12th ICCRTS

"Adapting C2 to the 21st Century"

Paper: **I-032**

Title: **Executable Architecture of Net Enabled Operations: State Machine of Federated Nodes**

Topics: Modeling and Simulation
C2 Metrics and Assessment
Network-Centric Experimentation and Applications

Authors: Mark Ball
Joint Staff Operational Research Team
Centre for Operational Research and Analysis
Ottawa, Ontario, Canada

Ronald Funk
Joint Staff Operational Research Team
Centre for Operational Research and Analysis
Ottawa, Ontario, Canada

Richard Sorensen
Principal Systems Engineer
Vitech Corporation
Vienna, Virginia, USA

Point of Contact:

Mark Ball
Centre for Operational Research and Analysis
National Defence Headquarters
101 Colonel By Drive
K1A 0K2
Office: (613) 992-4539
Fax: (613) 992-3342
Email: ball.mg@forces.gc.ca

Executable Architecture of Net Enabled Operations:
State Machine of Federated Nodes

By

Mark Ball, Ronald Funk and Richard Sorensen

The Defence Research and Development Canada (DRDC) Centre for Operational Research and Analysis (CORA) is developing capability-engineering analysis tools to support the building, demonstration, and analysis of executable architectures. Our paper to 11th ICCTS [1] described how to model workflows within an Operations Centre (OPCEN) employing a Net-Centric architecture. It used a State Machine (SM) model to simulate how multiple jobs can proceed in parallel when operators use Task, Post, Process, Use (TPPU) cycle to organize their work.

This paper extends the OPCEN SM model to track the interaction of work between OPCENs. The State Machine of Federated Nodes (SMOFN) model is organized around networked nodes that produce and consume products held in a virtual Repository. The data-driven simulation uses files to build customized job workflows and configure any combination of nodes without affecting the business logic. SMOFN also accounts for the following overhead activities:

(1) Tracking consumer perception of product utility as it accrues and decays;
(2) Consolidation of products into higher-level aggregated products; and
(3) Triggering new jobs where needed whenever relevant products become available.

Customization of SMOFN is underway to account for the data and product flows between OPCENs in new Canadian Forces Command structure.

Introduction

Background

The Defence Research and Development Canada (DRDC) Centre for Operational Research and Analysis (CORA) is developing capability-engineering analysis tools to support the building, demonstration, and analysis of executable architectures. Our paper to 11th ICCTS [1] described how to model workflows within an Operations Centre (OPCEN) employing an executable Net-Centric architecture. It used a State Machine (SM) model to simulate how multiple jobs proceed in parallel when operators use a Task, Post, Process, Use (TPPU) cycle to organize their work.

The OPCEN SM has been extended to account for Net-Enabled interactions between several OPCENs. In essence, the OPCEN SM accounted for the work done by a single OPCEN to produce several products based on data analysis. In the State Machine of Federated Nodes (SMOFN), not only are such production jobs tracked within several OPCENs, but the products they create are uploaded to a common, networked, Repository so that all products can be accessed and used by any OPCEN. The SMOFN is an attempt to capture the essential logic that governs the way work is conducted anywhere, but with particular emphasis on ensuring that it can be fully applied to networked OPCENs.

Primer on TPED vs. TPPU

Task, Process, Exploit, Disseminate (TPED) logic implies that each job is a serial process: jobs are worked on from start to finish without being interrupted. Flowchart diagrams (ie. in CORE) can simulate TPED by putting the job processes in a loop and repeating for each job. TPPU, on the other hand, allows jobs to be interrupted by higher priority jobs. As illustrated in Figure 1, updates of jobs processed through TPPU are posted at regular intervals and the utility of the job increases as more posts are made. This accrued utility is saved even if the job is interrupted. TPPU leads to concurrent behaviour (several partially processed jobs at a time) that is too complex to model using classical flowchart diagrams. The SM overcomes this by stopping time and executing the logic to assign operators to jobs in order of priority. It then steps forward in time and repeats the process. Progress within each job is tracked by recording the job's state and updating it at each time step.

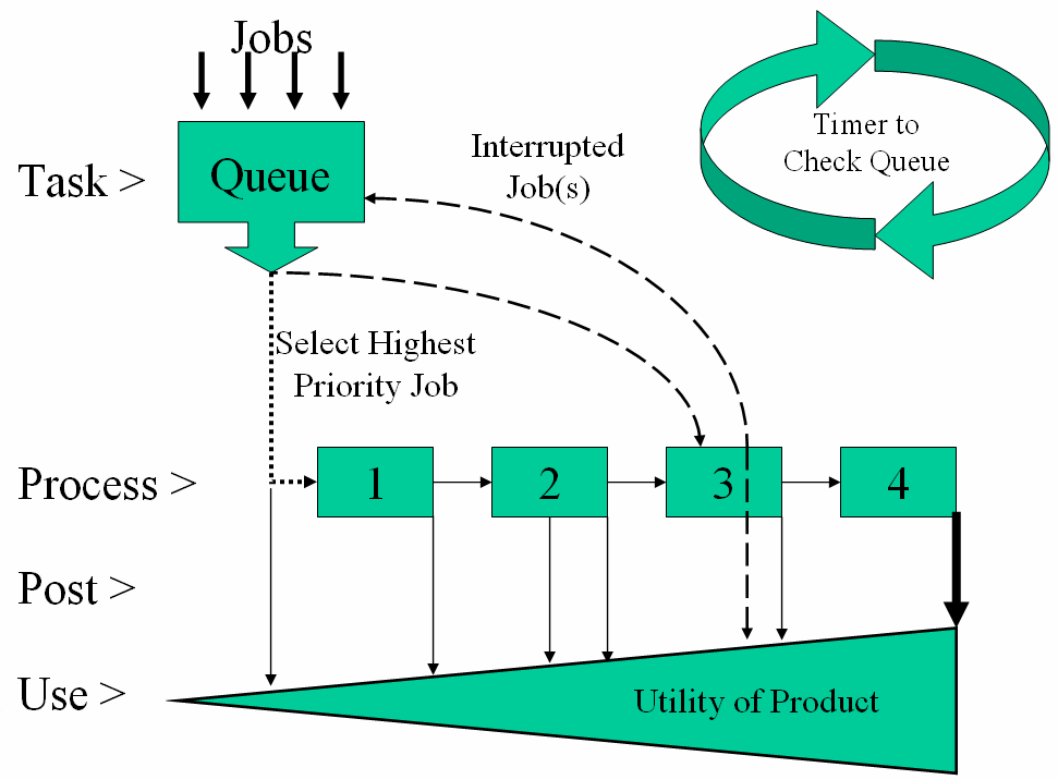

Figure 1: TPPU

Conceptual Basis for SMOFN

Figure 2 illustrates a conceptual Department of Defense Architecture Framework (DoDAF) OV-1 (high-level operational concept graphic). It is scale-free in the sense that it can represent interactions between nodes, or within a node, or even within a single operator's various responsibilities. The dashed lines show the net transfer of Products, Questions, Queries, Results, and Responses, but the solid lines serve as a reminder that all of these are actually transferred through the Repository, the interface to which is controlled by the Portal. This conceptual diagram was used as the basis for the logic controlling the interactions between the various nodes in the SMOFN. More specifically, it was used to define the interactions between each node and the Repository.

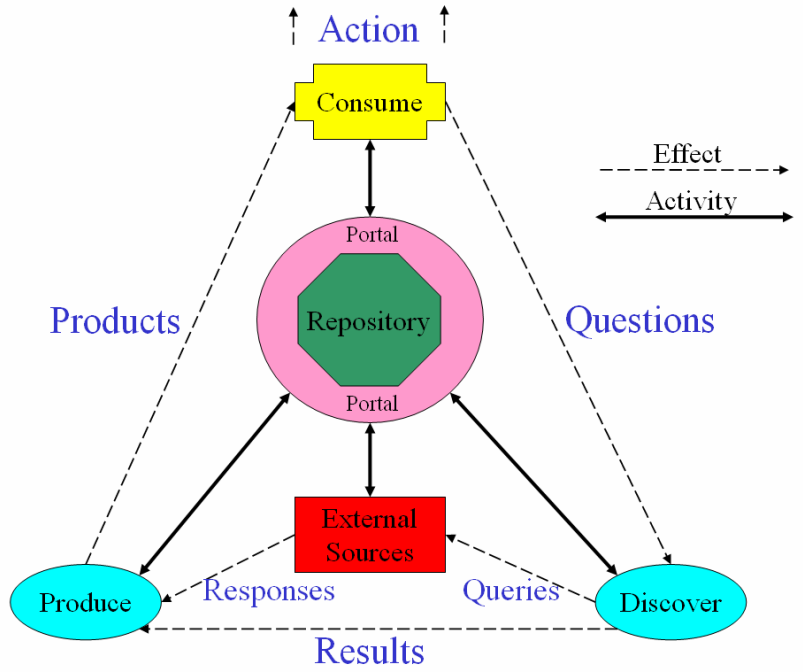

Figure 2: Scale-free OV-1

Producer-Repository-Consumer model logic

Figure 3 illustrates the ways in which Producer and Consumer nodes can communicate with each other and interact with the Repository. The most direct way for content to be transferred is for the Producer to push products directly to the Consumer. Note that in actual execution this transfer will likely take place over a network, and so the Repository still acts as a medium. Alternately, the Producer may post content to the Repository where it waits to be pulled by interested Consumers. In this case the Producer may notify the Consumer that the content is available, the Repository may have business rules to decide what content the Consumer should be notified of, or the Consumer may have their own way of searching for new information.

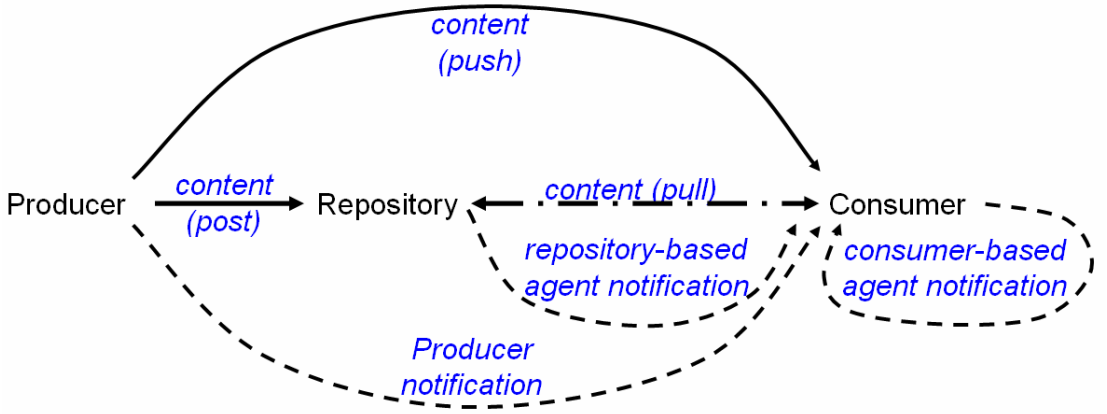

Figure 3: Producer-Repository-Consumer

Operational Decision Making Logic

Extensions of OPCEN SM to SMOFN

In terms of the Producer node, very little has changed in the SMOFN from the OPCEN SM described in [1]. However some work has been done to allow users to create more customized job threads. The OPCEN SM assumed every job involved 10 steps, although fewer steps could be handled by assigning zero time to some steps. SMOFN builds its job threads dynamically for any number of steps by assigning each step a unique name and linking them together until it reaches a final step called “COMPLETE”.

Producer logic

Coding the logic controlling Producer threads was the focus of the OPCEN SM. During each time step, the SMOFN checks for the arrival of new jobs to be worked on at each OPCEN. Each OPCEN then uses the same rules to assign operators to jobs. First the jobs are examined in order of priority and assigned to available operators with the skills necessary to perform the job. Some jobs that are in progress from the previous time step can be interrupted if a higher priority job requires the same operator, or a job step considered to be generic can be interrupted to free a skilled operator so they can move to a job requiring their skill, leaving the generic job for someone else. Although the utility of any job is determined by Consumers, Producers have their own estimate of what the utility will be and they will abandon jobs that are getting out of date by their utility decaying faster than the operators can increase it due to progressing on the job. Jobs will also be abandoned if they cannot be completed by their drop dead time, which is usually a set amount of time (different for each job type) after the job was added to the queue.

Consumer logic

In terms of SMOFN execution, the Consumer node’s job is very straightforward. Each time a product is received by the Consumer, there is a chance, based on the type of product received, that a question will be generated. Of course, a Consumer will not ask a question as soon as they receive a product, they require some time to review the product first. Because of this, each product type is associated with a certain delay time after which questions may be generated. If a question is generated, and once the delay time has passed, the question is sent to the Repository to be passed on to the Discover node.

Discover logic

The Discover node is responsible for answering questions generated by the Consumer. In real life terms, this tends to be done through a search for existing data. In modelling terms, it is handled through job threads similar to those used by the Producer. In fact, most of the logical scripting used by the Producer threads is shared with the Discover threads. There are a few important differences that should be mentioned though. First, the Discover logic has no accounting for Utility because when asked to find missing data, the Discoverers do not concern themselves with how useful they expect that data to be to the

Consumer. Similarly, there is no chance of a Discover thread returning “Nothing Significant to Report”. On the other hand, the data may not exist and External Sources can be tasked to collect it.

External Sources logic

The External Sources node receives Requests for Information (RFIs), or Queries, from the Discover node. Exactly how the requested information is found is beyond the scope of the SMOFN, but what is important to the model is how much information is found (in terms of file size) and how long it takes to find it. The found information will typically be in the form of raw data which is sent to the Producer, triggering a new data analysis job.

Repository logic

The repository is the medium between all of the other nodes. Its job is to transfer products from Producers to Consumers, questions from Consumers to Discoverers, RFI’s from Discoverers to External Sources, results from Discoverers to Producers, and responses from External Sources to Producers. The scripts to handle the logic of these data transfers are all similar, with nuances to account for differences between types of data or nodes, but more particularly for what type of information is relevant for the receiving node to execute its logic appropriately.

Implementation in COREsim

SMOFN top level

Figure 4 is the top-level diagram that controls SMOFN execution, colour-coded so each activity can be cross-referenced with the appropriate node in Figure 2. Unlike most diagrams in CORE, time does not increase as CORE steps through the process beginning at the left and moving to the right. In this case, time is actually stopped and the left-to-right process is the decision making process that takes place at any instant in time. The SMOFN then increments time and the process is repeated for the next time step. Another difference between Figure 4 and typical CORE diagrams is that the logic is not completely defined simply by the diagram itself. As mentioned earlier, the non-serial behaviour of TPPU logic is not scalable when modelled in classical flowchart diagrams, so scripting embedded within each activity in the diagram is responsible for tracking the state of each job.

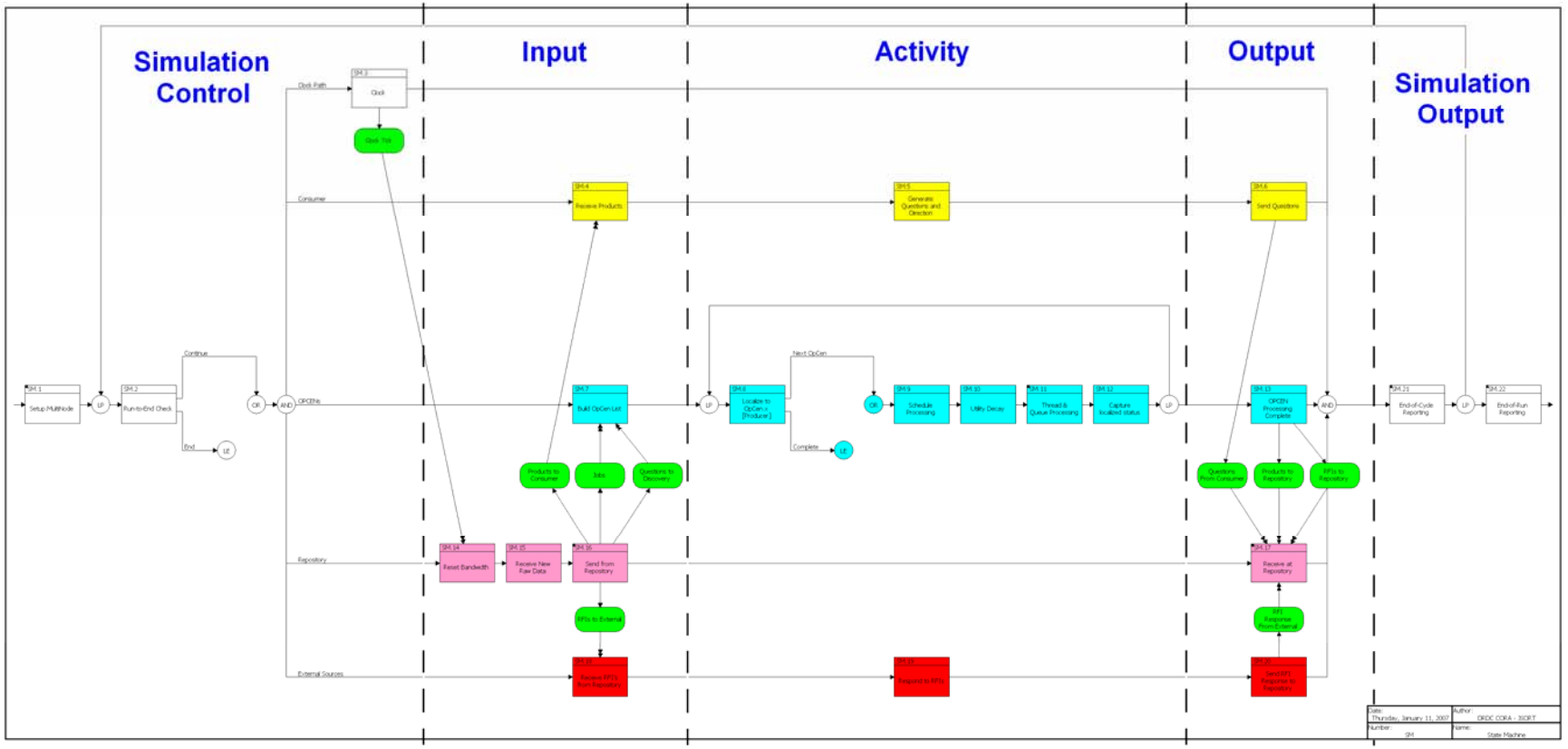

Figure 4: SMOFN Top-level

Logic within branches

The SMOFN logic in Figure 4 outside of the individual nodes is shown as white boxes in the simulation control and simulation output phases. When the simulation starts, the setup activity runs, reading data from input files and setting up global variables used to track data throughout the simulation. The SMOFN then enters a loop which repeats for each time step. First the Run-to-End Check executes to see if the end of the simulation has been reached (this will occur at a pre-set time or when there are no jobs left to do). If the end of the simulation is reached then the loop exits and the SMOFN skips ahead to the End-of-Run Reporting which outputs the data that was tracked through the simulation. If the end is not reached, the clock increments by one time step and each node's logic is executed for that time step, after which the current state of every task and every operator is saved to the global variables that will be the basis of the simulation output.

The simulation of the operational nodes is divided into three phases. Each node's activity is triggered by input received from the Repository, and that activity leads to an output that is sent back to the Repository.

Consumer activity in Figure 4 is represented by the yellow boxes. Each time step, the Consumer node begins by receiving products sent by the Repository. The Generate Questions activity then goes through all the products that have just been received and may generate questions based on those products. If any questions are generated, the delay before they are actually sent is determined and the time to send them is added to a schedule.

Producer and Discoverer logic elements are displayed by the blue boxes in Figure 4. The activity within both of these nodes is based on job threads so much of the scripting is the same. First a list of OPCENs is built and a loop is entered that is repeated for each OPCEN. The loop begins by choosing an OPCEN which has not been dealt with yet for this time step. The Schedule Processing activity then looks at the job arrival schedule and adds to the queue any job that arrives during this time step. Utility Decay then goes through all the jobs and checks to see how old their original data is. If the age of the data is such that the utility should decrease this time step, that is handled here. Thread & Queue Processing then goes through all jobs and assigns available operators to jobs in order of job priority. It is also here that the utility of jobs can increase as work progresses, or jobs can be abandoned if they cannot be completed on time. It is important to note that Produce and Discover jobs are allowed to draw from the same pool of operators so when the SMOFN sorts jobs in order of priority, it ignores whether the job is related to Production or Discovery. After these activities have executed, Capture Localized Status saves to memory any data that must be tracked into the next time step for the given OPCEN. After the loop has executed for each OPCEN, OPCEN Processing Complete signals to the Repository that it can now execute its logic for receiving products and RFIs.

The pink boxes represent the work done by the Repository during each time step. First of all, the global variable tracking the bandwidth between the Repository and each OPCEN must be reset as it is decremented whenever bandwidth is used to send or receive data. Bandwidth is tracked in terms of how much data can be transferred each time step. Next, the Repository checks for the arrival of new raw data. This check is based only on parameters read into the model during setup; raw data received from External Sources in response to RFIs is handled separately. Send From Repository contains the logic used to send products to the Consumer, jobs to the Producer, questions to the Discoverer, and RFIs to External Sources. This activity takes place before each of these other nodes executes during the time step so they can incorporate data as they receive it. Similarly, Receive At Repository waits until each node has completed their execution for the time step so it can receive data as soon as it is ready. This activity receives questions from Consumers, products from Producers, RFIs from Discoverers, and RFI responses from External Sources.

Finally, External Sources are simulated with the red boxes. They receive RFIs from the Repository, respond to those RFIs, and send the responses back to the Repository. As with the Consumer, the process required to respond to RFIs is not tracked in detail, only the amount of time required and the type of raw data returned are currently handled within the simulation. It is possible to model External Sources in more detail but it must be done in a way that does not detract from the OPCEN processes.

During each time step, sending items from the Repository must occur before any of the other nodes can execute their logic. Only after those nodes have executed can the Receive at Repository activity run. It should be noted that the Receive and Send activities of the Consumer and External Sources have no embedded logic scripts. Their logic is instead handled within the Send and Receive activities of the Repository, respectively. They are shown in the diagram to illustrate the Consumers' and External Sources' perspectives into what is taking place.

Input Data Files

One advantage of the SMOFN is that the model itself is only concerned with the execution logic. All operating parameters, such as the number of operators at an OPCEN and their skill sets, or the number of jobs to be done, are read from data files during the setup stage of the simulation.

Simulation Setup

The first data file read into the SMOFN is identified by the user and points to the data path where the rest of the files will be found. The next files read include a list of OPCENs and the data paths containing their setup information, a time at which the simulation will stop, and a switchboard allowing certain business rules to be specified.

OPCEN Inputs

Each OPCEN is defined by five different files read into the SMOFN model:

1. An events list;
2. A summary of thread types;
3. A matrix of operator skills;
4. A set of utility decay curves; and
5. The step-by-step definitions of each thread.

Events list

The purpose of the events list is to define what jobs will be added to the OPCEN job queue, and when. The main entries for each job are the thread name, priority, and the time at which the job will be added to the queue. In the OPCEN SM, this list would include all jobs that the OPCEN would process over the course of the simulation, however in the case of the SMOFN, it should only include jobs that are based on OPCEN operating procedures. Other jobs, based on the arrival of new data are more appropriately listed in the schedule of new jobs that the Repository will send to each OPCEN (described in the next section). For this reason, the utility of the job will typically begin to decay the moment the job is added to the queue. However it is possible to specify that data was collected at some earlier time, which will form the basis for utility decay. Finally, each job can have a specified deadline, though this can be set to zero and a standard slack time for the particular job type will be used.

Thread types

The thread types file defines the overall characteristics of each job type that is processed at the particular OPCEN. The details specified here reflect the job as a whole, whereas the characteristics of each step within the job thread are specified in a separate file. Each entry is identified by thread name and priority (entries in the events list will have their priority rounded to those identified here if necessary). The data that is used to describe a thread includes the percent chance that a job will return NSTR, the default slack time for the job, how many steps an operator can look ahead to estimate the expected utility gained over time spent on the job, whether the job is redundant with other jobs of its type (jobs that are redundant will be ignored if another job of the same type is in progress when a new one arrives in the queue), the utility of a job that returns NSTR, and the details of aggregating other jobs into this one. All of this data is used by the SMOFN to tailor its business rules to each job thread as appropriate.

For the SMOFN to properly account for the aggregation of several completed products into one new job, the following details need to be specified for the aggregating job thread: the name of the job step where aggregation will take place, the types of completed jobs to aggregate, the amount of time added to the aggregation step for each job aggregated, and the percentage of utility that carries over from the aggregated job to the aggregating job. The time added and utility carried over can be different for each job type that is aggregated.

Operator skills

The SMOFN refers to the operator skills matrix to assign operators in an OPCEN to queued jobs. Once assigned, this matrix also identifies how effectively each operator does their work. The operator skills matrix identifies each operator in an OPCEN by a "name" that is a string containing only letters and numbers. The information specified for each operator includes their speed and quality of work (both expressed as percentages where 100% speed means jobs are done in the standard required time, and 100% quality means the operator adds the expected utility to jobs they work on), the order in which they are assigned to generic work, the percent chance they are able to take on generic work when they are not already assigned any specific job, and a list of their skills, beginning with their primary skill and allowing up to ten entries. When a queued job requires a particular skill, the SMOFN will look to each operator's first skill, then each operator's second skill, and so on until an available operator with the required skill is found.

Utility decay curves

The utility decay curves are used by the SMOFN to describe how the utility of products decreases as those products age. Each line begins with a thread name and priority (the combination of which must match those in the thread types file) and a time expressed in units of the simulation's time steps (so far, we have used minutes for our simulations). Each line then ends with three numbers representing the lower bound, peak value, and upper bound of a triangular distribution. The logic of the utility decay is that after the product of a particular thread type has aged by the specified amount of time (age is counted from the moment the raw data was taken), its potential utility can be calculated by the specified triangular distribution. The SMOFN logic will ensure that potential utility will never go up as products age, even if the distributions allow it. Actual utility is the product of the utility accrued by work done on the job and the potential utility based on product age.

Thread definitions

The last file is used to describe the threads, in terms of how the SMOFN should handle each step within each job. This file uses one line of data for each step of each thread type (thread types must be the same, by name and priority, as those in the thread types file). The data for each line includes the thread type, the name of the step, the time and skill required to complete it, the number of posts (utility updates) during the step, whether or not it can be interrupted, the lower bound, peak, and upper bound defining a triangular distribution used to calculate the utility achieved by the end of the step, the file size of the resulting product (used to track uploading to the Repository and subsequent distribution to Consumers), the type of step (in terms of which logical script to execute within the SMOFN) and the name of the next step to move to after the current one.

Seven types of steps are currently defined in the SMOFN, the first three of which already existed in the OPCEN SM. Step type 1 involves no decision logic as to the following step, once the current step is complete the job will simply move on to the next. Step type

2 is the NSTR decision: if the job is to be marked NSTR, that will happen at the beginning of a step type 2 and it, along with any subsequent steps will be skipped. Step type 3 is the final step of any Production job, after which the job is marked complete and removed from the queue.

The remaining four steps are unique to Discovery jobs. Step type 4 is the decision as to the results of the Discovery process and has three possible exits. The first possible result is that no product exists, the second is that the desired product is found, and the final possibility is that some data is found but it is insufficient. In the case that no product exists, the Discovery job will move on to Step type 5, which is to create a request for new data that will be sent to External Sources. If the product is found, it must be analysed and put in the context of the originating question, so a new Production job is added to the appropriate OPCEN's schedule. The logic to do this is handled by step type 6. If insufficient data is found, then the Discovery will proceed to step type 7, which is a combination of the previous two. That is to say that a request will be sent to External Sources for more data, but in the mean time, a new Production job will be added to the OPCEN schedule to analyse whatever data was found.

Repository Inputs

While each OPCEN's initialization data is read into the SMOFN, a separate set of data files is used to describe the operating parameters of the Repository. These files are:

1. A schedule for the arrival of new data;
2. A schedule for the arrival of new products created outside the simulation;
3. A matrix organizing the delivery of products to the appropriate Consumers;
4. A list of each OPCEN's bandwidth; and
5. A list controlling the generation of questions.

Data arrival schedule

The data arrival schedule is similar to the events list for each OPCEN. Each line specifies the time that the new data was created, the time at which it will arrive at the Repository to be delivered to the Producer, the source of the data (such as a reconnaissance unit), the type and priority of the job thread that will be generated, the deadline for the data analysis job, the Producer OPCEN to which the data will be sent for analysis, the delivery method (i.e. push or pull), and the size of the file that must be delivered. This file is used to simulate the fact that many jobs within an OPCEN are triggered by the arrival of new data in the Repository, rather than by the OPCEN's own operating procedures.

New product arrival schedule

The schedule of outside products defines the arrival of products from Producers that are not tracked by the model. These products can then be forwarded to Consumer OPCENs as though they were created within the simulation. The required data includes the name of the producing OPCEN, the type of job (including the priority rounded to the nearest

appropriate thread type), the specific priority, the file size to be transferred, the time at which the original data was collected, the time at which the product will arrive at the Repository, the status of the job producing the product (i.e. complete, or the end of a specific job step, in which case the product corresponds to the most recent update), the utility achieved due to work on the producing job, and the decayed utility due to the data's age. This product arrival is an important consideration for the SMOFN so we can simulate how one OPCEN (or several) reacts to products created by other OPCENs, without having to model those other OPCENs and their own product creation processes in detail.

Delivery matrix

The SMOFN uses the delivery matrix to identify what products are sent from what Producers to what Consumers. This applies equally to products created within the SMOFN simulation as well as to products identified in the new product arrival schedule. A line of data contains the product type, the name of the Producer and Consumer OPCENs (one of each, products sent from one Producer to multiple Consumers must have one line for each Consumer), the percent probability that a product of the given type will be sent to the specified Consumer when created by the specified Producer, and the method of delivery (push or pull). This file is the main source of information for the SMOFN to account for data sharing between OPCENs.

Bandwidth

The bandwidth file is very simple, containing only a list of OPCENs that connect to the repository along with their bandwidth. This identifies to the SMOFN the amount of data that can be transferred between each OPCEN and the Repository during a single time step. Aside from the Producer and Consumer OPCENs, the Discover and External Sources nodes should also be included here. Units are at the discretion of the analyst, but must be consistent with the file sizes for products identified in the thread definitions and the new product arrival schedule, as well as raw data in the data arrival schedule.

Question generation

The questions file draws from the combinations of product types and Consumers that come out of the delivery matrix. For each such combination, the questions file identifies to the SMOFN the percent chance that a question will be generated by the Consumer, the file size required to contain that question, and the amount of time that the Consumer will take between receiving the product and sending out the question. This information is referred to whenever a product is sent to a Consumer so that questions are generated as appropriate.

Way Ahead

Work on the SMOFN model itself is nearing completion. The major issue now is to collect the data required to populate all of the initialization files described above. This

data must be representative of the operating parameters of Canadian Forces OPCENs, as the goal is to accurately simulate data flow between these. Particular emphasis is being placed on describing job threads that involve multiple OPCENs, such as the process to report and respond to events in theatre or the preparation of high-level daily briefs. Another major component is the composition of various OPCENs, as far as the number of operators on shift and the skills they are trained in. Although much of this data will be classified, the model itself will remain unclassified as it only reads the data upon execution.

The planned process for creating an operational architecture of the Canadian Forces (CF) command structure is illustrated in Figure 5. We begin by examining current C2 practices to capture them as executable threads, in light of the capabilities definition knowledge gained through previous work with the SMOFN model. These threads can then be converted into input to the SMOFN model. This, along with any developments of the SMOFN execution capability, leads to a more complete version of the SMOFN. The first payoff from the operator's perspective is that the processes captured as threads are now documented and can be used to create standard operating procedures. The second is that the inclusion of these threads into the SMOFN allows them to be analysed and potentially improved. The payoff from the modeller's perspective is the creation of a more complete definition of a CF C2 operational architecture.

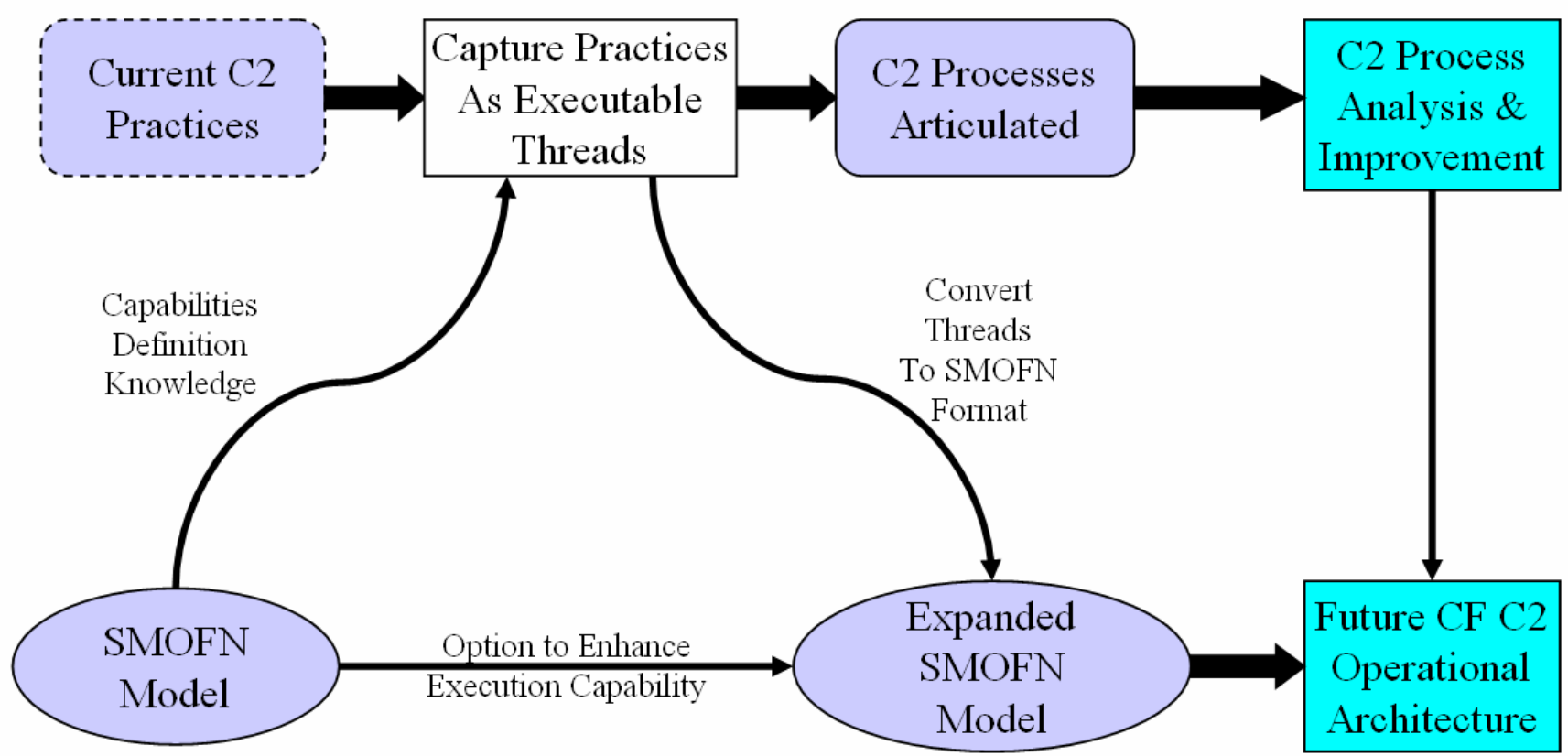

Figure 5: Planned Operational Architecture Creation Process

Reference

[1] Funk, R.W., M.G. Ball, and R. Sorensen, “Building Executable Architectures of Net Enabled Operations Using State Machines to Simulate Concurrent Activities”, presented to 11th ICCRTS, Cambridge, UK, September 28, 2006.