Quality of Security Service
Costing Demonstration for the MSHN Project

by

Evdokia Spyropoulou
Timothy Levin
Cynthia E. Irvine

April 2000

Approved for public release; distribution is unlimited.

Prepared for: Defense Advanced Research Project Agency
Information Technology Organization

20000522 036

DTIC QUALITY INSPECTED 2
This report was prepared for as part of the Naval Postgraduate School Center for Information Systems Security (INFOSEC) Studies and Research (NPS CISR) at the Naval Postgraduate School, as part of a project funded under the Defense Advanced Research Projects Agency/Information Technology Organization grant under the Quorum program.

This report was prepared by:

Evdokia Spyropoulou
Research Associate

Cynthia E. Irvine
Assistant Professor

Reviewed by:

Released by:

Neil C. Rowe
Associate Professor
Department of Computer Science

D. C. Boger, Chair
Department of Computer Science

D. W. Netzer
Associate Provost and Dean of Research
Security requirements for a task, system or network may permit the selection of a range of underlying services or security behaviors. When a range of services is available, variant security is possible. Variant security permits the notion of Quality of Security Service (QoSS) to be introduced. This paper describes a quality of security service demonstration, specifically with respect to costing. We describe the network as having three modes: normal, impacted, and emergency. For each of these modes, the user is given three possible security levels: low, medium and high. A variety of security services contribute to the overall security of each task. Each service has two costs: an initialization cost and a run-time cost. The demonstration illustrates the costs incurred as network modes and security levels are changed. High level and detailed specifications are provided.
Abstract

Quality of Security Service (QoSS) is possible when there is variant security, i.e. mechanisms that “allow a range of security behaviors” [3] and “offer the user various “degrees”, or strengths, of security”[1]. The base system security policy may impose certain minimum requirements for security. Assuming that underlying system mechanisms can provide variant security according to user choices for security level, an application could provide various degrees of security, always complying though with the restrictions placed by the system’s policy.

When an application / task is executed, it may access various network resources (e.g. bandwidth, CPU, storage). Security costs are fixed if there are no security selections. Variant security, on the other hand, causes a security overhead for the application which will depend on the user’s request. For each variant security mechanism we need information for the resource costs associated with the specific task. This way we are able to estimate a cost for security on a per-task, per-resource basis [3].

A preliminary taxonomy of security services, as the groundwork for a system to supply security-costing information to an RMS, is presented in [1], [2]. An application/task may invoke the use of various security services (e.g., authenticity, confidentiality, integrity, non-repudiation, etc.). The notion of service area [1] associates the security mechanism(s) that implement the service, with a topographical component of the network. A list of security services, mechanisms, and services is provided in [1].

A method for making the interaction with a wide range of security services and mechanisms comprehensible and easy for administrators and users is presented in [3].

A task is characterized by a set of security requirements that must be met. This set of requirements can vary if a dynamic network security policy is applied. This means that the network status or “mode” (e.g. normal, emergency, impacted mode) will influence the security restrictions and available security services for the task. We can predefine a set of alternate security policies. If the network administrator changes the network mode of operation, the appropriate policy will be employed, and the corresponding set of security requirements will apply to a specific task [3]. Thus, a high-level interface (mode selection) allows the alternation between security policies.

Similarly a user can specify the desired degree of security service, that is to be applied to the processing of the network task. Instead of presenting to the user all combinations of security

---

This research is supported by the DARPA/ITO Quorum Program
mechanisms and parameters for the variant services, we can offer a set of security level choices (e.g. high, medium, low). These abstract choices are translated to a pre-selected set of security mechanisms and settings. This way we can map “a simplified user abstraction of security to detailed underlying mechanisms, such that users can be presented with a coherent user-level view of available security options” [3].

The Quality of Security Service Costing Demonstration illustrates the concepts described above and provides a method for quantifying costs related to the security service. Using the taxonomy we identify services (discriminating between service areas) that tasks may invoke, and security mechanisms that implement them. We pre-define sets of security settings, corresponding to network modes and user security level choices. Costs for a task are calculated and expressed in terms of resources. These costs depend on the specific task’s security characteristics that were selected.

In Appendix A, the purpose and the requirements for the Quality of Security Service Costing Demonstration are presented.

Various logical structures are also introduced:

To describe security requirements posed from the network system for a specific task, a Task Requirement Vector is used consisting of service requirement components. Mode Service Matrix incorporates the idea of dynamic network and alternate sets of security policies.

To describe a set of specific security settings for a task, we use a Task Variable Vector. Availability of abstract user security level choices, leads to the need for a set of Task Variable Vectors, described by a Choice Variable Matrix. Furthermore the effect of mode selection on the security settings is expressed through the Mode Choice Matrix.

The cost for a resource during execution of a specific task is specified in a resource cost expression. Costs for all resources of a service are described in a Service Cost Vector and costs for all services invoked by a task are associated with a certain Cost Matrix.

Additionally, generic functions, which are necessary for the demo’s required processing logic, are presented in Appendix A.

In Appendix B the low-level specification for Quality of Security Service Costing Demonstration is presented. Objects and functions, for implementing Appendix’s A logical structures and relevant functionality, are specified, along with necessary constants and structures for storing costing information for tasks.

It should be noted that Appendix B is an evolving document (and SecurityCosts an application under further development). Future work will address among other issues:

- population of the demo with realistic costing data
- determination of best units for cost measures
- enumeration of specific security mechanisms with respect to the described taxonomy
- costing data storage issues.
References


Appendix A: MSHN Security Costing Demonstration Requirements

1 Overview

The MSHN project is designed to schedule multiple tasks into an environment that has a defined set of (finite) resources. In order for it to evaluate which tasks and resources it can “afford” to run concurrently in such an environment, it needs to know how much of each resource is consumed by each prospective task. With respect to security, the scheduler needs to know how much it will cost (viz, in terms of resources) to meet the relevant security requirements. For resources with fixed security needs, the overhead is fixed, and the scheduler does not have to perform any special calculations. On the other hand, for “variant” security, the scheduler needs to know the extent to which security will increase resource cost (i.e., level of resource usage) for a given “degree” of security service. A taxonomy of security services as well as an approach for specifying the cost of security for general types of security services are discussed in [1].

The purpose of the MSHN Security Costing Demonstration is to implement a prototype mechanism for storing, processing and retrieving security costing data for a range of representative security services. As a secondary purpose, the demo will illustrate the conceptual mapping of high-level security requirements to detailed services for different network operational modes (see [2]). The effect of this mapping in the demo will be that the security costs returned for a task will change according to the network mode and the user’s choice of high-level security requirements.

2 Requirements

The demo will produce the following items in pursuit of the purpose stated in Section 1. These items will be delivered in the form of one or more technical reports, and one or more laboratory demonstrations.

- A defined generic security service costing algorithm (see Section 4.1.1), derived from [1]
- A defined subset of security services for MSHN (derived from the taxonomy in [1]), and a particularization of the security costing algorithm for each of those services.
- A defined set of hypothetical tasks which invoke elements of the security service subset.
- A defined set of abstract user security choices, as per [2].
- A defined set of administrative network modes, as per [3].
- An automated mechanism and corresponding user manual for managing security costing data. The mechanism shall provide the following capabilities:

  - Take and store input for description of task and environmental security requirements.

---

1. a task has a name and a fixed application program; it may be invoked with various data sets.
2. security requirements for which the user is allowed a range of choices with respect to the degree of security applied. See [3] for examples.
-Process a task request, returning the estimated security costing data.
-Support abstract user security choices with respect to variant security requirements.
- Support network operational mode, returning different estimated security costing data for different modes.

3 Conventions

- Identifiers appear in italics.
- When it is being defined, an identifier is underlined.
- Identifiers for internal variables begin with an _underbar (“_”).
- Names for demo interface functions have a “demo_” prefix, as well as underbars between the words (e.g., demo_function_name), but are not italicized; internal demo functions have a “i_” prefix, instead.
- A “vector” structure is a series of value-holding items; a “matrix” structure is a series of vector structures.
- “<-” is the assignment operator: “A <- B” means that the value of B is assigned to A.

4 Development Task Descriptions

Items identified in Section 2 are described in further detail as development tasks in the sections below. There is a “definitions” task and an “automated mechanism” task, each with several subtasks.

4.1 Definitions Task

This subtask is to provide definitions and conceptual foundations for the second task (Section 4.2, “Automated Mechanism Task,” on page 9). The results of these subtasks may be consolidated in a technical report; they are integral to the development of the Automated Mechanism Task (Section 4.2).

4.1.1 Security Costing Algorithm

The purpose of this subtask is to define a generic security service costing algorithm, derived from [1]. For the security costing algorithm, the following logical structures are defined (see also, Figure 1 on page 4, and the Appendix):

- **task requirement vector**: A per-task collection of service requirement components. This structure is introduced in [3]. There is a system task requirement vector which contains network environment security requirements common to all tasks (in processing, the system vector may be logically appended to the task’s task requirement vector). Note that there are corresponding system components, as well as per-task components for each of the following type of structures.

- **service requirement component**: A boolean expression (possibly a compound of several boolean clauses) regarding the security requirements of a specific resource or security service, and
containing at most one variant security clause. A security service is a high-level type of resource, which is typically made up of other resources. Resources that are not defined in terms of other resources are termed elemental resources. A security service may be represented by one or more service requirement components.

**task cost matrix**: contains cost formulae for the task in the form of a vector of service cost vectors. In the task cost matrix there is a service cost vector corresponding to each security service or resource defined in the task requirement vector.

**service cost vector**: contains cost data for a specific security service or resource used by the task, in the form of a per service collection of resource cost expressions. There is one expression for each resource utilized in effecting the requirements of that service.

**resource cost expression**: contains a cost expression for a given resource. The value of this expression may be stated relative to the variant security variables in related service requirement components, e.g., a given cost may be dependent on another component’s cost expression. Each expression has the form: “start-up cost -- streaming cost”, where either “start-up cost” or “streaming cost” may be empty, but (realistically), not both.

**task variable vector**: a structure which is parallel to the task requirement vector, where each component is used to specify a value for the variant security variable found in the corresponding service requirement component.

**task result matrix**: a structure which is parallel to the task cost matrix, where each component is used to specify a vector of cost results, one result for each resource cost expression in the corresponding service cost vector.

Figure 1 on page 4 shows the relationships between these structures with an example. In this example, some of the elements (i.e., those so marked) represent variant security requirements, while others represent invariant security requirements.
To complete the algorithm to derive the task’s security service costs we need to define the processing for these vectors, which is simply the following. A user indicates specific Quality of Security
Service (QOSS) selections in a *task variable vector* submitted with the task request. The demo plugs these values into the corresponding *resource cost expressions* to derive specific resource costs for a *task result matrix*, which is returned to the requestor.

**4.1.2 Subset of Security Services**

The purpose of this subtask is to define a subset of the generic security services in [1] that are pertinent for MSHN and that will be useful in making this demonstration illustrative of its overall purposes. Also, this subtask must provide a particularization of the security costing algorithm for each element of the subset.

The list of security service categories from [1] is as follows

<table>
<thead>
<tr>
<th>Table 1: Security Services</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Data/object Confidentiality</td>
</tr>
<tr>
<td>Data/object Integrity</td>
</tr>
<tr>
<td>Traffic Flow Confidentiality</td>
</tr>
<tr>
<td>Authenticity</td>
</tr>
<tr>
<td>Non-Repudiation</td>
</tr>
<tr>
<td>Guarantee of Service</td>
</tr>
<tr>
<td>Availability</td>
</tr>
<tr>
<td>Audit and Intrusion Detection</td>
</tr>
<tr>
<td>Boundary Control</td>
</tr>
</tbody>
</table>

a. applies to protection of data objects as well as various types of other objects, such as network nodes, subnets, and devices.

As shown in Table 1, services in each of these categories may be employed for protection in intermediate network nodes (IN), network wire (NW), end systems (ES), with the exception of audit and boundary control, which effect security in the total subnet (TS) and non-repudiation, which effects only intermediate node and end system security.

**SUBSET**

From discussions with MSHN sponsoring organizations and review of MSHN planning documents (e.g., [5]) the following subset of generic security services is found to be relevant to the MSHN operational environment:

Data/object Confidentiality ES, IN (network objects are protected)
Data/object Integrity NW, ES (data transmission as well as network objects are protected)
EXAMPLE PARTICULARIZATION

This section provides an example of how the security service costing algorithm would be particularized to suit one of the elements of the subset, the "data integrity on the wire" security service.

- task requirement vector:
  a vector with three components is defined.

- service requirement component:
  The first component states that network communications using subnet Subnet_A will be cryptographically signed to provide integrity at the packet level, with a rate of integrity checking greater than 60%. Such a range of checking might be useful in transmitting images or other streams of data that are "averaged" somewhat by the receiver. The second component states that the symmetric key length used on Subnet_A will be in a certain range. The third component states that the user is authorized to use the subnet, and is an invariant expression. The first two components describe requirements for use of the "data integrity on the wire" security service.

  -(S1a) \( \forall p : \text{packets}, s: \text{subnet}((\text{send}(p, s) \& s = \text{Subnet}_A) \Rightarrow \text{i-packet\_integrity\_rate}(s) \geq 0.60) \)
  -(S1b) \( \forall p : \text{packets}, s: \text{subnet}((\text{send}(p, s) \& s = \text{Subnet}_A) \Rightarrow 56 \leq \text{i\_key\_length}(s) \leq 128) \)
  -(S2) \( \forall p : \text{packets}, s: \text{subnet}((\text{send}(p, s) \& s = \text{Subnet}_A) \Rightarrow \text{authorized\_access}(\text{user\_id}(p), s)) \)

- task cost matrix:
  a vector with two components is defined.

- service cost vector:
  (C1) contains cost data for use of the "data integrity on the wire" (S1) security service. This vector has three components, use of the cpu (C1/R1), memory (C1/R2), and bandwidth (C1/R3) resources. (Each expression has the form: start-up cost -- streaming cost.)

  -(C1/R1) 5000 processor clocks + (10 clocks x \( \text{i\_key\_length}(\text{Subnet}_A) \)) -- 40 Processor clocks per packet x \( \text{i\_packet\_integrity\_rate}(\text{Subnet}_A) \)
  -(C1/R2) (6144 + \( \text{i\_key\_length}(\text{Subnet}_A) \)) bytes -- (5120 + \( \text{i\_key\_length}(\text{Subnet}_A) \)) bytes
  -(C1/R3) 0 -- 8 bytes per packet x \( \text{i\_packet\_integrity\_rate}(\text{Subnet}_A) \)

  -(C2) empty (S2 is constant, so no expression is required here).

- task variable vector:
  a vector with three componentsis defined.

  -(V1a) \( \text{i\_packet\_integrity\_rate}(\text{Subnet}_A) = 0.8 \)
  -(V1b) \( \text{i\_key\_length}(\text{Subnet}_A) = 128 \)
  -(V2) empty (S2 is constant, so no expression is required here).

- task result matrix:
  a matrix with two vectors is defined, to contain security costs for the two defined services.
4.1.3 Hypothetical Tasks

The purpose of this subtask is to define a set of hypothetical tasks which invoke elements of the security service subset.

example

An example of an application or user task which would utilize the packet integrity service is a simple network file transfer program, like ftp.

4.1.4 Abstract User Security Choices

The purpose of this subtask is to define a set of abstract user security choices, as per [2], for users to symbolically select predefined task variable vectors. To represent this choice, we introduce the following structure:

choice variable matrix: a structure which maps user security choices to task variable vectors.

Mechanisms for managing the choice variable matrix are provided in Section 4.2.

example

For this example, we will use the example abstract user security choices from [2]:

user security choice::= (high I medium I low)

A set of mappings is defined for these user security choices and the variables in component V1 of the example task variable vector:

HIGH -> (V1a) i_packet_integrity_rate(Subnet_A) = 1; (V1b) i_key_length(Subnet_A) = 128
(V2) undefined

MEDIUM -> (V1a) i_packet_integrity_rate(Subnet_A) = .8; (V1b) i_key_length(Subnet_A) = 96
(V2) undefined

LOW -> (V1a) i_packet_integrity_rate(Subnet_A) = .6; (V1b) i_key_length(Subnet_A) = 56
(V2) undefined
A corresponding choice variable matrix for these mappings is shown in Table 2:

**Table 2: Choice Variable Matrix**

<table>
<thead>
<tr>
<th>User Security Choice</th>
<th>Low</th>
<th>Medium</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.6</td>
<td>.8</td>
<td>.6</td>
</tr>
<tr>
<td></td>
<td>56</td>
<td>96</td>
<td>128</td>
</tr>
<tr>
<td>undefined</td>
<td>undefined</td>
<td>undefined</td>
<td></td>
</tr>
</tbody>
</table>

### 4.1.5 Network Mode Choices

The purpose of this subtask is to define a set of administrator-selectable network modes, as per [2] [3]. The different modes determine different security characteristics for the tasks and the network system. To represent the semantics of these modes, we introduce the following structures:

- **mode service matrix**: a structure which maps network modes to task requirement vectors.
- **mode choice matrix**: a structure which maps network modes to choice variable matrixes.

Mechanisms for managing these matrices are provided in Section 4.2.

**Example: modes**

For this example, we will use the example network mode choices from [2] [3]:

- network mode::= (normal | impacted | emergency)

**Example: modes mapped to task requirement vectors**

The alternate mappings from these modes to different task requirement vectors forms a mode service matrix:

- normal -> (S1a) \( \forall p: \text{packets}, s: \text{subnet}((\text{send}(p, s) \& s = \text{Subnet}_A) \Rightarrow \text{packet integrity rate}(\text{Subnet}_A) \geq .60); \)
  
  (S1b) 56 \( \leq \) i_key_length(Subnet_A) \( \leq \) 128; (S2) authorized_access(user_id(p), Subnet_A)

- impacted -> (S1a) \( \forall p: \text{packets}, s: \text{subnet}((\text{send}(p, s) \& s = \text{Subnet}_A) \Rightarrow \)
  
  .20 \( \leq \) packet integrity rate(Subnet_A) \( \leq \) .60;
  
  (S1b) i_key_length(Subnet_A) = 56; (S2) authorized_access(user_id(p), Subnet_A)

- emergency -> (S1a) \( \forall p: \text{packets}, s: \text{subnet}((\text{send}(p, s) \& s = \text{Subnet}_A) \Rightarrow \) packet integrity rate(Subnet_A) = 1);
  
  (S1b) i_key_length(Subnet_A) = 128; (S2) authorized_access(user_id(p), Subnet_A)

**Example: modes mapped to user security choices**

The alternate mappings for user security choices, based on network modes are shown in Table 3. Logically, this forms a 3-component vector of choice variable matrixes, called a mode choice matrix.
Table 3: mode choice matrix

<table>
<thead>
<tr>
<th>Mode</th>
<th>User Security Choice</th>
<th>low</th>
<th>medium</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>\textit{V_{1a}}</td>
<td>.6</td>
<td>.8</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>\textit{V_{1b}}</td>
<td>56</td>
<td>96</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>\textit{V_{2}}</td>
<td>undefined</td>
<td>undefined</td>
<td>undefined</td>
</tr>
<tr>
<td>impacted</td>
<td>\textit{V_{1a}}</td>
<td>.2</td>
<td>.4</td>
<td>.6</td>
</tr>
<tr>
<td></td>
<td>\textit{V_{1b}}</td>
<td>56</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>\textit{V_{2}}</td>
<td>undefined</td>
<td>undefined</td>
<td>undefined</td>
</tr>
<tr>
<td>emergency</td>
<td>\textit{V_{1a}}</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>\textit{V_{1b}}</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>\textit{V_{2}}</td>
<td>undefined</td>
<td>undefined</td>
<td>undefined</td>
</tr>
</tbody>
</table>

\textbf{modes not mapped to task costs}

On the other hand, variable task costs are built into the cost expressions, so mappings external to those expressions are not required.

4.2 Automated Mechanism Task

The purpose of this task is to produce an automated mechanism and user manual for managing security costing data. Development of the mechanism consists of the following subtasks. Functions are presented in a generic syntax to show the required processing logic. The demo need not preserve the variable syntax or specific interfaces show here.

4.2.1 Specification of internal state

The purpose of this subtask is to consolidate in one place the logic for the internal state of the automated mechanism, such that the specification for the rest of the subtasks in this task can specify their “effects” (viz, changes) relative to that state. These logical constructs represent global variables in the demonstration mechanism. First we define an indexing mechanism:

\textit{task/system designator}: indicates a specific task or the network system. Typically, as below, a vector may have components for each task, as well as one for the network system.

We introduce a system constant of type \textit{task/system designator} to represent the network system: \textit{SYSTEM}.

\textbf{Global Variables}

\textit{current operational mode}: a \textit{network mode}, by default set to “normal.” This indicates the mode that is currently in effect for the network system as a whole.
_mode services_(task/system designator): the _mode service matrix_ for the system or designated task.

_**current service vector**_(task/system designator): a _task requirement vector_, which is currently in effect for the system or designated task. Changes to _**current operational mode**_ change the value of this variable.

_**mode choices**_(task/system designator): the _mode choice matrix_ for the system or designated task.

_**current choice matrix**_(task/system designator): a _choice variable matrix_, which is currently in effect for the system or designated task. Changes to _**current operational mode**_ change the value of this variable.

_**current cost matrix**_(task/system designator): the _task cost matrix_ for the system or designated task.

### 4.2.2 Task and environment setup

The purpose of this subtask is to create functions for the automated mechanism to take and store administrator input for specification of security attributes of tasks and the network environment.

**demo_set_task_services**

This function establishes/updates the security services and requirements for the system or a task.

**Input**

TSD: _task/system designator_ - indicates the task (or the system) for which to modify requirements

TSV: _task requirement vector_ - new security requirements

**Output**

none

**Processing**

none

**Effects**

update _**current service vector**_ for the system or task to the value of TSV:

_**current service vector**(TSD) <- TSV

**demo_set_task_costs**

This function establishes/updates the cost formulas for the system or a task.

**Input**

TSD: _task/system designator_ - indicates the task (or the system) for which to modify requirements
TCM: *task cost matrix* - new security costing information

**Output**

none

**Processing**

none

**Effects**

update *_current cost matrix* for the system or task to the value of TCM:

\[
_{current\ cost\ matrix}(TSD) \leftarrow TSV
\]

### 4.2.3 Process task request

The purpose of this subtask is to create functions for the automated mechanism to take user input for and process a task request. Processing will result in the return of estimated security costing data to the user.

**demo_task_request**

This function issues a request to the demo for a task’s security costing information. The security costing information is an estimate of the cost to access the variant security mechanisms associated with running the task.

**Input**

TSD: *task/system designator* - indicates the task for which to return information

TVV: *task variable vector* - specifies the values of task-specific variables for execution of this task

STVV: *task variable vector* - specifies the values of system-specific variables for execution of this task

**Output**

TRM: *task result matrix* - cost values for the task

STRM: *task result matrix* - cost values for the system.

Output format shall provide intuitive correspondence to the input (e.g., same nomenclature).

**Processing**

Use i_task_request with the inputs to obtain the output:

\[
\text{output} \leftarrow \text{i_task_request}(\text{TSD}, \text{TVV}, \text{STVV})
\]

**Effects**

none

**i_task_request**

This function calculates the costs of a task request
Input
TSD: task/system designator - indicates the task for which to return information
TVV: task variable vector - specifies the values of task-specific variables for execution of this task
STVV: task variable vector - specifies the values of system-specific variables for execution of this task
Output
TRM: task result matrix - cost values for the task
STRM: task result matrix - cost values for the system.
Processing
The security choices indicated in the task variable vectors (TRM and STRM) are plugged into _current cost matrix(TSD) and _current cost matrix(SYSTEM), respectively, to arrive at estimated costs.
Effects
none

4.2.4 Support abstract user security choices
The purpose of this subtask is to create functions for the automated mechanism to take and store administrative input that maps abstract user security choices (Section 4.1.4 ) to the exact security choices found in the task variable vector for the system or a task.
Additionally, this subtask modifies the demo_task_request interface (Section 3.2.2) to take an abstract user security choice. demo_task_request processing is modified to obtain exact security choices from the user security choice.
demo_map_user_choices
This function initializes or modifies the detailed security choices associated with each defined abstract user security choice.
Input
TSD: task/system designator - indicates the task (or the system) for which to modify security choices
CVM: choice variable matrix - a set of mapping statements, correlating each abstract user security choice to a task variable vector.
Output
none
Processing
none
Effects
The system or task’s _current choice matrix is overwritten with the input, i.e.:
_current choice matrix(TSD) <- CVM.

Note: optionally, this function could also take a user security choice, and work on one choice of the task’s _current choice matrix at a time.

demo_task_request_2

This function issues a request to the demo for a task’s security costing information.

Input
TSD: task/system designator - indicates the task for which to return information
USC: user security choice - the user’s choice of security for the invocation of this function.

Output
TRM: task result matrix - cost values for the task
STRM: task result matrix - cost values for the system

Processing
Use the abstract user security choice to obtain a task variable vector for the task and the system from the _current choice matrix. Internal variables _ITVV and _ISTVV of type task variable vector are introduced for illustration:

_ITVV <- (_current choice matrix(TSD)).USC
_INETV <- (_current choice matrix(SYSTEM)).USC

Use i_task_request with the resulting task variable vectors to obtain the output.

output <- i_task_request(TSD, _ITVV, _ISTVV)

Effects
none

4.2.5 Support network operational mode

The purpose of this subtask is to create functions for the automated mechanism to take and store administrative input to (1) set the _current operational mode and associated internal state, (2) set the per-task and system _mode services matrixes (viz, alternate task requirement vectors for the different modes), and (3) set the per-task and system _mode choices (viz, alternate abstract user security mappings relative to those vectors/modes).

demo_set_network_mode

This function sets the value of the network operational mode.

Input
MODE: network mode - new network operational mode, as per Section 4.1.5

Output
none
Processing

none

Effects

_current operational mode_ is set to value of MODE.

For each ENTRY of type task/system designator (viz, all tasks and the system entry), _current service vector_ and _current choice matrix_ are set to the component indicated by MODE in the corresponding _mode services_ and _mode choices_ matrixes:

\[
\text{current service vector}(\text{ENTRY}) \leftarrow \text{mode services}(\text{ENTRY}).\text{MODE}
\]
\[
\text{current choice matrix}(\text{ENTRY}) \leftarrow \text{mode choices}(\text{ENTRY}).\text{MODE}
\]

demo_set_mode_services

This function initializes or modifies the _mode services_ matrix for a task or the system. This function replaces demo_set_task_services.

Input

TSD: task/system designator - indicates the task (or the system) for which to

msm: mode service matrix - a set of alternate task requirement vectors, each mapped to a different network mode, per Section 4.1.5.

Output

none

Processing

none

Effects

System/Task's _mode services_ matrix is modified according to input:

\[
\text{mode services}(\text{TSD}) \leftarrow \text{MSM}
\]

Note: again, this function could just as well include a mode parameter, and effect only one mode of _mode services_, per task

demo_set_mode_maps

This function initializes or modifies the _mode choices_ matrix for a task or the system. This function replaces demo_map_user_choices.

Input

TSD: task/system designator - indicates the task (or the system) for which to set alternate user choices.

MCM: mode choice matrix - a set of choice variable matrixes, each mapped to a different network mode, per Table 3.
Output
none

Processing
none

Effects
System/Task’s _mode choices matrix is modified according to the input:

_mode choices(TSD) <- MCM

5 ERRATA

Interoperability with MSHN and/or the MSHN demos might be considered.

The RMS default values for task variables (e.g., for use when the user has specified a range rather than a specific value for a variant security variable) are not represented (see [4]). These values could be considered to be in special additional _current choice matrixes for each task, but the details need to be worked out.

It is not clear that _mode services or any task requirement vector or mode service matrix is actually needed to implement this demo. However these structures are included in the demo description as essential to understanding the service vector abstraction.
References


[4] Irvine C., Levin, T., Quantifying the Effects of User and RMS Security Choices on Metacomputer Efficiency, NPS Technical Presentation, Forthcoming

Appendix: BNF for Structures and Variables

The structure of the demo structure types and variables is represented here in a Backus-Naur form. In this form, "+" indicates, "one or more" of the item enclosed in preceding parentheses, and that some suitable separator is logically inserted between multiple items (e.g., ",").

**system**

_system current operational mode :::= mode_

**system and per-task**

_system and per-task mode services :::= mode service matrix_

_mode service matrix :::= (mode, task requirement vector)+_

_system current service vector :::= task requirement vector_

_task requirement vector :::= (service requirement component)+_

_service requirement component :::= boolean expression of requirements_

_system and per-task _current cost matrix :::= task cost matrix_

_task cost matrix :::= (service cost vector)+_

_service cost vector :::= (resource cost expression)+_

_system and per-task _mode choices :::= mode choice matrix_

_mode choice matrix :::= (mode, choice variable matrix)+_

_system and per-task _current choice matrix :::= choice variable matrix_

_choice variable matrix :::= (user security choice, task variable matrix)+_

_task variable matrix :::= (variable specification)+_

_user security choice :::= string_

_network mode :::= string_

**Output/Return**

_task result matrix :::= (service result vector)+_

_service result vector :::= (cost value)+_

Further derivation of and syntax for these non-terminal symbols are left to the reader: "boolean expression of requirements," "resource cost expression," "variable specification," "string," and "cost value."
APPENDIX B

LOW LEVEL SPECIFICATION

OF

Quality of Security Service Costing Demonstration for the MSHN Project

Version 0.3
A. INTRODUCTION ........................................................................................................ 4

B. APPLICATION FUNCTIONALITY ........................................................................... 4
   B.1 CONVENTIONS .................................................................................................. 4
   B.2 DESCRIPTION OF OPERATION ....................................................................... 5
   B.3 INTERFACES ................................................................................................... 6
      ➔ Initialization of files (Administrative Interface) ............................................. 6
      ➔ Input (User Interface) .................................................................................... 6
      ➔ Costing Request (User Interface) ................................................................. 6
      ➔ Display Cost Results (User Interface) ......................................................... 6
      ➔ Display Various Info (User Interface) .......................................................... 7
   B.4 COST FORMULAS AND COMPONENT VARIABLES ....................................... 7

C. APPLICATION CONSTANTS ............................................................................... 8

D. FILE STRUCTURES ............................................................................................ 10
   (FILE1) “TASK.DAT” ........................................................................................ 10
   (FILE2) “MSMtoTRV.DAT” ............................................................................... 11
   (FILE3.1 – FILE3.X) “TRV***.DAT” ................................................................ 11
   (FILE4) “MCMtoCVM .DAT” .............................................................................. 12
   (FILE5) “CVMtoTVV.DAT” ................................................................................. 13
   (FILE7.1 – FILE7.Z) “CM***.DAT” ................................................................ 14

E. SPECIFICATION OF ENTITIES ............................................................................ 15
   E.1 THE APPLICATION FRAME ............................................................................ 15
   E.2 OBJECT ENTITIES ......................................................................................... 16
      CSecurityCostsDoc .......................................................................................... 16
      Task .................................................................................................................. 19
      ModeServiceMtrx ............................................................................................. 21
      TaskReqVector ................................................................................................. 22
      ReqComponent ................................................................................................. 23
      ModeChoiceMtrx ............................................................................................... 24
      ChoiceVariableMtrx ........................................................................................ 25
      TaskVariableVector ........................................................................................ 26
      VariableValue ................................................................................................. 27
      CostMtrx ........................................................................................................... 28
      Service ............................................................................................................ 29
      Resource ........................................................................................................... 30

F. FUTURE REFINEMENTS ..................................................................................... 30

G. ARRAYS ............................................................................................................ 33

H. DATA BASE STRUCTURES ............................................................................... 35
A. INTRODUCTION

The Quality of Security Service Costing Demonstration for the MSHN Project illustrates a method for quantifying costs related to the security service. This document is the low-level specification for the demonstration. Objects that implement the needed logical structures, functions that provide the required functionality, along with structures for cost data are described here.

Pre-defined costing information for tasks and the security services they invoke, is necessary. This information can be stored in a database or in a set of files. Aspects of both approaches are covered here. We have selected to store the costing data in the layout that is closest to the needed logical structures for the costing application. This may allow some redundancy on the stored data, but facilitates significantly the application processing on files or data base tables.

Note 1: In the current software version costing info data are not stored in disk. Array structures similar to the disk structures are employed, with hardwired values for a couple of pre-defined tasks, and the ability to input data for new tasks.

Note 2: In all material following only absolutely necessary elements (fields of structures) are displayed. Additional fields (like version number, text descriptions, etc...) can be added later.

The layout of the rest of the document is:
B. Application Functionality
C. Application Constants
D. File Structures
E. Specification of entities
F. Future Refinements
G. Arrays
H. Data Base Structures

B. APPLICATION FUNCTIONALITY

B.1 Conventions

The abbreviations below are widely used throughout the document:
MSM: Mode Service Matrix
TRV: Task Requirement Vector
MCM: Mode Choice Matrix
CVM: Choice Variable Matrix
TVV: Task Variable Vector
CM: Cost Matrix

File structures are referred to as below:
Data base structures are referred to as below:

(DB1) TASK
(DB2) MSM_to_TRV
(DB3) REQUIREMENT_COMPONENTS
(DB4) TASK_REQUIREMENT_VECTOR
(DB5) MCM_TO_CVM
(DB6) CVM_TO_TVV
(DB7) VARIABLE_VALUES
(DB8) TASK_VARIABLE_VECTOR
(DB9) CM_TO_SERVICE_COSTS
(DB10) COST_MATRIX

Application functions using Files or DB structures have the comment below, next to their name:
//FileAccess (ODBC)
A DB structure field is referred to as StructureIndex.FieldName e.g. A1.Cost
A record’s specific field is referred to as Record[StructureIndex.FieldName]

B.2 Description of Operation

The costing demonstration was initially perceived to work on a “one-time request” basis. This means that the user requests one specific costing service each time, the request is processed, and for subsequent requests the cycle repeats exactly the same:

- User inputs info
  - task
  - security choice level for this task
  - network mode for this task (actually Administrator inputs this)

- Application processes
  - “loads” task relevant info from storage structures
  - selects current TRV, TVV and CM according to mode and choice selections
  - plugs TVV values and to cost formulas pointed by CM
  - fills results in CM

- Application displays results in layout selected by user

With this approach only one task’s info is cached into program’s memory, and it’s deleted each time a request for a different task is issued, so that the new info is loaded from the storage structures.
Variations on this concept of operation can exist:
For example the user could select only the specific task and then the application could calculate and display security costs for all possible combinations of choice and mode.

B.3 Interfaces

⇒ Initialization of files (Administrative Interface)

Functionality will be supplied to create new files, to add entries to existing files, but not to modify existing info at this stage.
This is because modification of “high-level” structures can be a complicated matter. Modification of a task’s MSM for example could mean various things:
- change 1, 2, or all 3 TRVs the MSM is pointing to, where change a TRV could mean
- make MSM point to a different but existing TRV
- make MSM point to a different TRV, and create TRV
- let MSM point to the same TRV and change the ReqComponents of the TRV

Initialization of files should be executed at least once (or implicitly invoked if files do not exist).

⇒ Input (User Interface)

User selects from existing lists the task id, network mode, and security choice level for the costing request.
Only an administrator can determine the network mode.
- An extra function should exist for data input, if they are needed for cost calculations.
- An extra function later for user defining specific component variable values (not abstract security choices)

⇒ Costing Request (User Interface)

User issues a costing request. This translates to:
- ensuring that task relevant info exist in program memory (retrieve from storage structures if needed)
- selecting TRV, TVV, CM for current mode, choice
- plugging values from TVV to compVariable[]
- calculating costs, by calling appropriate cost formulas

⇒ Display Cost Results (User Interface)

This could be a separate request or invoked along with the costing request, when user wishes other than the default display option.
The user can select to view
- all costs for all services
-costs for all services for a specific resource
-costs for all services for a specific resource and a specific cost type
-all costs for one service
-costs for a specific resource of a service
-costs for a specific resource and a specific cost type of a service

Display Various Info (User Interface)

User selects to display info for
-current CM formulas
-current TRV requirement components
-current TVV variable values

B.4 Cost Formulas and Component Variables

An explanation for the cost formulas and component variable values as visualized in the current approach should be given:
Storage of cost formulas (and generally mathematical expressions) in files or database tables could not be conceived in a straightforward and efficient way. It was thus chosen to store instead indexes to cost formulas that should be used for a specific task’s service resource cost. The various cost formulas are program functions. They are called through a function \( \text{costDispatcher()} \), which calls the appropriate cost formula based on the index (stored / loaded in memory).

Let’s assume that we have the “data integrity on the wire” security service of a task. The cost Formula for the start cost of the resource CPU for this service depends on the symmetric key length (this is the component variable used) and is something like:

\[(5000 + 10 \times \text{key_length}) \text{ clocks.}\]

When TVVs are stored or retrieved, what is actually stored/retrieved is a pair of (COMPONENT_VARIABLE constant, specific Value_RHS), for example (CV_KEY_LENGTH, 56).
In our application frame we do not define a variable name for each component variable (e.g. key_length) and then associate straight the name with its value. Instead we define an array

\[
\text{real compVariable[total number of COMPONENT VARIABLES]}
\]

The number of elements is equal to the number of all different component variables used in the various TRVs (and TVVs).
We refer to a specific component variable as:

\[
\text{compVariable[COMPONENT_VARIABLE constant]}
\]

So each variable always corresponds to the same array position, e.g. when we refer in our application to the component variable key_length, we use

\[
\text{compVariable[CV_KEY_LENGTH]}
\]

So, the cost formula above will be expressed in our program as:
\[
\text{real costFormula50} \{
\]
real x;
result = 5000 + 10 * compVariable[CV_KEY_LENGTH];
return x;
}

When a specific TVV has been selected and pairs like
CV_KEY_LENGTH, 56
have been “loaded”, then with function TaskVariableVector::plugValues, we do
something like:

\[
\text{compVariable}[\text{CV_KEY_LENGTH}] = 56;
\]

For the current task, not all component variables participate in its cost formulas (that is, not all components get a value from the specific TVV). So for each update of Task Variable Vector and before plugging new values to the \text{compVariable}[i], we should first null the array:

\[
\text{for i=0 to number of COMPONENT VARIABLES}
\]
\[
\quad \text{compVariable}[i] = \text{NULL}
\]

A function like the one below will be used, to invoke the appropriate cost formula, according to cost function id fields in info loaded from (FILE7.X) “CM***.dat” or (DB10) COST_MATRIX:

\[
\text{costDispatcher(function id, *result)}
\]
\[
\{ \quad \text{case function id of}
\]
\[
\quad 5:
\]
\[
\quad \text{*result} = \text{costFormula5}();
\]

As already mentioned, these formulas are expressed in terms of \text{compVariable}[i] array elements.

### C. APPLICATION CONSTANTS

**TASK constants**

<table>
<thead>
<tr>
<th>Define</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_FTP</td>
<td>0</td>
</tr>
<tr>
<td>T_WEB_BROWSER</td>
<td>1</td>
</tr>
<tr>
<td>T_UNDEFINED_1</td>
<td>2</td>
</tr>
<tr>
<td>T_UNDEFINED_2</td>
<td>3</td>
</tr>
<tr>
<td>T_UNDEFINED_3</td>
<td>4</td>
</tr>
</tbody>
</table>

//String description of tasks

**SERVICE constants**

<table>
<thead>
<tr>
<th>Define</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_CONFIDENTIALITY_NW</td>
<td>0</td>
</tr>
<tr>
<td>S_CONFIDENTIALITY_ES</td>
<td>1</td>
</tr>
<tr>
<td>S_INTEGRITY_NW</td>
<td>2</td>
</tr>
</tbody>
</table>

LOW LEVEL SPECIFICATION of QoS Costing Demonstration for MSHN 0.3
// String description of security services
const CString s_Service[6] = {"CONFIDENTIALITY_NetworkWire",
    "CONFIDENTIALITY_EndSystem", "INTEGRITY_NetworkWire",
    "INTEGRITY_EndSystem", "AUTHENTICITY_EndSystem",
    "AUDIT_TotalSubnet"};

SECURITY_CHOICE constants
#define CH_LOW 0
#define CH_MEDIUM 1
#define CH_HIGH 2
// String description of security level choices
const CString s_Choice[3] = {"LOW", "MEDIUM", "HIGH"};

NETWORK_MODE constants
#define M_NORMAL 0
#define M_IMPACTED 1
#define M_EMERGENCY 2
// String description of network modes of operation
const CString s_Mode[3] = {"NORMAL", "IMPACTED", "EMERGENCY"};

COMPONENT VARIABLE constants
#define CV_INTEGR_RATE 0
#define CV_SYM_KEY_LENGTH 1
#define CV_ACCESS 2
#define CV_ALGORITHM 3
#define CV_PUB_KEY_LENGTH 4
// String description of component variables
const CString s_CompVariable[5] = {"PACKET_INTEGRITY_RATE",
    "SYMMETRIC_KEY_LENGTH", "CLIENTAUTHORIZED_ACCESS",
    "SYMMETRICALGORITHM", "SERVERAUTHENTICATIONKEY_LENGTH"};

RESOURCE constants
#define R_CPU 0
#define R_MEMORY 1
#define R_BANDWIDTH 2
// String description of resources
const CString s_Resource[3] = {"CPU_A", "MEMORY", "BANDWIDTH"};
// Each resource's string description of start-up cost unit
const CString s_StartUnit[3] = {"clocks", "bytes", "bytes"};
// Each resource's string description of streaming cost unit
const CString s_StreamUnit[3] = {"clocks/packet", "bytes", "bytes/packet"};
// String description of formulas
const CString s_Formula[22] =

LOW LEVEL SPECIFICATION of QoSS Costing Demonstration for MSHN 0.3
"0",
"5000 + 10 x SYMMETRIC_KEY_LENGTH",
"40 x PACKET_INTEGRITY_RATE",
"6144 + SYMMETRIC_KEY_LENGTH",
"5120 + SYMMETRIC_KEY_LENGTH",
"8 x PACKET_INTEGRITY_RATE",
"200 x CLIENT_AUTHORIZED_ACCESS + 1000",
"2048 x CLIENT_AUTHORIZED_ACCESS + 67584",
"100",
"SYMMETRIC_ALGORITHM x (30000 + 100 x SYMMETRIC_KEY_LENGTH)",
"SYMMETRIC_ALGORITHM x (512 + 8 x SYMMETRIC_KEY_LENGTH)",
"SYMMETRIC_ALGORITHM x (8500 + 100 x SYMMETRIC_KEY_LENGTH)",
"SYMMETRIC_ALGORITHM x (6500 + 100 x SYMMETRIC_KEY_LENGTH)",
"SYMMETRIC_ALGORITHM x 2",
"SYMMETRIC_ALGORITHM x (5000 + 10 x SYMMETRIC_KEY_LENGTH)",
"SYMMETRIC_ALGORITHM x (40 x PACKET_INTEGRITY_RATE)",
"SYMMETRIC_ALGORITHM x (6144 + SYMMETRIC_KEY_LENGTH)",
"SYMMETRIC_ALGORITHM x (5120 + SYMMETRIC_KEY_LENGTH)",
"SYMMETRIC_ALGORITHM x (8 x PACKET_INTEGRITY_RATE)",
"200 x CLIENT_AUTHORIZED_ACCESS + 4000 + 10 x
SERVER_AUTHENTICATION_KEY_LENGTH",
"2048 x CLIENT_AUTHORIZED_ACCESS + 77584 + 5 x
SERVER_AUTHENTICATION_KEY_LENGTH",
"612 + SERVER_AUTHENTICATION_KEY_LENGTH" }

Application constants
const int MAX_TASK = 5; //Maximum number of tasks in demo
const int MAX_REQ_COMP = 5; //Maximum number of Requirement Components in a task
const int MAX_SERV = 3; //Maximum number of Services in a task

We use these values, in order to keep arrays with costing info (that need initializing...) to an easily manageable size for this version.

D. FILE STRUCTURES

(FILE1) "Task.dat"
Structure containing for each task corresponding indexes to Mode-Service, Mode-Choice and Cost Matrices.
Number of entries equals number of defined tasks.

Corresponding DB Structure: (DB1) TASK

<table>
<thead>
<tr>
<th>FIELD</th>
<th>DESCRIPTION</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>integer</td>
<td>0 (T_FTP)</td>
</tr>
</tbody>
</table>

LOW LEVEL SPECIFICATION of QoSS Costing Demonstration for MSHN 0.3
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>Normal_TRV</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>Impacted_TRV</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>Emergency_TRV</td>
<td>integer</td>
<td></td>
</tr>
</tbody>
</table>

**FIELD DESCRIPTION EXAMPLE**

- **ID**
  - Indicates Mode-Service Matrix's id

- **Normal_TRV**
  - Indicates id of Task Requirement Vector for normal mode

- **Impacted_TRV**
  - Indicates id of Task Requirement Vector for impacted mode

- **Emergency_TRV**
  - Indicates id of Task Requirement Vector for emergency mode

**NOTE:** for a different costing algorithm another field could be added, for an alternative cost matrix.

**FILE2** “MSMtoTRV.dat”

(MODE-SERVICE MATRIX to TASK REQUIREMENT VECTOR)

Structure containing for each Mode-Service Matrix indexes to Task Requirement Vectors according to network mode.

Number of entries equals number of defined tasks.

**Corresponding DB Structure:** (DB2) MSM_TO_TRV

**FILE3.1 – FILE3.X** “TRV***.dat”

A set of files with filename TRV***.dat, where *** is the id of TRV:

- TRV1.dat, ..., TRVx.dat

Number of files equals number of all possible Task Requirement Vectors

(x = number_of_Tasks * number_of_modes)
Number of entries in each file equals $\text{nrComp}_i = \text{number of requirement components of TRV } i$ (which is a matter of definition)

**Corresponding DB Structure**: specific (DB4) **TASK_REQUIREMENT_VECTOR**

Because if this set of files, a file corresponding to (DB3) **REQUIREMENT_COMPONENTS** is not needed.

$\text{nrComp}_i$ is the first info in file. The rest of the entries are as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>Variable_LHS</td>
<td>integer a COMPONENT_VARIABLE</td>
<td>1 (CV_SYM_KEY_LENGTH)</td>
</tr>
<tr>
<td></td>
<td>constant – indicates variable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>clause of component</td>
<td></td>
</tr>
<tr>
<td>Min_Range_Value</td>
<td>float a number</td>
<td>.6</td>
</tr>
<tr>
<td>Max_Range_Value</td>
<td>float a number</td>
<td>.8</td>
</tr>
</tbody>
</table>

NOTE: Additional fields e.g. for instantiated values can be included

(File4) “MCMtoCVM .dat” (MODE-CHOICE MATRIX to CHOICE VARIABLE MATRIX)

Structure containing for each Mode-Choice Matrix indexes to Choice Variable Matrixes according to network mode.

Number of entries equals number of defined tasks.

**Corresponding DB Structure**: (DB5) **MCM TO CVM**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>Normal_CVM</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>Impacted_CVM</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>Emergency_CVM</td>
<td>integer</td>
<td></td>
</tr>
</tbody>
</table>

LOW LEVEL SPECIFICATION of QoSS Costing Demonstration for MSHN 0.3
(FILE5) “CVMtoTVV.dat”
(CHOICE-VARIABLE MATRIX to TASK VARIABLE VECTOR)
Structure containing for each Choice Variable Matrix indexes to Task Variable Vectors according to security level choice.
Number of entries equals number_of_tasks * number_of_modes.

Corresponding DB Structure: (DB6) CVM_TO_TVV

<table>
<thead>
<tr>
<th>FIELD</th>
<th>DESCRIPTION</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>integer</td>
<td>indicates Choice-Variable Matrix’s id</td>
</tr>
<tr>
<td>Low_TVV</td>
<td>integer</td>
<td>indicates id of Task Variable Vector for low level security choice</td>
</tr>
<tr>
<td>Medium_TVV</td>
<td>integer</td>
<td>indicates id of Task Variable Vector for medium level security choice</td>
</tr>
<tr>
<td>High_TVV</td>
<td>integer</td>
<td>indicates id of Task Variable Vector for high level security choice</td>
</tr>
</tbody>
</table>

A set of files with filename TVV***.dat, where *** is the id of TVV:
TVV1.dat, ..., TVVy.dat
Number of files equals number of all possible Task Variable Vectors
(y = number_of_Tasks * number_of_modes * number_of_security_choices)
Number of entries equals to nrComp_i = number of requirement components of corresponding Task’s TRV i (which is a matter of definition)

Corresponding DB Structure: specific (DB8) TASK_VARIABLE_VECTOR
Because if this set of files, a file corresponding to (DB7) VARIABLE_VALUES is not needed.

nrComp_i       is the first info in file. The rest of the entries are as follows:

<table>
<thead>
<tr>
<th>FIELD</th>
<th>DESCRIPTION</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable_LHS</td>
<td>integer</td>
<td>a COMPONENT_VARIABLE constant – indicates variable clause of component</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 (CV_KEY_LENGTH)</td>
</tr>
<tr>
<td>Min_Value_RHS</td>
<td>float</td>
<td>number indicating minimum of acceptable value range</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.6</td>
</tr>
<tr>
<td>Max_Value_RHS</td>
<td>float</td>
<td>number indicating maximum of acceptable value range</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.7</td>
</tr>
</tbody>
</table>
A set of files with filename CM***.dat, where *** is the id of CM:
CM1.dat, ..., CMz.dat
Number of files equals to number of defined Tasks
Number of entries equals nrServicesi = number of services of Task i (which is a matter of definition)

**Corresponding DB Structure:** specific (DB10) COST_MATRIX
Because if this set of files, a file corresponding to (DB9) CM_TO_SERVICE_COSTS is not needed.

nrServicesi is the first info in file. The rest of the entries are as follows:

<table>
<thead>
<tr>
<th>FIELD</th>
<th>DESCRIPTION</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
<td>integer</td>
<td>3 (S_INTEGRITY_NW)</td>
</tr>
<tr>
<td></td>
<td>a SERVICE constant - indicates id of service</td>
<td></td>
</tr>
<tr>
<td>CPU_Start_Cost</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>indicates id of function used to calculate start cost for CPU.</td>
<td></td>
</tr>
<tr>
<td>CPU_Stream_Cost</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>indicates id of function used to calculate streaming cost for CPU.</td>
<td></td>
</tr>
<tr>
<td>memory_Start_Cost</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>indicates id of function used to calculate start cost for memory.</td>
<td></td>
</tr>
<tr>
<td>memory_Stream_Cost</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>indicates id of function used to calculate streaming cost for memory.</td>
<td></td>
</tr>
<tr>
<td>bandwidth_Start_Cost</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>indicates id of function used to calculate start cost for bandwidth.</td>
<td></td>
</tr>
<tr>
<td>bandwidth_Stream_Cost</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>indicates id of function used to calculate streaming cost for bandwidth.</td>
<td></td>
</tr>
</tbody>
</table>
E. SPECIFICATION OF ENTITIES

E.1 The Application Frame

The SecurityCosts application is implemented in Microsoft Visual C++ Version 6.0, using the Microsoft Foundation Class (MFC) Library. We could summarize the functionality of the application frame as follows: there is a loop running continuously and checking the event messages. While the message is not the "quit" message, the application frame sends the events to the objects that have the appropriate event handlers. A simplified example of the respective code is:

```cpp
getEvent(eventID);
while (eventID != QUIT)
{
    switch(eventID)
    {
        case MENU_USER_SELECT_TASK:
            CSecurityCostsDoc.OnUserTask0;
        ...
        case MENU_USER_SECURITY_LEVEL:
            CSecurityCostsDoc.OnUserSecurityLevel0;
        ...
        case MENU_ADMINISTRATOR_SETUPTASK:
            CSecurityCostsDoc.OnAdministratorSetupTask0;
        ...
        case MENU_USER_SECURITY_REQUIREMENTS_INFO:
            CSecurityCostsView.OnUserDisplayTRVsInfo0;
        ...
        case MENU_USER_SECURITY_CHOICES_INFO:
            CSecurityCostsView.OnUserDisplayTVVsInfo0;
        ...
        case MENU_USER_COST_FORMULAS:
            CSecurityCostsView.OnUserDisplayCostInfo0;
        ...
        case MENU_USER_DISPLAY_COST_RESULTS:
            CSecurityCostsView.OnUserDisplayCostResults0;
    }
    getEvent(eventID);
}
```

The CSecurityCostsDoc object handles events generated by the user’s menu requests for:
- selection of task (respective handler: OnUserTask0)
- selection of network mode (respective handler: OnAdministratorNetworkMode0)
- selection of security level (respective handler: OnUserSecurityLevel0)
- processing of costs (respective handler: OnUserProcessCosts0)
- set-up of a new task (respective handler: OnAdministratorSetupTask0)
The \textit{CSecurityCostsView} object handles events generated by the user's menu requests for display of:

- TRVs info (respective handler: \texttt{OnUserDisplayTRVsInfo()})
- TVVs info (respective handler: \texttt{OnUserDisplayTVVsInfo()})
- cost formulas (respective handler: \texttt{OnUserDisplayCostInfo()})
- cost results (respective handler: \texttt{OnUserDisplayCostResults()})

\section*{E.2 Object entities}

The \textit{CSecurityCostsView} object handling display of costing info and results is not described here, since it only involves the way that data are presented (which may change).

Interface dialog boxes are also not described in this document for similar reasons.

For the objects specified in this paragraph, description of constructors and destructors is included only when they perform a special action.

\textit{Get} functions for private members of objects are not described in this document, since their effect is trivial.

Objects and functions generated automatically by or related specifically to the MFC framework are also not described.

\begin{verbatim}
CSecurityCostsDoc

CSecurityCostsDoc::CSecurityCostsDoc //definition of entity
{
//MEMBERS
    Task    testTask;
    int     curTaskID;    //a TASK constant
    int     curModeID;    //a NETWORK_MODE constant
    int     curChoiceID;  //a SECURITY_CHOICE constant
    Task*   curTask;
    //ModeServiceMtrx*    curMSM;
    //TaskReqVector*      curTRV;
    ModeChoiceMtrx*      curMCM;
    ChoiceVariableMtrx*  curCVM;
    TaskVariableVector*  curTVV;
    CostMtrx*            curCM;
    float               compVariable[20];

//FUNCTIONS
    OnUserTask();
    OnAdministratorNetworkMode();
    OnUserSecurityLevel();
    OnUserProcessCosts();
    OnAdministratorSetupTask()
    selectTaskID();
    selectModelID();

CSecurityCostsDoc: CSecurityCostsDoc

CSecurityCostsDoc::CSecurityCostsDoc //definition of entity
{
//MEMBERS
    Task    testTask;
    int     curTaskID;    //a TASK constant
    int     curModeID;    //a NETWORK_MODE constant
    int     curChoiceID;  //a SECURITY_CHOICE constant
    Task*   curTask;
    //ModeServiceMtrx*    curMSM;
    //TaskReqVector*      curTRV;
    ModeChoiceMtrx*      curMCM;
    ChoiceVariableMtrx*  curCVM;
    TaskVariableVector*  curTVV;
    CostMtrx*            curCM;
    float               compVariable[20];

//FUNCTIONS
    OnUserTask();
    OnAdministratorNetworkMode();
    OnUserSecurityLevel();
    OnUserProcessCosts();
    OnAdministratorSetupTask()
    selectTaskID();
    selectModelID();

LOW LEVEL SPECIFICATION of \textit{QoSS Costing Demonstration for MSHN} 0.3

16
\end{verbatim}
selectChoiceID();
selectCurrentMatricesVectors();
costFormula0();
costFormula1();
...
costFormula21();
costDispatcher(int formula, float* result);
plugValues();
nullCompVariableArray();
calculateCosts();
}

CSecurityCostsDoc::OnUserTask()
- calls selectTaskID()
- calls testTask initialize(curTaskID)
- calls selectCurrentMatricesVectors()

CSecurityCostsDoc::OnAdministratorNetworkMode()
- calls selectModeID()
- calls selectCurrentMatricesVectors()

CSecurityCostsDoc::OnUserSecurityLevel()
- calls selectChoiceID()
- calls selectCurrentMatricesVectors()

CSecurityCostsDoc::OnUserProcessCosts()
- calls calculateCosts()

CSecurityCostsDoc:: OnAdministratorSetupTask()
In current version this function displays various dialog boxes, inputs task info from user and stores it in arrays
(dbTask[], dbMSMtoTRV[], dbTRV[], dbMCMtoCVM[], dbCVMtoTVV[],
dbTVV[], dbCM[])
Later task info will be stored in files/DB tables

CSecurityCostsDoc::selectTaskID()
- calls appropriate interface for input of task id value
- sets curTaskID to input value

CSecurityCostsDoc::selectModeID()
- calls appropriate interface for input of mode id value
- sets curModeID to input value

CSecurityCostsDoc::selectChoiceID()
- calls appropriate interface for input of choice id value
- sets curChoiceID to input value
CSecurityCostsDoc::selectCurrentMatricesVectors()

- sets \texttt{curTask} = \&\texttt{testTask}
- sets \texttt{curMCM} = \texttt{curTask->getTaskMCM}();
- sets \texttt{curCVM} = \texttt{curMCM->getModeCVM(curModeID)};
- sets \texttt{curTVV} = \texttt{curCVM->getChoiceTVV(curChoiceID)};
- sets \texttt{curCM} = \texttt{curTask->getTaskCM}();

CSecurityCostsDoc::calculateCosts()

- calls \texttt{nulCompVariableArray}()
- calls \texttt{plugValues}()
- for all services of \texttt{curCM}
  - for all resources
    - calls \texttt{costDispatcher} with id of \texttt{startupFunction} and the result put in \texttt{startupCost}
    - calls \texttt{costDispatcher} with id of \texttt{streamFunction} and the result put in \texttt{streamCost}

CSecurityCostsDoc::costDispatcher(int formula, float* result)

- based on a case \texttt{formula} of
  \texttt{XXX}:
  - calls appropriate \texttt{costFormulaXXX}() and sets \texttt{*result} to return value

SET of CSecurityCostsDoc::costFormulaXXX()

- calculates and return specific cost expression, using values of certain elements of \texttt{compVariable[]}. e.g.

CSecurityCostsDoc::costFormula0()

- returns value 0.

...

CSecurityCostsDoc::costFormula21()

- if \texttt{compVariable[CV_ACCESS]} equals 0
  - returns value 0.
else
  - returns value of expression  \(612 + \texttt{compVariable[CV_PUB_KEY_LENGTH]}\)

CSecurityCostsDoc::nullCompVariableArray()

- for i=0 to \texttt{total number of COMPONENT VARIABLES}
  - sets \texttt{compVariable[i]} to 0

CSecurityCostsDoc::plugValues ()

- \texttt{compVariable[]} elements whose corresponding variable is included in \texttt{curTVV}, get the corresponding value with the loop:
  for i=0 to \texttt{curTVV->getTVV_SIZE()}
    \texttt{compVariable[curTVV->getVarEntry(i)->getId]} = \texttt{curTVV->getVarEntry(i).getMinValue()}
Task

Task

//definition of entity

//MEMBERS
int id;
int nrComponents;
int nrServices;
ModeServiceMtrx taskMSM;
ModeChoiceMtrx taskMCM;
CostMtrx* taskCM;

//FUNCTIONS
initialize(int inp1);
setId(int input);
setNrComponents(int inp1);
setNrServices(int inp1);
setModeServiceMatrix(int input);
setModeChoiceMatrix(int input);
setCostMatrix(int input);
dbGetEntryTask(int* 1_nrComp, int* 1_nrServ, int* 1_msm, int* 1_mcm,
                  int* 1_cm);
dbGetSpecificCM(int input, *par);

Task::initialize(int inp1)
-defines local variables
    int locNrComponents, locNrServices
    int mtrxMS, mtrxMC, mtrxCM
-calls setId(inp1)
-calls dbGetEntryTask(&locNrComponents, &locNrServices, &mtrxMS, &mtrxMC,
                        &mtrxCM)
-calls setNrComponents(locNrComponents)
-calls setNrServices(locNrServices)
-calls setModeServiceMatrix(mtrxMS)
-calls setModeChoiceMatrix(mtrxMC)
-calls setCostMatrix(mtrxCM)

Task::setId(int input)
-sets id to input

Task::setNrComponents(int inp1)
-sets nrComponents to inp1

Task::setNrServices(int inp1)
-sets nrServices to inp1
### Task::setModeServiceMatrix(int input)
- calls `taskMSM.setId(input)`
- calls `taskMSM.setTRVs(nrComponents)`

### Task::setModeChoiceMatrix(int input)
- calls `taskMCM.setId(input)`
- calls `taskMCM.setCVMs(nrComponents)`

### Task::setCostMatrix(int input)
- creates dynamically a new `CostMtrx` with `nrServices` services
- assigns it to `taskCM`
- calls `taskCM->setId(input)`
- calls `taskCM->setServices()`

### Task::dbGetEntryTask(int* l_nrComp, int* l_nrServ, int* l_msm, int* l_mcm, int* l_cm)

**FILE case**
- (FILE1) "Task.dat"  

**DB case**
- selects record `recX` in (DB1) TASK for which `A1.ID = id`
- sets `*l_msm = recX[A1.Mode-Service]`
- sets `*l_mcm = recX[A1.Mode-Choice]`
- sets `*l_cm = recX[A1.Cost]`
- sets `*l_nrComp = recX[A1.NrComponents]`
- sets `*l_nrServ = recX[A1.NrServices]`

**NOTE:** In current version this function accesses elements of array `dbTask[][]`

### Task::dbGetSpecificCM(int input, *par)

**FILE case**
- gets specific (FILE7.1 – FILE7.Z) "CM*.*.dat" filename by appending to "CM"+string(input)+ "_.dat"
  (e.g. for input = 2, filename is “CM2.dat”)
- puts info for filename in `par`

**DB case**
- invokes DB operation for generation of (DB10) COST_MATRIX, from (DB9) CM_TO_SERVICE_COSTS with `A9.CM=Id`
- puts info for specific structure addressing in `par`

**NOTE:** In current version this function performs no action
ModeServiceMtrx

ModeServiceMtrx:: ModeServiceMtrx

//definition of entity
{
//MEMBERS
    int id; //id number of MSM
    TaskReqVector* modeTRV[3]; //array of pointers to TRVs for each mode

//FUNCTIONS
    ~ModeServiceMtrx() //destructor
    setId(input);
    setTRVs(int nrReqComp);
    setModeTRV(int idTRV, int index, int nrEntries);
    dbGetEntryMSMtoTRV(int* _normID, int* _impID, int* _emID);
    dbGetSpecificTRV(int input, *par);
}

ModeServiceMtrx:: ~ModeServiceMtrx()

- for i=0 to 3
    - frees memory reserved by modeTRV[i]

ModeServiceMtrx::setId(input)

- sets id to input

ModeServiceMtrx::setTRVs(int nrReqComp)

- defines local variables
    int normTRV, impTRV, emTRV
- calls dbGetEntryMSMtoTRV(&normTRV, &impTRV, &emTRV)
- calls setModeTRV(normTRV, M_NORMAL, nrReqComp)
- calls setModeTRV(impTRV, M_IMPACTED, nrReqComp)
- calls setModeTRV(emTRV, M_EMERGENCY, nrReqComp)

ModeServiceMtrx::setModeTRV(int idTRV, int index, int nrEntries)

- calls dbGetSpecificTRV(idTRV)
- creates dynamically a new TaskReqVector with nrEntries requirement components
- assigns it to modeTRV[index]
- calls modeTRV[index].setId(idTRV)
- calls modeTRV[index].setRequirementComponents()

ModeServiceMtrx::dbGetEntryMSMtoTRV(int* _normID, int* _impID,
                                        int* _emID)  // FileAccess (ODBC)

FILE case
(FILE2) "MSMtoTRV.dat"

DB case

- selects record recX in (DB2) MSM_TO_TRV for which A2.ID = id
- sets *l_normID = recX[A1.Normal/TRV]
-sets *l_implID = recX[A1. Impacted_TRV]
-sets *l_emID = recX[A1. Emergency_TRV]

NOTE: In current version this function accesses elements of array dbMSMtoTRV[][]

**TaskReqVector::dbGetSpecificTRV(int input, *par)**  //FileAccess (ODBC)

**FILE case**

-gets specific (FILE3.1 – FILE3.X) “TRV***.dat” filename by appending to “TRV”+string(input)+ “.dat”
(e.g. for input = 30, filename is “TRV30.dat”)  
-puts info for filename in par

**DB case**

-invokes DB operation for generation of (DB4) TASK_REQUIREMENT_VECTOR, from (DB3) REQUIREMENT_COMPONENTS with A3.TRV=id  
-somehow puts info for specific structure addressing in par

NOTE: In current version this function performs no action

**TaskReqVector**

**TaskReqVector::TaskReqVector**  //definition of entity

{ //MEMBERS
  int id;  //id number of TRV
  const int TRV_SIZE;  //number of components in req_comp[]
  ReqComponent** req_comp;  //a pointer to an array of TRV_SIZE
                          //pointers to ReqComponent components

  //FUNCTIONS
  TaskReqVector (int size);  //constructor
  ~TaskReqVector ();  //destructor
  setId(int input);
  setRequirementComponents();
  dbGetNextEntryTRV(input, int* idReqComp, int* idCompVar,
                   float* minVal, float* maxVal)
}

**TaskReqVector::TaskReqVector (int size)**

-sets TRV_SIZE to size

-for i=0 to TRV_SIZE
   -dynamically creates a new ReqComponent
   -assigns it to req_comp[i]
TaskReqVector::~TaskReqVector()

- for i = 0 to TRV_SIZE
  - frees memory reserved by req_comp[i]

TaskReqVector::setId(int input)

- sets id to input

TaskReqVector::setRequirementComponents()

- defines local variables
  int reqcomp_id, compvar_id
  int compvar_min, compvar_max

- for i = 0 to TRV_SIZE
  - calls dbGetNextEntryTRV(i, &reqcomp_id, &compvar_id, &compvar_min, &compvar_max)
  - calls req_comp[i]->setId(reqcomp_id)
  - calls req_comp[i]->setVariable(compvar_id)
  - calls req_comp[i]->setExpression(compvar_min, compvar_max)

TaskReqVector::dbGetNextEntryTRV(input, int* idReqComp, int* idCompVar,
float* minVal, float* maxVal)

// FileAccess (ODBC)

FILE case
(input needed for keeping track of last file position) (FILE3.X) “TRV***.dat”

DB case
(input needed for keeping track of last record)
-selects next record recX in structure (DB4) TASK_REQUIREMENT_VECTOR
- sets *idReqComp = recX[A4.ID]
- sets *idCompVar = recX[A4.Value_LHS]
- sets *minVal = recX[A4.Min_Range_Value]
- sets *maxVal = recX[A4.Max_Range_Value]

NOTE: In current version this function accesses elements of array dbTRV[][][]

ReqComponent

ReqComponent::ReqComponent // definition of entity

{ // MEMBERS
  int id; // id number of requirement component
  int comp_variable; // id of component variable
  float min_value; // minimum acceptable value for variable
  float max_value; // maximum acceptable value for variable
}

// FUNCTIONS

LOW LEVEL SPECIFICATION of QoS Costing Demonstration for MSHN 0.3
setld(int input);
setVariable(int input);
setExpression(input1, input2, ...);

ReqComponent::setld(int input)
-set id to input

ReqComponent::setVariable(int input)
-sets comp_variable to input

ReqComponent::setExpression(float input1, float input2)
-sets min_value, max_value to input1, input2 respectively

ModeChoiceMtrx

ModeChoiceMtrx::ModeChoiceMtrx //definition of entity
{/MEMBERS
  int id;
  ChoiceVariableMtrx modeCVM[3];

  //FUNCTIONS
  setId(int input);
  setCVMs(int nrCompVar);
  setModeCVM(int idCVM, int index, int nrEntries);
  dbGetEntryMCMtoCVM(int* normID, int* implID, int* emID);
}

ModeChoiceMtrx::setId(int input)
-set id to input

ModeChoiceMtrx::setCVMs(int nrCompVar)
-defines local variables
- int normCVM, impCVM, emergCVM
-calls dbGetEntryMCMtoCVM(&normCVM, &impCVM, &emergCVM)
-calls setModeCVM(normCVM, M_NORMAL, nrCompVar)
-calls setModeCVM (impCVM, M_IMPACTED, nrCompVar)
-calls setModeCVM (emergCVM, M_EMERGENCY, nrCompVar)

ModeChoiceMtrx::setModeCVM(int idCVM, int index, int nrEntries)
-calls modeCVM[index].setId(idCVM)
-calls modeCVM[index].setTVVs(nrEntries)

ModeChoiceMtrx::dbGetEntryMCMtoCVM(int* normID, int* implID, int* emID)
//FileAccess (ODBC)

FILE case
DB case

-selects record recX in (DB5) MCM_TO_CVM for which A5.ID = input
-sets *normID = recX[A5.Normal_CVM]
-sets *impID = recX[A5.Impacted_CVM]
-sets *emID = recX[A5.Emergency_CVM]

NOTE: In current version this function accesses elements of array dbMCMtoCVM[]

ChoiceVariableMtrx:

ChoiceVariableMtrx::ChoiceVariableMtrx

//definition of entity

{ //MEMBERS
  int id;
  int nrCompVariables;
  TaskVariableVector* choiceTVV[3];

  //FUNCTIONS
  ~ChoiceVariableMtrx();
  setId(int input);
  setTVVs(int nrVar);
  setChoiceTVV(int input1, int input2);
  dbGetEntryCVMtoTVV(int input, int* parl, int* par2, int* par3);
  dbGetSpecificTVV(int input, *par);
}

ChoiceVariableMtrx:: ~ ChoiceVariableMtrx()

-for i = 0 to 3
  -frees memory reserved by choiceTVV[i]

ChoiceVariableMtrx::setId(int input)

-sets id to input

ChoiceVariableMtrx::setTVVs(int nrVar)

-defines local variables
  int lowTVV, medTVV, highTVV
-calls dbGetEntryCVMtoTVV(&lowTVV, &medTVV, &highTVV)
-calls setChoiceTVV(lowTVV, CH_LOW, nrVar)
-calls setChoiceTVV(medTVV, CH_MEDIUM, nrVar)
-calls setChoiceTVV(highTVV, CH_HIGH, nrVar)

ChoiceVariableMtrx::setChoiceTVV(int idTVV, int index, int nrEntries)

-calls dbGetSpecificTVV(idTVV);
-creates dynamically a new TaskVariableVector with nrEntries component variables
-assigns it to \textit{choiceTVV[index]}
-calls \textit{choiceTVV[index].setId(input2)}
-calls \textit{choiceTVV[index].setVariables()}

\begin{verbatim}
ChoiceVariableMtrx::dbGetEntryCVMtoTVV(int* lowID, int* medID, int* highID) //FileAccess (ODBC)

FILE case
(FILE5) "CVMtoTVV.dat"

DB case

-selects record \textit{recX} in (DB6) CVM\_TO\_TVV, for which \textit{A6.ID} = \textit{id}
-sets \*\textit{lowID} = \textit{recX[A6.Low\_TVV]}
-sets \*\textit{medID} = \textit{recX[A6.Medium\_TVV]}
-sets \*\textit{highID} = \textit{recX[A6.High\_TVV]}

\end{verbatim}

NOTE: In current version this function accesses elements of array \textit{dbCVMtoTVV[][]}

\begin{verbatim}
ChoiceVariableMtrx:: dbGetSpecificTVV(int input, *par) //FileAccess (ODBC)

FILE case

-gets specific (FILE6.1 – FILE6.Y) "TVV***.dat" filename by appending to
 "TVV"+string(input)+ "\_dat" (e.g. for input = 45, filename is "TVV45.dat")
-puts info for filename in \textit{par}

DB case

-invokes DB operation for generation of (DB8) TASK\_VARIABLE\_VECTOR, from (DB7)
VARIABLE\_VALUES with \textit{A7.TVV=id}
-puts info for specific structure addressing in \textit{par}

\end{verbatim}

NOTE: In current version this function performs no action

\textbf{TaskVariableVector}

\begin{verbatim}
TaskVariableVector::TaskVariableVector() //definition of entity
{
//MEMBERS
    int id;
    const int TVV\_SIZE; //number of components in \textit{var_entry[]}
    VariableValue** var_entry; //a pointer to an array of TVV\_SIZE
    //pointers to \textit{VariableValue} components
}

//FUNCTIONS
    TaskVariableVector(int size)
    ~TaskVariableVector()
    setId(int input)
\end{verbatim}

LOW LEVEL SPECIFICATION of \textit{QoSS Costing Demonstration for MSHN 0.3} 26
setVariables()
    dbGetNextEntryTVV(input, int* idTVV, float* minV, float* maxV)
}

TaskVariableVector::TaskVariableVector(int size)
- sets $TVV\_SIZE$ to size
- for $i = 0$ to $TVV\_SIZE$
  - dynamically creates a new VariableValue
  - assigns it to $var\_entry[i]$

TaskVariableVector::~TaskVariableVector()
- for $i = 0$ to $TVV\_SIZE$
  - frees memory reserved by $var\_entry[i]$

TaskVariableVector::setId(int input)
- set $id$ to $input$

TaskVariableVector::setVariables() //FileAccess (ODBC)
- defines local variables
  int $var\_id$
  float $minRHS$, $maxRHS$
- for $i = 0$ to $TVV\_SIZE$
  - calls $dbGetNextEntryTVV(i, &var\_id, &minRHS, &maxRHS)$
  - calls $var\_entry[i]->setId(var\_id)$
  - calls $var\_entry[i]->setMinMaxValue(minRHS)$
  - calls $var\_entry[i]->setMinMaxValue(maxRHS)$

TaskVariableVector::dbGetNextEntryTVV(input, int* idTVV, float* minV, float* maxV) //FileAccess (ODBC)

FILE case
(input needed for keeping track of last file position) (FILE6.Y) “TVV***.dat”

DB case
(input needed for keeping track of last record)
- selects next record $recX$ in structure (DB8) TASK\_VARIABLE\_VECTOR
- sets $*idTVV = recX[A8.Variable\_LHS]$
- sets $*minV = recX[A8.Min\_Value\_RHS]$
- sets $*maxV = recX[A8.Max\_Value\_RHS]$

NOTE: In current version this function accesses elements of array $dbTVV[][][]$

VariableValue

VariableValue::VariableValue //definition of entity
{/MEMBERS

LOW LEVEL SPECIFICATION of QoSS Costing Demonstration for MSHN 0.3
int id;
float min_value;
float max_value;

//FUNCTIONS
setId(int input);
setMinValue(input);
setMaxValue(input);

VariableValue::setId(int input)
- set id to input

VariableValue::setMinValue(input)
- sets min_value to input

VariableValue::setMaxValue(input)
- sets max_value to input

CostMtrx

CostMtrx::CostMtrx //definition of entity
{//MEMBERS
 int id;
 const int CM_SIZE; //number of services in serv[]
 Service** serv[]; //a pointer to an array of CM_SIZE
 //pointers to Service components

 //FUNCTIONS
 setId(int input);
 setServices();
 calculateCosts();
 displayResults();
 displayResults(int resource1);
 displayResults(int resource1, int cost_type);
 dbGetNextEntryCM(int inp1, int* servID, int* cost1, int* cost2);
}

CostMtrx:: CostMtrx (int size)
- sets CM_SIZE to size
- for i=0 to CM_SIZE
  - dynamically creates a new Service
  - assigns it to serv[i]

CostMtrx::~CostMtrx ()
- for i=0 to CM_SIZE
  - frees memory reserved by serv[i]
CostMtrx::setId(int input)
-sets id to input

CostMtrx::setServices()
//FileAccess (ODBC)
-defines local variables
  int serv_id
  int start_cost[3]
  int stream_cost[3],
-for i=0 to CM_SIZE
  -calls dbGetNextEntryCM(i, &serv_id, start_cost, stream_cost)
  -calls serv[i]->setId(serv_id)
  -calls serv[i]->setResources(start_cost, stream_cost)

CostMtrx::dbGetNextEntryCM(int inpl, int* servID, int* cost1, int* cost2)
//FileAccess (ODBC)

FILE case
(input needed for keeping track of last file position) (FILE7.Z) “CM***.dat”

DB case
(input needed for keeping track of last record)
-selects next record recX in structure (DB10) COST_MATRIX
-sets *servID = recX[A10.Service]
-sets cost1[R_CPU] = recX[A10.CPU_Start_Cost]
-sets cost2[R_CPU] = recX[A10.CPU_Stream_Cost]
-sets cost1[R_MEMORY] = recX[A10.memory_Start_Cost]
-sets cost2[R_MEMORY] = recX[A10.memory_Stream_Cost]
-sets cost1[R_Bandwidth] = recX[A10.bandwidth_Start_Cost]
-sets cost2[R_Bandwidth] = recX[A10.bandwidth_Stream_Cost]

NOTE: In current version this function accesses elements of array dbCM[][][]

Service
//definition of entity

Service::Service
{//MEMBERS
  int id;
  Resource resourceCost [3];

  //FUNCTIONS
  setId(int input);
  setResources(int *cost1, int *cost2);
}

Service::setId(int input)
-sets id to input
Service::setResources(int *cost1, int *cost2)

-for i = R_CPU to R_BANDWIDTH
  -calls resourceCost[i].setStartupFunction(cost1[i])
  -calls resourceCost[i].setStreamFunction(cost2[i])

Resource::Resource

{ //definition of entity

  //MEMBERS
  int startupFunction;
  float startupCost;
  int streamFunction;
  float streamCost;

  //FUNCTIONS
  setStartupFunction(input);
  setStreamFunction(input);
}

Resource::setStartupFunction(input)

-sets startupFunction to input

Resource::setStreamFunction(input)

-sets streamFunction to input

F. FUTURE REFINEMENTS

The application should not necessarily be restrained into having only one task’s info cached into program’s memory. Info for a predefined maximum number of tasks (MAX_SIZE) could be kept in program’s memory. This would:

➢ reduce the amount of “interaction” with the storage structures (files or data base)
➢ enable a form of fast “switching” between tasks (which could be used later for satisfying “multiple” requests)

A track of usage statistics can be kept, so info for frequently used tasks is maintained in memory. When there’s a request for a task not present in memory, and maximum number of tasks in memory is already reached, the least used task could be deleted, to make space for loading the new task’s info.

MAX_SIZE can be decided after examining the average size of a task’s info and the memory available for program operation.

This concept is illustrated below with the CSecurityCostsDoc using the array

```
Task* applicationTask[MAX_SIZE]
```

and integer member nrTasks, along with the general functions

```
setupCurrentTask()
```
checkInList()
leastUsedTask()
and the Task entity having additionally
a usage member and
an increaseUsage() member function

CSecurityCostsDoc:: CSecurityCostsDoc //definition of entity
{ //MEMBERS
    Task* applicationTask[MAX_SIZE]; //instead of one task
    int nrTasks; //additional member

    int curTaskID; //a TASK constant
    int curModeID; //a NETWORK_MODE constant
    int curChoiceID; //a SECURITY_CHOICE constant
    Task* curTask;
    //ModeServiceMtrx* curMSM;
    //TaskReqVector* curTRV;
    ModeChoiceMtrx* curMCM;
    ChoiceVariableMtrx* curCVM;
    TaskVariableVector* curTVV;
    CostMtrx* curCM;
    float compVariable[20];

    //FUNCTIONS
    OnUserTask();
    OnAdministratorNetworkMode();
    OnUserSecurityLevel();
    OnUserProcessCosts();
    OnAdministratorSetupTask()
    selectTaskID();
    selectModeID();
    selectChoiceID();
    selectCurrentMatricesVectors();
    costFormula0();
    costFormula1();
    ...
    costFormula21();
    costDispatcher(int formula, float* result);
    plugValues();
    nullCompVariableArray();
    calculateCosts();
}

CSecurityCostsDoc::OnUserTask()  
-calls selectTaskID()  
-calls setupCurrentTask()  //different function call

LOW LEVEL SPECIFICATION of QoSS Costing Demonstration for MSHN 0.3  
31
CSecurityCostsDoc::setupCurrentTask()
- defines local variable
  int k
- if selected task is not the previous current task  (if curTask->id <> curTaskID)
  - if selected task does not exist in applicationTask[]  (if !checkInList(curTaskID, &k))
  - if task list is full (if nrTasks >= MAX_SIZE)
    - calls leastUsedTask() and sets k to its return value
    - calls delete applicationTask[k]
    - decreases nrTasks = nrTasks - 1
  else
    - sets k = nrTasks
  - creates dynamically applicationTask[k] = new Task
  - increases nrTasks = nrTasks + 1
  - calls applicationTask[k]->initialize(curTaskID)
    - sets curTask = applicationTask[k]
- calls curTask->increaseUsage()

CSecurityCostsDoc::checkInList(int inp, int* index)
- searches elements of applicationTask[], for a task with applicationTask[]->id equal to inp.
  If it finds it, it puts the element position in the array in *index and returns TRUE, otherwise it returns FALSE.
  (  bool result = FALSE;
      for i=0 to nrTasks
       if applicationTask[i]->id == id
        *index = i
        result = TRUE
        break
      return result
  )

CSecurityCostsDoc::leastUsedTask()
- finds and returns the position in the applicationTask[] of the task less used (that is of the task with the minimum usage member)
  (int leastIndex = 0
   for i=1 to (nrTasks-1)
    if applicationTask[i]->usage < applicationTask[leastIndex]->usage
      leastIndex = i
   return leastIndex
 )

Task::Task
{  //definition of entity
   //MEMBERS
   int id;

LOW LEVEL SPECIFICATION of QoSS Costing Demonstration for MSHN 0.3 32
int nrComponents;
int nrServices;
long int usage; //additional member
ModeServiceMtrx taskMSM;
ModeChoiceMtrx taskMCM;
CostMtrx* taskCM;

//FUNCTIONS
Task(int inp1, int inp2, int inp3);
initialize(int inp1);
setId(int input);
setMode(int input);
setChoice(int input);
setMatrices();
setModeServiceMatrix(int input);
setModeChoiceMatrix(int input);
setCostMatrix(int input);
increaseUsage();
calculateCosts();
dbGetEntryTask(int input, int *par1, int *par2, int *par3);
dbGetSpecificCM(int input, *par);
dbGetNrEntriesCM(input, *par);

Task::Task() //One of the constructors
-sets usage=0

Task::increaseUsage()
-sets usage = usage+1

G. ARRAYS

As previously mentioned, in current version of SecurityCosts application we are using arrays to store the costing information for each task. The set of arrays used is described below:

- int dbTask[MAX_TASK][5]
  1st dimension
  size: number of defined tasks
  description: id of task (incremental index)
  2nd dimension
  size: 5
  description: [][0] number of Requirement Components (= number of Component Variables) of Task
number of Services of Task
id of MSM
id of MCM
id of CM

> int dbMSMtoTRV[MAX_TASK][3]
1st dimension
size: number of defined tasks (number of MSMs)
description: id of MSM (incremental index)
2nd dimension
size: 3
description: 
[0] id of normal TRV
[1] id of impacted TRV
[2] id of emergency TRV

> float dbTRV[MAX_TASK*3][MAX_REQ_COMP][4]
1st dimension
size: number of possible TRVs (number of MSMs * number of Modes)
description: id of TRV (incremental index)
2nd dimension
size: number of Requirement Components of Task with max nr of Requirement Components (i.e. max of dbTask[][][0] column)
description: incremental index of Task's Requirement Components
3rd dimension
size: 4
description: 
[0] id of Requirement Component
[1] id of Component Variable in Requirement Component
[2] minimum acceptable range value
[3] maximum acceptable range value

> int dbMCMtoCVM[MAX_TASK][3]
1st dimension
size: number of defined tasks (number of MCMs)
description: id of MCM (incremental index)
2nd dimension
size: 3
description: 
[0] id of normal CVM
[1] id of impacted CVM
[2] id of emergency CVM

> int dbCVMtoTVV[MAX_TASK*3][3]
1st dimension
size: number of possible CVMs (number of MCMs * number of Modes)
description: id of CVM (incremental index)
2nd dimension
size: 3
description: []0 id of low TVV
[]1 id of medium TVV
[]2 id of impacted TVV

> float dbTVV[MAX_TASK*3][MAX_REQ_COMP][3]
1st dimension
size: number of possible TVVs (nr of MCMs * nr of Modes * nr of Choices)
description: id of TVV (incremental index)
2nd dimension
size: number of Requirement Components of Task with max nr of Requirement Components (i.e. max of dbTask[][0] column)
description: incremental index of Task's Variable Components
3rd dimension
size: 3
description: []0 id of Component Variable
[]1 minimum user accepted value
[]2 maximum user accepted value

> int dbCM[MAX_TASK][MAX_SERV][7]
1st dimension
size: number of defined tasks (number of CMs)
description: id of CM (incremental index)
2nd dimension
size: number of Services of Task with max number of Services (i.e. max of dbTask[][1] column)
description: incremental index of Task's Services
3rd dimension
size: 7
description: []0 id of Service
[]1 id of CPU start-up CostFormula
[]2 id of CPU streaming CostFormula
[]3 id of MEMORY start-up CostFormula
[]4 id of MEMORY streaming CostFormula
[]5 id of BANDWIDTH start-up CostFormula
[]6 id of BANDWIDTH streaming CostFormula

H. DATA BASE STRUCTURES

(DB1) TASK
Structure containing for each task corresponding indexes to Mode-Service, Mode-Choice and Cost Matrices.
Number of entries equals number of defined tasks.
<table>
<thead>
<tr>
<th><strong>FIELD</strong></th>
<th><strong>DESCRIPTION</strong></th>
<th><strong>EXAMPLE</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>integer</td>
<td>0 (T_FTP)</td>
</tr>
<tr>
<td>Mode-Service</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>Mode-Choice</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>Cost</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>NrComponents</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>NrServices</td>
<td>integer</td>
<td></td>
</tr>
</tbody>
</table>

NOTE: for a different costing algorithm another field could be added, for an alternative cost matrix.

**DB2** MSM_TO_TRV
(MODE-SERVICE MATRIX to TASK REQUIREMENT VECTOR)
Structure containing for each Mode-Service Matrix indexes to Task Requirement Vectors according to network mode.
Number of entries equals number of defined tasks.

<table>
<thead>
<tr>
<th><strong>FIELD</strong></th>
<th><strong>DESCRIPTION</strong></th>
<th><strong>EXAMPLE</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>Normal_TRV</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>Impacted_TRV</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>Emergency_TRV</td>
<td>integer</td>
<td></td>
</tr>
</tbody>
</table>

**DB3** REQUIREMENT_COMPONENTS
Structure containing all possible requirement components, related to their containing Task Requirement Vector.
Number of entries equals to \( \sum_{i=1}^{nrTRV} nrCompi \), where

\[
\begin{align*}
nrTRV &= \text{number of all possible Task Requirement Vectors} \\
&= \text{number of Tasks } \times \text{number of modes} \\
nrCompi &= \text{number of requirement components of TRV i (which is a matter of definition)}
\end{align*}
\]

NOTE: If the same component belongs to a different task, there will be a separate entry for it.

<table>
<thead>
<tr>
<th>FIELD</th>
<th>DESCRIPTION</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>indicates Requirement Component’s id</td>
<td></td>
</tr>
<tr>
<td>TRV</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>indicates id of Task Requirement Vector to which component belongs</td>
<td></td>
</tr>
<tr>
<td>Variable_LHS</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a COMPONENT VARIABLE constant – indicates variable clause of component</td>
<td>1 (CV_KEY_LENGTH)</td>
</tr>
<tr>
<td>Min_Range_Value</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a number</td>
<td>.6</td>
</tr>
<tr>
<td>Max_Range_Value</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a number</td>
<td>.7</td>
</tr>
</tbody>
</table>

NOTE: Additional fields e.g. for instantiated values can be included

(DB4) TASK_REQUIREMENT_VECTOR
In order to create a specific Task Requirement Vector X:
SELECT in REQUIREMENT_COMPONENTS (TRV = X)
PROJECT (all fields but TRV)

<table>
<thead>
<tr>
<th>FIELD</th>
<th>DESCRIPTION</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>indicates Requirement Component’s id</td>
<td></td>
</tr>
<tr>
<td>Variable_LHS</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a COMPONENT_VARIABLE constant – indicates variable clause of component</td>
<td>1 (CV_KEY_LENGTH)</td>
</tr>
<tr>
<td>Min_Range_Value</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a number</td>
<td>.6</td>
</tr>
<tr>
<td>Max_Range_Value</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a number</td>
<td>.7</td>
</tr>
</tbody>
</table>
(DB5) MCM_TO_CVM
(MODE-CHOICE MATRIX to CHOICE VARIABLE MATRIX)
Structure containing for each Mode-Choice Matrix indexes to Choice Variable Matrixes according to network mode.
Number of entries equals number of defined tasks.

<table>
<thead>
<tr>
<th>FIELD</th>
<th>DESCRIPTION</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>indicates Mode-Choice Matrix’s id</td>
<td></td>
</tr>
<tr>
<td>Normal_CVM</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>indicates id of Choice-Variable Matrix for normal mode</td>
<td></td>
</tr>
<tr>
<td>Impacted_CVM</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>indicates id of Choice-Variable Matrix for impacted mode</td>
<td></td>
</tr>
<tr>
<td>Emergency_CVM</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>indicates id of Choice-Variable Matrix for emergency mode</td>
<td></td>
</tr>
</tbody>
</table>

(DB6) CVM_TO_TVV
(CHOICE-VARIABLE MATRIX to TASK VARIABLE VECTOR)
Structure containing for each Choice Variable Matrix indexes to Task Variable Vectors according to security level choice.
Number of entries equals number_of_tasks * number_of_modes.

<table>
<thead>
<tr>
<th>FIELD</th>
<th>DESCRIPTION</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>indicates Choice-Variable Matrix’s id</td>
<td></td>
</tr>
<tr>
<td>Low_TVV</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>indicates id of Task Variable Vector for low level security choice</td>
<td></td>
</tr>
<tr>
<td>Medium_TVV</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>indicates id of Task Variable Vector for medium level security choice</td>
<td></td>
</tr>
<tr>
<td>High_TVV</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>indicates id of Task Variable Vector for high level security choice</td>
<td></td>
</tr>
</tbody>
</table>

(DB7) VARIABLE_VALUES
Structure containing all possible variable values, related to their containing Task Variable Vector.

Number of entries equals to \( \sum_{i=1}^{nrTVV} nrComp_i \), where

\[ nrTVV = \text{number of all possible Task Variable Vectors} = \text{number_of_Tasks} * \text{number_of_modes} * \text{number_of_security_choices} \]
nrComp_i = number of requirement components of corresponding Task’s TRV i (which is a matter of definition)

NOTE: If the same variable with the same value belongs to a different Task Variable Vector, there will be a separate entry for it.

<table>
<thead>
<tr>
<th>FIELD</th>
<th>DESCRIPTION</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TVV</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>indicates id of Task Variable Vector to which variable belongs</td>
<td></td>
</tr>
<tr>
<td>Variable_LHS</td>
<td>integer</td>
<td>1 (CV_KEY_LENGTH)</td>
</tr>
<tr>
<td></td>
<td>a COMPONENT_VARIABLE constant – indicates variable clause of component</td>
<td></td>
</tr>
<tr>
<td>Min_Value_RHS</td>
<td>float</td>
<td>.6</td>
</tr>
<tr>
<td></td>
<td>number indicating minimum of acceptable value range</td>
<td></td>
</tr>
<tr>
<td>Max_Value_RHS</td>
<td>float</td>
<td>.6</td>
</tr>
<tr>
<td></td>
<td>number indicating maximum of acceptable value range</td>
<td></td>
</tr>
</tbody>
</table>

(DB8) TASK_VARIABLE_VECTOR
In order to create a specific Task Variable Vector X:
SELECT in VARIABLE_VALUES (TVV = X)
PROJECT (all fields but TVV)

<table>
<thead>
<tr>
<th>FIELD</th>
<th>DESCRIPTION</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable_LHS</td>
<td>integer</td>
<td>1 (CV_KEY_LENGTH)</td>
</tr>
<tr>
<td></td>
<td>a COMPONENT_VARIABLE constant – indicates variable clause of component</td>
<td></td>
</tr>
<tr>
<td>Min_Value_RHS</td>
<td>float</td>
<td>.6</td>
</tr>
<tr>
<td></td>
<td>number indicating minimum of acceptable value range</td>
<td></td>
</tr>
<tr>
<td>Max_Value_RHS</td>
<td>float</td>
<td>.6</td>
</tr>
<tr>
<td></td>
<td>number indicating maximum of acceptable value range</td>
<td></td>
</tr>
</tbody>
</table>

(DB9) CM_TO_SERVICE_COSTS
(COST MATRIX to SERVICE RESOURCE COSTS):
Structure containing Service Costs related to their containing Cost Matrix.

Number of entries equals \( \sum_{i=1}^{nrTasks} nrServices_i \), where

\( nrTasks = \text{number of Tasks} \)
\( nrServices_i = \text{number of services of Task } i \) (which is a matter of definition)
NOTE: If the same service belongs to a different task, there will be a separate entry for it.

<table>
<thead>
<tr>
<th>FIELD</th>
<th>DESCRIPTION</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
<td>integer a SERVICE constant - indicates id of service</td>
<td>3 (S_INTEGRITY_NW)</td>
</tr>
<tr>
<td>CM</td>
<td>integer indicates id of Cost Matrix to which service belongs.</td>
<td></td>
</tr>
<tr>
<td>CPU_Start_Cost</td>
<td>integer indicates id of function used to calculate start cost for CPU.</td>
<td></td>
</tr>
<tr>
<td>CPU_Stream_Cost</td>
<td>integer indicates id of function used to calculate streaming cost for CPU.</td>
<td></td>
</tr>
<tr>
<td>memory_Start_Cost</td>
<td>integer indicates id of function used to calculate start cost for memory.</td>
<td></td>
</tr>
<tr>
<td>memory_Stream_Cost</td>
<td>integer indicates id of function used to calculate streaming cost for memory.</td>
<td></td>
</tr>
<tr>
<td>bandwidth_Start_Cost</td>
<td>integer indicates id of function used to calculate start cost for bandwidth.</td>
<td></td>
</tr>
<tr>
<td>bandwidth_Stream_Cost</td>
<td>integer indicates id of function used to calculate streaming cost for bandwidth.</td>
<td></td>
</tr>
</tbody>
</table>

(DB10) COST_MATRIX
In order to create a specific Cost Matrix X:
SELECT in CM to SERVICE COSTS (CM = X)
PROJECT (all fields but CM)

<table>
<thead>
<tr>
<th>FIELD</th>
<th>DESCRIPTION</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
<td>integer a SERVICE constant - indicates id of service</td>
<td>3 (S_INTEGRITY_ES)</td>
</tr>
<tr>
<td>CPU_Start_Cost</td>
<td>integer indicates id of function used to calculate start cost for CPU.</td>
<td></td>
</tr>
<tr>
<td>CPU_Stream_Cost</td>
<td>integer indicates id of function used to calculate streaming cost for CPU.</td>
<td></td>
</tr>
<tr>
<td>memory_Start_Cost</td>
<td>integer indicates id of function used to calculate streaming cost for CPU.</td>
<td></td>
</tr>
</tbody>
</table>

LOW LEVEL SPECIFICATION of QoSS Costing Demonstration for MSHN 0.3
<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>memory_Stream_Cost</code></td>
<td>integer</td>
<td>indicates id of function used to calculate streaming cost for memory.</td>
</tr>
<tr>
<td><code>bandwidth_Start_Cost</code></td>
<td>integer</td>
<td>indicates id of function used to calculate start cost for bandwidth.</td>
</tr>
<tr>
<td><code>bandwidth_Stream_Cost</code></td>
<td>integer</td>
<td>indicates id of function used to calculate streaming cost for bandwidth.</td>
</tr>
<tr>
<td></td>
<td>Distribution List</td>
<td>Quantity</td>
</tr>
<tr>
<td>---</td>
<td>------------------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>2.</td>
<td>Dudley Knox Library, Code 013&lt;br&gt;Naval Postgraduate School&lt;br&gt;Monterey, CA 93943-5100</td>
<td>2</td>
</tr>
<tr>
<td>3.</td>
<td>Research Office, Code 09&lt;br&gt;Naval Postgraduate School&lt;br&gt;Monterey, CA 93943-5138</td>
<td>1</td>
</tr>
<tr>
<td>4.</td>
<td>Dr. Cynthia E. Irvine&lt;br&gt;Code CS/Ic&lt;br&gt;Department of Computer Science&lt;br&gt;Naval Postgraduate School&lt;br&gt;Monterey, CA 93943-5118</td>
<td>10</td>
</tr>
</tbody>
</table>