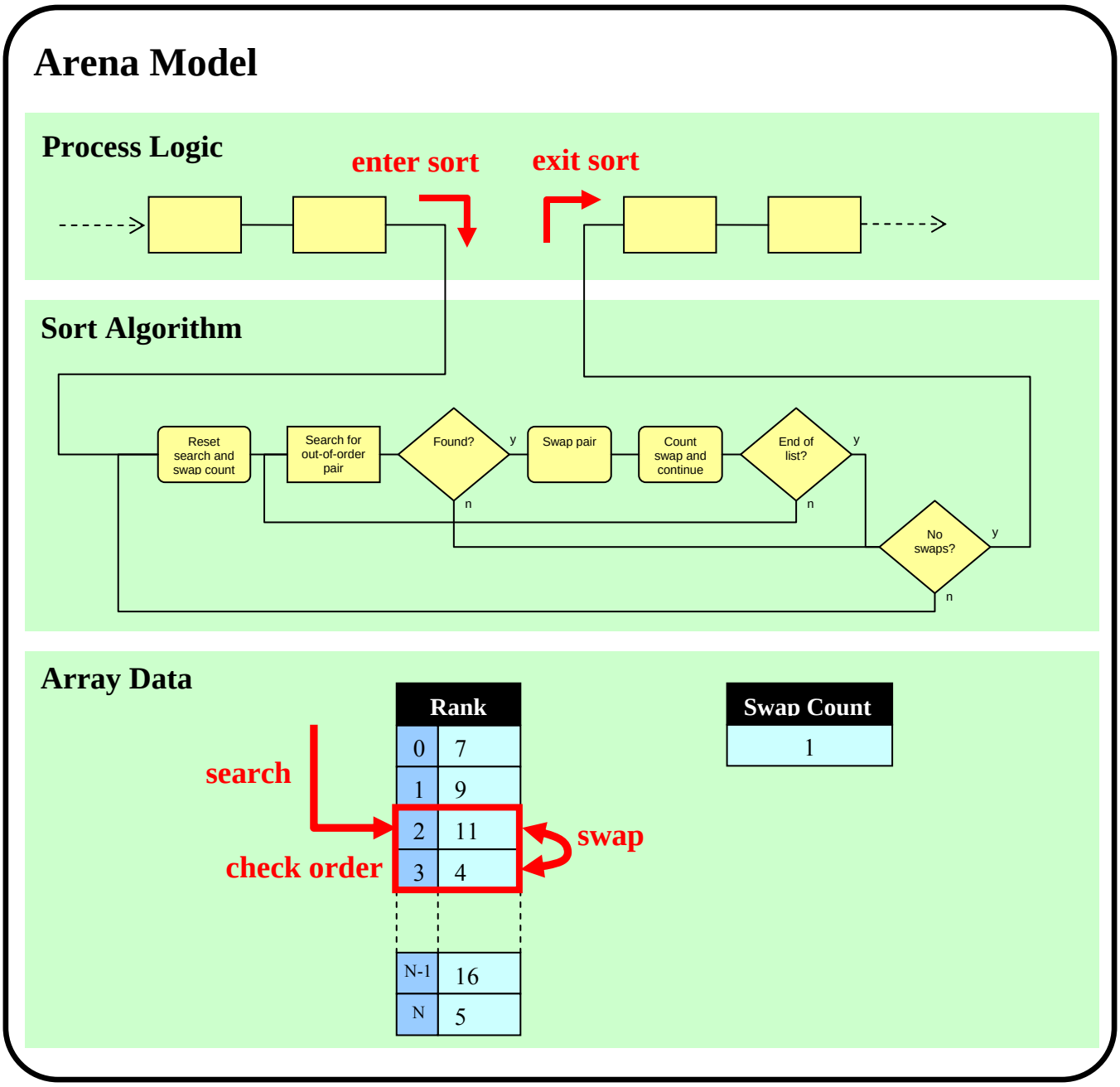

Figure 6: Example of a MARS V1 bubble sort algorithm implemented in-line with the top-level process logic acting on array data containing rank values.

5 MARS V1 Technical Limitations

Using the architecture described in the previous section, MARS V1 was used to create scenarios to evaluated the Army's managed readiness system [5]. However, in conducting this analysis, some aspects of this architecture proved challenging to work with in practice and some scenarios which would be considered of only moderate size and complexity were approaching the technical limits of what could be achieved in MARS V1. This was expected to seriously hinder important future work in MARS. In particular, the implementation of a model of establishment dynamics was considered infeasible in this architecture. The limitations identified in MARS V1 fall into four main areas which are described in the next four sub-sections.

5.1 Data Volume

MARS V1, and simulations in general, require the model and data to be loaded from the hard drive into memory for the simulation to execute. But because memory space is small compared to hard drive space, this limits the volume of data that can be used by the simulation and limits complexity of the model compared to what can be stored on the hard drive. In the case of MARS V1, for a scenario of moderate size and complexity, the Arena application allocated approximately 700 megabytes of memory on initialization to store the model and data. This approached the memory limits for a single program on current PC hardware. While most modern hardware supports several gigabytes of memory, memory is a shared resource competed for by all running processes and their data. As a result, allocating this much memory can produce unpredictable behaviour. The operating system may deny the application's attempt to allocate this much memory causing the simulation to fail, or sluggish performance and system instability may result as other processes compete for scarce memory and get swapped into virtual memory.

The fact that memory limitations were encountered in a moderately sized simulation scenario meant that future development options were very limited. It would not be possible to import more data to represent a more comprehensive scenario (e.g. including the Navy and Air Force establishments in addition to the Army establishment) or to implement new features requiring substantial additional data (e.g. modeling training courses and enforcing qualification requirements). No such practical limitation exists for the data and model specification when they reside on the hard drive. Ideally the simulation should be able to process data of any size and models of any complexity that are available and relevant to a given scenario.

5.2 Data Structures

The MARS V1 model stored all scenario data in Arena's arrays for simulation execution. Arena arrays are static, meaning their size is fixed and must be declared before the run starts. However, for the purposes of the managed readiness model, static arrays require careful maintenance, are inefficient, and are prone to errors. This is because the size of each array must be chosen carefully to ensure that it is equal to or exceeds the number of rows that will be used when the simulation runs.

In some cases, the required number of rows is known, so it is possible to set the array size to the correct value. For example, the number of rows required for input data is known before the simulation starts, but this means that anytime the input data changes, the corresponding Arena array size must change with it. It is possible to automate this array size management in Arena but this was not done in MARS V1 because the implementation requires advanced knowledge of Arena and involves low-level coding to adjust the model.

However, in some cases the required array size is not known for a given scenario. This is typically the case for arrays used to store intermediate results, such as a list of resource candidates used to fill theatre slots. Usually the number of rows required for these intermediate results is small, but they must also be able to handle boundary cases where potentially very large intermediate datasets are generated. Otherwise the array will overflow producing a fatal execution error.

In MARS V1, the approach in sizing arrays was to make them larger, by some safety factor, than an estimate of what would be required. But this approach is bound to fail eventually as increasingly complex simulation scenarios consume and process greater volumes of data. Because MARS V1 was also running into memory limitations, increasing arrays sizes on a trial-and-error basis was not a sustainable practice. The conventional solution to this problem is the use of a dynamic data structure such as a list or dictionary but Arena does not yet support these features.

5.3 Algorithms

The discrete event process paradigm is not well suited to implementing algorithms. Data operations like searching, sorting, matching and updating are CPU-heavy procedures involving intensive looping and data access which require a considerable number of blocks and complex program flow to implement as a DES process. Inserting these DES algorithms in-line with the process logic adds considerable bulk and complexity to the model. Further, the knowledge and time required to implement the algorithms in Arena is substantial, and the probability of introducing programming errors is increased. In MARS V1 approximately 90% of the overall Arena model consisted of algorithms with the remaining 10% devoted to the top level process logic. While other programming languages typically provide data manipulation operations as low-level, optimized functions for advanced data structures, Arena does not include such functionality and provides only limited features to support working with its static arrays.

Because MARS V1 combines the high-level process logic and the low-level algorithms into one large DES process, it became a challenge to debug, modify and enhance the model. In Arena, all objects and variables have global scope and there is no ability to encapsulate the implementation of a portion of the process analogous to a function or object method in other programming languages. Thus any change to one part of the model has the potential to affect or break other parts of the model which means the developer effort required to implement small changes is very high. This presents a serious challenge to maintaining MARS in the long term because debugging and feature enhancement are regular activities of the evolutionary development approach being employed.

5.4 Performance

The run time for a MARS V1 simulation included a significant overhead required to import scenario data from the database into Arena, to initialize the model, and to export simulation output back to the database. Twenty to thirty minutes of the total simulation run time was devoted to this overhead which was largely independent of the specific scenario being simulated. This made regular testing and debugging of the model very slow because any scenario, no matter how small, needed these twenty to thirty minutes as a minimum to produce results.

Table 1 shows the times required to complete various stages of a simulation run in a realistic scenario that consisted of seven large Task rotations each allocating approximately 2000 resources to 18 Activities from an establishment of 20 000. In this example, the total overhead was 25m 10s for a simulation that took over six hours to complete. The large size of the Arena model was responsible for the long initialization time of 9m 42s. The large volume of input data and complex data import process were responsible for the 12m 58s data import time. The export of simulation output back to the database was not a significant time cost at 2m 30s. This is because the raw output data was already in text form after the execution step completed, and the bulk transfer of text data to database is relatively fast.

Table 1: Sample MARS V1 simulation run times.

Run Step	Time (hh:mm:ss)
Initialize	0:09:42
Import Data	0:12:58
Execute	6:08:58
Export Data	0:02:30
Total	6:34:08

The technical limitations described in this section led to an investigation of possible technological solutions.

6 Proposed Technical Approaches

Various options were available to address the technical issues encountered in MARS V1. Several of the issues described above related to difficulties in implementing the managed readiness model in Arena, therefore two potential options were to either replace Arena with another commercial simulation package or to contract the development of a customised simulation engine. A third alternative was to retain Arena but to exploit other technology to handle those aspects of the model that are not well suited to Arena's feature set.

An option assessment was carried out by the MARS development team which included recommendations from the Land Software Engineering Centre. This assessment concluded that Arena should be retained as the simulation engine for MARS for several reasons. First, no comparable DES software offers features that would specifically address the issues identified in MARS V1. Second, contracting the development of a custom simulation engine for MARS would transfer the in-depth knowledge of the engine from LFORT to the contractor. This would be a substantial hindrance to progress from LFORT's perspective because the simulation engine is in a state of regular development and refinement. Finally, the initial reasoning behind the choice of Arena for MARS remained valid: There is a knowledge base of experienced Arena users within CORA and the Director General Military Personnel Research and Analysis, and the other major models with which MARS is likely to interface in the future are also built in Arena. These models include the Arena Career Modelling Environment [6], the Production Management Tool [7] and the Human Resources Operational Sustainability Model [8].

Based on this reasoning, LFORT proposed a solution that retained Arena as the simulation engine but that leveraged other technology to address the issues identified in the previous section. The limitations discussed above generally relate to the challenge of implementing the data and algorithm aspects of the managed readiness model in Arena. Therefore, it was proposed that relational database technology could provide an ideal platform for implementing the large-scale data storage and data manipulation algorithms required by the MARS model. This implies the use of a database, not only as a data source, but also as the runtime computational workspace used by the Arena simulation. This approach was termed a Simulation Runtime Database (SRDB).

7 The Simulation Runtime Database Approach

In the SRDB approach, the database's tables replace the runtime memory arrays used by Arena, and the database's own algorithms replace significant portions of the DES algorithms previously implemented within the Arena process. Further, the scenario data never leave the database for runtime processing. Instead, the simulation directly manipulates the database to advance the simulation state. Access to the database is managed using Arena's VBA automation capability. Figure 7 illustrates the model architecture using the SRDB. Note that the data and algorithm parts of the model have been moved into the SRDB which drastically simplifies the model architecture compared to MARS V1 as shown in Figure 5, above.

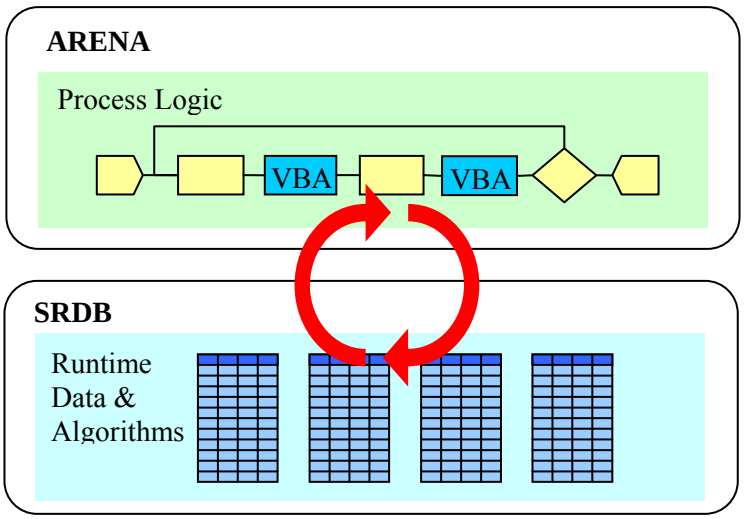

Figure 7: Architecture and data flow of the MARS V2 Managed Readiness Model using the SRDB.

The SRDB approach specifically addresses the major technical issues identified in MARS V1. The impact of the SRDB on the four problem areas described in the Section 5 are detailed below.

7.1 Data Volume

A database provides data storage that is not limited by available memory because it resides on the hard drive. Databases are still limited in their size, in some cases by the software (MS Access has a 2 gigabyte file size limit), and ultimately by hard drive space (on the order of 100s of gigabytes to terabytes). But even in the case of MS Access, the 2 gigabyte file size limit provides substantially more data processing capacity than a typical 2 gigabyte memory space because memory must be shared with all other data and programs running on the computer, and access to memory is controlled by the operating system.

Database interfaces are also standardized for most programming languages which means that one database can be swapped for another with minimal impact on the other parts of the managed readiness model. For example, if the 2 gigabyte file size cap on MS Access becomes a limitation,

it should be possible to upgrade to a more powerful database management system without altering the Arena model.

7.2 Data structures

Databases store data in tables which are very powerful data structures that provide many advantages over the static arrays provided by Arena. Most importantly, tables are dynamic in that they automatically grow and shrink anytime records are added to or deleted from the table. Thus, the simulation developer is no longer responsible for managing the size of the simulation's runtime data structures. Further, tables provide convenient features making them ideal for large-scale data management: they are inherently 2 dimensional with named columns and column data formats; column relationships can be established (1:1, 1:∞, ∞:1) that can be enforced by the database; and the data are concurrently accessible through standard programming interfaces by any software.

Databases also provide advanced features designed to prevent data corruption while actions (known as transactions) are performed on it. For example, during a data update step in MARS V1, an unhandled algorithm error condition could leave the in-memory data in an invalid state where some but not all of the updates completed successfully. The simulation would then proceed using this corrupted data. The same data update step performed on a database would report the error, stopping the simulation, and roll all the data back to its initial valid state before the transaction started. This facilitates debugging and improves the reliability of simulation results.

7.3 Algorithms

Relational databases provide a standard language for manipulating data stored in tables called Structured Query Language (SQL) [9]. An SQL engine is implemented by the database which parses and executes SQL statements. SQL queries can be used to insert and delete data, sort data, search for data linked across multiple tables subject to complex criteria, aggregate results providing properties of the data (counts, sum, mean, standard deviation, variance, minimum, maximum) update data using regular mathematical expressions and parameters, among many other capabilities.

Within the context of the managed readiness model, SQL statements can be used at runtime to instruct the database to carry out the complex data processing algorithms that were previously implemented within the Arena DES process in MARS V1. This would replace the largest and most complex parts of the MARS V1 model with concise SQL statements that are easy to create and modify. For example, the bubble sort algorithm shown in Figure 6, which requires seven blocks and several custom variables and attributes, would be replaced by one line of VBA code that executes an SQL statement: *SELECT * FROM Table ORDER BY Rank*.

7.4 Performance

The SRDB approach almost completely eliminates the overhead time observed in MARS V1 simulations. Firstly, the data import and export steps are eliminated because the data never leave

the database. Runtime data processing simply occurs directly on the source database. Secondly, the smaller size of the DES model, by virtue of replacing DES algorithms with SQL, substantially reduces the number of initialization steps that precedes a run. Small simulation scenarios will therefore run very quickly because of the small overhead.

However, because the SRDB is an entirely different architectural approach from MARS V1, it was not clear how the execution speed of the simulation would be affected. Intensive hard drive input/output (I/O) activity during runtime was expected to impose a performance penalty. But the use of optimized database algorithms and the simpler model implementation had the potential to improve execution speed. A full scale trial simulation of comparable scale and complexity to those conducted in MARS V1 would be required to completely assess the performance impact of the SRDB approach. Such a trial will be discussed later in Section 8.

8 Feasibility of Implementing the SRDB in Arena

In order to assess the technical feasibility of the SRDB approach, a small prototype was developed in Arena and a simple SRDB was built in MS Access. This prototype had to successfully demonstrate the following capabilities in order to prove that the SRDB approach was feasible:

1. Establish a connection to the database that can be used during Arena execution.

2. Invoke SQL commands at runtime from within the Arena discrete event process to manipulate data in the database.

3. Read and react to SQL query results within the Arena discrete event process.

Arena's VBA programming interface provides low level control over the simulation including access to variables, entity attributes, and states of blocks in the process diagram. The user can implement VBA subroutines that are automatically called before simulation execution starts and after it finishes. Additionally, Arena process building blocks include a VBA block which can be embedded in the discrete event logic. The VBA block executes a user-implemented subroutine when an Arena entity passes through it. The integration of VBA in Arena provided the flexibility needed to establish a user-controlled interface with the database.

For the prototype, a minimal DES process was built and the following elements were added to demonstrate the three requirements listed above.

1. A VBA subroutine was written that establishes a global connection to the database using the Microsoft Data Access Objects (DAO) 3.6 Object Library. This subroutine was set up so that it would run just prior to starting the simulation so that the database connection would be open during execution.

2. A VBA block was inserted into the process diagram so that when an entity enters the block during the simulation, a subroutine is called that sends an SQL update command via the DAO interface to alter data in the database.

3. A second VBA block was inserted in the process diagram that sends an SQL query to the database and reads the resulting dataset. The code then adjusts the attributes of the entity depending on those results.

The third element above is a crucial test demonstrating that the state of an entity can be altered depending on the result of a database query. This allows the simulation to respond to information obtained from the database. For example, when an activity has assembled its resources, it must query the database to determine if the resources it found are sufficient for the activity to run. The attributes of the active Arena entity would then be updated in the VBA code based on the query result. Subsequent DES process logic can then respond to the entity's attributes by directing the activity either to run as scheduled if sufficient resources were found or else to be cancelled.

Testing of this prototype was successful in demonstrating the three criteria listed above. This result proved that the SRDB approach was feasible within Arena. While the full performance impact of this approach was not known, its potential to address all the major issues identified in MARS V1 justified undertaking the development effort to implement MARS V2 using an SRDB.

9 MARS V2 Model Architecture

Implementing the SRDB for the MARS V2 model consisted, first, of modifying the original scenario database from MARS V1 in order to take on the role of the SRDB and, second, of rebuilding the managed readiness model to take advantage of the new capabilities provided by the SRDB.

9.1 Implementing the MARS V2 SRDB

To create the SRDB, the original scenario database from MARS V1 was altered to handle runtime data processing tasks. This consisted of adding tables to handle two major runtime functions. First, object state tables were created to store the current state of all dynamic objects in the simulation. These include the resource attribute table, the slot attribute table, various tables that define resource groups and their current activities, and several tables tracking the states of activities and tasks. Second, temporary tables were created to store the intermediate results of data processing steps. These tables were used for many operations throughout the simulation including storing attribute requirement lists, attribute update instructions, part type requirements, lists of required resources and lists of resource candidates.

The storage of simulation input and output data was already a function of the MARS V1 scenario database so these tables did not change except in how they were used. In MARS V2, the model did not import the input data at the start of the simulation; it accessed it directly when required at runtime. And the model did not export simulation output back to the database after the simulation; it wrote directly to the output tables as simulation events occurred.

9.2 Implementing the MARS V2 Arena Process Logic

Integrating the SRDB into the MARS V2 model process logic followed the methodology used in the feasibility study described in Section 8. A VBA subroutine that ran just before the start of the simulation established a global connection to the SRDB using the DAO library. Then, VBA blocks placed within the process logic carried out the data processing tasks associated with simulation events. The VBA code used SQL to manipulate and query the database and reacted to query results.

The DAO interface allows SQL queries to take two forms. First, SQL scripts can be defined directly in the VBA code and sent to the database. This method is appropriate for simple queries such as adding and deleting records and for querying single tables with straightforward criteria. However, for more complex queries, the SQL statement can become very long and difficult to interpret. For these larger queries, a second method was implemented which involves defining the query as an object in the database. Most databases provide this capability. In MS Access, SQL query objects can be designed using a visual interface that allows very complex queries to be built quickly and easily. Query fields, table joins, nested queries, field criteria and parameters are represented graphically in the MS Access query design tool. The query is given a name and saved as an executable object in the Access database. The VBA block code in the Arena model then calls the query object by name, passing in the necessary parameters, and retrieves the result

and acts on it if necessary. This approach very efficiently exploits a powerful commercial tool to implement the complex data-processing algorithms used in the MARS model. Data operations that would take weeks to implement as a DES process in MARS V1 can be developed in minutes using the Access query design tool.

All the algorithms implemented in Arena in MARS V1 were replaced with VBA code blocks that sent SQL commands to the database by one of the two methods described above. This drastically simplified the Arena DES process. For example, the Arena-implemented sorting algorithm shown previously in Figure 6 would be implemented in MARS V2 as shown in Figure 8 using a VBA block and SQL. In this case, when the SQL command is invoked, the database responds by retrieving the requested data, sorting it, creating a temporary result recordset in memory and returning this to the Arena VBA block to be read and acted upon. In general the SRDB approach makes available the most powerful aspects of database technology to the MARS developer in implementing the managed readiness model.

For further detail, Annex A provides instructions for setting up an Arena model to interact with MS Access using VBA. Annex B provides VBA sample code that shows how to connect to the database, send SQL commands, and retrieve query results from within Arena.

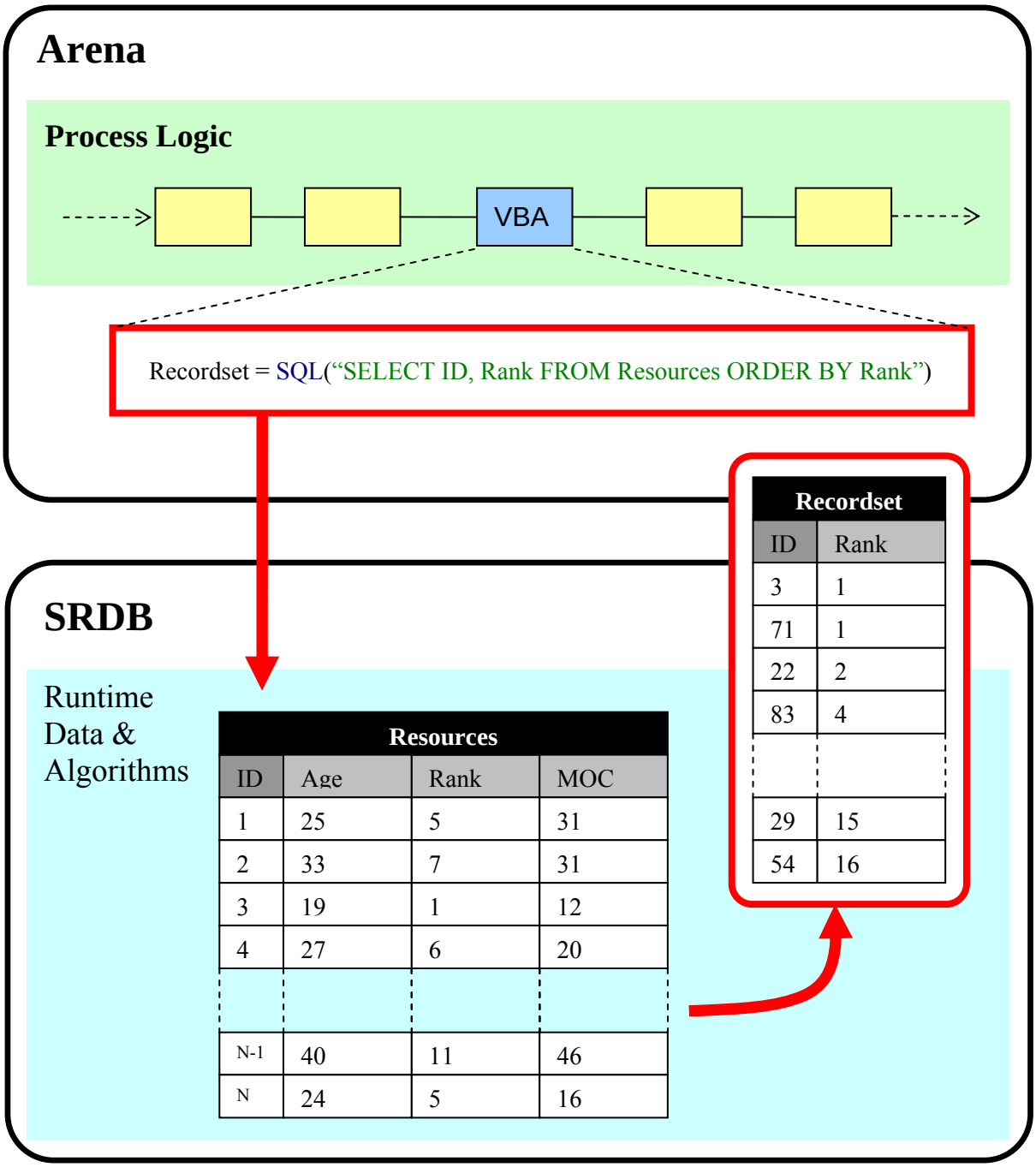

Figure 8: MARS V2 Implementation of a sorting operation using an SQL command sent to the SRDB.

10 Performance of the SRDB

Because the MARS V2 runtime data resides in the SRDB on the hard drive, the approach was expected to suffer a performance penalty as hard disk I/O is significantly slower than memory I/O. However, other aspects of the SRDB approach improved overall performance and compensated for the cost of increased hard disk I/O. Ultimately, the SRDB approach was found to perform better than the conventional in-memory approach in MARS V1. Table 2 compares the run times measured previously in MARS V1 to those of MARS V2 using a scenario that matched as closely as possible the scenario described in Section 5.4. While MARS V2 saved time by nearly eliminating the runtime overhead, note that the pure execution time was also less than half that of MARS V1.

Table 2: Comparison of MARS V1 and V2 simulation run times.

	Time (hh:mm:ss)	
Run Step	MARS V1	MARS V2
Initialize	0:09:42	0:00:42
Import Data	0:12:58	0:00:00
Execute	6:08:58	2:49:57
Export Data	0:02:30	0:00:00
Total	6:34:08	2:50:39

The improved execution speed of MARS V2 using the SRDB was attributed to several factors that positively affect performance. The impact of these factors is described below.

If the simulation runs in memory, the input data must be moved from the hard disk to memory before the simulation starts and the output data must be moved back the hard disk after it finishes. The process of writing from the database to text files, reading the text files into Arena's memory, writing output events to text files and reading the output text files back to the database involves multiple read/write operations on the hard drive for the entire input and output datasets. Thus, even for memory-based models, extensive hard-disk I/O is unavoidable. Additionally, much of the data import step is often wasted effort because a given simulation scenario will frequently only access a fraction of the whole input dataset. The SRDB approach is more efficient in that it only accesses what data are necessary. Thus, it is not necessarily true that memory-based simulations involve less hard disk activity than the SRDB approach.

Developing algorithms for processing data structures is a complex subject. Optimizing these algorithms for scale, performance, and memory efficiency is the occupation of a field of computer science. The algorithms provided by databases and invoked through SQL have the benefit of being mature, robust, and specifically optimized for scale. Conversely, in most instances algorithms built using DES process logic are not well tested and are unlikely to scale well as the problem size increases. For example, in MARS V1, DES search algorithms were implemented as exhaustive searches which scale proportionally with the number of items to be searched, $O(n)$ for n items in big-O notation [10]. In MARS V2 using the SRDB, the indexing of table fields allows searches to be performed much more efficiently than $O(n)$. For MS Access tree-based indexes, the search time is proportional to the logarithm of the number of items, $O(\log(n))$[10]. While

each individual data access event is slower in the database than in memory, given a list of 1000 items, the database's search requires two orders of magnitude fewer steps on average than an exhaustive search. Further optimizations in databases, such as keeping indexes in memory, and the efficient use of the hard drive cache bring overall database performance close to memory I/O performance for many data operations.

Moving the data and algorithm components of MARS into the database also provides options for speed optimization independent of the process logic which remains in Arena. In the MARS V2 SRDB, it was possible to improve performance substantially through the use of table field indexing and the design of queries. If further performance gains are required in the future, it is possible to switch to one of several databases designed specifically for performance. If true memory-based performance is desired, some databases allow the creation of temporary tables that reside in memory, such as MySQL [11], and some databases can be run entirely in memory, such as SQLite [12]. These in-memory database options would provide the benefits of tables and SQL and would eliminate the performance penalty of hard-disk access. However, an in-memory database would still have to import from and export to a permanent database on the hard-disk.

11 Conclusions

This paper described technical limitations in the architecture of the MARS V1 managed readiness model. LFORT devised a novel solution to address these limitations in which the powerful data management and data processing capabilities of relational databases were incorporated into simulation execution. This was termed a Simulation Runtime Database, or SRDB, approach and was implemented in MARS V2. This provided MARS V2 the ability to process a much larger volume of data using more powerful data structures. It also allowed SQL to be used to invoke complex data processing algorithms and produced a substantial performance increase compared to MARS V1. Ultimately, the SRDB architecture provided a more powerful foundation on which to implement important aspects of modelling readiness for the CF that were not possible in the MARS V1 architecture, including the dynamics of the establishment. The design and implementation of these new features will be described in future publications on MARS V2. In general, a much greater degree of technical flexibility was achieved by implementing the SRDB architecture, and this will allow MARS V2 to meet the increasingly complex simulation needs of CF decision makers.

References

[1] Ormrod, M., Young, C., and Pall R. (2007). Modelling Force Generation with the Managed Readiness Simulator (MARS): Modelling Concept and Requirements for MARS v1.0. DRDC CORA Technical Memorandum TM 2007-65.

[2] Pall, R., Young, C., and Ormrod, M. (2007). Modelling Force Generation with the Managed Readiness Simulator (MARS): Implementation of MARS v1.0 in a Discrete Event Simulation Environment. DRDC CORA Technical Memorandum TM 2007-52.

[3] Young, C., Pall, R., and Ormrod, M. (2007). A Framework & Prototype for Modelling Army Force Generation. DRDC CORA Technical Memorandum TM 2007-54.

[4] Kelton, D.W., Sadowski, R. P., Sturrock, D. R. (2004) Simulation with Arena. 3rd ed. Boston: McGraw-Hill Higher Education, 2004.

[5] Ormrod, M., Young, C. (2007). Preliminary Analysis of Task Force Afghanistan Sustainability Using MARS. DRDC CORA Technical Memorandum TM 2007-40

[6] Isbrandt, S., Zegers, A. (2006). The Arena Career Modelling Environment Individual Training and Education (ACME IT & E) Projection Tool. DRDC CORA Technical Report TR 2006-03.

[7] Straver, M. C., Okazawa, S., Wind, A., Moorhead, P. (2009) Training Pipeline Modelling Using the Production Management Tool. DRDC DGMPRA Technical Memorandum TM (DRAFT).

[8] Moorhead, P., Wind, A., Halbrohr, M. (2008) A Discrete Event Simulation Model for Examining Future Sustainability of Canadian Forces Operations. Proceedings of the 2008 Winter Simulation Conference. Mason, S. J., Hill, R. R., Mönch, L., Rose, O., Jefferson, T., Fowler, J. W. eds.

[9] Bowman, J. S., Emerson S. L., Darnovsky, M. (1996) The Practical SQL Handbook: Using Structured Query Language. 3rd ed. Addison-Wesley Developers Press, 1996.

[10] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. (2007) Introduction to Algorithms. 2nd ed. The MIT Press, 2007. pp. 44 - 45, 432.

[11] MySQL 5.0 Reference Manual: The Memory (Heap) Storage Engine. http://dev.mysql.com/doc/refman/5.0/en/memory-storage-engine.html

[12] SQLite: In-Memory Databases. http://www.sqlite.org/inmemorydb.html

Annex A Setting up Arena for MS Access Connectivity

1. Attach the Blocks template (if not already present in the templates panel)
   a. Open an Arena project file
   b. Under the **File** menu, select **Template Panel** > **Attach…**
   c. Select **Blocks.tpo** and click **Open**
2. Add a VBA block from the Blocks template to the model, as in Figure A-9

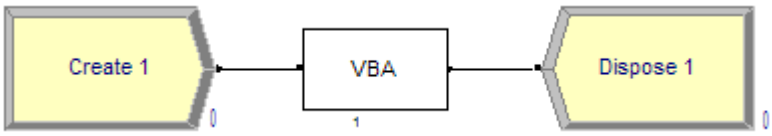

Figure A-9: Sample Arena model with VBA block

3. Double-click the VBA block to enter the VB Editor
4. Setup VB References to connect to Access databases
   a. In the VB Editor, under the **Tools** menu, select **References…**
   b. Check **Microsoft DAO 3.6 Object Library**
   c. Click **OK**

Annex B VBA Sample Code for Interacting with MS Access from Arena

```
'connect to a database
Dim dbScenarioDB As Database
Set dbScenarioDB = OpenDatabase("Data.mdb")


'execute action SQL queries
dbScenarioDB.Execute "DELETE * FROM [Scenario Information]", dbFailOnError
dbScenarioDB.Execute "INSERT INTO [Scenario Information] VALUES (365, 20)",
dbFailOnError


'open the result of an SQL select query
Dim rsTable1 As Recordset
Set rsTable1 = dbScenarioDB.OpenRecordset("SELECT * FROM [Personnel] WHERE Age > 25")


'read the values from the query result
Dim Age
Dim QualLevel

If rsTable1.RecordCount > 0 Then

    rsTable1.MoveFirst

    Do While Not rsTable1.EOF

        Age = rsTable1("Age")
        QualLevel = rsTable1("Qual Level")
        rsTable1.MoveNext

    Loop

End If

rsTable1.Close


'execute a query created in Access with parameters
Dim QDef As QueryDef
Set QDef = dbScenarioDB.QueryDefs("UpdateAge")
QDef.Parameters("pYears") = 1
'QDef.Parameters("2nd parameter") = x
'QDef.Parameters("3rd parameter") = y
'...
QDef.Execute dbFailOnError
QDef.Close


'open the result of a select query created in Access with parameters
Set QDef = dbScenarioDB.QueryDefs("QualifiedPers")
```

```
QDef.Parameters("pMinQual") = 10
Set rsTable1 = QDef.OpenRecordset
QDef.Close
'access the records in rsTable1 as above
rsTable1.Close

dbScenarioDB.Close
```

This page intentionally left blank.

Distribution list

Document No.: DRDC CORA TM [enter number only: 9999-999]

(Report distributed by CD unless otherwise noted)

LIST PART 1: Internal Distribution by Centre

3 Authors (Hard Copies)
1 DG DRDC CORA
1 DDG DRDC CORA
1 Chief Scientist DRDC CORA
1 Section Head, Land OR (Email: Dean.Haslip@forces.gc.ca)
1 LFORT
1 LCDORT
1 DRDC Valcartier OR
2 DRDC CORA Library (1 Hard Copy, 1 CD)
1 DGMPRA Personnel Generation Research Section

14 TOTAL LIST PART 1

LIST PART 2: External Distribution by DRDKIM

1 ADM(S&T) (for distribution)
1 Director S&T Land
1 DRDKIM 3
1 DG DRDC Valcartier
1 CF College Library
1 Fort Frontenac Library
1 RMC personnel as required
1 COS(Ops) [DGLS]
1 COS(Strat) [DGLCD]
1 LFDTS
1 DGL Res as required
1 DLS [DLSP]
1 G1 [DLPM]
1 G3 [DLFR]
1 G4 [DLSS]
1 DLCD
1 DLFD
1 DLR
1 DLCI
1 DAT
1 DAD
1 DLSE
1 CISTI

1 Document Exchange Manager
DSTO Research Library
Defence Science & Technology Organisation
PO Box 44
Pyrmont NSW 2009
AUSTRALIA

1 Dr. Neville J Curtis
Research Leader Land Operations Research
75 Labs
Land Operations Division
PO Box 1500
Edinburgh SA 5111
AUSTRALIA

1 Chief Analyst
Land Battlespace Systems
Dstl Integrated Systems
Room 31, Bldg A3, Fort Halstead
Sevenoaks, Kent, UK, TN14 7BP

1 Dr. Jason Field
Land Battlespace Systems
Dstl Integrated Systems
Fort Halstead
Sevenoaks, Kent, UK, TN147BP

1 Director, US AMSAA
ATTN: AMSRD-AMS-S)
392 Hopkins Road
APG, MD 21005-5071

1 Mr. Patrick O'Neill
Chief, Combat Support Analysis Division USAMSAA (ATTN: AMSRD-AMS-S)
392 Hopkins Road
APG, MD 21005-5071

1 Dr. James T. Treharne
OCA Division
Center for Army Analysis
6001 Goethals Road
Fort Belvoir, VA 22060-5230

1 Mr. Robert Barrett
Chief, International Activities
Center for Army Analysis
6001 Goethals Road
Fort Belvoir, VA 22060-5230

1 Mr. John Hughes
HQ, TRADOC Analysis Center (TRAC)
Programs & Resources Directorate (PRD)
255 Sedgwick Avenue
Fort Leavenworth, Kansas 66027-2345

1 Ms. Belinda Smeenk
TNO Defence, Security and Safety
Information and Operations
P.O. Box 96864, 2509 JG
The Hague, The Netherlands

1 Mr. Bob Barbier
TNO Defence, Security and Safety
Information and Operations
P.O. Box 96864, 2509 JG
The Hague, The Netherlands

35 TOTAL LIST PART 2

49 TOTAL COPIES REQUIRED

This page intentionally left blank.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Defence R&D Canada – CORA
101 Colonel By Drive
Ottawa, Ontario K1A 0K2

2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.)

UNCLASSIFIED

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

Managed Readiness Simulator (MARS) V2: Assessment of a Simulation Runtime Database Approach

4. AUTHORS (last name, followed by initials – ranks, titles, etc. not to be used)

Okazawa, S.; Ormrod, M.; Young, C.

5. DATE OF PUBLICATION (Month and year of publication of document.)

September 2009

6a. NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.)

49

6b. NO. OF REFS (Total cited in document.)

12

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Memorandum

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

Defence R&D Canada – CORA
101 Colonel By Drive
Ottawa, Ontario K1A 0K2

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)

9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)

10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)

DRDC CORA TM 2009-042

10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

Unlimited

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected.))

Unlimited

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

This paper describes technical enhancements implemented in the second version of the Managed Readiness Simulator (MARS V2). These enhancements were required to address limitations in the technology and implementation of the initial prototype (MARS V1). The limitations related to data volume, data structures, algorithms and performance, which prevented larger and more complex scenarios from being simulated. Therefore, a range of potential approaches was identified, and a novel model architecture was devised that would address these limitations. The approach consisted of exploiting database technology during simulation execution which was termed a Simulation Runtime Database (SRDB). This allowed the simulation to use the database's table data structures for runtime data storage and to invoke data processing algorithms through Structured Query Language commands. A feasibility study was conducted that proved that the SRDB could be implemented in the MARS simulation engine, Arena. This justified the implementation of the MARS V2 model using the SRDB approach. Testing of the completed MARS V2 model showed that it successfully addressed all four of the limitations identified in MARS V1. Ultimately, the SRDB approach provided a much more capable platform for large, complex simulation scenarios that will meet current and future client needs.

Dans ce document, on décrit les améliorations techniques apportées à la deuxième version du programme de simulation de gestion de la disponibilité opérationnelle (MARS V2). Ces améliorations étaient nécessaires pour éliminer des faiblesses dans la technologie et la mise en œuvre du prototype initial (MARS V1). Les faiblesses, reliées au volume de données, aux structures de données, aux algorithmes et aux performances, empêchaient la simulation de scénarios plus grands et plus complexes. Par conséquent, on a identifié un éventail d'approches potentielles et on a créé une architecture de modèle novatrice permettant d'éliminer ces faiblesses. L'approche adoptée consistait à exploiter une technologie des bases de données pendant l'exécution des simulations, que l'on a baptisée SRDB (Simulation Runtime Database). Cela permettait à la simulation d'utiliser les structures de données des tables de la base de données pour le stockage des données pendant l'exécution et d'invoquer des algorithmes de traitement des données au moyen de commandes SQL (Structured Query Language). Une étude de faisabilité a prouvé que la technologie SRDB pouvait être mise en œuvre dans Arena, le moteur de simulation du programme MARS. Cela justifiait la mise en œuvre du modèle MARS V2 à l'aide de l'approche à SRDB. La mise à l'essai du modèle MARS V2 terminé a démontré que ce dernier éliminait avec succès les quatre faiblesses identifiées dans MARS V1. En fin de compte, l'approche SRDB a permis d'obtenir une plateforme beaucoup plus puissante pour de grands scénarios de simulation complexes qui répondra aux besoins présent et futurs des clients.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Managed Readiness Simulator; MARS; MARS V2; Simulation Runtime Database; SRDB; Simulation; Forecast; Readiness

DRDC CORA

www.drdc-rddc.gc.ca