Proceedings of the 2013 Winter Simulation Conference
R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, eds

A DISCRETE EVENT SIMULATION ENVIRONMENT TAILORED TO THE NEEDS OF MILITARY HUMAN RESOURCES MANAGEMENT

Stephen Okazawa

Defence Research and Development Canada
Department of National Defence
Ottawa, ON K1A 0K2, CANADA

ABSTRACT

The management of military human resources (HR) is a complex problem. Discrete event models of military HR systems are used by the Canadian Department of National Defence to provide military decision makers with greater knowledge of the outcome of possible courses of action. However, models of military HR systems have unique characteristics that most discrete event simulation software products do not cater to. Specifically, military HR models tend to be complex rule-based models, they process large amounts of data, and the ability to interconnect models is very desirable. This paper presents a novel discrete event simulation environment being developed by Defence Research and Development Canada called “Right Person, Right Qualification, Right Place, Right Time Human Resources” which is tailored to the particular needs of military HR modeling and simulation.

1 INTRODUCTION

The management of military human resources (HR) is a complex area in which decision making affecting the careers of military personnel often produces unforeseen and long-lasting consequences. Without careful analysis, a decision to change personnel force structure, training programs, retention policies, or operational commitments can unknowingly produce shortages and/or excesses of personnel in various occupations at various ranks at various times. Ultimately, these unexpected outcomes may result in a loss of operational readiness and can take years, if not decades, to correct.

Within Defence Research and Development Canada (DRDC), the Director General Military Personnel Research and Analysis (DGMPRA) and the Centre for Operational Research and Analysis (CORA) apply scientific methods to provide knowledge and advice to military managers and decision makers in the Canadian Department of National Defence (DND). DGMPRA’s Workforce Modeling and Analysis Team and various teams in CORA develop software models of DND HR systems and conduct simulations and analyses in order to provide HR decision makers with greater knowledge of the outcomes of possible courses of action. Examples of HR-related information produced by this work include forecasts of: training demand, training output and occupation health (Corbett 2013; Latchman and Hunter 2002; Straver, Okazawa, and Wind 2009; Seguin 2011), manning levels at all ranks in various occupations (Zegers and Isbrandt 2010), and readiness for deployed operations (Scales, Okazawa, and Ormrod 2011; Moorhead, Wind and Halbrohr 2008).

Many of DRDC’s HR models are implemented in the discrete event simulation (DES) environment, Arena, while a smaller number are implemented in other environments such as PowerSim or in programming languages like C++. While each of these environments and languages is powerful in its own right, none of them was designed with a military HR context in mind. Models of military HR systems have unique characteristics that most simulation software is not ideally suited to addressing. As a consequence, the process of developing and running DND HR models is not as efficient or as ambitious as it could be. This paper presents a novel discrete event simulation environment being developed by DGMPRA called Right Person, Right Qualification, Right Place, Right Time Human Resources (R4 HR) which is tailored to the specific needs of military HR decision making. The objective of developing this new software is to provide more accurate and more comprehensive HR-related information to military decision makers in less time.

2 CHARACTERISTICS OF MILITARY HR MODELS

In our experience, military HR models present a challenge for DES environments, such as Arena, because they tend to be complex rule-based systems, they make use of very large datasets, and the ability to interconnect models is very desirable. This section elaborates on these characteristics and the challenges they pose. Note that we frequently draw comparisons with Arena throughout the paper because it is a widely used DES environment and because of the number of DRDC's military HR models that are currently implemented in it.

2.1 Rule-Based Models

Military HR models often contain logic in the form of complex sets of rules representing the policies and procedures that affect the careers of military members. For example, in a personnel force structure model, the following is a set of rules that could be used to carry out promotions:

1. Find all unfilled positions in the next rank and sort in order of priority
2. Find all promotable individuals in the preceding rank, i.e., individuals who:
    a. have spent a certain minimum amount of time in the preceding rank
    b. have completed necessary training in the preceding rank
3. For each available position:
    a. Find all promotable individuals that meet the position's minimum qualifications
    b. Randomly select an individual from this group and promote to the position

The accuracy of military HR models is generally a function of the correctness of these rules, rather than the correctness of properties like processing times, transit times, and queue sizes as is the case in many other applications of DES. It is also the nature of military organizations that policies and procedures are constantly changing. Each new simulation scenario typically involves a request to analyze the effect of changing one or more of these rules.

Using standard DES components like assign blocks, decide blocks and delay blocks, it can become very difficult to implement such a rule-based system. The resulting logic tends to involve deeply nested loops that search and sort large amounts of data. This quickly becomes very challenging to debug, validate and modify. In such cases, for example Okazawa, Ormrod and Young (2009), it is often more appropriate to implement this type of model logic using an embedded programming language like Arena's Visual Basic for Applications (VBA). However, this presents its own challenges. Writing portions of model logic in code requires knowledge of the simulation environment's programming interface, moves model logic out of the graphical interface into a separate code editor, and makes it less easy to understand and change the model as simulation requirements change.

2.2 Large Datasets

In Arena, and other similar DES environments, simulation data from spreadsheets and/or databases is imported to in-memory data structures for processing as the simulation runs. On most current personal computer (PC) hardware, this amounts to an upper limit of a few gigabytes of data; though, the limit is less in reality because memory is a shared resource. This places a relatively low ceiling on the scale of simulation that can be carried out from a military HR perspective. A large-scale DND HR simulation could make use of data from a variety of large databases including: regular force, reserve force and civilian personnel records, individual employment and training histories, establishment and theatre unit structures, organizational structure, and planned intake and manning levels. By a rough estimate, importing these datasets would require on the order of tens of gigabytes of memory space, exceeding the available memory on most modern PCs. In general, however, there is no limit on the amount or type of data that a simulation may need. As DND's data warehousing technology improves, the amount of data available for simulation will only increase. Therefore, the volume of data required for military HR simulations will most likely continue to exceed the limits imposed by in-memory data structures even as PC hardware improves over time.

2.3 Interconnecting Models

Maintaining a capability to develop and operate DND HR models such as a personnel force structure model or a deployment model is a substantial task typically involving a team of two or three scientists having specialized knowledge in that area. The ground-up development and operation of a unified military HR model that handles these and other areas at once is an enormous task beyond the scope of any one team because of the diverse expertise required and the complexity that such a model would entail.

However, the need for such a unified model exists because the various components of military HR are interconnected. For example, a deployment model cannot accurately determine the availability of personnel over time without also modeling personnel force structure and training; a training model cannot accurately determine student intake without also modeling personnel force structure; and it cannot accurately determine if it is meeting demand without also modeling deployments. Without a unified model, the effects of these interactions must either be significantly simplified or ignored.

The most feasible approach to achieving a unified military HR model is to allow parallel execution of multiple sub-models that have been developed by different teams of scientists. In this context, parallel execution means running two or more sub-models in the same simulation instance using the same event queue and having the ability to exchange data. However, to the author's knowledge, Arena and similar DES environments are not designed to facilitate this. For example, in Arena, block names, variable names, signal values and the simulation clock all have global scope. If two or more separately-developed models are imported into the same simulation instance, it is unlikely that no naming collisions will occur and unlikely that the simulation clock has been used in a consistent way (e.g., assuming the same zero-time). It is very hard to detect and debug the resulting errors, mostly likely producing a broken model.

3 THE R4 HR SIMULATION ENVIRONMENT

The R4 HR simulation environment was developed specifically to address these unique characteristics of military HR models. This section introduces the main concepts and features of the R4 HR simulation environment that distinguish it from other DES software and allow it to meet the needs of military HR modeling and simulation.

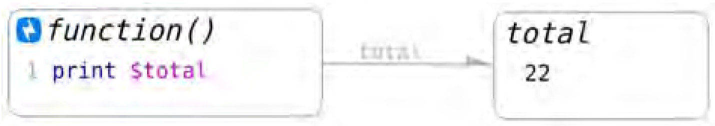
3.1 Python

As discussed above, many aspects of the rule-based model logic that is common in military HR models are better suited to a programming language implementation rather than a graphical process implementation using DES building blocks. However, verbose and unintuitive syntax are significant barriers to developing and maintaining models that implement some logic in code. For example, to access a simulation variable named "total" in an Arena VBA block, the following code is used:

```
ThisDocument.Model.SIMAN.VariableArrayValue(
   ThisDocument.Model.SIMAN.SymbolNumber(“total”))
```

The R4 HR simulation environment attempts to provide the advantages of a programming language but employs a clear, simple syntax that non-programmers will be comfortable with and seamlessly integrates the code with the graphical interface. In the R4 HR environment, the following code would be used to access a variable named "total": `$total`.

The R4 HR environment's programming language is Python. The model developer can create Python functions that integrate with the rest of the model and are visible and editable directly in the modeling interface. Figure 1 illustrates how the "total" example would appear in the R4 HR environment. Note, the `$` that precedes the variable name is not standard Python syntax and indicates that "total" is an alias for a model object and not a regular python variable (the use of aliases will be described in more detail below).



```
print $total
```

Figure 1: A Python function accesses and prints a simulation variable named "total".

Python was chosen because it is a powerful, general purpose programming language supported by an impressive variety of libraries that extend its capabilities in many scientific fields. It is a language that emphasizes simple, readable syntax that is friendly to non-programmers. Finally, it is a widely-used language that has proven itself in ambitious scientific applications. For example, Python was used to streamline NASA's Space Shuttle mission design (python.org 2013).

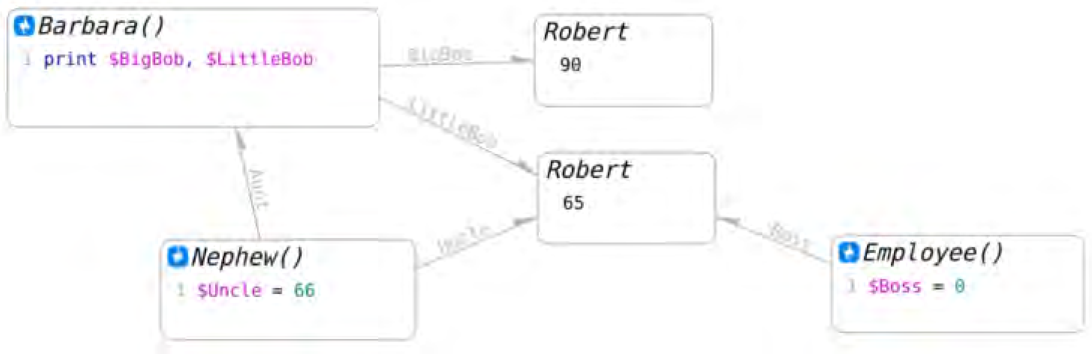
3.2 Modular Model Logic

The R4 HR environment implements a modular architecture that allows any model logic, whether a single function or an entire model, to be duplicated and reconnected elsewhere in the same workspace or imported into another model. The architecture also allows separately developed models to be integrated and executed in parallel. This is achieved in large part by a naming system that avoids naming collisions.

Naming collisions arise because software normally organizes objects into scopes. Implicitly, any object can interact with any other object in the same scope, so all names in the scope must be unique. We cannot transplant objects and code from one scope to another if they use names that have already been claimed in the new scope. The objects must be renamed and any code that accesses them must be updated, which is a difficult and error-prone process. In Arena, for example, all model objects exist in the same scope, so naming collisions are very hard to avoid when moving or duplicating model logic.

To avoid these problems, R4 HR uses a naming system similar to the way people use names. Many people share the same name, and yet society functions despite the ambiguity. This works because peoples' interactions are not managed with scopes; they are managed with one-to-one connections. So it does not matter that there may be many people named, for example, Robert. What matters is who is connected to each Robert and what they call him. A Robert may indeed be called Robert by some, but others will call him Bob or boss or uncle depending on their relationship to him. In other words, Robert is given an alias by each person that interacts with him. If another person, say Barbara, knows more than one Robert, she will give each one a different alias, like Big Bob and Little Bob. If a group of people move from one place to another, they retain the connections and aliases that already existed within the group so they can function as before. To interact with their new setting, new one-to-one connections are formed.

R4 HR applies this naming system to all model objects. Therefore, if one model object needs to interact with another, it forms a connection to that object and then refers to it using the alias. In the Python code, aliases are prefixed with a `$` (as seen in Figure 1, above). This system ensures that any selection of model logic can be safely duplicated, moved and reconnected elsewhere without affecting its functionality. We will refer to this naming system as the graph-alias naming system because the objects and their connections form a directed graph where each connection has an alias. Figure 2 shows a number of objects and connections and illustrates how the graph-alias naming system handles multiple objects with the same name. In the R4 HR interface, model objects are surrounded by a rectangular border, connections appear as arrows and aliases are shown next to the connection arrow.



```
print $BigBob, $LittleBob
```

```
$Uncle = 66
```

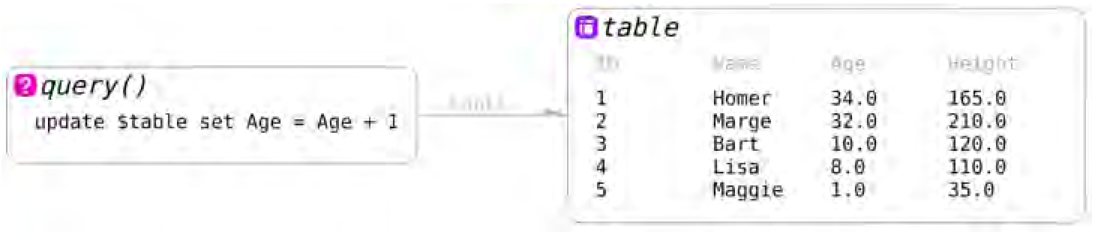
```
$Boss = 0
```

Figure 2: A model in which there are two variables named Robert. A function named Barbara connects to both Roberts with different aliases. The functions Nephew and Employee form other connections with their own aliases.

3.3 Relational Database

The vast majority of military HR data resides in databases. For Arena-based models, we import this data into in-memory data arrays. As discussed above, this places a low ceiling on the size of data that can be processed. It also presents another disadvantage. A relational database can be processed using structure query language (SQL) which is ideally suited to the large-scale searching, sorting and updating operations that are common in military HR models. But when the data is imported into an in-memory array, the ability to use SQL is lost, and these data operations must then be re-implemented by the model developer.

In order to conveniently access and process very large datasets, the R4 HR environment provides a built-in relational database. Simulation data can be imported into the built-in database without changing form and can be directly processed using SQL as the simulation runs. This approach was previously implemented in Arena using extensive VBA code (Okazawa, Ormrod, and Young 2009). In R4 HR, the approach has been seamlessly integrated into the environment, so the model developer can use database tables and SQL without knowing the details of opening database connections and executing transactions. Figure 3 shows an example of an SQL query acting on a database table in the R4 HR environment.



```
update $table set Age = Age + 1
```

ID	Name	Age	Height
1	Homer	34.0	165.0
2	Marge	32.0	210.0
3	Bart	10.0	120.0
4	Lisa	8.0	110.0
5	Maggie	1.0	35.0

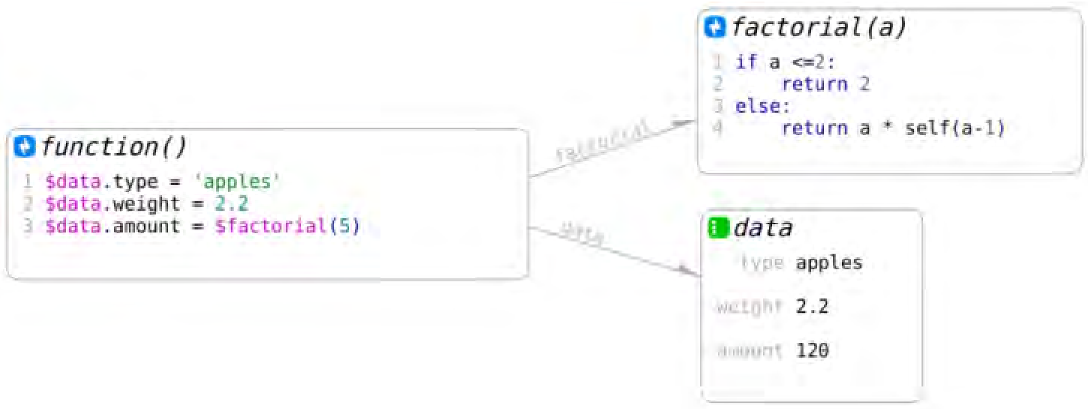
Figure 3: An SQL query is applied to a relational database table in the R4 HR environment.

3.4 Model Objects and Connections

As with most DES simulation environments, models in R4 HR are created by adding model objects to the workspace and connecting them together. The environment provides a selection of basic modeling objects that will be useful in many scenarios. These include function objects for executing Python code, data objects for storing and processing data in various formats, plotting objects that display information in graphical form, and other objects that are useful for managing larger, more complex models.

Each of the basic modeling objects is a special Python object created for the R4 HR software, but any Python object can actually be used as a model object. This provides a great deal of flexibility because it allows application-specific Python objects, such as a numpy array (an optimized data structure for matrix operations) or a custom file reader, to be easily incorporated into a model.

When one object, A, is connected to another object, B, it means that A can interact with B. For example, Figure 4 shows a function object that is connected to another function object and a data object. Through the connections, the function is able to use the other objects. In this case, it assigns data to the data object and calls the factorial function object. Note that function objects can have parameters and can be recursive as the factorial function illustrates. Connections are uni-directional, so object B cannot interact with object A unless another connection is created from B to A. Connections are the only way that one model object can interact with another which respects the graph-alias naming system and thus preserves modularity.



```
function()
$data.type = 'apples'
$data.weight = 2.2
$data.amount = $factorial(5)
```

```
factorial(a)
if a <=2:
    return 2
else:
    return a * self(a-1)
```

```
data
  type  apples
weight  2.2
amount  120
```

Figure 4: A function object is connected to and uses two other objects: another function object called factorial and a data object.

3.5 Events and Signals

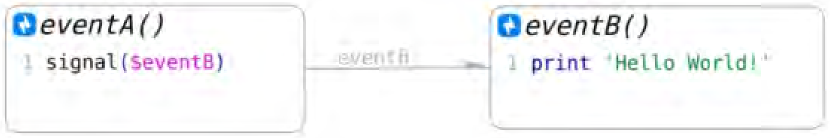
A discrete event simulation consists of items, conventionally called entities, that flow through a process. Each step in the process is an event which may apply changes to an entity's attributes, changes to simulation variables, delays, decisions, and the use of resources. In R4 HR, the Python function objects introduced above are used to define events. In most cases, events will consist of a few simple lines of code, but the full flexibility of Python and its libraries is available in any function object when needed.

We normally think of entities as physical objects such as a parcel being shipped or a hospital patient, and events are triggered when the entities arrive at the various steps in the process. But a more general interpretation is that an entity is something that carries information between process steps. The information could describe a physical object, and information on the same object could be passed through the entire process (which is analogous to traditional DES entities); but we can also allow entities to carry arbitrary information. This is the approach used in R4 HR. There is no assumption of what an entity is or that it persists throughout the process. Entities simply communicate whatever information needs to be passed within the process, and because we think of entities as information rather than objects, we use the term signal in place of entity. (This use of the term signal is not to be confused with Arena signals which are event triggers that are broadcast simulation-wide and do not carry information.)

The advantage of this approach is that the information that moves from one process step to another is unrestricted. The signal might only be a trigger for another event, in which case no information needs to be sent; the signal could carry a simple message such as a value or string; or the signal could carry complex information such as a list of items or a custom class instance. If the developer wishes to implement traditional DES entities, the signal can carry an entity message that stores information in attributes.

Because there is no assumption that signals persist throughout a process, all signals must be explicitly created and propagated as needed. This is done by calling a signal function within an event's Python code. The signal function must be called with at least one parameter specifying the target event that will receive the signal, and a connection must exist to the target event; a second optional parameter is used to attach a message to the signal; a third optional parameter is used to specify the time at which the target event will be triggered; and a fourth optional parameter assigns a priority to the signal.

When the signal function is called, the target event, message, time and priority are stored in the R4 HR software's event queue (often called the future event list). As with other DES software, the event queue manages all pending events in the simulation and decides which event will execute next. When one event has completed, the event queue will select the event with the nearest time to execute next. If there is more than one such event, it will select the one with the highest priority. If there is still more than one such event, it will select the one that was placed on the event queue first. Figure 5 shows a simple example of one event signaling another without passing any information.



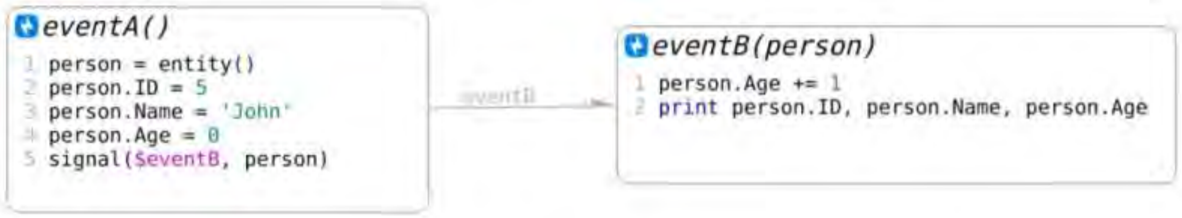
```
eventA()
signal($eventB)
```

```
eventB()
print 'Hello World!'
```

Figure 5: EventA sends a trigger signal to eventB containing no information. In this example, eventB responds by printing “Hello World!”, but the event could carry out any action required by the model.

Because events and functions are one and the same, any function object can be called like a function or signaled like an event. The execution of a called function is nested within the execution of the code that called it, so it runs immediately and can return a value (as in the factorial function example in Figure 3, above). Conversely, signaling a function does not execute it, it places it in the event queue for execution later in the simulation. The code that created the signal finishes before the signaled function runs, therefore a signaled function cannot return a value to the code that sent the signal.

The parameters of a function define the inputs that the function expects when it is called. When a function with parameters is signaled, the signal must include a message that matches the function's parameters. This is how a signal's message gets passed to the target event. Figure 6 shows an example of sending a signal with a message. Note that the event that receives the message has a parameter called “person” in its title line, so any signals that trigger this event must include a person as a message.



```
eventA()
person = entity()
person.ID = 5
person.Name = 'John'
person.Age = 0
signal($eventB, person)
```

```
eventB(person)
person.Age += 1
print person.ID, person.Name, person.Age
```

Figure 6: EventA creates an entity called “person” and assigns attributes for its ID, name and age. It then sends a signal to eventB, including the person as a message. This is analogous to an entity creation step in traditional DES. EventB receives the message as a function parameter, increments the person's age, and then prints its attributes. This is analogous to an entity attribute assignment step in traditional DES.

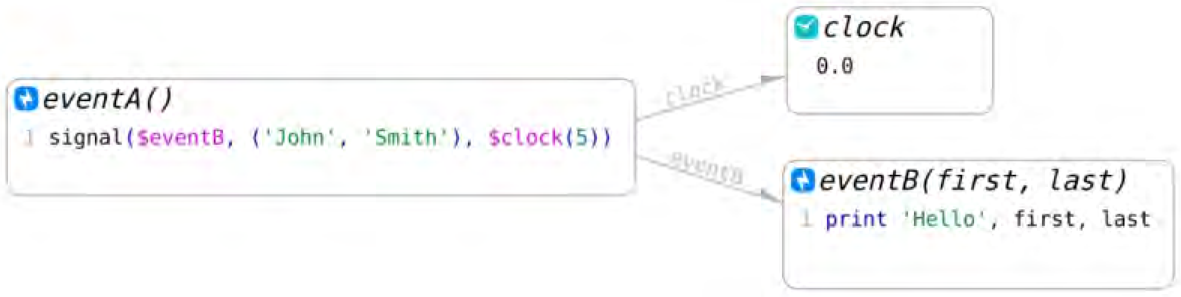
3.6 Clocks

The examples of signaling described in the previous section did not involve time. If no time is provided in the signal call, the current simulation time is used by default. In order to schedule events to occur in the future, we need to introduce the concept of clocks.

In the R4 HR environment, the model developer does not have access to a global simulation clock. Instead, R4 HR uses multiple local clocks. This is consistent with the way we use time in reality as there is no absolute clock that started ticking when the universe began; we use wrist watches, wall clocks, CPU clocks, and other devises that tick along with time but which do not necessarily display the same time or tick at the same rate. In the same way, R4 HR's local clocks can be individually set to different times and different speeds.

This allows the model developer to use clocks in any way (s)he sees fit without worrying about future integration issues. One developer may decide that a clock tick corresponds to an hour and that the initial time at simulation start is zero, while another developer may decide that a clock tick corresponds to a day with an initial time of 365. If these two models were later integrated for parallel execution, each model brings its own clock, so there will be no conflicts in how time is used. The hour-clock will start at zero and will tick 24 times for each tick of the day-clock which will start at 365. Note that clock times are real numbers, not integers, so events can be scheduled at any real number value on a given clock.

To signal an event at a future time, the signal defines a time or a delay according a local clock. The event that is sending the signal must have a connection to the clock it is using. Figure 7 shows an example of an event setting up a signal to trigger another event at time 5 on a local clock. Note that the target event accepts two parameters, so the signal sends a message with two elements: *('John', 'Smith')*.



```
signal($eventB, ('John', 'Smith'), $clock(5))
```

```
print 'Hello', first, last
```

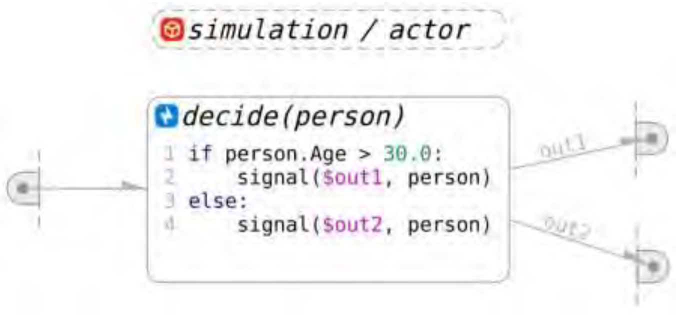
Figure 7: EventA creates a signal with a two-parameter message *('John', 'Smith')* that will trigger eventB at time 5 according to the clock. This is analogous to a delay step in traditional DES.

3.7 Actors and Sockets

While it is possible to implement an entire model in a single workspace, such a model would become difficult to understand because both high-level and low-level logic would be mixed together. Usually, it is preferable to hide lower-level details from a higher-level view. This makes the structure of the model clearer and helps others understand how it works.

To achieve this, the R4 HR environment allows model logic to be enclosed within an object called an actor. Actors can also contain other actors, so the model can be organized hierarchically. When a model object inside an actor needs to interact with an object outside the actor (and vice versa), connections can cross actor boundaries using another object called a socket. Actors and sockets are purely cosmetic; they have no impact on how the model works, only on how it is organized and displayed in the interface. This means the model developer can enclose arbitrary portions of logic inside actors without having to make any changes to the model implementation.

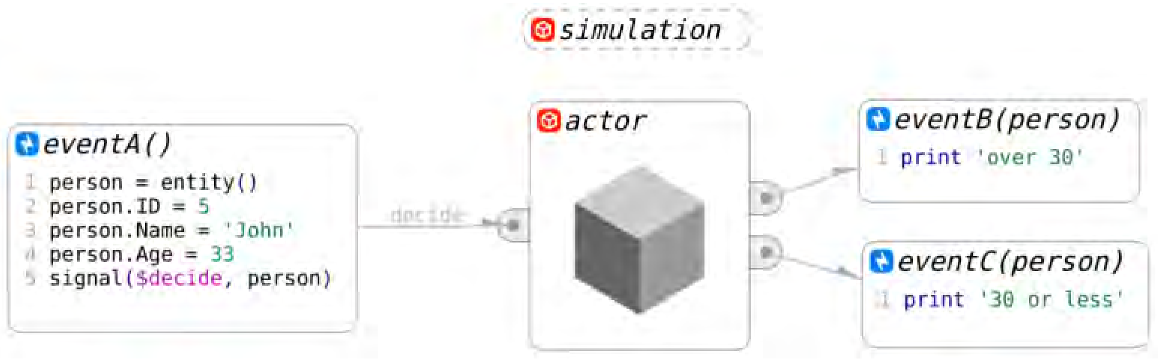
Figure 8 shows a simple example of a decision event that is enclosed within an actor. In R4 HR, there is a top-level actor called "simulation" within which all model logic is created. The actor currently being viewed is displayed within a dashed border and shows its path to the top-level simulation actor. The actor shown in the figure contains an event that receives information about a person. The code resends the signal with the same information to one of two outputs depending on the person's age. The objects to the left and right of the event are sockets. The dashed edge on each socket indicates the side of the actor that the socket is attached to. The dots within the sockets are called nodes, and their role is to break up connections into segments.



```
if person.Age > 30.0:
    signal($out1, person)
else:
    signal($out2, person)
```

Figure 8: The internal logic of an actor showing an event that signals one of two output sockets depending on the value of an incoming entity attribute. This is analogous to a decision step in traditional DES.

Figure 9 shows the same actor viewed from the outside. The sockets now appear on the outside of the actor and show how the actor's internal logic is connected to external logic.



```
person = entity()
person.ID = 5
person.Name = 'John'
person.Age = 33
signal($decide, person)
```

```
print 'over 30'
```

```
print '30 or less'
```

Figure 9: The external logic that interacts with the actor by connecting to its sockets.

3.8 Modeling Approach

In practice, we anticipate that most models, at a high level, will consist of a number of interconnected actors. Each actor would be responsible for a different aspect of the model and would pass information to other actors using sockets as needed. For example, Figure 10 shows how a personnel force structure model could be structured in the R4 HR environment. Personnel force structure models simulate the progression of military personnel from recruitment, up through the ranks until they retire.

In this example, the Recruit actor sends signals to the private rank (PTE) actor when new recruits enter the system. Each rank actor maintains a list of current personnel and ensures that they complete the requirements of the rank such as training and enforcing a minimum amount of time in the rank. When these criteria are met, an individual becomes promotable. When a private becomes promotable, the private actor immediately signals the corporal (CPL) actor to promote the individual. Beyond the rank of corporal, promotions only occur if there is a position available in the rank above, so three connections are needed to manage the back-and-forth communication. The first connection allows the lower rank to signal the upper rank when an individual has become promotable. If a position is available at that time, the upper rank will signal back through the second connection. Finally, the lower rank responds by sending a signal with information about the individual to be promoted through the third connection, and the promotion is complete.

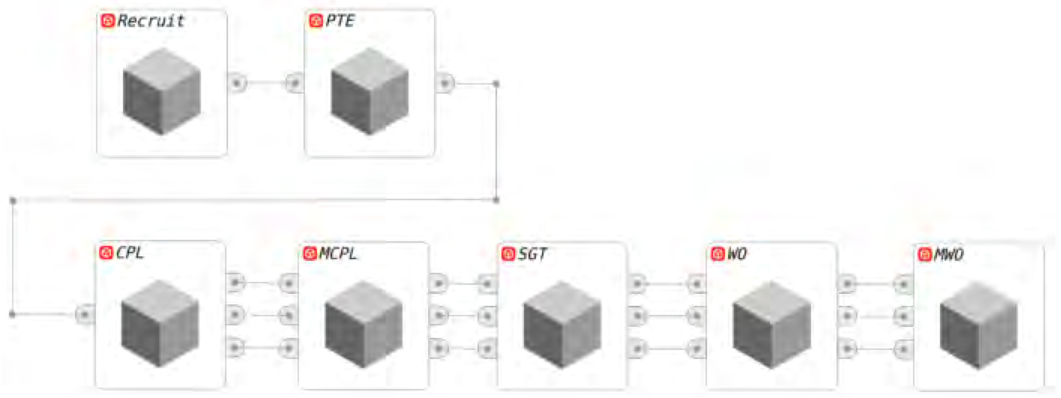


Figure 10: A sample layout for a personnel force structure model composed of actors, where each actor is responsible for a rank and communicates with other actors using sockets.

Alternately, if no position was available for the promotion, the upper rank would not have responded and the individual would wait in the lower rank. Eventually, when a position does open, the upper rank would signal the lower rank through the second connection. Then the lower rank would select an individual from all those who are waiting and send the individual's information through the third connection to complete the promotion.

This example highlights how actors and sockets contribute to modular model development. Actors neatly hide the detailed implementation of model components and clearly indicate the connections that must be made to "plug" the component into the model. This allows actors to be easily disconnected, moved, duplicated, swapped and reconnected.

By extension, actors and sockets also provide a convenient means to interconnect several complete models. It is therefore feasible to create a unified military HR model in the R4 HR environment. Each model would be enclosed in its own actor and sockets would be used to allow information exchange between them. Figure 11 illustrates a simple structure that could be used to interconnect a personnel force structure model, a training model and a deployment model. In this minimal example, the force structure model would send information to the training model about students that need training, and the training model would send information back whenever students complete training. The deployment model would send requests for deployable personnel to the force structure model, the force structure model would respond by identifying whatever qualified and available personnel can be assigned to the deployment, and finally, the deployment model would inform the force structure model when the deployment is complete.

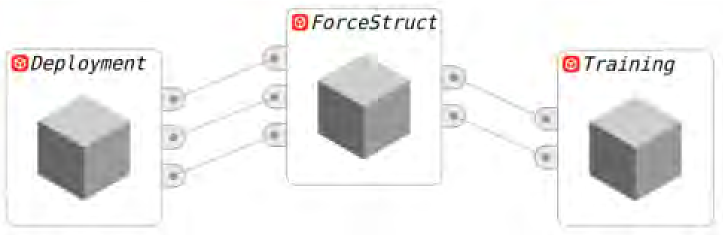


Figure 11: A simple structure for a unified military HR model in which a personnel force structure model, training model and deployment model are enclosed within actors and communicate using sockets.

3.9 Additional Model Objects

In addition to the features described above, the R4 HR environment provides a number of other model objects that are useful in developing and running models, but which do not need to be described in detail for the purposes of this paper. These include:

- a sheet object that stores data in a grid and provides some common features of spreadsheets,
- a hub object that allows one object to connect to many other related objects by making a single connection to the hub, and
- a data visualization object that displays two-dimensional (2D) and three-dimensional (3D) graphs and plots that animate as the simulation runs.

3.10 3D Graphical User Interface

Arena and other DES environments generally employ multiple workspaces for different aspects of a simulation project including a graphical modeling space, a code editor, a 3D animation space, and a results analysis space. The R4 HR environment integrates all these workspaces into a single 3D space. All model objects including functions, data and actors occupy a position in this 3D space. Actors can be assigned a user-specified 2D or 3D icon. The positions of all model objects can be controlled within the simulation, so it is possible to create an animated 3D representation of the events occurring in the simulation. For example, if an actor was used to model the activities of personnel deployed to a naval vessel, a ship icon could be assigned to the actor, and the actor could be moved between a shore position and a sea position to indicate its operational status.

4 DEVELOPMENT STATUS

DGMPRA has developed a functioning prototype of the R4 HR environment. All of the figures in this paper are screen captures from the prototype, and all of the features introduced above are present in the prototype. It is possible to carry out simulations in the software as it stands. However, the prototype was not developed to be used for real modeling and simulation work; it was developed to validate and refine R4 HR concepts. As such, it contains bugs, incomplete features, and the performance is not optimized. DGMPRA is in the process of launching a contract to have the R4 HR software professionally developed. The timeframe for completion of the contract is currently 2016.

5 CONCLUSION

The R4 HR simulation environment is intended to be a platform for future military HR modeling and simulation in DND. It provides many convenient features that were specifically designed with this task in mind including an embedded programming language that is both powerful and friendly to non-programmers; an architecture that allows model logic to be easily moved, duplicated and reconnected; an integrated relational database for processing large amounts of data; a flexible signaling system for communication between simulation events; and a 3D graphical user interface that integrates modeling, simulation and analysis activities into a single workspace.

Through these features R4 HR will enable military HR models to be developed faster and modified more easily as simulation requirements change. As a result, simulation support to decision making will be more responsive to operational and strategic needs. However, a key advantage of R4 HR is that it makes feasible the prospect of creating unified models that interconnect the various sub-systems of military HR. By capturing the interactions between these sub-systems, the R4 HR environment will provide foresight to DND decision making that is more accurate and comprehensive than our current modeling and simulation approaches allow.

ACKNOWLEDGMENTS

The R4 HR project could not have achieved its current momentum and progress without the hard work of the following individuals: Stan Isbrandt, Patricia Moorhead, Sonia Latchman, Zakia Bouayed, Abe Jesion, Albert Chan, Benoit Paquin, and Alex Kwiatkowski.

REFERENCES

Corbett, N. *Modelling the Production and Absorption of Pilots: The Development of the Production, Absorption and Retention Simulation (PARSim)*. DRDC Technical Report 2013-023, Center for Operational Research and Analysis, Ottawa, Canada.

Latchman, S., and C. Hunter. 2002. *Preliminary Results from the Pilot Production/Absorption/Retention Simulation (PARSim) Model*. 1 CAD/CANR Headquarters Research Note 0202, Centre for Operational Research and Analysis, Ottawa, Canada.

Moorhead, P., A. Wind, and M. Halbrohr. 2008. "A Discrete Event Simulation Model for Examining Future Sustainability of Canadian Forces Operations." In *Proceedings of the 2008 Winter Simulation Conference*, edited by S.J. Mason, R. Hill, L. Moench, and O. Rose, 1164 - 1172. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Okazawa, S., M. Ormrod, and C. Young. 2009. *Managed Readiness Simulation (MARS) V2: Assessment of a Simulation Runtime Database Approach*. DRDC Technical Memorandum 2009-042, Center for Operational Research and Analysis, Ottawa, Canada.

python.org. 2013. Python Streamlines Space Shuttle Mission Design. Accessed March 15. http://www.python.org/about/success/usa/.

Scales, C., S. Okazawa, and M. Ormrod. 2011. "The Managed Readiness Simulator: A Force Readiness Model." In *Proceedings of the 2011 Winter Simulation Conference*, edited by S. Jain, R.R. Creasey, J. Himmelspach, K.P. White, and M. Fu, 2519 – 2529. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Seguin, R. 2011. "1 Canadian Forces Flying Training School (1 CFFTS) Resource Allocation Simulation Tool." In *Proceedings of the 2011 Winter Simulation Conference*, eWSC Editing Team dited by S. Jain, R.R. Creasey, J. Himmelspach, K.P. White, and M. Fu, 2519 – 2529. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Straver, M., S. Okazawa S., and A. Wind. 2009. *Training Pipeline Modelling Using the Production Management Tool*. DRDC Technical Memorandum 2009-019, Director General Military Personnel Research and Analysis, Ottawa, Canada.

Zegers, A., and S. Isbrandt. 2010. "The Arena Career Modelling Environment – a New Workforce Modelling Tool for the Canadian Forces." In *Proceedings of the 2010 Summer Computer Simulation Conference*, Edited by G. Wainer, 94 – 101. Curran Associates, Inc.

AUTHOR BIOGRAPHY

STEPHEN OKAZAWA is a defence scientist in Defence Research and Development Canada. His research focusses on discrete event modeling methods in support of military decision making. He received his MEng in Electro-Mechanical Engineering from the University of British Columbia, Canada. His email address is stephen.okazawa@forces.gc.ca.