NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

OPTIMAL SENSOR ALLOCATION FOR A DISCRETE EVENT COMBAT SIMULATION

by

Thomas Doll

June 2004

Thesis Advisor:	Matt Carlyle
Second Reader:	Donovan Phillips

Approved for public release; distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** June 2004	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Optimal Sensor Allocation for a Discrete Event Combat Simulation			**5. FUNDING NUMBERS**
6. AUTHOR(S) Thomas Doll			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Army TRADOC Analysis Center Monterey, CA			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited			**12b. DISTRIBUTION CODE** unlimited

13. ABSTRACT

The U.S. Army's Future Force is being developed as a faster, lighter, more rapidly deployable alternative to the current force structure. The Future Force will feature a smaller in-theater footprint and require the ability to cover a larger area of the battle space with intelligence-gathering assets. To support this development the Naval Postgraduate School and TRAC Monterey began to conduct research in the area of allocation of Future Force sensor platforms.

A previous thesis developed the Sensor Allocation Model (SAM) for finding an appropriate mix and allocation strategy for organic Unit of Action sensors in a given threat scenario. The mix suggested by the model is robust to uncertainties in sensor performance and target quantity and location. SAM shows great promise for use as a screening tool in support of analysis of alternatives studies as well as in support of Army and Joint war fighting experimentation. It also has potential for use as an operational decision support tool for unit commanders.

This thesis discusses three improvements to SAM. First, SAM has been translated into a programming language that easily can be implemented into any simulation environment. Second, it now contains more realistic constraints on sensor platform employment duration and distance. Third, the model estimates of sensor performance have been improved with a Probability Line of Sight model. Together, these improvements have greatly improved SAM's usability.

14. SUBJECT TERMS Optimization, Objective Force, Unit of Action, Unmanned Aerial Vehicles, Sensors, Sensor Allocation, Mixed Integer Program, MIP, Operations Research, Java			**15. NUMBER OF PAGES** 89
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

OPTIMAL SENSOR ALLOCATION FOR A DISCRETE EVENT COMBAT SIMULATION

Thomas M. Doll
Captain, German Army
Diploma in Electrical Engineering, University of the German Armed Forces Munich

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
June 2004

Author: Thomas M. Doll

Approved by: W. Matthew Carlyle
Thesis Advisor

Donovan Phillips
Second Reader

James N. Eagle
Chairman, Department of Operations Research

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The U.S. Army's Future Force is being developed as a faster, lighter, more rapidly deployable alternative to the current force structure. The Future Force will feature a smaller in-theater footprint and require the ability to cover a larger area of the battle space with intelligence-gathering assets. To support this development the Naval Postgraduate School and TRAC Monterey began to conduct research in the area of allocation of Future Force sensor platforms.

A previous thesis developed the Sensor Allocation Model (SAM) for finding an appropriate mix and allocation strategy for organic Unit of Action sensors in a given threat scenario. The mix suggested by the model is robust to uncertainties in sensor performance and target quantity and location. SAM shows great promise for use as a screening tool in support of analysis of alternatives studies as well as in support of Army and Joint war fighting experimentation. It also has potential for use as an operational decision support tool for unit commanders.

This thesis discusses three improvements to SAM. First, SAM has been translated into a programming language that easily can be implemented into any simulation environment. Second, it now contains more realistic constraints on sensor platform employment duration and distance. Third, the model estimates of sensor performance have been improved with a Probability Line of Sight model. Together, these improvements have greatly improved SAM's usability.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my thesis team of Professor Matthew Carlyle and Major Donovan Phillips for agreeing to accept me as the thesis student to work on this project six months ago and direct me through the process. Numerous additional thanks go out to the many professors in the NPS OR department who helped further direct me in a forward direction.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

The U.S. Army's Future Force is being developed as a faster, lighter, more rapidly deployable alternative to the current force structure. The Future Force will feature a smaller in-theater footprint and require the ability to cover a larger area of the battle space with intelligence-gathering assets. To support this development the Naval Postgraduate School and TRAC Monterey began to conduct research in the area of allocation of Future Force sensor platforms. One result of that effort was the Sensor Allocation Model (SAM), the subject of a prior report.

This thesis seeks to improve the Sensor Allocation Model (SAM) and translate it into a programming language that can easily be implemented into a simulation environment. Initially the model did not include realistic constraints on sensor platform employment duration and distance, and it overestimated sensor performance.

The model developed in this thesis ensures that sensor platforms are represented realistically in terms of range, time on station, and performance level. Two major improvements were made: the implementation of a Probability of Line-of-Sight (PLOS) approach to represent the effects of terrain on sensor performance and 2) the refinement of the sweep width sub-model to more accurately assess sensor detection capabilities based on the geometry of the sensors and platforms.

The model is demonstrated using an unclassified set of sensor performance data similar in type and format to the classified data currently available from the US Army Materiel Systems Analysis Activity (AMSAA). The data set includes ten platforms, 92 consolidated 'packages' of sensor platforms, ten target clusters and four enemy order of battle configurations.

This research was sponsored by the U.S. Army Training and Doctrine Command Analysis Center – Monterey (TRAC-Monterey). The Java SAM output has the potential for use as input in future simulation studies.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

This thesis seeks to improve the Sensor Allocation Model (SAM) developed in [Tutton, 2003] and translate it into a programming language that can easily be implemented into any simulation environment. Since the new model is programmed in Java it is called the *Java-Sensor-Allocation-Model* (JSAM). At the outset of this research, SAM did not include realistic constraints on sensor platform employment duration and distance. These have now been added to the model. The model also overestimated sensor performance. This issue has been resolved using the geometry of the sensors and the target areas, as well as adding a probabilistic model of the existence of line-of-sight from sensor to target to capture terrain effects, so that the model represents sensor performance realistically.

To define a starting point for this thesis the following paragraphs summarize the research that has been done in this area before.

1. Background on the Original Sensor Allocation Model

The original version of SAM used results from search theory to estimate the performance of various allocations of sensors. It employed the following levels of sensor aggregation: a *sensor* is a specific piece of technology and is the fundamental unit we discuss; a *platform* can carry (multiple) sensors on one body (e.g., an airframe); and a *package* consists of multiple platforms that will be deployed simultaneously to a single geographical area. Sensors are identified as specific technologies or capabilities, such as infrared (IR), acoustic, and radar, and their performance can be measured by a probability of detection at a given range against a specific target type. Platforms are further identified as ground or aerial, and as moving or stationary. Finally, packages consist of either a single platform type, or multiple platforms. Single-platform packages are referred to as basic packages. Basic packages can be combined to form consolidated packages, and are generated automatically by SAM. SAM uses the performance of each package to

determine an effective assignment of packages to target clusters. Figure 1 gives an example for target clustering.

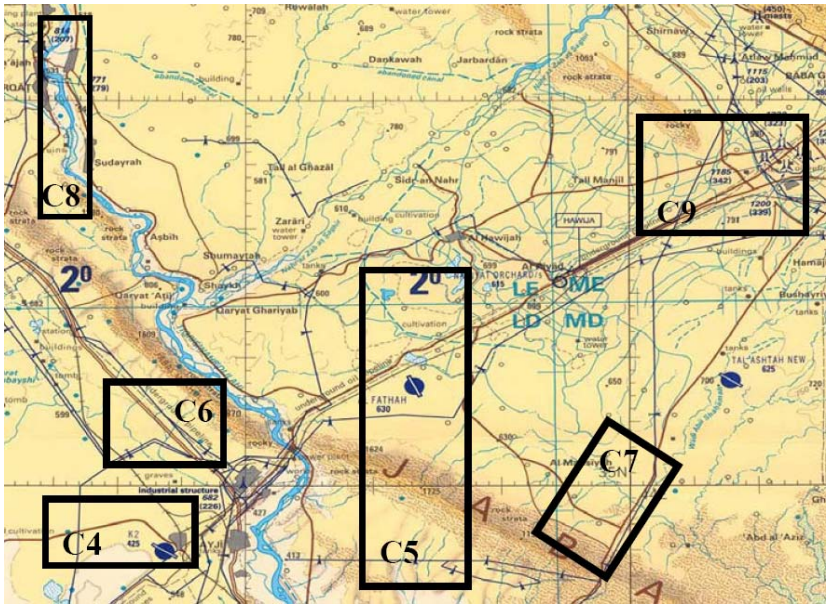

Figure 1. Example of Target Clustering (From [Tutton, 2003])

2. Sensors, Platforms and Packages in the Old Model

Following the sweep width idea in search theory the sensor sweep width in SAM was gained by numerical integration over the sensor's lateral range curve [Wagner, 1999].

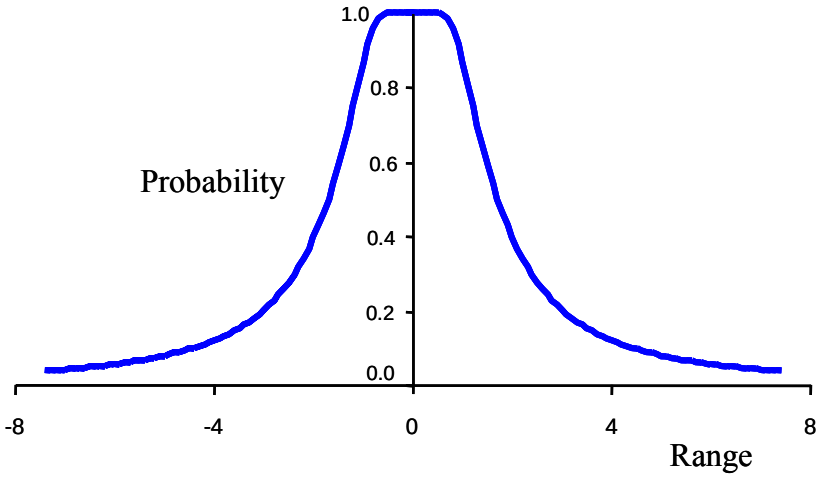

Figure 2. Example of a Lateral Range Curve. The vertical axis represents detection probability, or a similar measure for ranged devices. (From [Tutton, 2003])

Each sensor detects targets with a certain probability at a certain range resulting in a lateral range curve for each specific sensor/target pair. Sensors are not guaranteed to move directly toward a target but pass the target at some lateral range within the sensor's detection zone. Figure 3 below illustrates a sensor detection zone.

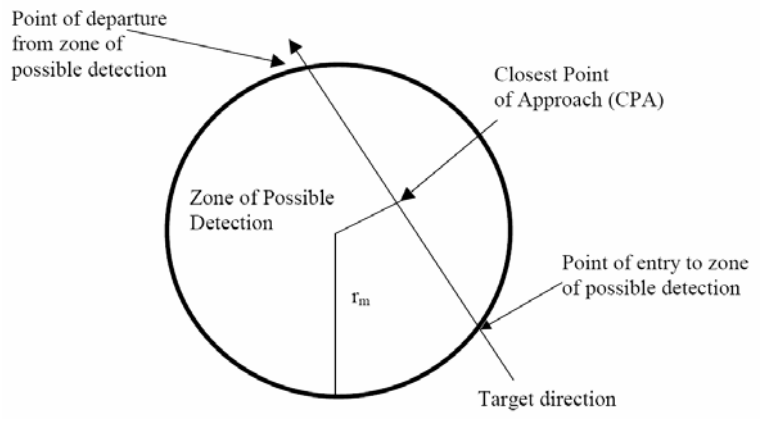

Figure 3. Sensor Detection Zone (From [Tutton, 2003])

Sweep width is a scalar measure of the search effectiveness of a sensor. By definition, sweep width is equal to the area under the lateral range curve and represents the effective width of the sensor detection zone (see equation below). Using this approach in the old Sensor Allocation Model, sensors were modeled as 'cookie cutter' sensors with sweep width, W, defined as:

$$W = \int_{-\infty}^{\infty} F_L(x)dx$$

The Cumulative Detection Probability (CDP) is the probability that a platform searching for a target over a specific time interval detects that target at least once and is determined by the total amount of time spent over any single area in the region surveyed, by its adjusted sweep width (explained in the next paragraph) and the speed with which the platform moves over or within the search area.

Multiple sensors mounted on the same platform perform at least at the level of the best sensor, and no better than the sum of all sensors. This aggregate platform performance can be depicted with an *adjusted sweep width*. In the absence of actual performance data for the platform, its adjusted sweep width is assumed to be slightly

greater than the sweep width of the best-performing sensor on the platform. Transit speed is defined as the speed at which the platform can travel to, and return from the search area. Operational time is the amount of time a platform can remain operational, including transit time and search time. Using this an associated *time on station* (time available over the search area) can be determined.

A CDP was calculated for each individual platform (for each target type) using the platform's sensing velocity, adjusted sweep width, time on station, and area of the target cluster. Since a uniform target distribution was assumed, the overall 'effectiveness' of a package was simply the average of the CDPs, which represents the expected proportion of all targets detected within a cluster. Thus the package CDP could be determined using the velocity, sweep width, and time on station of any platform in the package because each platform had the same CDP once the effective proportion of search area was determined.

3. Enemy Order of Battle and System Characteristics

Four potential enemy orders of battle (EOBs) were generated to model the uncertainty associated with target location, type and number of entities for each identified target cluster. Each EOB had a probability of occurrence that the model considered when determining a robust allocation of sensors to target areas.

Sensor platform combinations were evaluated using the four characteristics of cost, logistical footprint, perishability (opposite of survivability), and latency. Each characteristic can be calculated at each level of aggregation, and used at the package level to assess overall characteristics of the suite of packages employed. SAM allowed the analyst or decision maker to provide a relative weighting to each characteristic category according to importance in the scenario or outcome. The objective function of SAM incorporated these characteristics by minimizing their effects while maximizing expected number of targets detected in the search area. The solution of SAM, for any threat scenario, is a proposed packaging of available sensors, and an assignment of those packages to target areas, in order to maximize the effectiveness of the sensors employed.

See appendix for the description of the optimization model in NPS format.

B. PROBLEM DEFINITION

Tutton [Tutton, 2003] extended the basic idea of sensor employment in SAM to find an appropriate mix and allocation strategy for organic Unit of Action sensors in a given threat scenario. The mix suggested by the model is robust to uncertainties in sensor performance and target quantity and location, as modeled by a finite (but possibly large) number of credible scenarios. This includes the ability to model weather effects as well. The model shows great promise for use as a screening tool in support of analysis of alternatives (AOA) studies as well as in support of Army and Joint war fighting experimentation. The model also has potential for use as an operational decision support tool for unit commanders.

This thesis builds on the model described above. The two principal research questions we address are as follows: "How can SAM be improved and brought to the point where it can be validated for use in Future Combat Systems studies and analyses?" and "How can SAM be translated into a programming language that easily can be implemented into any simulation environment?"

The first question is especially interesting because of several problems that evolved when the model was designed in the first place. Initial testing revealed that SAM overestimated sensor performance. The calculated sweep width for the sensors on the various platforms are too wide compared to data generated with real sensor platforms. A probability-of-line-of-sight approach for representing sensor degradation due to terrain solves part of the problem, but the model needs to be further redefined and extended. Constraints for operational time and distance also needed to be improved.

To understand the second question, it is necessary to describe the model setup a little bit more in detail. The model actually has two parts, a preprocessor part and an optimization part. The preprocessor calculation section uses inputs from the user and data management sections to calculate required input for the optimization part. Required preprocessor calculations include such things as platform time on station (in hours), target cluster search area (in square kilometers), platform characteristic information, and consolidated package performance measures. The optimization part of the model is defined in the General Algebraic Modeling System (GAMS), a mathematical modeling

language especially suited for optimization. To embed SAM into a simulation, both the preprocessor and the model need to be included into a programming environment. At the moment it takes several manual steps to set the preprocessor and the model up for an optimization run. We have developed an implementation of SAM in Java, for use with Simkit, a Java based simulation environment. This implementation needs to have a structure that makes it easy to exchange data, and evolving data issues still need to be identified and resolved.

C. SCOPE, LIMITATIONS AND ASSUMPTIONS

To answer the stated research questions we developed a Java program to do all the preprocessor calculations, call GAMS, and capture the GAMS output. Within this setup the model has been improved to produce usable output results, and distance and time constraints have been added to the model. Probability of Line of Sight (PLOS) calculations have also been added to the model to allow more realistic estimates of sensor performance in a variety of scenarios.

For each scenario we assume all sensor platforms are located at a single, predefined entry point. We use ten different target types and ten cluster areas different in location and dimension. For testing purposes the target distribution within a cluster is generated by the program on a random basis. However seeds can be used to fix the produced distributions. The scenarios used are different in location based on the underlying terrain and on the target distribution generated by the program. In the analysis part of this thesis we distinguish between six different scenarios. Three have fixed terrain using the PLOS model, but different seeds for the target distribution, and three have the same seed but with different terrains.

One optimization run takes about 30 seconds. The optimization model that needs to be solved consists of 416 Continuous Variables, 7680 Integer Variables and 7680 Binary Variables. There are two sets of constraints defined in the model. One with a total number of 7744 constraints and one with a total number of 416 constraints. Considering the described complexity of the Optimization Program, 30 seconds is not a bad runtime. Although not perfect it is still possible to use the model in a simulation environment.

More than 99% of the time is used to solve the optimization model. There is also a good chance that the runtime can be significantly decreased by using a faster commercial solver than the XP-Solver in the GAMS environment.

THIS PAGE INTENTIONALLY LEFT BLANK

II. IMPROVEMENTS TO SAM

A. REFINEMENT OF SENSOR ALLOCATION LOGIC

1. Employment Probability of Line Of Sight

One of the most serious limitations of the first version of SAM was that it did not take the effects terrain on sensor performance into account. The model assumed the ground to be a flat plane. Observer to target line-of-sight interruption affected by roughness of terrain remained unaccounted for. To overcome this limitation we implemented a Probability of Line of Sight model following the JWARS approach [Blacksten, 2002]. The most basic idea behind the implementation is that the sensor performance data provided by AMSAA describes the conditional probability to detect a target given that line-of-sight exists, and the PLOS methodology provides the probability that line of sight exists. Using the following basic probabilistic formula for conditioning we achieve the 'true' probability of detection.

$$P_D(R) = P_D(R \mid LOS) \bullet P_D(LOS) \tag{1}$$

Based on operational altitude of the observer and given terrain characteristics the JWARS PLOS model provides the probability to detect a target at a specific range. As a measure of precision this distribution can be compared to empirical data that can be generated. Empirical PLOS curves are constructed by randomly dropping observer-target pairs on the terrain in question, each pair separated by a specified range and observer height. This sampling is repeated for some number of trials to establish an average PLOS for the terrain type, range and observer height used. This is repeated for a series of ranges to give an empirical PLOS curve for the specified observer height. The process is then repeated for other observer heights of interest to yield a family of curves. An example is given in Figure 4.

The JWARS PLOS model is subdivided into three parts that build on each other. In the first step the parametric effective roughness curve is designed. As a first implementation in SAM the data in column 6 (Southern Iran, Moderate mountainous) was used.

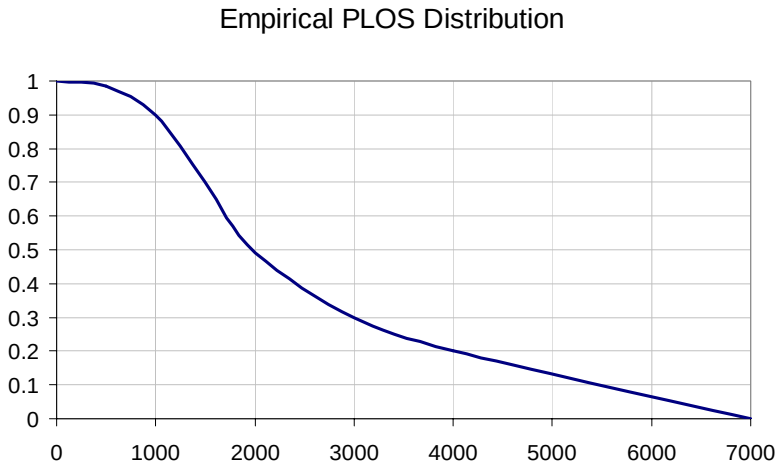

Figure 4. Empirical PLOS Distribution

The four parameters in this formula are Corrected Roughness Ceiling, Corrected Roughness Growth Rate, Extinction Ceiling and Extinction Growth Rate. Corrected Roughness Ceiling is the effective maximum value of roughness as ranges become large. Corrected Roughness Growth Rate is the growth rate parameter for the model. The dimensions are meters and meters^{-1}. Extinction Ceiling is the highest value possible for the extinction factor in the model and the Extinction Growth Rate is a measure of how fast the extinction coefficient grows with roughness. These parameters have to be defined to give the best fit to the empirical PLOS curves.

Lat Long	20N 50W	23N 48W	25N 50W	28N 48W	28N 52W	28N 57W	30N 55W	33N 48W	33N 53W
Location	Southeast central Saudi Arabia	Eastern central Saudi Arabia	Eastern coastal Saudi Arabia	North eastern coastal Saudi Arabia	Western Iran	Southern Iran	Central Iran	Central Iran	Central Iran
Description	Desert dunes?	Piedmont?	Flat coastal?	Flat coastal?	Moderate mountainous	Moderate mountainous	Mountainous	Mountainous?	Piedmont?
Corrected Roughness Ceiling (m)	6.59E+00	3.46E+00	4.31E+00	1.12E+01	7.54E+02	1.45E+01	2.15E+01	5.69E+01	1.44E+03
Corrected Roughness Growth Rate (m/m)	1.22E-03	6.83E-04	2.25E-04	5.69E-04	4.33E-05	7.04E-04	6.99E-05	3.40E-04	2.61E-06
Extinction Ceiling (m^{-1})	2.13E-04	7.54E-05	2.52E-04	1.20E-04	7.43E-04	6.05E-04	5.26E-04	4.78E-04	3.60E-04
Extinction Growth Rate (m/m)	8.34E-01	8.356E-01	1.08E+00	5.53E-01	8.60E-01	5.69E-01	1.30E+00	1.06E+00	1.31E+00

Figure 5. PLOS Parameters for predefined locations

The effect of PLOS implementation in the SAM can easily be shown graphically. Figure 6 shows the probability of detection graph without PLOS correction and Figure 7 shows the same graph after the PLOS correction. It can be seen that the probability to detect a target decreases significantly even for small ranges at around 500 meters. This curve is much more realistic than the one without the PLOS correction. It does not overestimate the sensor capabilities as much as the pure probability curve.

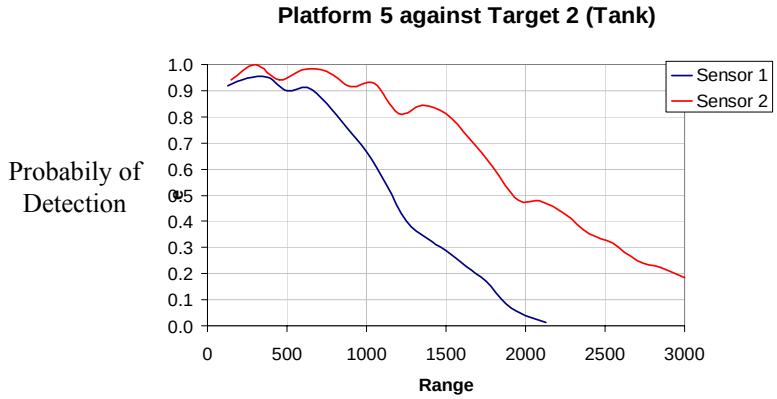

Figure 6. Probability of Detection without PLOS

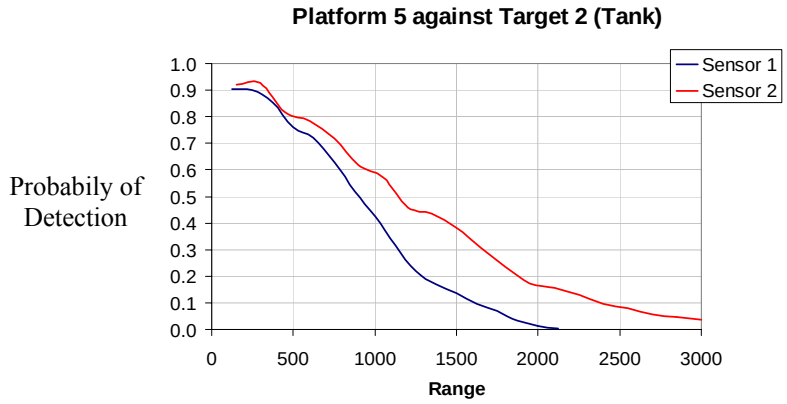

Figure 7. Probability of Detection after PLOS correction

2. Implementation of Distance and Time Constraints

The implementation of constraints for operational time and radius turned out to be relatively easy in Java. The operational time constraint was already implemented in the spreadsheet preprocessor developed by Tutton and Olson [SMM_Preprocessor, 2003]. In

this modeling approach the preprocessing part was done in an Excel workbook with several spreadsheets. As a first step for the implementation two floating point arrays were used to store operational time and radius values for the different platforms. The values used for this thesis are shown in Table 1. They are taken from the platform description in the 'Army Future Combat Systems Unit of Action Systems Book Version 3.0' [AMSAA, 22 May 2003].

Platform	UAV Class I	UAV Class II	UAV Class III	UAV Class IVa	ARV RSTA	UGS
Operational Time [hours]	1	2	6	5	72	72
Operational Radius [km]	16	30	40	75	50	assumed unlimited

Table 1. Operational Time and Radii for different Platforms

It is worth mentioning that the UGS is the only stationary platform. Its search speed is zero. The CDP for a UGS is calculated based on target speed which usually is much smaller than the average search speed of a UAV. It is designed to operate for 72 hours and decreasing this time decreases its cumulative detection probability significantly. The Operational Radius is assumed to be unlimited because these platforms can be delivered by several long-distance means (aircraft, artillery tube, etc.).

The values for operational time and radius are used to calculate the *time-on-station* for a platform. Using travel speed and distance from the entry point to the closest point of a cluster we get the travel time. This time can be subtracted from the operational time to give the time-on-station. If this value is negative it is set to zero, which essentially means that the platform cannot get to an area (and back) within the time frame of the scenario, and therefore will have no time to search for targets. If a cluster lies outside the operational radius of a platform the time on station also is set to zero. This simple adjustment to the data makes sure that a platform is not used outside its operational radius

or outside its operational time. (The described calculation is performed in the timeOnStation method in the class Platforms, see Appendix B.

3. Adjustments to Represent Sensor Performance Realistically

To determine how best to model the performance of sensor platforms in our model, we must first have a sense as to how the platform performs operationally. Additionally, since most of these platforms exist in concept only, their operational employment techniques can only be surmised. One expert opinion on how these platforms may be employed is summarized as follows (an airborne platform with an optical or IR sensor is assumed): as the platform proceeds along its search path, it captures three 'still' images – left, center, and right with respect to its flight direction. Still images from a moving platform are obtained by vertically (in the direction of flight) moving the sensor lens to briefly (1-2 seconds) compensate for the platform's forward movement. The sensor lens is then shifted horizontally (orthogonal to flight direction) to obtain the center and right images in similar fashion. This process is repeated continually as the platform proceeds along its search path, resulting in an effective search width of three times the sensor's field of view (FOV). See Figure 8.

A new modeling approach was developed to represent sensor platform employment as described above. The sensor itself is restricted by vertical and horizontal angles within which they can observe – its FOV. Within this FOV a sensor can detect targets with specific probability that is defined by the observing sensor, sensor-target range, and target type. Outside the FOV, the probability to detect a target is zero. In our modeling approach one or more sensors are mounted on each platform and each of them looks in a specific observation angle forward and captures still images as described in the paragraph above and Figure 8.

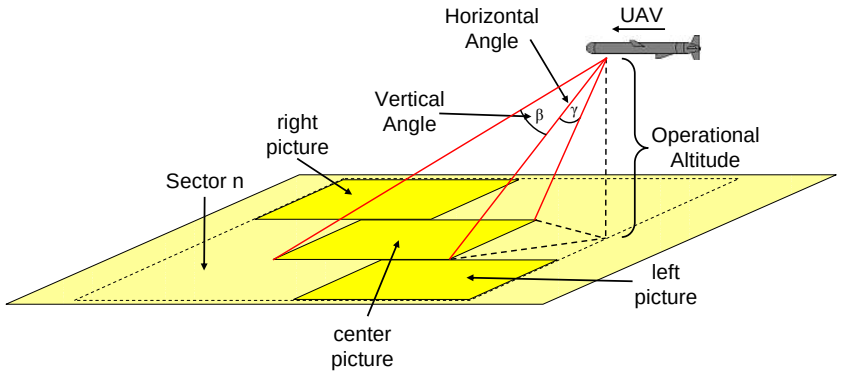

Figure 8. Basic modeling idea

Our model represents sensor performance as if the sensor takes three pictures in a row before the platform moves on to the next sector, as shown in Figure 9. Platform speed and altitude are assumed fixed (user input data).

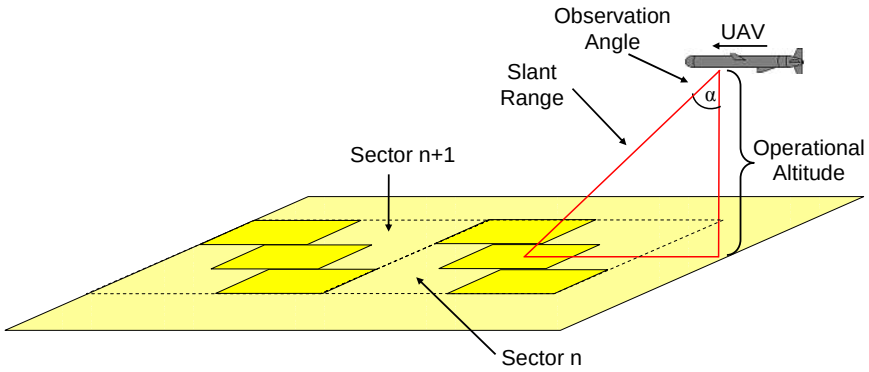

Figure 9. Sectors

Figure 10 shows how the ground distances from the sensor to the three pictures (left, center and right) in one sector are calculated in JSAM. Depending on the geometric differences in surface range and slant range to the sensor the center picture and the side pictures (left and right) have a different geometry. In this diagram the pictures are drawn side by side. This is not how they really are distributed in the sector but is necessary to calculate the distances correctly. Based on the surface range from the sensor to the center of each picture, an 'average' detection probability is interpolated from sensor performance data input. The sensor is assumed to perform at this 'average' level at all points within the picture.

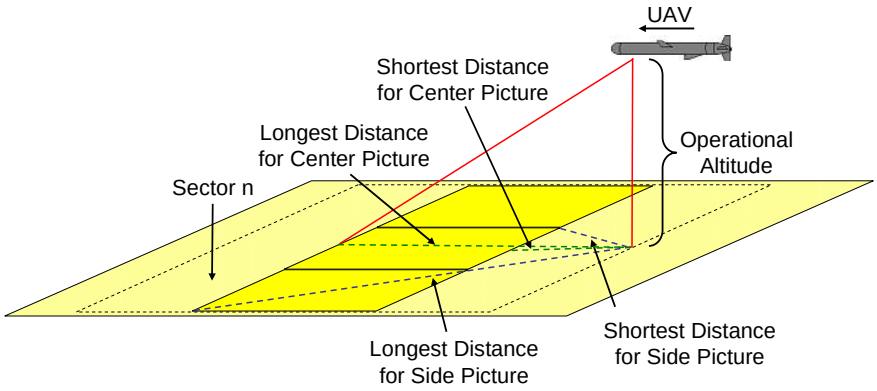

Figure 10. Ground Distances

Picture length, picture width and probability to detect a target within a picture are used to define a volume that represents the detection capability of the sensor in one picture. This volume can be transformed into a cube with probability of detection one and an *effective detection area* smaller than the original picture area. The volumes of the two cubes are, of course, identical. The effective detection area is the area relative to the original picture within which the probability to detect a target would be one (note: this is the three-dimensional analog to the two-dimensional Sweep Width concept discussed in section I.2). See Figure 11.

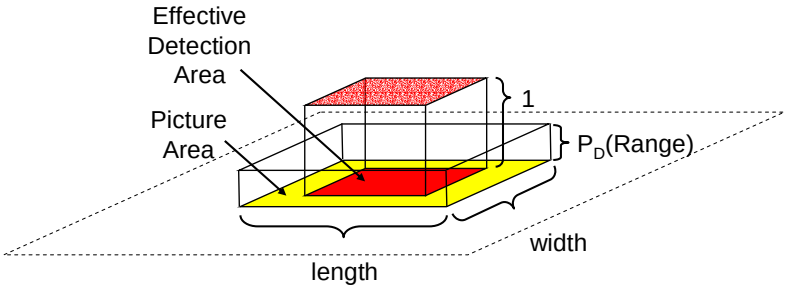

Figure 11. Area Transformation

Finally we can sum the effective detection areas of the tree pictures in a sector to get the total effective area covered in a sector. To simplify the model, the sector length is assumed to be twice the length of a picture (in reality, this dimension depends on several factors: length of time spent observing a single picture area; altitude; speed; and

observation angle). As shown in Figure 12 this now can easily be transformed into a sensor sweep width.

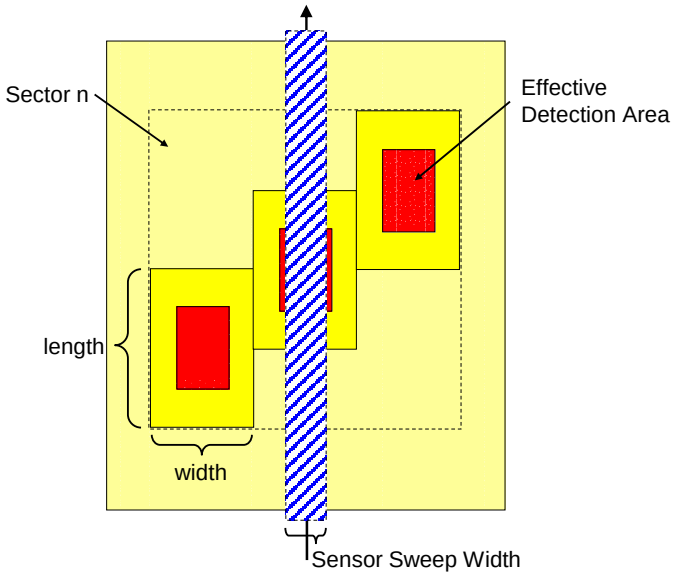

Figure 12. Transformation for Sensor Sweep Width

B. ADAPTATION OF SAM FOR INTEGRATION WITH A SIMULATION ENVIRONMENT

Initially SAM was designed in two parts. An Excel Workbook with various macros was used to do all the preprocessing. The optimization model itself was and still is defined in the General Algebraic Modeling System (GAMS). The way the model was set up the user had to do several steps manually before he could start the optimization in. To port SAM into Java the system had to be refined and restructured. Objective was to create a program that in one single step does all the preprocessing calculations, produces the necessary GAMS input files, starts GAMS and captures the GAMS output. The final design setup is shown in Figure 13. It uses interfaces as well as inheritance.

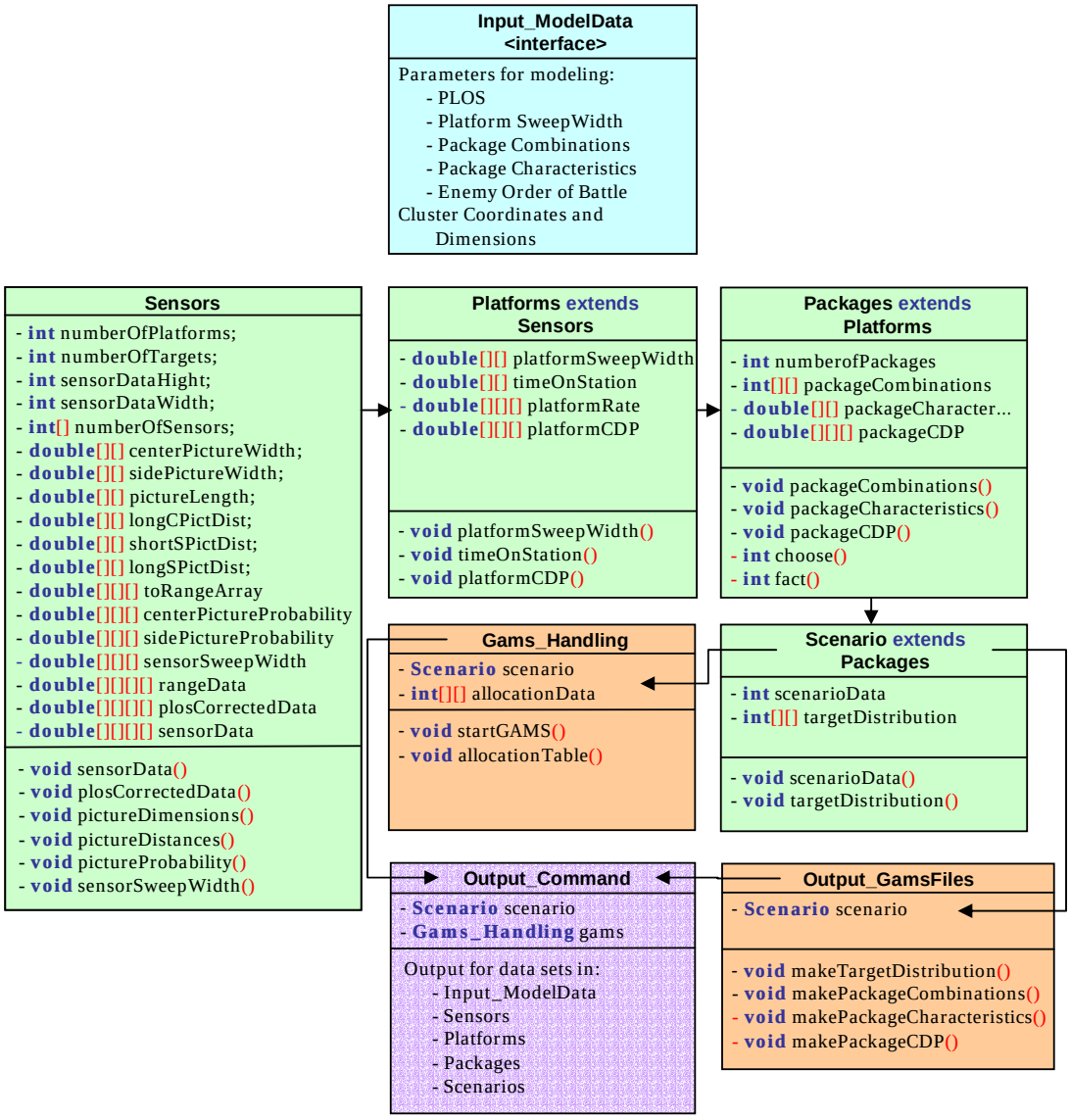

Figure 13. JSAM Design Concept

JSAM consists of eight classes shown in the diagram and the Main class from which the program is executed. The classes Sensors, Platforms, Packages and Scenario inherit from each other. This makes sense since Sensors are mounted on Platforms and Platforms are combined to Packages. Finally Packages are employed in Scenarios. Another good reason for doing this is that parameters defined in parent classes are available in child classes. In our case this means that all instance variables defined in the parent classes are available in the child class Scenario. This class is the only one of the four that needs to be instantiated in Main. The object instantiated from Scenario is called scenario. The

interface ‘Input ModelData’ is implemented by all other classes besides Main. It is used to control the model. All model parameters are defined within this class. For the user this is very convenient. The complete parameter set up can be done from within this class. ‘Output Command’ handles all output methods that print to the command line. This again is very convenient for the user. If one of the output methods needs to be adjusted he just needs to focus on this one class. The ‘Output Command’ is instantiated in Main. The object is called output.

The Optimization Part of the model is executed in GAMS. JSAM handles this with the ‘Output GamsFiles’ and ‘Gams Handling’ classes. ‘Output GamsFiles’ produces four GAMS input files. ‘Gams Handling’ starts GAMS as a Java encapsulated process. This process runs completely in the background, invisible to the user. It also captures the GAMS output. ‘Output GamsFiles’ as well as ‘Gams Handling’ are instantiated in Main. The objects are called gamsFiles and gams respectively.

III. ANALYSIS

A. TEST AND EVALUATION OF THE JSAM OUTPUT FOR DIFFERENT TARGET DISTRIBUTIONS

The main purpose of this chapter is to test the program and to analyze and evaluate the optimization output. In this first subchapter the factor of variance is chosen to be the distribution of targets on the battlefield. It was mentioned earlier in the text that JSAM is capable of producing target distributions randomly. In the following test setup three different seeds are used to generate three different target distributions. The objective of this test is to see if the program can handle different types of input and if the optimization results generated make sense. It is expected that the output for each run is different. The distribution of targets within clusters is one of the main factors for the GAMS optimization model. The different seeds for runs one to three are as follows:

1st run 13769 2nd run 45832 3rd run 91273

The values chosen for all other model variables and parameters are shown in Figure 14. This is to make sure that the analysis can easily be reproduced. The reader just needs to copy the content and paste it into the class Input_ModelData. The values for the seeds of course need to be set accordingly. I used JSAM Version 1.5 for the analysis. It is more convenient than Version 1.6, which is optimized for the implementation into an Event Graph Simulation. Besides this the two versions produce exactly the same output.

```
// parameters for modeling PLOS
double sigma_ceiling          = 14.5;
double g_rgh                  = 0.000704;
double c_ceiling              = 0.000605;
double g_xtnct                = 0.569;

double[] operatonalAltitude    = {150.0, 300.0, 600.0, 2000.0, 3.0, 1.0};

// parameters for modleing platform SweepWidth
double[][] horizontalAngle   = {
  {20.00, 15.00},
  {20.00, 9.200},
  {8.920, 8.900},
  {8.920, 8.900},
  {20.00, 15.00},
  {10.00, 7.500}
  };                  //[Platform][Sensor] horizontal observation angle

double[][] verticalAngle  = {
  {15.00, 11.25},
  {15.00, 6.900},
  {6.680, 6.700},
  {6.680, 6.700}
  };                  //[Platform][Sensor] vertical observation angle

double[] observationAngle = {45.0, 45.0, 45.0, 45.0};   //[Platform]

double swag_dependance = 0.1;
```

```
 // parameters for modeling package combinations
 int[][] basicPackages = {
   { 1, 0, 0, 0, 0, 0},    //PlatformCombination for Package 1
   { 0, 1, 0, 0, 0, 0},    //PlatformCombination for Package 2
   { 0, 0, 1, 0, 0, 0},    //PlatformCombination for Package 3
   { 0, 0, 0, 1, 0, 0},    //PlatformCombination for Package 4
   { 0, 0, 0, 0, 1, 0},    //PlatformCombination for Package 5
   { 0, 0, 0, 0, 0, 1},    //PlatformCombination for Package 6
   { 1, 1, 0, 0, 0, 0},    //PlatformCombination for Package 7
   { 0, 0, 1, 1, 0, 0},    //PlatformCombination for Package 8
 };

double package_enhFact = 1.1;

// parameters for modleing Package Characteristics
double[] platformLatency         = {0.6, 0.5, 0.3, 0.1, 0.4, 1.0};
double[] platformCost            = {0.1, 0.3, 0.7, 1.0, 0.8, 0.2};
double[] platformPerishability   = {0.2, 0.2, 0.8, 1.0, 0.4, 0.2};
double[] platformLogistics       = {0.3, 0.4, 0.5, 0.7, 0.1, 0.1};

double[] travelSpeed          = {80.0,  100.0, 200.0, 135.0,  200.0, 200.0};
double[] searchSpeed          = {60.0,  80.0,  140.0, 100.0,  20.0,  0.0  };
double[] operationalTime      = {2.0,   2.0,   6.0,   5.0,    72.0,  72.0 };
double[] operationalRadius    = {16.0,  30.0,  40.0,  75.0,   800.0, 800.0};

boolean[] airDropped       = {false, false, false, false,  true,  true,};

 // cluster coordinates and dimensions
 double[] entryPoint = {19.0, 50.0};

 double[][] clusterData = {
   {30.5, 54.0, 31.0, 22.0},  //cluster_01 {xCoord, yCoord, ewDimension, nsDimension}
   {20.5, 39.0, 21.0, 28.0},  //cluster_02
   {36.0, 44.0, 20.0, 18.0},  //cluster_03
   {49.0, 63.5, 32.0, 23.0},  //cluster_04
   {50.5, 49.0, 29.0, 26.0},  //cluster_05
   {48.0, 32.0, 24.0, 28.0},  //cluster_06
   {65.0, 53.0, 20.0, 34.0},  //cluster_07
   {62.5, 32.0, 25.0, 28.0},  //cluster_08
   {55.5, 20.5, 39.0, 15.0},  //cluster_09
   {62.0, 10.0, 28.0, 31.0},  //cluster_10
};

double[] targetSpeed = {1.0, 5.0, 4.0, 7.0, 5.0, 6.0, 6.0, 6.0, 7.0, 5.0};

// enemy order of battle
long scenario_Seed            = 13769;
int max_NumberOfTargets       = 100;
double[] eOB_Factor           = {0.45, 0.50, 0.60, 0.65};
```

Figure 14. Variable and Parameter setup for Analysis with different target distributions

For this modeling set up the following ten targets were used:

Target 1	Rifleman, RPG, SOF
Target 2	Tracked Main Battle Tank
Target 3	Special Purpose Artillery
Target 4	Wheeled Light Transport
Target 5	Tracked Armor Vehicle
Target 6	Heavy Wheeled Transport
Target 7	Towed Artillery
Target 8	Wheeled Armor Vehicle
Target 9	Engineer Vehicle
Target 10	Tracked Missile Launcher

Table 2. Target Description

The target distribution generated for the first run with seed 13769 is shown in Table 3.

```
------- Target Distribution EOB01 --------
Cl T01 T02 T03 T04 T05 T06 T07 T08 T09 T10

01 026 089 069 000 025 006 071 000 072 000
02 088 027 000 000 040 000 000 099 000 000
03 000 000 000 065 000 000 000 000 000 020
04 000 000 000 000 000 000 000 013 000 045
05 000 000 000 000 000 000 000 000 000 000
06 035 002 000 083 000 055 053 000 000 000
07 000 000 000 000 000 081 033 000 000 000
08 000 005 076 018 063 000 000 000 000 000
09 013 024 000 009 000 012 000 000 000 000
10 000 002 000 000 000 000 099 026 099 000

------- Target Distribution EOB11 --------
Cl T01 T02 T03 T04 T05 T06 T07 T08 T09 T10

01 026 000 069 013 000 000 000 000 072 000
02 000 027 047 000 040 000 000 000 000 000
03 000 000 078 065 000 053 073 056 060 000
04 000 100 044 070 000 000 000 000 000 045
05 000 050 000 000 011 076 000 091 029 049
06 000 002 000 083 013 000 053 047 057 000
07 064 000 075 000 080 081 033 028 000 000
08 000 000 000 000 063 064 060 075 000 099
09 000 000 000 009 000 000 000 000 019 022
10 021 000 091 093 089 000 099 000 000 044

------- Target Distribution EOB21 --------
Cl T01 T02 T03 T04 T05 T06 T07 T08 T09 T10

01 026 000 069 013 000 000 000 013 000 091
02 088 000 047 000 040 000 097 000 036 000
03 000 089 078 065 000 053 000 056 060 000
04 035 100 044 070 000 000 004 013 000 000
05 000 000 055 017 011 000 000 091 029 049
06 035 000 038 000 013 055 000 047 000 000
07 000 036 075 000 000 081 033 028 000 000
08 024 000 076 018 063 064 000 000 000 000
09 000 000 077 000 015 000 000 043 000 000
10 000 000 091 000 089 015 000 000 099 044

------- Target Distribution EOB31 --------
Cl T01 T02 T03 T04 T05 T06 T07 T08 T09 T10

01 026 089 069 013 025 006 000 013 000 091
02 088 027 000 060 040 092 097 099 036 000
03 000 000 000 065 000 053 073 000 060 000
04 000 100 044 070 000 037 004 013 000 045
05 000 050 055 017 000 076 072 091 029 000
06 000 002 038 083 013 000 053 047 000 000
07 000 000 000 000 080 081 000 000 061 050
08 000 005 076 018 063 000 000 075 073 000
09 000 024 077 009 015 012 000 043 000 000
10 021 002 091 093 089 015 099 026 000 044
```

Table 3. Randomly generated target distribution (seed 13769)

The tables are very easy to read. Each sub table consists of ten rows for ten clusters and ten columns for ten targets. The number 026 in the first row first column means that there are 26 targets of type one in cluster one. In cluster 07 there are 81 targets of type 6. The target distribution generated for the second run with seed 45832 is shown in Table 4.

```
------- Target Distribution EOB01 --------
Cl T01 T02 T03 T04 T05 T06 T07 T08 T09 T10

01 000 068 000 058 000 000 087 033 087 052
02 094 000 000 000 079 018 000 000 000 000
03 000 000 099 000 037 067 000 000 038 072
04 066 000 043 083 000 035 000 000 000 044
05 000 000 015 000 000 000 053 041 049 000
06 000 000 091 063 058 000 024 000 000 000
07 000 000 000 012 000 000 035 000 000 087
08 016 020 000 000 000 095 007 000 070 000
09 000 048 040 091 000 088 076 000 000 000
10 000 049 000 000 063 000 000 000 000 000

------- Target Distribution EOB11 --------
Cl T01 T02 T03 T04 T05 T06 T07 T08 T09 T10

01 096 068 000 000 000 012 087 033 000 000
02 094 000 044 000 000 018 000 080 000 000
03 000 000 099 000 037 067 000 090 000 000
04 000 000 043 083 029 000 093 042 050 044
05 000 000 015 059 000 033 053 000 049 064
06 066 081 091 063 000 000 024 000 000 000
07 031 000 000 012 021 073 000 044 000 087
08 000 000 028 000 051 000 007 000 000 000
09 000 000 000 000 091 000 000 076 000 000 055
10 000 000 000 003 063 096 000 000 002 000

------- Target Distribution EOB21 --------
Cl T01 T02 T03 T04 T05 T06 T07 T08 T09 T10

01 000 068 071 058 059 012 087 000 000 052
02 094 000 044 000 079 000 000 080 044 000
03 076 082 099 051 037 067 045 090 000 072
04 066 000 043 000 029 035 093 000 050 000
05 000 047 015 000 037 000 053 041 049 000
06 000 081 091 063 058 000 000 054 000 000
07 031 050 007 012 021 073 035 044 000 000
08 016 000 000 083 000 095 000 000 000 000
09 055 000 040 091 005 088 076 000 049 055
10 000 049 000 003 063 096 006 064 002 072

------- Target Distribution EOB31 --------
Cl T01 T02 T03 T04 T05 T06 T07 T08 T09 T10

01 000 068 071 058 059 012 000 033 087 000
02 094 000 000 000 079 000 027 080 000 000
03 076 000 099 051 037 000 045 090 000 072
04 066 028 043 083 000 035 093 042 050 044
05 008 047 000 059 037 000 053 041 049 000
06 066 081 000 063 058 023 024 054 100 004
07 031 050 000 012 000 073 035 044 030 000
08 016 020 028 000 000 095 000 000 070 000
09 055 000 040 000 000 088 000 000 049 000
10 077 049 000 000 063 096 006 000 002 072
```

Table 4. Randomly generated target distribution (seed 45832)

Table 5 shows the target distribution generated for the third run with seed 91273.

------- Target Distribution EOB01 --------									
Cl	T01	T02	T03	T04	T05	T06	T07	T08	T09	T10
01	000	062	000	070	085	045	000	003	080	002
02	000	000	000	082	000	000	000	043	100	057
03	034	000	000	000	000	049	075	094	000	055
04	000	044	000	000	055	000	035	000	000	051
05	000	096	000	049	000	060	000	000	081	000
06	074	000	000	011	000	000	006	000	000	000
07	000	053	000	014	043	016	000	095	000	050
08	002	000	072	000	000	000	091	000	000	064
09	000	009	078	028	013	000	039	000	000	025
10	062	000	000	092	000	019	015	000	071	000

------- Target Distribution EOB11 --------									
Cl	T01	T02	T03	T04	T05	T06	T07	T08	T09	T10
01	069	000	000	070	085	045	038	003	080	000
02	000	062	088	082	065	073	070	043	000	057
03	000	060	000	065	000	049	000	094	063	055
04	000	000	097	000	055	024	035	000	000	051
05	000	096	089	000	005	000	000	096	081	000
06	000	069	000	011	055	000	000	032	000	042
07	000	053	093	014	043	016	049	000	063	000
08	002	000	000	077	012	047	091	000	039	064
09	095	000	000	028	013	022	000	000	090	025
10	062	046	090	000	061	019	015	006	000	097

------- Target Distribution EOB21 --------									
Cl	T01	T02	T03	T04	T05	T06	T07	T08	T09	T10
01	069	062	000	070	000	045	038	000	000	002
02	022	062	088	000	000	000	070	043	100	057
03	034	000	047	065	067	049	075	094	063	055
04	000	044	000	031	000	024	000	000	099	000
05	000	096	000	000	005	060	000	096	081	000
06	074	069	035	000	000	000	000	032	000	000
07	097	053	093	014	043	000	000	095	000	000
08	002	000	000	000	012	047	091	000	039	000
09	095	000	078	000	013	000	000	078	090	025
10	062	046	090	092	000	019	015	000	000	097

------- Target Distribution EOB31 --------									
Cl	T01	T02	T03	T04	T05	T06	T07	T08	T09	T10
01	069	000	000	070	000	045	038	003	080	000
02	022	062	088	082	000	000	070	043	100	000
03	034	060	047	000	000	049	075	094	063	055
04	037	044	097	000	055	024	000	016	099	051
05	090	000	089	049	005	000	054	096	081	000
06	074	069	035	000	055	065	000	032	053	042
07	097	000	093	014	043	000	000	095	000	000
08	002	056	072	077	012	000	091	081	000	064
09	000	000	078	028	000	022	039	078	090	025
10	062	000	090	092	061	019	015	006	071	097

Table 5. Randomly generated target distribution (seed 91273)

The quadrants two, three and four are for the other three 'enemy orders of battle' constellations. A simple visual inspection of the three distribution tables shows that they are completely different. Table 6 shows the output allocations of the three runs.

1st run

----- Platf Alloc -----						
	P1	P2	P3	P4	P5	P6
C01	--	--	--	10	--	10
C02	16	16	--	08	--	--
C03	--	20	10	--	--	--
C04	--	--	--	02	04	--
C05	--	--	--	02	04	--
C06	--	--	--	03	03	--
C07	--	--	--	--	04	08
C08	--	--	--	--	04	08
C09	--	--	--	02	02	--
C10	--	--	--	--	06	03
Sum	16	36	10	27	27	29

2nd run

----- Platf Alloc -----						
	P1	P2	P3	P4	P5	P6
C01	20	20	--	10	--	--
C02	08	--	--	--	04	--
C03	--	06	--	--	03	--
C04	--	--	--	06	03	--
C05	--	--	06	06	--	12
C06	--	--	02	02	04	--
C07	--	--	--	--	04	08
C08	--	--	--	01	02	--
C09	--	--	--	02	04	--
C10	--	--	--	--	03	06
Sum	28	26	08	27	27	26

3rd run

----- Platf Alloc -----						
	P1	P2	P3	P4	P5	P6
C01	--	--	--	10	--	10
C02	--	16	--	08	--	--
C03	--	18	09	--	--	--
C04	--	--	--	02	04	--
C05	--	--	--	02	04	--
C06	--	--	03	03	03	--
C07	--	--	--	--	04	08
C08	--	--	--	--	04	08
C09	--	--	--	02	02	--
C10	--	--	--	--	06	06
Sum	00	34	12	27	27	32

Table 6. Platform Allocations for the three test runs

In these output tables we can directly see which platforms are assigned to which clusters. In the allocation table for run one for example there are 16 UAV II (Platform 2) sent to cluster 2 and 20 are sent to cluster 3. For this modeling set up the following six platforms were used:

Platform 1	Unmanned Aerial Vehicle – Class I
Platform 2	Unmanned Aerial Vehicle – Class II
Platform 3	Unmanned Aerial Vehicle – Class III

Platform 4	Unmanned Aerial Vehicle – Class IVa
Platform 5	Armed Robotic Vehicle - RSTA
Platform 6	Unattended Ground Sensors

Table 7. Platform Description

Due to the complexity of the optimization model it is still hard to analyze the output but there are at least a few things that can be mentioned. First of all we can see that the results for the three runs are different from each other. This makes sense since the model is expected to react differently for different target distributions. The following platform distribution was used:

Platform	1	2	3	4	5	6
# available	108	36	12	27	27	33

Table 8. Number of Platforms available

By comparison with the total number of platforms of each type that are used by the model we see that in average 90% of the platforms 2 to 6 are chosen. Platform one is rarely chosen. Platform one has a limited range of 16 km and a limited operational time of 120 minutes. The output below was generated with the setup for the 3rd run above. Just the value for the operational time for platform 1 was increased by one hour to a total of three hours. We can see that the number of platforms of type one increased from a total of 0 to a total of 36.

-----	Platf	Alloc	-----			
	P1	P2	P3	P4	P5	P6
C01	20	20	10	--	--	--
C02	16	16	--	08	--	--
C03	--	--	02	02	02	--
C04	--	--	--	--	04	08
C05	--	--	--	06	03	--
C06	--	--	--	--	04	02
C07	--	--	--	02	04	--
C08	--	--	--	09	--	18
C09	--	--	--	--	04	02
C10	--	--	--	--	06	03
Sum	51	36	12	27	27	33

Table 9. Platform Allocations for 3rd run with increased operational time for Platform 1

To see how the model reacts with only airborne sensor platforms available we also generated the three runs just with platforms 1 to 4. The parameters p_avail(p) in the GAMS Model for platforms 5 and 6 were set to be zero. The operational time for platform 1 was set back to 2 hours. The result is shown in Table 10.

1st run

----- Platf Alloc -----						
	P1	P2	P3	P4	P5	P6
C01	30	20	--	--	--	--
C02	24	16	--	--	--	--
C03	--	--	02	03	--	--
C04	--	--	04	02	--	--
C05	--	--	03	03	--	--
C06	--	--	03	02	--	--
C07	--	--	--	04	--	--
C08	--	--	--	05	--	--
C09	--	--	--	01	--	--
C10	--	--	--	07	--	--
Sum	34	36	12	27	00	00

2nd run

----- Platf Alloc -----						
	P1	P2	P3	P4	P5	P6
C01	08	08	--	04	--	--
C02	30	20	--	--	--	--
C03	--	08	--	04	--	--
C04	--	--	06	02	--	--
C05	--	--	03	02	--	--
C06	--	--	03	03	--	--
C07	--	--	--	04	--	--
C08						
C09	--	--	--	05	--	--
C10	--	--	--	03	--	--
Sum	38	36	12	27	00	00

3rd run

----- Platf Alloc -----						
	P1	P2	P3	P4	P5	P6
C01	30	20	--	--	--	--
C02	14	14	07	--	--	--
C03	--	02	--	04	--	--
C04	--	--	02	02	--	--
C05	--	--	02	03	--	--
C06	--	--	01	01	--	--
C07	--	--	--	04	--	--
C08	--	--	--	04	--	--
C09	--	--	--	04	--	--
C10	--	--	--	05	--	--
Sum	44	36	12	27	00	00

Table 10. Platform Allocations just for platforms 1 to 4

B. TEST AND EVALUATION OF THE JSAM OUTPUT FOR DIFFERENT PLOS PARAMETERS

In chapter II A 1 we described how the JWARS PLOS model was implemented into JSAM. In this chapter we want to test the implementation and show how JSAM reacts to different types of terrain. Three different terrain types were chosen to be tested. The PLOS parameters for these terrain types are shown below.

Set Up 1: Southern Iran (moderate mountainous)

sigma_ceiling	= 14.5;
g_rgh	= 0.000704;
c_ceiling	= 0.000605;
g_xtnct	= 0.569;

Set Up 2: Southeast Central Arabia (desert dunes)

sigma_ceiling	= 6.59;
g_rgh	= 0.00122;
c_ceiling	= 0.000213;
g_xtnct	= 0.834;

Set Up 3: Central Iran (mountainous)

sigma_ceiling	= 56.9;
g_rgh	= 0.00034;
c_ceiling	= 0.000478;
g_xtnct	= 1.3;

The setup for all other model parameters is shown in Figure 15:

```
// parameters for modeling PLOS
double[] operatonalAltitude         = {150.0, 300.0, 600.0, 2000.0, 3.0, 1.0};

// parameters for modleing platform SweepWidth
double[][] horizontalAngle   = {
  {20.00, 15.00},
  {20.00, 9.200},
  {8.920, 8.900},
  {8.920, 8.900},
  {20.00, 15.00},
  {10.00, 7.500}
  };                    //[Platform][Sensor] horizontal observation angle

double[][] verticalAngle  = {
  {15.00, 11.25},
  {15.00, 6.900},
  {6.680, 6.700},
  {6.680, 6.700}
  };                    //[Platform][Sensor] vertical observation angle

double[] observationAngle = {45.0, 45.0, 45.0, 45.0};   //[Platform]

double swag_dependance = 0.1;

// parameters for modeling package combinations
int[][] basicPackages = {
  { 1, 0, 0, 0, 0, 0},    //PlatformCombination for Package 1
  { 0, 1, 0, 0, 0, 0},    //PlatformCombination for Package 2
  { 0, 0, 1, 0, 0, 0},    //PlatformCombination for Package 3
  { 0, 0, 0, 1, 0, 0},    //PlatformCombination for Package 4
  { 0, 0, 0, 0, 1, 0},    //PlatformCombination for Package 5
  { 0, 0, 0, 0, 0, 1},    //PlatformCombination for Package 6
  { 1, 1, 0, 0, 0, 0},    //PlatformCombination for Package 7
  { 0, 0, 1, 1, 0, 0},    //PlatformCombination for Package 8
};

double package_enhFact = 1.1;

// parameters for modleing Package Characteristics
double[] platformLatency         = {0.6, 0.5, 0.3, 0.1, 0.4, 1.0};
double[] platformCost            = {0.1, 0.3, 0.7, 1.0, 0.8, 0.2};
double[] platformPerishability   = {0.2, 0.2, 0.8, 1.0, 0.4, 0.2};
double[] platformLogistics       = {0.3, 0.4, 0.5, 0.7, 0.1, 0.1};

double[] travelSpeed          = {80.0,  100.0, 200.0, 135.0,  200.0, 200.0};
double[] searchSpeed          = {60.0,  80.0,  140.0, 100.0,  20.0,  0.0  };
double[] operationalTime      = {1.0,   2.0,   6.0,   5.0,    72.0,  72.0 };
double[] operationalRadius    = {16.0,  30.0,  40.0,  75.0,   800.0, 800.0};

boolean[] airDropped       = {false, false, false, false,  true,  true,};

// cluster coordinates and dimensions
double[] entryPoint = {19.0, 50.0};

double[][] clusterData = {
  {30.5, 54.0, 31.0, 22.0},  //cluster_01 {xCoord, yCoord, ewDimension, nsDimension}
  {20.5, 39.0, 21.0, 28.0},  //cluster_02
  {36.0, 44.0, 20.0, 18.0},  //cluster_03
  {49.0, 63.5, 32.0, 23.0},  //cluster_04
  {50.5, 49.0, 29.0, 26.0},  //cluster_05
  {48.0, 32.0, 24.0, 28.0},  //cluster_06
  {65.0, 53.0, 20.0, 34.0},  //cluster_07
  {62.5, 32.0, 25.0, 28.0},  //cluster_08
  {55.5, 20.5, 39.0, 15.0},  //cluster_09
  {62.0, 10.0, 28.0, 31.0},  //cluster_10
};

double[] targetSpeed = {1.0, 5.0, 4.0, 7.0, 5.0, 6.0, 6.0, 6.0, 7.0, 5.0};

// enemy order of battle
long scenario_Seed            = 12345;
int max_NumberOfTargets       = 100;
double[] eOB_Factor           = {0.45, 0.50, 0.60, 0.65};
```

Figure 15. Variable and Parameter setup used for Analysis with different PLOS parameters

The output generated with the three different terrains is shown in Table 11. It is worth mentioning that this time the target distribution was not changed at all. Each run was generated with the seed 45832 and the corresponding target distribution can be seen in Table 4.

Set up 1

-----	Platf	Alloc	-----			
	P1	P2	P3	P4	P5	P6
C01	--	--	--	10	--	10
C02	16	16	--	08	--	--
C03	--	20	10	--	--	--
C04	--	--	--	02	04	--
C05	--	--	--	02	04	--
C06	--	--	--	03	03	--
C07	--	--	--	--	04	08
C08	--	--	--	--	04	08
C09	--	--	--	02	02	--
C10	--	--	--	--	06	03

Set up 2

-----	Platf	Alloc	-----			
	P1	P2	P3	P4	P5	P6
C01	20	20	--	10	--	--
C02	08	--	--	--	04	--
C03	--	06	--	--	03	--
C04	--	--	--	06	03	--
C05	--	--	06	06	--	12
C06	--	--	02	02	04	--
C07	--	--	--	--	04	08
C08	--	--	--	01	02	--
C09	--	--	--	02	04	--
C10	--	--	--	--	03	06

Set up 3

-----	Platf	Alloc	-----			
	P1	P2	P3	P4	P5	P6
C01	20	20	--	10	--	--
C02	14	14	--	07	--	--
C03	--	02	--	--	04	--
C04	--	--	08	--	04	--
C05	--	--	--	10	--	10
C06	--	--	04	--	04	--
C07	--	--	--	--	04	04
C08	--	--	--	--	03	03
C09	--	--	--	--	04	08
C10	--	--	--	--	04	08

Table 11. Allocation Output for different types of terrain

The variability between the three different tables is not as significant as it was in the previous chapter. Some of the allocations are equal. On the other hand it is quite obvious that terrain modeled as Probability of Line of Sight has a significant effect on JSAM and it's output. The reason why platform one is used much less than all the other platforms is exactly the same as in the previous subchapter. Its operation time is set to be just two hours and its operational range is just 16 km. It is capable to operate for a limited time in clusters one and two and it is not capable to reach the other clusters at all.

IV. CONCLUSIONS AND RECOMMENDATIONS

The Java SAM allocates sensor platforms to target clusters on the battlefield. The model ensures that platforms have sufficient range, time on station, and performance level if it allocates it to a specific cluster in a specific enemy order of battle scenario. Two major improvements were made. First, the JWARS PLOS methodology was implemented to represent the effects of terrain on sensor performance. Second, the sweep width sub-model was refined to incorporate the constraints on sensor performance imposed by the sensor's Field of View. These improvements result in a significantly more accurate representation of sensor performance in the model.

A. RECOMMENDED MODEL REFINEMENTS

1. Classified Data

Classified sensor data is available from AMSAA sources. The AMSAA data was reviewed to determine type and format available and a surrogate data set was generated to mirror the classified data and to develop our models. The assumption was made in development of the optimization models that platform and package performance data would become available as further experimentation and research is conducted. The availability of such data would eliminate the need to surrogate.

2. Tuning

A good mathematical model should always build on and be checked against historical data. Data that describes how many targets were detected by some platform on a battlefield would be perfect to tune this model. Checking against such a data set would make sure that the results produced are reliable and usable.

3. Verification and Validation

Verification and Validation have not been conducted on this model. TRAC-Monterey is in the process of developing the Dynamic Allocation of Fires and Sensor (DAFS) simulation. The JSAM output can directly be used as input to the DAFS model. Validation would be accomplished by comparing the performance (in DAFS or other simulations) of sensor allocations suggested by JSAM to those derived by other means.

4. Model Split

At the moment the model mixes the use of Unmanned Aerial and Unmanned Ground Operating Vehicles. Both systems are modeled differently but resulting Sweep Width for both models is used as a decision parameter for the optimization part. The model might be much more reliable if the two types of platforms were handled separately by the model.

B. SUGGESTED FURTHER RESEARCH

Many possibilities can be pursued to extend or to improve the model presented in this thesis. This thesis only addresses the allocation of organic Unit of Action sensor assets. The model could be modified to include joint assets at the higher echelons. At the moment the model also just uses a single entry point as the single coordinate for all available sensors. To represent dislocated troops on the battlefield realistically many sensor locations need to be handled by the model.

Another more challenging project would be to create a dynamic model significantly improving the utility of the presented model. There are two components for consideration in the development of a dynamic model. The inclusion of multiple time periods would take into account equipment or platform resupply, attrition rates, maintenance, follow-on missions, and new launch sites. The second more difficult component would involve the allocation of a package to a higher priority or just-identified target area. This reallocation would also apply to reallocation to a secondary target area if the allocation to the original target area was no longer required [Tutton, 2003].

LIST OF REFERENCES

1. Tutton, Stephanie, [2003] Optimizing the Allocation of Sensor Assets for the Unit of Action, Monterey: Naval Postgraduate School, Master's Thesis

2. Wagner, Daniel H., [1999] Naval Operations Analysis, 3rd Edition, Naval Intitute Press

3. Blacksten, H. Ric, Burdick, Charles D., Grell, Mihly G. [2002], Probability of Line-of-Sight (PLOS) and Roughness Enhancements for JWARS 1.5, San Jose: INFORMS/MAS

4. Tutton, Stephanie, Olson, Keith, [2003], SMM Preprocessor.xls, Monterey, Naval Postgraduate School, Excel Spreadsheet Model

5. Director US Army Materiel Systems Analysis Activity (AMSAA) [2003], Army Future Combat Systems Unit of Action Systems Book Version 3.0, Aberdeen: AMSAA

6. Rutherford, Thomas F., [2003], Rutherford's GAMS Programming Tools 2003, Denver: University of Colorado, Department of Economics, Available: http://debreu.colorado.edu/inclib/tools.htm

7. McCarl, Bruce A. McCarl's GAMS User Guide 2003, Texas: A&M University, Department of Agricultural Economics Available: http://www.gams.com/dd/docs/bigdocs/gams2002/

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A OPTIMIZATION MODEL IN NPS FORMAT

The mixed-integer program makes the best overall allocation of packages based on the mix available, taking into account the characteristic weightings of each package, target type weights, and sensor/platform performance. The decision variables for the Sensor Allocation Model are integer and indicate how many sensor packages of a certain type to allocate to a target cluster.

A. INDICES

p	platform type	{'UAV', 'ARV', 'UGS',...}
k	package configuration	{'K1', 'K2', 'K3',...}
t	target type	{'INF', 'Main Battle Tank', ...}
c	target cluster	{'C1', 'C2', 'C3',...}
ch	sensor characteristic	{'latency', 'cost', 'logistics', ...}
w	outcome_scenario	{'W1', 'W2', 'W3', ...}
eob	enemy order of battle	{'EOB1', 'EOB2', 'EOB3', ...}
n	number of packages of type *k* to cluster *c*	{' N1','N2', ..., 'N10'}

B. PARAMETERS

1. Asset Data

$plat\_pkg_{p,k}$	number of platforms of type *p* required for one package of type *k*
p_avail_{p}	number of platforms of type *p* available (inventory)
$pkg\_char_{k,ch}$	value of package *k* contribution to each characteristic *ch*
$cdp_{c,t,k,w}$	Cumulative Detection Probability for package *k* against target *t* type in cluster *c* in outcome *w*

2. Target Data

$num\_tgt_{t,c,eob}$	number of targets of type *t* in cluster *c* for a specific *eob*

3. Parameter Weights

wt_tgt_{t}	value of detecting target type *t*
wt_char_{ch}	platform characteristic weights *ch*
pr_eob_{eob}	probability of a specific *eob* occurring

pr_out_w	probability of a specific *w* occurring
alpha_det	overall weight for expected targets detected portion of the objective function
alpha_char	overall weight for characteristic portion of the objective function

4. Derived Data

$cdpe_{c,t,k,w,n}$	Cumulative Detection Probability Enumerated for *n* packages of type *k* against target type *t* in cluster *c* in outcome *w*

The model inputs are CDPs (indexed by target cluster, target type, package, and outcome) for one package, and the MIP precomputes the CDPs for assignment of up to ten packages of a single type assigned to a target cluster against a specific target type and indexes them by n.

C. DECISION VARIABLES

Unrestricted continuous variables in the model:

$CH\_OBJ_{ch,w}$	value of characteristic weights *ch* over all packages assigned to all clusters for an outcome *w*
$EXP\_TGT_{t,c,w}$	expected number of targets detected by target type *t* in cluster *c* for outcome *w*
OBJ	objective function value

Integer variables in the model:

$KTOC_{c,k,w}$	Integer Variable: number of packages of type *k* assigned to cluster *c* in outcome *w*

Binary variables in the model:

$$IND\_VAR_{c,k,n,w} = \begin{cases} 1 \text{ if } n \text{ packages of type } k \text{ are assigned to cluster } c \text{ in outcome } w \\ 0 \text{ otherwise} \end{cases}$$

The key decision variables in the Sensor Allocation Model are integer and allow for the selection of which consolidated package, and how many are assigned to each target cluster.

D. CONSTRAINTS

The model requires two main constraints. The first constraint set ensures that only one package type (regardless of configuration) is assigned to a target cluster.

$$\sum_{k,n} IND\_VAR_{c,k,n,w} \leq 1; \qquad \forall w, c$$

$$KTOC_{c,k,w} = \sum_{n} ord(n) * IND\_VAR_{c,k,n,w} \qquad \forall c, k, w$$

The second constraint ensures that only available platforms are used:

$$\sum_{c,k} plat\_pkg_{p,k} * KTOC_{c,k,w} \leq p\_avail_{p}; \qquad \forall p, w$$

The next two constraints calculate terms in the objective function.

$$\mathrm{EXP\_TGT_{t,c,w}} = \sum_{eob,k,n} num\_tgt_{c,t,eob} * wt\_tgt_{t} * pr\_eob_{eob} * cdpe_{c,t,w,k,n} * IND\_VAR_{c,k,w,n}$$

$$CH\_OBJ_{ch,w} = \sum_{c} \sum_{k} KTOC_{c,k,w} * pkg\_char_{k,ch} * wt\_char_{ch,w}$$

The final constraint defines the objective as a weighted combination of expected, weighted targets detected and weighted sensor characteristics.

$$OBJ = alpha\_det * EXP\_TGT_{t,c,w} - alpha\_char * OBJ\_CH_{ch,w}$$

E. OBJECTIVE FUNCTION

The objective in this model is to maximize the weighted combination of expected number of weighted detections and overall sensor characteristic penalties.

Maximize *OBJ*

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B DOCUMENTATION OF THE JAVA SENSOR ALLOCATION PROGRAM

A. JSAM DOCUMENTATION AND PROGRAM DESCRIPTION

In this subchapter we describe JSAM, its classes and methods in more detail. See Appendix ( ) for the JSAM code. 'Input ModelData' is used to handle and capture all the data input. It is subdivided into several paragraphs. These are for Parameters to model the Probability of Line of Sight, the Sweep Width of a Platform, the Package Combinations and Characteristics as well as the Enemy Order of Battle. Cluster Coordinates and Cluster Dimensions are defined here too. The idea is that all input and modeling data can be set up and changed in one single place. An excerpt that shows how some of the parameters mentioned above are defined is shown in Figure 16.

To make sure that whenever something is changed in the interface all classes in the program are compiled again, the text file 'PackageClasses.txt' was added to the package folder. This file contains the names of all classes in the program and the command 'javac @PackageClasses' compiles all classes listed in the text file. Compiling just the interface is insufficient for a correct JSAM run.

```
// parameters for modeling PLOS
double sigma_ceiling           = 14.5;
double g_rgh                   = 0.000704;
double c_ceiling               = 0.000605;
double g_xtnct                 = 0.569;

double[] operatonalAltitude           = {150.0, 300.0, 600.0, 2000.0, 3.0, 1.0};
double[] operatonalObservationRange   = {150.0, 300.0, 600.0, 2000.0, 3.0, 1.0};

// parameters for modleing platform SweepWidth
double[][] horizontalAngle   = {
   {20.00, 15.00},
   {20.00, 9.200},
   {8.920, 8.900},
   {8.920, 8.900},
   {20.00, 15.00},
   {20.00, 15.00}
   };                    //[Platform][Sensor] horizontal observation angle

double[][] verticalAngle  = {
   {15.00, 11.25},
   {15.00, 6.900},
   {6.680, 6.700},
   {6.680, 6.700}
   };                    //[Platform][Sensor] vertical observation angle

double[] observationAngle = {45.0, 45.0, 45.0, 45.0};   //[Platform]

double swag_dependance = 0.1;
```

Figure 16. Excerpt of the class Input_ModelData

The Sensors class handles data that has to do with sensors mounted on a platform. It provides six methods. The first one is called sensorData. It is used to read the actual sensor performance data into the Java Program. For this purpose the data file was restructured similar to a network data file. The two numbers in the first row define width and height of the data block. The first column is for the platform, the second is for the sensor and the third column specifies different types of targets. '1 2 1' in row three for instance means Sensor 2 on platform 1 observing target 1. The next three columns are for the ranges within which data are provided. Column four is for start range, column five is for end range and column six defines the range increments in which data is provided. The combination '0 1200 100,' for instance, means that data is provided between 0 and 1200 meters in increments of 100 meters. All other columns provide data for a specific sensor on a specific platform against a specific target at a specific range. The name of the file is 'SENSOR_DATA.CSV'.

27	120										
1	1	1	0	6000	500	1	0.911596	0.978829	0.957341	0.967688	0.924096
1	2	1	0	1200	100	1	0.991558	0.951871	0.898665	0.931452	0.917393
1	1	2	0	6000	500	1	0.991627	0.964177	0.908344	0.89414	0.93644
1	2	2	0	1200	100	1	0.957563	0.969453	0.961055	0.917477	0.875669
1	1	3	0	6000	500	1	0.908911	0.960705	0.914058	0.918571	0.883918
1	2	3	0	1200	100	1	0.980121	0.910741	0.947372	0.852149	0.823776
1	1	4	0	2400	200	1	0.989058	0.900173	0.948794	0.936191	0.947119
1	2	4	0	1200	100	1	0.965566	0.928927	0.950795	0.919499	0.884167
1	1	5	0	2400	200	1	0.999963	0.919988	0.900865	0.885747	0.859609
1	2	5	0	1200	100	1	0.918437	0.982633	0.912627	0.955035	0.892936
1	1	6	0	2400	200	1	0.921525	0.92376	0.985256	0.89319	0.816344
1	2	6	0	1200	100	1	0.969328	0.95034	0.928775	0.859611	0.769106
1	1	7	0	6000	500	1	0.948291	0.930717	0.949882	0.911142	0.937599
1	2	7	0	1200	100	1	0.97779	0.948167	0.970712	0.981202	0.895133
1	1	8	0	6000	500	1	0.952212	0.946774	0.930221	0.950682	0.935158
1	2	8	0	1200	100	1	0.904197	0.972343	0.95461	0.954945	0.881419
1	1	9	0	6000	500	1	0.970761	0.907008	0.905394	0.906715	0.876577
1	2	9	0	1200	100	1	0.980482	0.947848	0.91017	0.867379	0.774993
1	1	10	0	2400	200	1	0.957574	0.969968	0.904337	0.916768	0.951048
1	2	10	0	1200	100	1	0.99842	0.92159	0.971495	0.891958	0.826662

Table 12. Excerpt of Sensor_Data.csv

JSAM stores this information in three multidimensional arrays, per platform, sensor, target and a range index. The method plosCorrectedData computes a correction factor to take Probability of Line of Sight into account. The basic theory behind this is described in chapter II A 1. The next three methods in this class are pictureDimensions, pictureDistances and pictureProbability. They are used to compute dimensions, distances and probabilities assigned to a picture taken by a sensor. These data are necessary to calculate the sweep width of a sensor, which finally is done by sensorSweepWidth.

The Platforms class handles all the calculations to produce platform data. Since this class extends Sensors it has access to its methods and instance variables. The latter is used extensively in this program. Platforms has three methods. The method platformSweepWidth computes the adjusted sweep width for a platform. Apart from a few necessary 'if-statements' it simply applies the following formula:

$$W_{adj} = \max(w_S) + \alpha_{plt} \cdot (\sum w_S - \max(w_S)) \quad (6)$$

W_{adj} is the adjusted sweep width for a platform. w_s represents the individual sweep width of a sensor and α_{plt} is the positive independence factor indicating added benefit of multiple sensors mounted on one platform. α_{plt} is one of the parameters that are defined in the interface class. Here it is called swag dependence. Typically it is a value between 0 and 1. 0 means that just the sensor with the maximum sweep width contributes to the platform sweep width. 1 means that the sweep width of all sensors mounted on a platform are added up. timeOnStation calculates the distance form the entry point to each of the clusters and finally the time on station for each platform. It also checks the operational radius of a platform. Is a platform out of a clusters range, its time on station for this cluster is set to be zero. PlatformCDP finally is used to compute the Cumulative Probability of Detection of a target within a specific cluster. The calculation relies on the following basic formula from random search theory:

$$CPD_{plt} = 1 - e^{-\frac{vW_{adj}t}{A}} \quad (7)$$

A is the total area of a specific cluster, v is the search speed and t is the time on station of the platform. Depending on whether the platform is stationary or moving the search speed v is either the actual search speed of the platform itself or the speed of the target that is observed.

A Package is a combination of Platforms which itself somehow is a combination of Sensors. It just makes sense that the class Packages extends Platforms as Platforms extends Sensors. This class has three main methods and two helper methods. The helper

methods are needed to compute ‘n factorial’ and the probabilistic formula ‘n choose k’. They are easy to understand and will not be explained here. packageCombinations adds up combinations of platforms to packages. This is done intelligently based on user input data. Firstly the method ads each of the rows in ‘basicPackages’ to each other row in this array. Then it adds each row multiplied by two to each other row in the array. Finally it adds each row to each other row in the array after this was multiplied by two. packageCharacteristics takes the arrays platformLatency, platformCost, platformLatency and platformCost as well as the results from the previous described method and computes the characteristics for the whole package. It just adds up all the values. The third method in this class finally calculates the Cumulative Probability of Detection of the package. The theory behind this is explained extensively in [Tutton, 2003]. The basic formulas used to compute the package CDP are shown below:

$$Rate_{plt} = vW_{adj}t \tag{8}$$

$$Effprop = \frac{Rate_{plt}}{\sum Rate_{plt}} \tag{9}$$

$$CDP_{pkg} = 1 - e^{-\frac{Rate_{plt}}{A \cdot Effprop \cdot \beta_{pkg}}} \tag{10}$$

Like in the formulas before v is defined to be the search speed, t is the time on station and W_{adj} is the adjusted sweep width for a platform. Effprop is the effective proportion of the cluster that is searched by the platform for which the Rate was calculated. Finally the Cumulative Detection Probability for each package is calculated. β_{pkg} is a positive dependence factor associated with a specific configuration of individual platforms designed as a package.

Scenario has two methods which both somehow produce target distributions. scenarioData uses the Java built in random number generator to produce a target distribution per cluster and targetDistribution uses the Enemy Order of Battle factors defined as user input data to vary the target distribution produced by the previous method. The eob-factor defines a uniform distribution that is used to decide whether a target is absent or present.

There are three more classes in the program that need to be explained. Two of them are for output handling and the third class takes care of the GAMS environment. Let us have a look at the output classes first. Output_Command holds all the print methods of the program that print tables and arrays to the command line, either for error checking or for a better understanding of the program itself. Which method is called or not can be defined in the constructor. By default just the final program result is printed out.

Output_GamsFiles produces the input files needed for the GAMS optimization. The class needs an instance variable of Type Scenario to have access to the data needed. Basically the methods in this class just transform the Java data into a format that can be red in by GAMS.

Gams_Handling finally is responsible for the GAMS environment. It uses the Java Runtime feature to start GAMS. The started application runs completely in the background. The output file created by the GAMS application is captured. A GAMS Plug In is used to modify the GAMS Output file. The new format is much easier to capture. The Plug In is called 'Gams2csv' and can be downloaded for free form Thomas F. Rutherford's GAMS programming tools homepage [Rutherford, 2003]. The following explanation is from Mc Carl's GAMS User Guide [Mc Carl, 2003]. GAMS users do not have to do put file programming to move data in CSV format. Rather they can use a libinclude routine called Gams2csv developed by Rutherford and associates at the University of Colorado.

Gams2csv is invoked as follows:

```
FILE localname /externalname/;
PUT localname;
$LIBInclude gams2csv [row domain [column domain] ] item[.suffix]
[item[.suffix] ...]
```

However the method allocationTable takes the GAMS output and reads its content. It stores assigned packages to clusters. This actually is the final result and output produced by JSAM.

B. DATA EXTRACTION AND DOCUMENTATION

To finally prepare JSAM for Implementation into a discrete event simulation all the modeling data that is set up within the interface has to be extracted and stored in text files. The class Read_DataFiles has to be added to read the generated text files back in. The advantage of this structure is that the simulation can generate the modeling input necessary for a Java Sensor Allocation run. The disadvantage of this setup is that it is not as user friendly as the setup with the interface. The four new files have the following names:

PLOS_MODELING.CSV
SCENARIO_MODELING.CSV
PACKAGE_MODELING.CSV
FIELD_OF_VIEW_MODELING.CSV

They keep all the input data needed for a Java Sensor Allocation run. Figure 17 shows the content of PLOS_MODELING.CSV. Although for a better reading the data below is shown tabulator delimited, the real data files are all comma delimited.

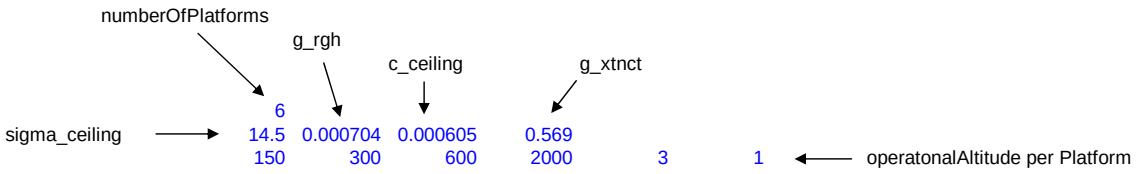

Figure 17. Description of plos_modeling

The number in row one is for the number of platforms used in the model, row two is for the four PLOS parameters and row three is for the operational altitude of the different platforms. Figure 18 shows the content of SCENARIO_MODELING.CSV.

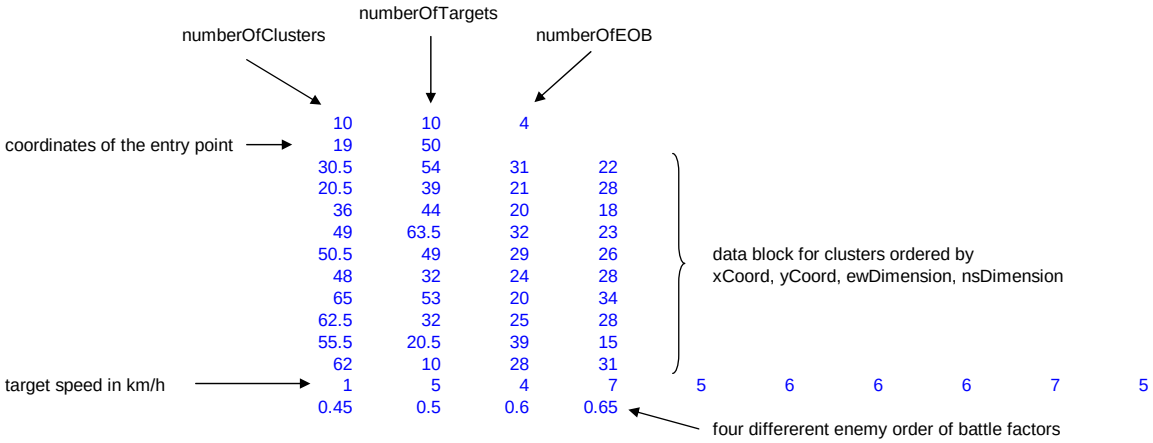

Figure 18. Description of scenario_modeling

The file provides x- and y-Coordinates for the entry point as well as Cluster Coordinates, Cluster Dimensions and speed values for the different types of targets used in the model. The last row shows the four eob parameters used to generate enemy order of battle setups. PACKAGE_MODELING is shown in Figure 19.

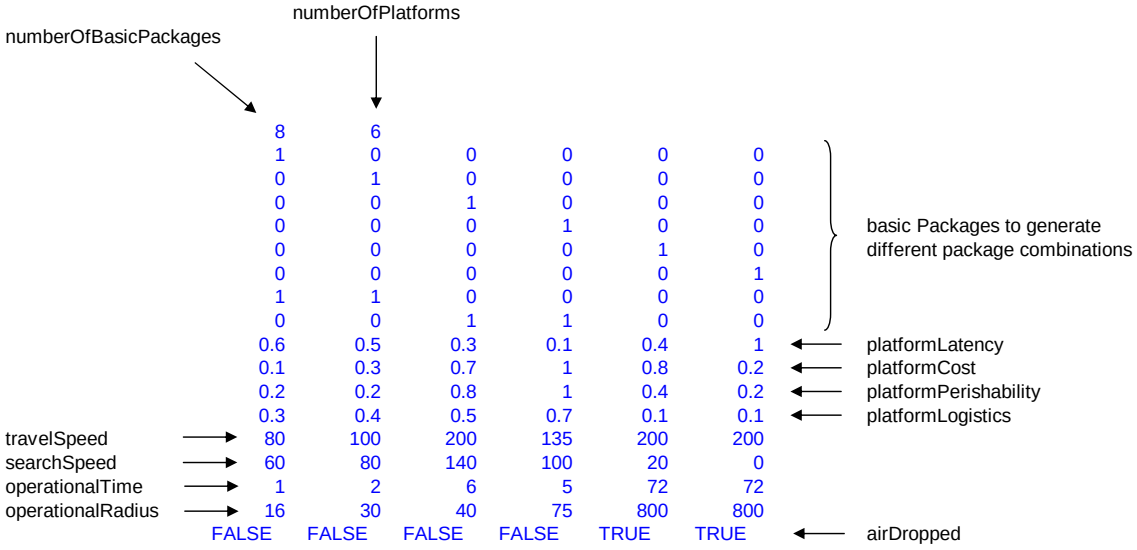

Figure 19. Description package_modeling

In the first row the number of basic packages that are shown in the next eight rows is defined. The columns in this file are for data related to different platforms. Data in column one is for platform one, data in column two is for platform two and so on. All other necessary explanations are shown in the picture.

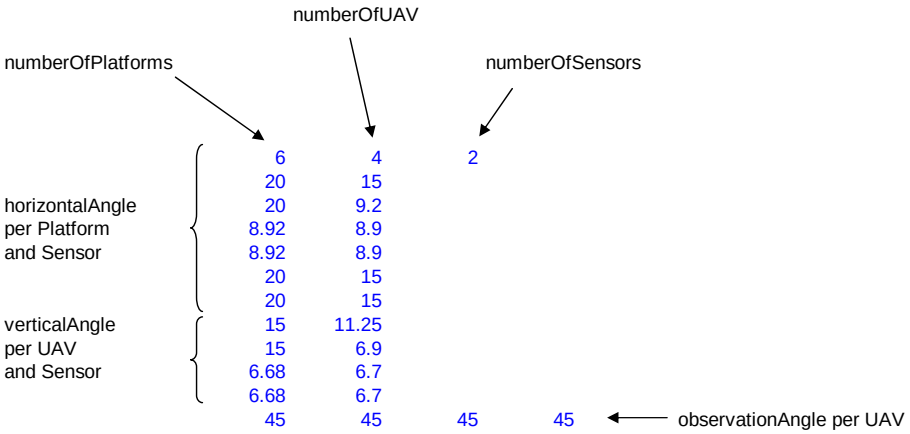

Figure 20. Description of field_of_view_modeling

This file provides data for horizontal angles per platform and sensor, vertical angles per UAV and Sensor. In our set up platforms 5 and 6 are not UAV's. Operating on the ground they are modeled just on basis of their horizontal angle. The observation angles for the four UAV's are shown in the last row.

APPENDIX C CODE FOR THE JAVA SENSOR ALLOCATION PROGRAM

A. MAIN

```
package jSAM;


/*
 * File: Main.java
 * Created: 25th October 2003, 18:00
 */

/**
 * @author Thomas Doll
 */

import java.util.*;
import java.io.*;
import java.text.*;

public class Main {

    public static void main(String[] args) {

        jSAM.Scenario scenario = new jSAM.Scenario();
        jSAM.Output_GamsFiles gamsFiles = new jSAM.Output_GamsFiles(scenario);
        jSAM.Gams_Handling gams = new jSAM.Gams_Handling(scenario);
        jSAM.Output_Command output = new jSAM.Output_Command(scenario, gams);

    }
}
```

B. INPUT_MODELDATA

```
package jSAM;

/*
 * File: Input_ModelData.java
 * Created: 25th November 2003, 18:00
 */

/**
 * @author Thomas Doll
 */

import java.text.DecimalFormat;

public interface Input_ModelData {

    // parameters for modeling PLOS
    double sigma_ceiling           = 14.5;
    double g_rgh                   = 0.000704;
    double c_ceiling               = 0.000605;
    double g_xtnct                 = 0.569;

    double[] operatonalAltitude            = {150.0, 300.0, 600.0, 2000.0, 3.0, 1.0};
    double[] operatonalObservationRange    = {150.0, 300.0, 600.0, 2000.0, 3.0, 1.0};

    // parameters for modleing platform SweepWidth
    double[][] horizontalAngle   = {
       {20.00, 15.00},
       {20.00, 9.200},
       {8.920, 8.900},
       {8.920, 8.900},
       {20.00, 15.00},
       {20.00, 15.00}
       };                  //[Platform][Sensor] horizontal observation angle
```

```
double[][] verticalAngle  = {
   {15.00, 11.25},
   {15.00, 6.900},
   {6.680, 6.700},
   {6.680, 6.700}
   };                    //[Platform][Sensor] vertical observation angle

double[] observationAngle = {45.0, 45.0, 45.0, 45.0};   //[Platform]

double swag_dependance = 0.1;

// parameters for modeling package combinations
int[][] basicPackages = {
   { 1, 0, 0, 0, 0, 0},    //PlatformCombination for Package 1
   { 0, 1, 0, 0, 0, 0},    //PlatformCombination for Package 2
   { 0, 0, 1, 0, 0, 0},    //PlatformCombination for Package 3
   { 0, 0, 0, 1, 0, 0},    //PlatformCombination for Package 4
   { 0, 0, 0, 0, 1, 0},    //PlatformCombination for Package 5
   { 0, 0, 0, 0, 0, 1},    //PlatformCombination for Package 6
   { 1, 1, 0, 0, 0, 0},    //PlatformCombination for Package 7
   { 0, 0, 1, 1, 0, 0},    //PlatformCombination for Package 8
};

double package_enhFact = 1.1;

// parameters for modleing Package Characteristics
double[] platformLatency         = {0.6, 0.5, 0.3, 0.1, 0.4, 1.0};
double[] platformCost            = {0.1, 0.3, 0.7, 1.0, 0.8, 0.2};
double[] platformPerishability   = {0.2, 0.2, 0.8, 1.0, 0.4, 0.2};
double[] platformLogistics       = {0.3, 0.4, 0.5, 0.7, 0.1, 0.1};

double[] travelSpeed          = {80.0,  100.0, 200.0, 135.0,  200.0, 200.0};
double[] searchSpeed          = {60.0,  80.0,  140.0, 100.0,  20.0,  0.0  };
double[] operationalTime      = {1.0,   2.0,   6.0,   5.0,    72.0,  72.0 };
double[] operationalRadius    = {16.0,  30.0,  40.0,  75.0,   800.0, 800.0};

boolean[] airDropped       = {false, false, false, false,  true,  true,};

// cluster coordinates and dimensions
double[] entryPoint = {19.0, 50.0};

double[][] clusterData = {
   {30.5, 54.0, 31.0, 22.0},  //cluster_01 {xCoord, yCoord, ewDimension, nsDimension}
   {20.5, 39.0, 21.0, 28.0},  //cluster_02
   {36.0, 44.0, 20.0, 18.0},  //cluster_03
   {49.0, 63.5, 32.0, 23.0},  //cluster_04
   {50.5, 49.0, 29.0, 26.0},  //cluster_05
   {48.0, 32.0, 24.0, 28.0},  //cluster_06
   {65.0, 53.0, 20.0, 34.0},  //cluster_07
   {62.5, 32.0, 25.0, 28.0},  //cluster_08
   {55.5, 20.5, 39.0, 15.0},  //cluster_09
   {62.0, 10.0, 28.0, 31.0},  //cluster_10
};

double[] targetSpeed = {1.0, 5.0, 4.0, 7.0, 5.0, 6.0, 6.0, 6.0, 7.0, 5.0};

// enemy order of battle
long scenario_Seed            = 12345;
int max_NumberOfTargets       = 100;
double[] eOB_Factor           = {0.45, 0.50, 0.60, 0.65};

// common used objects
DecimalFormat formatProb            = new DecimalFormat("0.000 ");
DecimalFormat formatIdentifier      = new DecimalFormat("00 ");
DecimalFormat formatNumber          = new DecimalFormat("000 ");
DecimalFormat formatPackage         = new DecimalFormat("0000 ");
DecimalFormat formatRange           = new DecimalFormat("00000 ");
DecimalFormat formatCharacteristics = new DecimalFormat("0.0 ");
DecimalFormat formatTime            = new DecimalFormat("00.00 ");
DecimalFormat formatSweepWidth      = new DecimalFormat("00000 ");
DecimalFormat formatCDP             = new DecimalFormat("0.000 ");
DecimalFormat formatRate            = new DecimalFormat("0000.00 ");
DecimalFormat formatProb2           = new DecimalFormat("0.00000");
DecimalFormat formatIdentifier2     = new DecimalFormat("00");
DecimalFormat formatNumber2         = new DecimalFormat(" 0  ");
DecimalFormat formatCoord           = new DecimalFormat("000.000 ");
DecimalFormat formatNumber3         = new DecimalFormat(" 0 ");

// control panel
```

```
    boolean debugging_Mode          = false; //for error checking
    boolean print_Verbose           = true;  //prints everything to the command line
    boolean use_operationalRadius   = true;  //use operational radius or not
    boolean run_Mode                = true;  //prints some information about the run

} // end Input_ModelData interface
```

C. SENSORS

```
package jSAM;


/*
 * File: Sensor.java
 * Created: 25th November 2003, 18:00
 */

/**
 * @author Thomas Doll
 */

import java.util.*;
import java.io.*;
import java.text.*;


public class Sensors implements Input_ModelData {
    // class constants
    final static String sensorDataFile = "SENSOR_DATA_MOD.CSV";

    // class variables
    // instance variables
    private int numberOfPlatforms;
    private int numberOfTargets;
    private int sensorDataHight;
    private int sensorDataWidth;

    private int[] numberOfSensors;              // [Platform] Indicates the number of
                                                   sensors per platform

    private double[][] centerPictureWidth;      // [Platform][Sensor]
    private double[][] sidePictureWidth;        // [Platform][Sensor]
    private double[][] pictureLength;           // [Platform][Sensor]
    private double[][] shortCPictDist;          // [Platform][Sensor] shortest ground
                                                   Distance form UAV to Center Picture
    private double[][] longCPictDist;           // [Platform][Sensor] longest ground
                                                   Distance form UAV to Center Picture
    private double[][] shortSPictDist;          // [Platform][Sensor] shortest ground
                                                   Distance form UAV to one of the Side
                                                   Picture
    private double[][] longSPictDist;           // [Platform][Sensor] longest ground
                                                   Distance form UAV to one of the Side
                                                   Picture

    private double[][][] toRangeArray;             // [Platform][Sensor][Target]
    private double[][][] centerPictureProbability; // [Platform][Sensor][Target]
    private double[][][] sidePictureProbability;   // [Platform][Sensor][Target]
    private double[][][] sensorSweepWidth;         // [Platform][Sensor][Target]

    private double[][][][] rangeData;          // [Platform][Sensor][Target][RangeIndex]
    private double[][][][] sensorData;         // [Platform][Sensor][Target][SensorIndex]
    private double[][][][] plosCorrectedData;  // [Platform][Sensor][Target][Index]

    // class methods
    // constructor methods
    /**
     * constructor for parameter inputString.
     */
    public Sensors() {
        // this if for debugging
        if (run_Mode) {
            System.out.println("\nJAVA SENSOR ALLOCATION MODEL (JSAM)
                                NAVAL POSTGRADUATE SCHOOL");
            System.out.println("    Version: 1.5");
            System.out.println("    Date   : January 19 2004");
            System.out.println("    Author : Dipl.-Ing. Thomas Doll tdoll@nps.navy.mil");
            System.out.println("    Contact: tdoll@nps.navy.mil");
```

```
            System.out.println("\n->Start Preprocessor");
        }
        if (debugging_Mode) {
            System.out.println("\n-------------- Debugging Output for Class Sensors -----
                               ----------");
        }

        sensorData();

        this.centerPictureWidth = new double[getNumberOfUAV()][];
        this.sidePictureWidth = new double[getNumberOfUAV()][];
        this.pictureLength = new double[getNumberOfUAV()][];
        pictureDimensions();

        this.shortCPictDist = new double[getNumberOfUAV()][];
        this.longCPictDist = new double[getNumberOfUAV()][];
        this.shortSPictDist = new double[getNumberOfUAV()][];
        this.longSPictDist = new double[getNumberOfUAV()][];
        pictureDistances();

        plosCorrectedData();
        pictureProbability();
        sensorSweepWidth();
    }

    // instance methods

    // reads the sensor data into the new data structure from a textfile
    public void sensorData() {
        String          inputString;
        FileReader      inputFile;
        BufferedReader  inputUnit;
        StringTokenizer tk;

        // this is to predefine the values for numberOfPlatforms and numberOfTargets
        try {
            inputFile = new FileReader(sensorDataFile);
            inputUnit = new BufferedReader(inputFile);
            inputString = inputUnit.readLine();

            tk = new StringTokenizer(inputString, ",");

            sensorDataWidth = Integer.parseInt(tk.nextToken());
            sensorDataHight = Integer.parseInt(tk.nextToken());

            for (int i = 0; i < sensorDataHight; i++) {
                inputString = inputUnit.readLine();
                tk = new StringTokenizer(inputString, ",");

                int p = (int) Double.parseDouble(tk.nextToken());
                int s = (int) Double.parseDouble(tk.nextToken());
                int t = (int) Double.parseDouble(tk.nextToken());

                if (p > numberOfPlatforms) {
                    numberOfPlatforms = p;
                }
                if (t > numberOfTargets) {
                    numberOfTargets = t;
                }
            }
            inputUnit.close();
        } catch (FileNotFoundException e) {
            System.out.println(e);
            return;
        } catch (IOException e) {
            System.out.println(e);
            return;
        }

        // this is to predefine the values for numberOfSensors
        numberOfSensors = new int[numberOfPlatforms];
        try {
            inputFile = new FileReader(sensorDataFile);
            inputUnit = new BufferedReader(inputFile);
            inputString = inputUnit.readLine();

            tk = new StringTokenizer(inputString, ",");

            for (int i = 0; i < sensorDataHight; i++) {
```

```
            inputString = inputUnit.readLine();
            tk = new StringTokenizer(inputString, ",");

            int p = (int) Double.parseDouble(tk.nextToken());
            int s = (int) Double.parseDouble(tk.nextToken());
            int t = (int) Double.parseDouble(tk.nextToken());

            if (s > numberOfSensors[p - 1]) {
                numberOfSensors[p - 1] = s;
            }
        }
        inputUnit.close();
    } catch (FileNotFoundException e) {
        System.out.println(e);
        return;
    } catch (IOException e) {
        System.out.println(e);
        return;
    }

    // this is to define the the size of rangeData[][][][], sensorData[][][][],
       plosCorrectedData[][][][], pictureProbability[][][][],
    // and sensorSweepWidth[][][]
    this.rangeData = new double[numberOfPlatforms][][][];
    this.sensorData = new double[numberOfPlatforms][][][];
    this.plosCorrectedData = new double[numberOfPlatforms][][][];
    this.centerPictureProbability = new double[numberOfPlatforms][][];
    this.sidePictureProbability = new double[numberOfPlatforms][][];
    this.toRangeArray = new double[numberOfPlatforms][][];
    this.sensorSweepWidth = new double[numberOfPlatforms][][];
    try {
        inputFile = new FileReader(sensorDataFile);
        inputUnit = new BufferedReader(inputFile);
        inputString = inputUnit.readLine();

        tk = new StringTokenizer(inputString, ",");

        for (int i = 0; i < sensorDataHight; i++) {
            inputString = inputUnit.readLine();
            tk = new StringTokenizer(inputString, ",");

            int p = (int) Double.parseDouble(tk.nextToken());

            this.rangeData[p - 1] = new double[numberOfSensors
                                    [p - 1]][numberOfTargets][sensorDataWidth - 6];
            this.sensorData[p - 1] = new double[numberOfSensors
                                    [p - 1]][numberOfTargets][sensorDataWidth - 6];
            this.plosCorrectedData[p - 1] = new double[numberOfSensors
                                    [p - 1]][numberOfTargets][sensorDataWidth - 6];
            this.centerPictureProbability[p - 1] = new double[numberOfSensors
                                    [p - 1]][numberOfTargets];
            this.sidePictureProbability[p - 1] = new double[numberOfSensors
                                    [p - 1]][numberOfTargets];
            this.toRangeArray[p - 1] = new double[numberOfSensors
                                    [p - 1]][numberOfTargets];
            this.sensorSweepWidth[p - 1] = new double[numberOfSensors
                                    [p - 1]][numberOfTargets];
        }
        inputUnit.close();
    } catch (FileNotFoundException e) {
        System.out.println(e);
        return;
    } catch (IOException e) {
        System.out.println(e);
        return;
    }

    // this finally is to read in the data
    try {
        inputFile = new FileReader(sensorDataFile);
        inputUnit = new BufferedReader(inputFile);
        inputString = inputUnit.readLine();

        tk = new StringTokenizer(inputString, ",");

        for (int i = 0; i < sensorDataHight; i++) {
            inputString = inputUnit.readLine();
            tk = new StringTokenizer(inputString, ",");

            int p = (int) Double.parseDouble(tk.nextToken());
```

```
                int s = (int) Double.parseDouble(tk.nextToken());
                int t = (int) Double.parseDouble(tk.nextToken());
                double fromRange = Double.parseDouble(tk.nextToken());
                double toRange = Double.parseDouble(tk.nextToken());
                double increment = Double.parseDouble(tk.nextToken());
                double range = fromRange;

                toRangeArray[p - 1][s - 1][t - 1] = toRange;

                for (int j = 6; j < sensorDataWidth; j++) {
                    if (range <= toRange) {
                        rangeData[p - 1][s - 1][t - 1][j - 6] = range;
                        range += increment;
                    } else {
                        rangeData[p - 1][s - 1][t - 1][j - 6] = 0.0;
                    }
                    sensorData[p - 1][s - 1][t - 1][j - 6] =
                                                    Double.parseDouble(tk.nextToken());

                }

            }

            if (debugging_Mode) {
                for (int p = 0; p < getNumberOfPlatforms(); p++) {
                    for (int s = 0; s < getNumberOfSensors(p); s++) {
                        for (int t = 0; t < getNumberOfTargets(); t++) {
                            for (int index = 0; index < getSensorDataSize(); index++) {
                                // System.out.println("rangeData[p-1][s-1][t-1][j-6]= " +
                                                        rangeData[p-1][s-1][t-1][j-6]);
                                System.out.println("sensorData[p-1][s-1][t-1][j-6]= " +
                                                    sensorData[p][s][t][index]);
                                // System.out.println("getSensorData[p-1][s-1][t-1][j-6]=
                                                " + getSensorData(p-1,s-1,t-1,index));
                            }
                        }
                    }
                }

                inputUnit.close();
            } catch (FileNotFoundException e) {
                System.out.println(e);
                return;
            } catch (IOException e) {
                System.out.println(e);
                return;
            }
        }

        // PLOS correction of sensorData
        public void plosCorrectedData() {
            double slantRange = 0.0;

            for (int p = 0; p < getNumberOfPlatforms(); p++) {
                for (int s = 0; s < getNumberOfSensors(p); s++) {
                    for (int t = 0; t < getNumberOfTargets(); t++) {
                        for (int index = 0; index < getSensorDataSize(); index++) {
                            slantRange = Math.sqrt(rangeData[p][s][t][index] *
                                        rangeData[p][s][t][index] + operatonalAltitude[p] *
                                        operatonalAltitude[p]);
                            double sigma_rgh = sigma_ceiling * (1 - Math.exp(-g_rgh *
                                        slantRange));
                            double c_xtnct = c_ceiling * (1 - Math.exp(-g_xtnct * sigma_rgh /
                                        operatonalAltitude[p]));
                            double plos = Math.exp(-c_xtnct * slantRange);

                            plosCorrectedData[p][s][t][index] = sensorData[p][s][t][index] *
                                        plos;
                        }
                    }
                }
            }
        }

        // method to compute the Dimensions of a single observaton picture taken by a
    platform
        public void pictureDimensions() {
            for (int p = 0; p < getNumberOfUAV(); p++) {
                this.centerPictureWidth[p] = new double[getNumberOfSensors(p)];
```

```
            this.sidePictureWidth[p] = new double[getNumberOfSensors(p)];
            this.pictureLength[p] = new double[getNumberOfSensors(p)];
            for (int s = 0; s < getNumberOfSensors(p); s++) {
                double slantRange = 0.0;
                double groundDist = 0.0;

                // this is for the width of the areas
                slantRange = operatonalAltitude[p] /
                             Math.cos(Math.toRadians(observationAngle[p]));
                centerPictureWidth[p][s] = 2 * (slantRange *
                             Math.tan(Math.toRadians(horizontalAngle[p][s] / 2)));
                sidePictureWidth[p][s] = slantRange * Math.tan(Math.toRadians(3 *
                             horizontalAngle[p][s] / 2)) - centerPictureWidth[p][s] / 2;

                // this is for the length of the areas
                groundDist = operatonalAltitude[p] *
                             Math.tan(Math.toRadians(observationAngle[p] –
                             verticalAngle[p][s] / 2));
                pictureLength[p][s] = (operatonalAltitude[p] *
                             Math.tan(Math.toRadians(observationAngle[p] +
                             verticalAngle[p][s] / 2))) - groundDist;
            }
        }
    }

    // method to compute the Ground Distancies from the UAV to the different Pictures
    public void pictureDistances() {
        for (int p = 0; p < getNumberOfUAV(); p++) {
            this.shortCPictDist[p] = new double[getNumberOfSensors(p)];
            this.longCPictDist[p] = new double[getNumberOfSensors(p)];
            this.shortSPictDist[p] = new double[getNumberOfSensors(p)];
            this.longSPictDist[p] = new double[getNumberOfSensors(p)];
            for (int s = 0; s < getNumberOfSensors(p); s++) {
                double slantRange = 0.0;
                double groundDist = 0.0;

                // this is for the distancies to the center picture
                shortCPictDist[p][s] = operatonalAltitude[p] *
                                       Math.tan(Math.toRadians(observationAngle[p]));

                if (debugging_Mode) {
                    System.out.println("p= " + p + " operatonalAltitude[p] " +
                                       operatonalAltitude[p]);
                    System.out.println("p= " + p + " observationAngle[p] " +
                                       observationAngle[p]);
                    System.out.println("p= " + p + " s= " + s + " shortCPictDist[p][s]= "
                                       + shortCPictDist[p][s]);
                }

                longCPictDist[p][s] = Math.sqrt((shortCPictDist[p][s] +
                                      pictureLength[p][s]) * (shortCPictDist[p][s] +
                                      pictureLength[p][s]) + (centerPictureWidth[p][s] /
                                      2) * (centerPictureWidth[p][s] / 2));

                // this is for the distancies to the side picture
                shortSPictDist[p][s] = Math.sqrt(shortCPictDist[p][s] *
                                       shortCPictDist[p][s] + (centerPictureWidth[p][s] /
                                       2) * (centerPictureWidth[p][s] / 2));
                longSPictDist[p][s] = Math.sqrt((shortCPictDist[p][s] +
                                      pictureLength[p][s]) * (shortCPictDist[p][s] +
                                      pictureLength[p][s]) + (3 *
                                      centerPictureWidth[p][s] / 2) * (3 *
                                      centerPictureWidth[p][s] / 2));
            }
        }
    }

    // method to compute the mean probability of detection in a picture
    public void pictureProbability() {
        double shortCenterProb = 0.0;
        double longCenterProb = 0.0;
        double shortSideProb = 0.0;
        double longSideProb = 0.0;

        for (int p = 0; p < getNumberOfUAV(); p++) {
            for (int s = 0; s < getNumberOfSensors(p); s++) {
                for (int t = 0; t < getNumberOfTargets(); t++) {
                    for (int index = 0; index < getSensorDataSize() - 1; index++) {

                        if (shortCPictDist[p][s] >= toRangeArray[p][s][t]) {
```

```
                shortCenterProb = 0.0;
            } else if (shortCPictDist[p][s] >= rangeData[p][s][t][index] &&
                    shortCPictDist[p][s] < rangeData[p][s][t][index + 1])
                    {
                // interpolation
                double probDiff = Math.abs(plosCorrectedData[p][s][t][index]
                        - plosCorrectedData[p][s][t][index + 1]);
                double rangeDiff = Math.abs(rangeData[p][s][t][index + 1] -
                        rangeData[p][s][t][index]);
                double rangeIntercept = Math.abs(shortCPictDist[p][s] -
                        rangeData[p][s][t][index]);
                double probIntercept = (probDiff / rangeDiff) *
                        rangeIntercept;

                shortCenterProb = plosCorrectedData[p][s][t][index] -
                        probIntercept;
            }

            if (longCPictDist[p][s] >= toRangeArray[p][s][t]) {
                longCenterProb = 0.0;
            } else if (longCPictDist[p][s] <= rangeData[p][s][t][index + 1]
                    && shortCPictDist[p][s] > rangeData[p][s][t][index +
                    1]) {
                // interpolation
                double probDiff = Math.abs(plosCorrectedData[p][s][t][index]
                        - plosCorrectedData[p][s][t][index + 1]);
                double rangeDiff = Math.abs(rangeData[p][s][t][index + 1] -
                        rangeData[p][s][t][index]);
                double rangeIntercept = Math.abs(shortCPictDist[p][s] -
                        rangeData[p][s][t][index]);
                double probIntercept = (probDiff / rangeDiff) *
                        rangeIntercept;

                longCenterProb = plosCorrectedData[p][s][t][index] -
                        probIntercept;
            }

            if (shortSPictDist[p][s] >= toRangeArray[p][s][t]) {
                shortSideProb = 0.0;
            } else if (shortSPictDist[p][s] >= rangeData[p][s][t][index] &&
                    shortSPictDist[p][s] < rangeData[p][s][t][index + 1])
                    {
                // interpolation
                double probDiff = Math.abs(plosCorrectedData[p][s][t][index]
                        - plosCorrectedData[p][s][t][index + 1]);
                double rangeDiff = Math.abs(rangeData[p][s][t][index + 1] -
                        rangeData[p][s][t][index]);
                double rangeIntercept = Math.abs(shortCPictDist[p][s] -
                        rangeData[p][s][t][index]);
                double probIntercept = (probDiff / rangeDiff) *
                        rangeIntercept;

                shortSideProb = plosCorrectedData[p][s][t][index] -
                        probIntercept;
            }

            if (longSPictDist[p][s] >= toRangeArray[p][s][t]) {
                longSideProb = 0.0;
            } else if (longSPictDist[p][s] <= rangeData[p][s][t][index + 1
                    ] && shortSPictDist[p][s] > rangeData[p][s][t][index +
                    1]) {
                // interpolation
                double probDiff = Math.abs(plosCorrectedData[p][s][t][index]
                        - plosCorrectedData[p][s][t][index + 1]);
                double rangeDiff = Math.abs(rangeData[p][s][t][index + 1] -
                        rangeData[p][s][t][index]);
                double rangeIntercept = Math.abs(shortCPictDist[p][s] -
                        rangeData[p][s][t][index]);
                double probIntercept = (probDiff / rangeDiff) *
                        rangeIntercept;

                longSideProb = plosCorrectedData[p][s][t][index] -
                        probIntercept;
            }
        }
        centerPictureProbability[p][s][t] = (shortCenterProb +
                longCenterProb) / 2;
        sidePictureProbability[p][s][t] = (shortSideProb + longSideProb) / 2;
    }
}
```

```
    }
}

// method to compute the SensorSweepWidth based on the reltive area covered with
   pictures
public void sensorSweepWidth() {
    double centerSweepArea = 0.0;
    double sideSweepArea = 0.0;
    double coveredSegmentArea = 0.0;

    for (int p = 0; p < getNumberOfPlatforms(); p++) {
        for (int s = 0; s < getNumberOfSensors(p); s++) {
            for (int t = 0; t < getNumberOfTargets(); t++) {
                if (p < getNumberOfUAV()) {    // This is for UAV's flying at an
                                                  operational altitude
                    centerSweepArea = centerPictureWidth[p][s] * pictureLength[p][s]
                                    * centerPictureProbability[p][s][t];
                    sideSweepArea = sidePictureWidth[p][s] * pictureLength[p][s] *
                                    sidePictureProbability[p][s][t];
                    coveredSegmentArea = centerSweepArea + (2 * sideSweepArea);

                    sensorSweepWidth[p][s][t] = coveredSegmentArea / (2 *
                                   pictureLength[p][s]);
                } else {  // This is for UAV's operting on the ground (also referred
                             to as airdropped)
                    // Integration over plosCorrectedData to get the SensorSweepWidth
                       for stationary groundoperating platforms
                    double increment = rangeData[p][s][t][1];
                    double sectorPart = 0.0;
                    double sectorProb = 0.0;
                    double coveredSectorArea = 0.0;

                    for (int index = 0; index < getPlosCorrectedDataSize()-1;
                         index++) {
                        if (rangeData[p][s][t][index+1] != 0.0) {
                            sectorPart =(Math.PI * rangeData[p][s][t][index+1] *
                                         rangeData[p][s][t][index+1] *
                                         horizontalAngle[p][s]/360.0) –
                                         (Math.PI * rangeData[p][s][t][index] *
                                         rangeData[p][s][t][index]*
                                         horizontalAngle[p][s]/360.0);
                            sectorProb = (sensorData[p][s][t][index] +
                                         sensorData[p][s][t][index+1]) / 2.0;
                            coveredSectorArea += sectorPart * sectorProb;
                            if (debugging_Mode) {
                                if (p==5 && s==1 && t==1) {
                                    System.out.println ("SectorPart(" +
                                    rangeData[p][s][t][index+1] + ") = " +
                                    sectorPart);
                                    System.out.println ("SectorProb(" +
                                    (rangeData[p][s][t][index] + (increment/2.0)) +
                                    ") = " + sectorProb);
                                }
                            }
                        }
                    }
                    sensorSweepWidth[p][s][t] = coveredSectorArea /
                                                toRangeArray[p][s][t];
                    if (debugging_Mode) {
                        if (p==5 && s==1 && t==1) {
                            System.out.println("coveredSectorArea(Plat 5, Sensor 1,
                                                Target 1) = " + coveredSectorArea);
                            System.out.println("sweepWidth(Plat 5, Sensor 1, Target
                                                1) = " + sensorSweepWidth[5][1][1]);
                        }
                    }
                }
            }
        }
    }
}

// getter methods for instance variables
public int getNumberOfPlatforms() {
    return numberOfPlatforms;
}

public int getNumberOfUAV() {
    int numberOfUAV = 0;
```

```
    for (int p = 0; p < getNumberOfPlatforms(); p++) {
        if (!airDropped[p]) {
            ++numberOfUAV;
        }
    }
    return numberOfUAV;
}

public int getNumberOfTargets() {
    return numberOfTargets;
}

public int getNumberOfSensors(int index) {
    return numberOfSensors[index];
}

public int getNumberOfSensorsLength() {
    return numberOfSensors.length;
}

public int getSensorDataHight() {
    return sensorDataHight;
}

public int getSensorDataWidth() {
    return sensorDataWidth;
}

public double getRangeData(int indexP, int indexS, int indexT, int indexR) {
    return rangeData[indexP][indexS][indexT][indexR];
}

public int getRangeDataSize() {
    return rangeData[0][0][0].length;
}

public double getSensorData(int indexP, int indexS, int indexT, int indexR) {
    return sensorData[indexP][indexS][indexT][indexR];
}

public int getSensorDataSize() {
    return sensorData[0][0][0].length;
}

public double getPlosCorrectedData(int indexP, int indexS, int indexT, int indexR) {
    return plosCorrectedData[indexP][indexS][indexT][indexR];
}

public int getPlosCorrectedDataSize() {
    return plosCorrectedData[0][0][0].length;
}

public double getCenterPictureWidth(int indexP, int indexS) {
    return centerPictureWidth[indexP][indexS];
}

public int getCenterPictureWidthHight() {
    return centerPictureWidth.length;
}

public int getCenterPictureWidthWidth() {
    return centerPictureWidth[0].length;
}

public double getSidePictureWidth(int indexP, int indexS) {
    return sidePictureWidth[indexP][indexS];
}

public int getSidePictureWidthHight() {
    return sidePictureWidth.length;
}

public int getSidePictureWidtWidth() {
    return sidePictureWidth[0].length;
}

public double getPictureLength(int indexP, int indexS) {
    return pictureLength[indexP][indexS];
}
```

```
    public int getPictureLengthHight() {
        return pictureLength.length;
    }

    public int getPictureLengthWidth() {
        return pictureLength[0].length;
    }

    public double getShortCPictDist(int indexP, int indexS) {
        return shortCPictDist[indexP][indexS];
    }

    public double getLongCPictDist(int indexP, int indexS) {
        return longCPictDist[indexP][indexS];
    }

    public double getShortSPictDist(int indexP, int indexS) {
        return shortSPictDist[indexP][indexS];
    }

    public double getLongSPictDist(int indexP, int indexS) {
        return longSPictDist[indexP][indexS];
    }

    public double getCenterPictureProbability(int indexP, int indexS, int indexT) {
        return centerPictureProbability[indexP][indexS][indexT];
    }

    public double getSidePictureProbability(int indexP, int indexS, int indexT) {
        return sidePictureProbability[indexP][indexS][indexT];
    }

    public double getSensorSweepWidth(int indexP, int indexS, int indexT) {
        return sensorSweepWidth[indexP][indexS][indexT];
    }

    public int getNumberOfCluster() {
        return clusterData.length;
    }

    public String getSensorDataFile() {
        return sensorDataFile;
    }

} // end Sensor
```

D. PLATFORMS

```
package jSAM;


/*
 * File: Platform.java
 * Created: 25th November 2003, 18:00
 */

/**
 * @author Thomas Doll
 */

import java.util.*;
import java.io.*;
import java.text.*;


public class Platforms extends Sensors implements Input_ModelData {
    // class constants
    // class variables
    // instance variables
    private double[][] platformSweepWidth;   // [Platform][Target]
    private double[][] timeOnStation;     // [Platform][Cluster]
    private double[][][] platformRate;    // [Platform][Cluster][Target]
    private double[][][] platformCDP;     // [Platform][Cluster][Target]

    // class methods
    // constructor methods
    /**
```

```
 * constructor for parameter inputString.
 */
public Platforms() {
    super();
    // this if for debugging
    if (debugging_Mode) {
        System.out.println("\n-------------- Debugging Output for Class Platforms ---
                           ----------");
    }

    this.platformSweepWidth = new double
                              [getNumberOfPlatforms()][getNumberOfTargets()];
    platformSweepWidth();

    this.timeOnStation = new double[getNumberOfPlatforms()][getNumberOfCluster()];
    timeOnStation();

    this.platformRate = new double
         [getNumberOfPlatforms()][getNumberOfCluster()][getNumberOfTargets()];
    this.platformCDP = new double
         [getNumberOfPlatforms()][getNumberOfCluster()][getNumberOfTargets()];
    platformCDP();

}

// instance methods

// computes the sweep width for the platform
public void platformSweepWidth() {
    double sensor1_SweepWidth = 0.0;
    double sensor2_SweepWidth = 0.0;
    double sensor3_SweepWidth = 0.0;

    for (int p = 0; p < getNumberOfPlatforms(); p++) {
        for (int t = 0; t < getNumberOfTargets(); t++) {

            if (getNumberOfSensors(p) == 1) {
                platformSweepWidth[p][t] = getSensorSweepWidth(p, 0, t);
            }
            if (getNumberOfSensors(p) == 2) {
                sensor1_SweepWidth = getSensorSweepWidth(p, 0, t);
                sensor2_SweepWidth = getSensorSweepWidth(p, 1, t);
                double max = Math.max(sensor1_SweepWidth, sensor2_SweepWidth);

                platformSweepWidth[p][t] = max + (sensor1_SweepWidth +
                                           sensor2_SweepWidth - max) *
                                           swag_dependance;
                if (debugging_Mode) {
                    if (p == 0 && t == 1) {
                        System.out.println("This Platform has 2 Sensors");
                        System.out.println("\tmaximum Sensor Sweep Width New   : " +
                                           max);
                        System.out.println("\tsweep width for the platform New : " +
                                           platformSweepWidth[p][t]);
                    }
                }
            }
            if (getNumberOfSensors(p) == 3) {
                sensor1_SweepWidth = getSensorSweepWidth(p, 0, t);
                sensor2_SweepWidth = getSensorSweepWidth(p, 1, t);
                sensor3_SweepWidth = getSensorSweepWidth(p, 2, t);
                double max1 = Math.max(sensor1_SweepWidth, sensor2_SweepWidth);
                double max = Math.max(max1, sensor3_SweepWidth);

                platformSweepWidth[p][t] = max + (sensor1_SweepWidth +
                                           sensor2_SweepWidth + sensor3_SweepWidth -
                                           max) * swag_dependance;
            }
        }
    }
}

// computes the time on station for the platform
public void timeOnStation() {
    for (int p = 0; p < getNumberOfPlatforms(); p++) {
        for (int c = 0; c < getNumberOfCluster(); c++) {
            double distance = Math.sqrt((entryPoint[0] - clusterData[c][0]) *
                              (entryPoint[0] - clusterData[c][0]) + (entryPoint[1] -
                              clusterData[c][1]) * (entryPoint[1] -
                              clusterData[c][1]));
```

```
            double time = operationalTime[p] - (2 * distance / travelSpeed[p]);

            if (use_operationalRadius) {
                if (distance <= operationalRadius[p]) {
                    if (airDropped[p]) {
                        timeOnStation[p][c] = operationalTime[p];
                    } else {
                        if (time > 0.0) {
                            timeOnStation[p][c] = time;
                        } else {
                            timeOnStation[p][c] = 0.0;
                        }
                    }
                } else {
                    timeOnStation[p][c] = 0.0;
                }
            } else {
                if (airDropped[p]) {
                    timeOnStation[p][c] = operationalTime[p];
                } else {
                    if (time > 0.0) {
                        timeOnStation[p][c] = time;
                    } else {
                        timeOnStation[p][c] = 0.0;
                    }
                }
            }
        }
    }
}

// computes the CDP for the platforms
public void platformCDP() {
    for (int c = 0; c < getNumberOfCluster(); c++) {
        for (int t = 0; t < getNumberOfTargets(); t++) {
            for (int p = 0; p < getNumberOfPlatforms(); p++) {
                if (searchSpeed[p] > 0.0) {
                    platformRate[p][c][t] = searchSpeed[p] *
                                            (platformSweepWidth[p][t] / 1000) *
                                            timeOnStation[p][c];
                } else {
                    platformRate[p][c][t] = targetSpeed[t] *
                                            (platformSweepWidth[p][t] / 1000) *
                                            timeOnStation[p][c];
                }
                platformCDP[p][c][t] = 1.0 - Math.exp(-platformRate[p][c][t] /
                (clusterData[c][2] * clusterData[c][3]));
                // this if for debugging
                if (debugging_Mode) {
                    if (c == 0 && t == 0 && p == 1) {
                        System.out.println("\nc = " + c + " t = " + t + " p = " + p);
                        System.out.println("platformRate[p=" + p + "][c=" + c +
                                           "][t=" + t + "]    : " +
                                           formatRate.format(platformRate[p][c][t]));
                        System.out.println("platformCDP[p=" + p + "][c=" + c + "][t="
                                           + t + "]     : " +
                                          formatCDP.format(platformCDP[p][c][t]));
                    }
                }
            }
        }
    }
}

// getter methods for instance variables
public double getPlatformSweepWidth(int indexP, int indexT) {
    return platformSweepWidth[indexP][indexT];
}

public int getPlatformSweepWidthHight() {
    return platformSweepWidth.length;
}

public int getPlatformSweepWidthWidth() {
    return platformSweepWidth[0].length;
}

public double getTimeOnStation(int indexI, int indexJ) {
    return timeOnStation[indexI][indexJ];
}
```

```
    public int getTimeOnStationHight() {
        return timeOnStation.length;
    }

    public int getTimeOnStationWidth() {
        return timeOnStation[0].length;
    }

    public double getPlatformRate(int indexP, int indexC, int indexT) {
        return platformRate[indexP][indexC][indexT];
    }

    public int getPlatformRateHight() {
        return platformRate.length;
    }

    public int getPlatformRateWidth() {
        return platformRate[0].length;
    }

    public int getPlatformRateDepth() {
        return platformRate[0][0].length;
    }

    public double getPlatformCDP(int indexP, int indexC, int indexT) {
        return platformCDP[indexP][indexC][indexT];
    }

    public int getPlatformCDPHight() {
        return platformCDP.length;
    }

    public int getPlatformCDPWidth() {
        return platformCDP[0].length;
    }

    public int getPlatformCDPDepth() {
        return platformCDP[0][0].length;
    }

} // end Platform
```

E. PACKAGES

```
package jSAM;


/*
 * File: Package.java
 * Created: 25th November 2003, 18:00
 */

/**
 * @author Thomas Doll
 */

import java.util.*;
import java.io.*;
import java.text.*;


public class Packages extends Platforms {
    // class constants
    // class variables
    // instance variables
    int numberOfPackages;
    int[][] packageCombinations;       // [Package][Platform]
    double[][] packageCharacteristics; // [Package][Latency, Cost, Perishability,
                                          Logistics]
    double[][][] packageCDP;           // [Package][Cluster][Target]

    // class methods
    // constructor methods
    /**
     * constructor for parameter inputString.
     */
```

```
public Packages() {
    super();
    // this if for debugging
    if (debugging_Mode) {
        System.out.println("\n--------------- Debugging Output for Class Packages ----
                           ----------");
    }

    this.numberOfPackages = basicPackages.length + 3 * choose(basicPackages.length,
                            2);
    this.packageCombinations = new int[numberOfPackages][getNumberOfPlatforms()];
     packageCombinations();
    this.packageCharacteristics = new double[numberOfPackages][4];
    packageCharacteristics();
    this.packageCDP = new double
                      [numberOfPackages][getNumberOfCluster()][getNumberOfTargets()];
    packageCDP();
}

// instance methods

// method to combine basic packages to consolidated packages
public void packageCombinations() {
    int combinationHight = choose(basicPackages.length, 2);
    int count = 0;

    for (int k = 0; k < basicPackages.length; k++) {
        for (int p = 0; p < getNumberOfPlatforms(); p++) {
            packageCombinations[k][p] = basicPackages[k][p];
        }
        for (int k1 = k + 1; k1 < basicPackages.length; k1++) {
            for (int p = 0; p < getNumberOfPlatforms(); p++) {
                packageCombinations[basicPackages.length + count][p] =
                                    basicPackages[k][p] + basicPackages[k1][p];
                packageCombinations[basicPackages.length + combinationHight +
                                    count][p] = 2 * basicPackages[k][p] +
                                                basicPackages[k1][p];
                packageCombinations[basicPackages.length + 2 * combinationHight +
                                    count][p] = basicPackages[k][p] + 2 *
                                                basicPackages[k1][p];

            }
            count++;
        }
    }
}

// method to compute the package characteristics
public void packageCharacteristics() {
    for (int k = 0; k < numberOfPackages; k++) {
        double packageLatency = 0.0;
        double packageCost = 0.0;
        double packagePerishability = 0.0;
        double packageLogistics = 0.0;

        for (int p = 0; p < getNumberOfPlatforms(); p++) {
            packageLatency += packageCombinations[k][p] * platformLatency[p];
            packageCost += packageCombinations[k][p] * platformCost[p];
            packagePerishability += packageCombinations[k][p] *
                                    platformPerishability[p];
            packageLogistics += packageCombinations[k][p] * platformLogistics[p];
        }
        packageCharacteristics[k][0] = packageLatency;
        packageCharacteristics[k][1] = packageCost;
        packageCharacteristics[k][2] = packagePerishability;
        packageCharacteristics[k][3] = packageLogistics;
    }
}

// method to compute the Package CDP
public void packageCDP() {
    for (int c = 0; c < getNumberOfCluster(); c++) {
        for (int t = 0; t < getNumberOfTargets(); t++) {
            for (int k = 0; k < numberOfPackages; k++) {
                int firstPlatformInPackage = 0;
                boolean found = false;
                double firstPlatformInPackageRate = 0.0;
                double allPlatformInPackageRate = 0.0;
                double effectiveProportionCovered = 0.0;
                double areaPrime = 0.0;
                int numberOfPlatformsInPackage = 0;
```

```
                    for (int p = 0; p < getNumberOfPlatforms(); p++) {
                        if (packageCombinations[k][p] >= 1.0 && !found &&
                            getPlatformRate(p, c, t) != 0.0) {
                            firstPlatformInPackage = p;
                            firstPlatformInPackageRate = getPlatformRate(p, c, t);
                            found = true;
                            // this if for debugging
                            if (debugging_Mode) {
                                if(c==0 && t==0 && k==19) {
                                    System.out.println("c = " + c +" t = " + t + " k = "
                                                       + k + " p = " + p);
                                    System.out.println("packageCombinations[k="+k+"]
                                                       [p="+p+"] : " +
                                                        packageCombinations[k][p]);
                                }
                            }
                        }
                        allPlatformInPackageRate += packageCombinations[k][p] *
                                                    getPlatformRate(p, c, t);
                        numberOfPlatformsInPackage += packageCombinations[k][p];
                    }
                    if (firstPlatformInPackageRate == 0.0 && allPlatformInPackageRate ==
                        0.0) {
                        effectiveProportionCovered = 1.0;
                    } else {
                        effectiveProportionCovered = firstPlatformInPackageRate /
                                                     allPlatformInPackageRate;
                    }
                    areaPrime = effectiveProportionCovered * (clusterData[c][2] *
                                clusterData[c][3]);
                    if (numberOfPlatformsInPackage == 1) {
                        packageCDP[k][c][t] = 1.0 - Math.exp(-firstPlatformInPackageRate
                        / areaPrime);
                    } else {
                        packageCDP[k][c][t] = 1.0 - Math.exp(-firstPlatformInPackageRate
                       / areaPrime * package_enhFact);
                    }
                    // this if for debugging
                    if (debugging_Mode) {
                        if (c == 0 && t == 0 && k == 19) {
                            System.out.println("firstPlatformInPackage         : " +
                                    formatIdentifier.format(firstPlatformInPackage));
                            System.out.println("firstPlatformInPackageRate     : " +
                                    formatRate.format(firstPlatformInPackageRate));
                            System.out.println("allPlatformInPackageRate       : " +
                                    formatRate.format(allPlatformInPackageRate));
                            System.out.println("effectiveProportionCovered     : " +
                                    formatProb.format(effectiveProportionCovered));
                            System.out.println("areaPrime                      : " +
                                    formatRate.format(areaPrime));
                            System.out.println("packageCDP[k="+k+"][c="+c+"][t="+t+"]
                                    : " + formatCDP.format(packageCDP[k][c][t]));
                         }
                    }
                }
            }
        }
    }

    // probabilistic method to compute n choose k
    public int choose(int n, int k) {
        return fact(n) / (fact(k) * fact(n - k));
    }

    // probabilistic method to compute faculty of n
    public int fact(int n) {
        int answer = 1;    // Accumulate the product

        for (int i = 1; i <= n; i++) {
            answer *= i;
        }

        return answer;
    }

    // getter methods for instance variables
    public double getPackageCombinations(int indexI, int indexJ) {
        return packageCombinations[indexI][indexJ];
    }
```

```
    public int getPackageCombinationsHight() {
        return packageCombinations.length;
    }

    public int getPackageCombinationsWidth() {
        return packageCombinations[0].length;
    }

    public double getPackageCharacteristics(int indexI, int indexJ) {
        return packageCharacteristics[indexI][indexJ];
    }

    public int getPackageCharacteristicsHight() {
        return packageCharacteristics.length;
    }

    public int getPackageCharacteristicsWidth() {
        return packageCharacteristics[0].length;
    }

    public int getNumberOfPackages() {
        return numberOfPackages;
    }

    public double getPackageCDP(int indexK, int indexC, int indexT) {
        return packageCDP[indexK][indexC][indexT];
    }

    public int getPackageCDPHight() {
        return packageCDP.length;
    }

    public int getPackageCDPWidth() {
        return packageCDP[0].length;
    }

    public int getPackageCDPDepth() {
        return packageCDP[0][0].length;
    }

} // end Platform
```

F. SCENARIO

```
package jSAM;


/*
 * File: Scenario.java
 * Created: 25th November 2003, 18:00
 */

/**
 * @author Thomas Doll
 */

import java.util.*;
import java.io.*;
import java.text.*;

public class Scenario extends Packages implements Input_ModelData {
    // class constants
    // class variables
    // instance variables
    Random random;
    private int [][] scenarioData;          //[Clsuter][Target]
    private int [][][] tagetDistribution; //[EnemyOrderOfBattle][Clsuter][Target]

    // class methods
    // constructor methods
    /**
     * constructor for parameter inputString.
     */
    public Scenario() {
        super();
        // this if for debugging
```

```
        if (debugging_Mode) {
            System.out.println("\n--------- Debugging Output for Class ScenarioGenerator
                               ----------");
        }
        this.random = new Random(scenario_Seed);
        this.scenarioData = new int[getNumberOfCluster()][getNumberOfTargets()];
        scenarioData();
        this.tagetDistribution = new int
                          [getNumberOfEOB()][getNumberOfCluster()][getNumberOfTargets()];
        targetDistribution();
    }

    // instance methods

    // generates basic target distribution in clusters
    public void scenarioData() {
        for (int c=0; c<getNumberOfCluster(); c++) {
            for (int t=0; t<getNumberOfTargets(); t++) {
                scenarioData[c][t] = random.nextInt(max_NumberOfTargets+1);
            }
        }
    }

    // generates targetDistribution according enemyOrderOfBattle_Factor
    public void targetDistribution() {
        for (int e=0; e<getNumberOfEOB(); e++) {
            for (int c=0; c<getNumberOfCluster(); c++) {
                for (int t=0; t<getNumberOfTargets(); t++) {
                    double decisionValue = random.nextDouble();
                    if (decisionValue <= eOB_Factor[e]) {
                        tagetDistribution[e][c][t] = scenarioData[c][t];
                    } else {
                        tagetDistribution[e][c][t] = 0;
                    }
                }
            }
        }
    }


    // getter methods for instance variables
    public int getScenarioData(int indexC, int indexT) {
        return scenarioData[indexC][indexT];
    }

    public int getScenarioDataHight() {
        return scenarioData.length;
    }

    public int getScenarioDataWidth() {
        return scenarioData[0].length;
    }

    public int getTagetDistribution(int indexE, int indexC, int indexT) {
        return tagetDistribution[indexE][indexC][indexT];
    }

    public int getTagetDistributionHight() {
        return tagetDistribution.length;
    }

    public int getTagetDistributionWidth() {
        return tagetDistribution[0].length;
    }

    public int getTagetDistributionDepth() {
        return tagetDistribution[0][0].length;
    }
    public int getNumberOfEOB() {
      return eOB_Factor.length;
    }
  }

} // end Sensor
```

G. GAMS_HANDLING

```
package jSAM;
```

```java
/*
 * File: Gams_Handling.java
 * Created: 25th November 2003, 18:00
 */

/**
 * @author Thomas Doll
 */

import java.util.*;
import java.io.*;
import java.text.*;


public class Gams_Handling implements Input_ModelData {
    // class constants
    final static String sensorData = "ALLOCATION_TABLE.CSV";
    // class variables
    // instance variables
    Scenario scenario;
    int[][] allocationTable;
    // class methods
    // constructor methods
    /**
     * constructor for parameter inputString.
     */
    public Gams_Handling(Scenario scenario) {
        this.scenario = scenario;
        this.allocationTable = new int
                        [scenario.getNumberOfCluster()][scenario.getNumberOfPackages()];

        startGAMS();
        allocationTable();
    }

    // instance methods

    // method to start GAMS
    public void startGAMS() {
        if (run_Mode) {
            System.out.println("\n->Start GAMS");
        }

        String[] cmdArray = new String[5];
        cmdArray[0] = "C:\\Program Files\\Gams21.1\\gams.exe";
        cmdArray[1] = "C:\\Java_Classpath\\jSAM\\GamsModel.gms";
        cmdArray[2] = "WDIR=C:\\Java_Classpath\\jSAM";
        cmdArray[3] = "SCRDIR=C:\\Java_Classpath\\jSAM";
        cmdArray[4] = "LO=2";

        try {
            Process gamsProcess = Runtime.getRuntime().exec(cmdArray);
            gamsProcess.waitFor();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        catch (InterruptedException ex) {
            ex.printStackTrace();
        }

        if (run_Mode) {
            System.out.println("\n->End GAMS");
        }

    }

    // reads AllocationTable form Gams Lst File
    public void allocationTable() {
        String          inputString;
        FileReader      inputFile;
        BufferedReader  inputUnit;
        StringTokenizer tk;

        try {
            inputFile = new FileReader(sensorData);
            inputUnit = new BufferedReader(inputFile);
            inputString = inputUnit.readLine();
```

```
            inputString = inputUnit.readLine();
            inputString = inputUnit.readLine();

            tk = new StringTokenizer(inputString, ",");

          for (int c = 0; c < scenario.getNumberOfCluster(); c++) {
              inputString = inputUnit.readLine();
              tk = new StringTokenizer(inputString, ",");
              for (int k = 0; k < scenario.numberOfPackages; k++) {
                  if(k<=1) {
                      tk.nextToken();
                  } else {
                      Double number = new Double(Double.parseDouble(tk.nextToken()));
                      allocationTable[c][k] = number.intValue();
                  }
              }
          }
          inputUnit.close();
          } catch (FileNotFoundException e) {
              System.out.println(e);
              return;
          } catch (IOException e) {
              System.out.println(e);
              return;
          }
    }

    // getter methods for instance variables
    public int getAllocationTable(int indexC, int indexK) {
        return allocationTable[indexC][indexK];
    }

    public int getAllocationTableHight() {
        return allocationTable.length;
    }

    public int getAllocationTableWidth() {
        return allocationTable[0].length;
    }


} // end Gams_Handling
```

H. OUTPUT_GAMSFILES

```
package jSAM;


/*
 * File: Output_GamsFiles.java
 * Created: 25th November 2003, 18:00
 */

/**
 * @author Thomas Doll
 */

import java.util.*;
import java.io.*;
import java.text.*;


public class Output_GamsFiles implements Input_ModelData {
    // class constants
    final static String targetDistribution = "TARGET_DISTRIBUTION.CSV";
    final static String packageCombinations = "PACKAGE_COMBINATIONS.CSV";
    final static String packageCharacteristics = "PACKAGE_CHARACTERISTICS.CSV";
    final static String packageCDP = "PACKAGE_CDP.CSV";

    // class variables
    // instance variables
    Scenario scenario;

    // class methods
    // constructor methods
    /**
     * constructor for parameter inputString.
```

```
 */
public Output_GamsFiles(Scenario scenario) {
    if (run_Mode) {
        System.out.println("\n->Generate GAMS Include Files");
    }
    this.scenario = scenario;
    try {
        makeTargetDistribution();
        makePackageCDP();
        makePackageCharacteristics();
        makePackageCombinations();
    }
    catch (IOException ex) {
        ex.printStackTrace();
    }

}

// instance methods

// method to create the textfile TARGET_DISTRIBUTION.CSV
public void makeTargetDistribution() throws IOException {
    FileWriter filewriter = new FileWriter(targetDistribution);
    BufferedWriter bufferedWriter = new BufferedWriter(filewriter);
    PrintWriter printWriter = new PrintWriter(bufferedWriter);
    String outputLine = new String();
    outputLine = "dummy,dummy";
    for (int c=0; c<scenario.getNumberOfCluster(); c++) {
        outputLine += (",C" + (c+1));
    }
    printWriter.print(outputLine + "\n");
    for (int e=0; e<scenario.getNumberOfEOB(); e++) {
        for (int t = 0; t < scenario.getNumberOfTargets(); t++) {
            outputLine = "";
            for (int c = 0; c < scenario.getNumberOfCluster(); c++) {
                outputLine += ("," +
                                formatIdentifier2.format(scenario.getTagetDistribution
                                (e, c, t)));
            }
            printWriter.print("T" + (t+1) + ".EOB" + (e+1));
            printWriter.print(outputLine + "\n");
        }
    }
    printWriter.close();
    if (run_Mode) {
        System.out.println("    TARGET_DISTRIBUTION.CSV");
    }
}

// method to create the textfile PACKAGE_COMBINATIONS.CSV
public void makePackageCombinations() throws IOException {
    FileWriter filewriter = new FileWriter(packageCombinations);
    BufferedWriter bufferedWriter = new BufferedWriter(filewriter);
    PrintWriter printWriter = new PrintWriter(bufferedWriter);
    String outputLine = new String();
    outputLine = "dummy";
    for (int p = 0; p < scenario.getNumberOfPlatforms(); p++) {
        outputLine += (",P" + (p+1));
    }
    printWriter.print(outputLine + "\n");
    for (int k=0; k<scenario.getNumberOfPackages(); k++) {
        outputLine = "";
        for (int p = 0; p < scenario.getNumberOfPlatforms(); p++) {
            outputLine += ("," +
                formatIdentifier2.format(scenario.getPackageCombinations(k, p)));
        }
        printWriter.print("k" + (k+1));
        printWriter.print(outputLine + "\n");
    }
    printWriter.close();
    if (run_Mode) {
        System.out.println("    PACKAGE_COMBINATIONS.CSV");
    }
}

// method to create the textfile PACKAGECHARACTERISTICS.CSV
public void makePackageCharacteristics() throws IOException {
    FileWriter filewriter = new FileWriter(packageCharacteristics);
    BufferedWriter bufferedWriter = new BufferedWriter(filewriter);
    PrintWriter printWriter = new PrintWriter(bufferedWriter);
```

```
        String outputLine = new String();
        outputLine = "dummy,Latency,Cost,Perishability,Logistics";
        printWriter.print(outputLine + "\n");
        for (int k=0; k<scenario.getNumberOfPackages(); k++) {
            outputLine = "";
            for (int a = 0; a < scenario.getPackageCharacteristicsWidth(); a++) {
                outputLine += ("," +
                formatCharacteristics.format(scenario.getPackageCharacteristics(k, a)));
            }
            printWriter.print("k" + (k+1));
            printWriter.print(outputLine + "\n");
        }
        printWriter.close();
        if (run_Mode) {
            System.out.println("    PACKAGE_CHARACTERISTICS.CSV");
        }
    }

    // method to create the textfile PACKAGE_CDP.CSV
    public void makePackageCDP() throws IOException {
        FileWriter filewriter = new FileWriter(packageCDP);
        BufferedWriter bufferedWriter = new BufferedWriter(filewriter);
        PrintWriter printWriter = new PrintWriter(bufferedWriter);
        String outputLine = new String();
        outputLine = "dummy,dummy";
        for (int k=0; k<scenario.getNumberOfPackages(); k++) {
            outputLine += (",k" + (k+1));
        }
        printWriter.print(outputLine + "\n");
        for (int c=0; c<scenario.getNumberOfCluster(); c++) {
            for (int t=0; t<scenario.getNumberOfTargets(); t++) {
                outputLine = "";
                for (int k=0; k<scenario.getNumberOfPackages(); k++) {
                    outputLine += ("," + formatProb2.format(scenario.getPackageCDP
                            (k, c, t)));
                }
                printWriter.print("C" + (c+1) + ".T" + (t+1));
                printWriter.print(outputLine + "\n");
            }
        }
        printWriter.close();
        if (run_Mode) {
            System.out.println("    PACKAGE_CDP.CSV");
        }
    }

} // end Output_GamsFiles
```

I. OUTPUT_COMMAND

```
package jSAM;


/*
 * File: Output_Command.java
 * Created: 25th November 2003, 18:00
 */

/**
 * @author Thomas Doll
 */

import java.util.*;
import java.io.*;
import java.text.*;


public class Output_Command implements Input_ModelData {
    // class constants
    // class variables
    // instance variables
    Scenario scenario;
    Gams_Handling gams;

    // class methods
    // constructor methods
```

```
/**
 * constructor for parameter inputString.
 */
public Output_Command(Scenario scenario, Gams_Handling gams) {
    this.scenario = scenario;
    this.gams = gams;

    if (print_Verbose) {
        //printClusterData();
        //printPlatformSensorInformation();
        //printPictureDimensions();
        //printPictureDistancies();
        //printSensorData();
        printRangeData();
        printPlosCorrectedData();
        //printPictureProbability();
        printSensorSweepWidth();
        printPlatformSweepWidth();
        //printTimeOnStation();
        printPackageCombinations();
        //printPlatformCDP();
        //printPlatformRate();
        //printPackageCharacteristics();
        printPackageCDP();
        //printScenarioData();
        //printTargetDistribution();
        //printAllocationTableFull();
        printAllocationTable();
    } else {
        printAllocationTable();
    }
}

// instance methods

// output for data sets in Input_ModelData
public void printClusterData() {

  System.out.print("\n----- Clusters and Target Data ----");
    System.out.print("\n    X-Coord Y-Coord EW-Dime NS-Dime");
    System.out.print("\n");
    for (int c = 0; c < scenario.getNumberOfCluster(); c++) {
        System.out.print("C" + formatIdentifier.format(c+1));
        for (int s = 0; s<clusterData[0].length; s++) {
           System.out.print(formatCoord.format(clusterData[c][s]));
        }
        System.out.print("\n");
    }
}
// output for data sets in Sensor
public void printPlatformSensorInformation() {
    System.out.println("\n----- Platform Sensor Information ------\n");
    System.out.println("Name of the data file:\t" + scenario.getSensorDataFile());
    System.out.println("Number of Platforms  :\t" + scenario.getNumberOfPlatforms());
    System.out.println("Number of Targets    :\t" + scenario.getNumberOfTargets());

    System.out.println("Platform Sensor Table:\tPlatform  Senors");
    for (int p = 0; p < scenario.getNumberOfSensorsLength(); p++) {
        System.out.println("\t\t\t" + (p + 1) + "\t  " +
                           scenario.getNumberOfSensors(p));
    }
}

public void printSensorData() {
    System.out.println("\n-------------------------------------------------------------
                       -- SensorData -----------------------------------------------
                       -------------");
    System.out.println("Pl Se Tg  ----------------------- This part of the array
                       shows probabilities of detection at a specific range ---------
                       ---------------\n");
    for (int p = 0; p < scenario.getNumberOfPlatforms(); p++) {
        for (int s = 0; s < scenario.getNumberOfSensors(p); s++) {
            for (int t = 0; t < scenario.getNumberOfTargets(); t++) {
                System.out.print(formatIdentifier.format(p+1) +
                                 formatIdentifier.format(s+1) +
                                 formatIdentifier.format(t+1));
                for (int index = 0; index < scenario.getSensorDataSize(); index++) {
                    System.out.print(formatProb.format
                                 (scenario.getSensorData(p,s,t,index)));
                }
```

```
                System.out.print("\n");
            }
        }
    }
}

public void printRangeData() {
    System.out.println("\n------------------------------------------------------------
                       -- RangeData ------------------------------------------------
                       -------------");
    System.out.println("Pl Se Tg  ----------------------- This part of the array
                       shows probabilities of detection at a specific range ---------
                       ---------------\n");
    for (int p = 0; p < scenario.getNumberOfPlatforms(); p++) {
        for (int s = 0; s < scenario.getNumberOfSensors(p); s++) {
            for (int t = 0; t < scenario.getNumberOfTargets(); t++) {
                System.out.print(formatIdentifier.format(p+1) +
                    formatIdentifier.format(s+1) + formatIdentifier.format(t+1));
                for (int index = 0; index < scenario.getRangeDataSize(); index++){
                     System.out.print(formatRange.format
                     (scenario.getRangeData(p,s,t,index)));
                }
                System.out.print("\n");
            }
        }
    }
}

public void printPlosCorrectedData() {
    System.out.println("\n------------------------------------------------------------
                       PLOS Corrected Data -----------------------------------------
                       ---------------");
    System.out.println("Pl Se Tg  ----------------------- This part of the array
                       shows probabilities of detection at a specific range ---------
                       ---------------\n");
    for (int p = 0; p < scenario.getNumberOfPlatforms(); p++) {
        for (int s = 0; s < scenario.getNumberOfSensors(p); s++) {
            for (int t = 0; t < scenario.getNumberOfTargets(); t++) {
                System.out.print(formatIdentifier.format(p+1) +
                                 formatIdentifier.format(s+1) +
                                formatIdentifier.format(t+1));
                for (int index = 0; index < scenario.getRangeDataSize(); index++) {
                    System.out.print(formatProb.format
                                     (scenario.getPlosCorrectedData(p,s,t,index)));
                }
                System.out.print("\n");
            }
        }
    }
}

public void printPictureProbability() {
    System.out.println("\n--- Picture Prob ---");
    System.out.println("Pl Se Tg CProb SProb\n");
    for (int p = 0; p < scenario.getNumberOfUAV(); p++) {
       for (int s = 0; s < scenario.getNumberOfSensors(p); s++) {
          for (int t = 0; t < scenario.getNumberOfTargets(); t++) {
             System.out.print(formatIdentifier.format(p+1) +
                              formatIdentifier.format(s+1) +
                              formatIdentifier.format(t+1) +
              formatProb.format(scenario.getCenterPictureProbability(p,s,t)) +
              formatProb.format(scenario.getSidePictureProbability(p,s,t)));

              System.out.print("\n");
          }
       }
    }
}

public void printPictureDimensions() {
    System.out.println("\n----- Picture Dimensions ----");
    System.out.println("Pl Se CentW CentL SideW SideL\n");
    for (int p = 0; p < scenario.getNumberOfUAV(); p++) {
        for (int s = 0; s < scenario.getNumberOfSensors(p); s++) {
            System.out.print(formatIdentifier.format(p + 1) +
                     formatIdentifier.format(s + 1) +
                     formatSweepWidth.format(scenario.getCenterPictureWidth(p, s)) +
                     formatSweepWidth.format(scenario.getPictureLength(p, s)) +
                     formatSweepWidth.format(scenario.getSidePictureWidth(p, s)) +
                     formatSweepWidth.format(scenario.getPictureLength(p, s)));
```

```
            System.out.print("\n");
        }
    }
}

public void printPictureDistancies() {
  System.out.println("\n----- Picture Distancies ----");
    System.out.println("Pl Se shorC longC shorS longS\n");
    for (int p = 0; p < scenario.getNumberOfUAV(); p++) {
        for (int s = 0; s < scenario.getNumberOfSensors(p); s++) {
           System.out.print(formatIdentifier.format(p + 1) +
                        formatIdentifier.format(s + 1) +
                        formatSweepWidth.format(scenario.getShortCPictDist(p, s)) +
                        formatSweepWidth.format(scenario.getLongCPictDist(p, s)) +
                        formatSweepWidth.format(scenario.getShortSPictDist(p, s)) +
                        formatSweepWidth.format(scenario.getLongSPictDist(p, s)));
           System.out.print("\n");
        }
    }
}

public void printSensorSweepWidth() {
    System.out.println("\n-- SeSweepW --");
    System.out.println("Pl Se Tg Sweep\n");
    for (int p = 0; p < scenario.getNumberOfPlatforms(); p++) {
        for (int s = 0; s < scenario.getNumberOfSensors(p); s++) {
            for (int t = 0; t < scenario.getNumberOfTargets(); t++) {
                System.out.print(formatIdentifier.format(p+1) +
                         formatIdentifier.format(s+1) +
                         formatIdentifier.format(t+1) +
                         formatSweepWidth.format
                         (scenario.getSensorSweepWidth(p,s,t)));
                System.out.print("\n");
            }
        }
    }
}

// output for data sets in Platform
public void printPlatformSweepWidth() {
    System.out.println("\n----------------- Platform Sweep Width New ---------------
                       -");
    System.out.println("Tgt01 Tgt02 Tgt03 Tgt04 Tgt05 Tgt06 Tgt07 Tgt08 Tgt09
                        Tgt10\n");
    for (int p = 0; p < scenario.getPlatformSweepWidthHight(); p++) {
        for (int t = 0; t < scenario.getPlatformSweepWidthWidth(); t++) {
            System.out.print(formatRange.format(scenario.getPlatformSweepWidth(p,
                             t)));
        }
        System.out.print("\n");
    }
}

public void printTimeOnStation() {
    System.out.println("\n---------------- Platform Time on Station -----------------
                       ");
    System.out.println("Clu01 Clu02 Clu03 Clu04 Clu05 Clu06 Clu07 Clu08 Clu09
                        Clu10\n");
    for (int p = 0; p < scenario.getTimeOnStationHight(); p++) {
        for (int c = 0; c < clusterData.length; c++) {
            System.out.print(formatTime.format(scenario.getTimeOnStation(p, c)));
        }
        System.out.print("\n");
    }
}

public void printPlatformRate() {
    System.out.println("\n------------------- Platform Rate --------------------");
    System.out.println("Cl Tg Platf01 Platf02 Platf03 Platf04 Platf05 Platf06\n");
    for (int c = 0; c < scenario.getNumberOfCluster(); c++) {
        for (int t = 0; t < scenario.getNumberOfTargets(); t++) {
            System.out.print(formatIdentifier.format(c + 1) +
                             formatIdentifier.format(t + 1));
            for (int p = 0; p < scenario.getNumberOfPlatforms(); p++) {
                System.out.print(formatRate.format(scenario.getPlatformRate(p, c,
                t)));
            }
            System.out.print("\n");
        }
    }
```

```
}

public void printPlatformCDP() {
    System.out.println("\n------------- Platform CDP --------------");
    System.out.println("Cl Tg Plat1 Plat2 Plat3 Plat4 Plat5 Plat6\n");
    for (int c = 0; c < scenario.getNumberOfCluster(); c++) {
        for (int t = 0; t < scenario.getNumberOfTargets(); t++) {
            System.out.print(formatIdentifier.format(c + 1) +
                             formatIdentifier.format(t + 1));
            for (int p = 0; p < scenario.getNumberOfPlatforms(); p++) {
                System.out.print(formatCDP.format(scenario.getPlatformCDP(p, c, t)));
            }
            System.out.print("\n");
        }
    }
}

// output for data sets in Package
public void printPackageCombinations() {
    System.out.println("\n---- Package Comb -----");
    System.out.println("      P1 P2 P3 P4 P5 P6\n");
    for (int k = 0; k < scenario.getPackageCombinationsHight(); k++) {
        System.out.print("Pa" + formatNumber.format(k + 1));
        for (int p = 0; p < scenario.getNumberOfPlatforms(); p++) {
            if (p < basicPackages[0].length) {
                if (scenario.getPackageCombinations(k, p) != 0) {
                    System.out.print(formatIdentifier.format
                                     (scenario.getPackageCombinations(k, p)));
                } else {
                    System.out.print("-- ");
                }
            } else {
                System.out.print(formatCharacteristics.format
                                 (scenario.getPackageCombinations(k, p)));
            }
        }
        System.out.print("\n");
    }
}

public void printPackageCharacteristics() {
    System.out.println("\n--- Package Char ----");
    System.out.println("      Lat Cos Per Log\n");
    for (int k = 0; k < scenario.getPackageCharacteristicsHight(); k++) {
        System.out.print("Pa" + formatNumber.format(k + 1));
        for (int a = 0; a < scenario.getPackageCharacteristicsWidth(); a++) {
            System.out.print(formatCharacteristics.format
                             (scenario.getPackageCharacteristics(k, a)));
        }
        System.out.print("\n");
    }
}

public void printPackageCDP() {
    System.out.print("\n------------- Package CDP ---------------");
    for (int i = 6; i < scenario.getNumberOfPackages(); i++) {
        System.out.print("------");
    }
    System.out.print("\nCl Tg ");
    for (int i = 0; i < scenario.getNumberOfPackages(); i++) {
        System.out.print("Pa" + formatNumber.format(i + 1));
    }
    System.out.println("\n");
    for (int c = 0; c < scenario.getNumberOfCluster(); c++) {
        for (int t = 0; t < scenario.getNumberOfTargets(); t++) {
            System.out.print(formatIdentifier.format(c + 1) +
                             formatIdentifier.format(t + 1));
            for (int k = 0; k < scenario.getNumberOfPackages(); k++) {
                System.out.print(formatCDP.format(scenario.getPackageCDP(k, c, t)));
            }
            System.out.print("\n");
        }
    }
}

// output for data sets in ScenarioGenerator
public void printScenarioData() {
    System.out.println("\n------------- Scenario Data --------------");
    System.out.println("Cl T01 T02 T03 T04 T05 T06 T07 T08 T09 T10\n");
    for (int c = 0; c < scenario.getNumberOfCluster(); c++) {
```

```
            System.out.print(formatIdentifier.format(c + 1));
            for (int t = 0; t < scenario.getNumberOfTargets(); t++) {
                System.out.print(formatNumber.format(scenario.getScenarioData(c, t)));
            }
            System.out.print("\n");
        }
    }

    public void printTargetDistribution() {
        for (int e = 0; e < scenario.getNumberOfEOB(); e++) {
            System.out.println("\n------- Target Distribution EOB" + e + 1 + " --------
                               ");
            System.out.println("Cl T01 T02 T03 T04 T05 T06 T07 T08 T09 T10\n");
            for (int c = 0; c < scenario.getNumberOfCluster(); c++) {
                System.out.print(formatIdentifier.format(c + 1));
                for (int t = 0; t < scenario.getNumberOfTargets(); t++) {
                    System.out.print(formatNumber.format(scenario.getTagetDistribution(e,
                                     c, t)));
                }
                System.out.print("\n");
            }
        }
    }

    public void printAllocationTableFull() {
        System.out.print("\n------------- AllocationTable --------------");
        for (int i = 14; i < scenario.getNumberOfPackages(); i++) {
            System.out.print("---");
        }
        System.out.print("\nCl ");
        for (int p = 0; p < scenario.getNumberOfPackages(); p++) {
            System.out.print("" + formatIdentifier.format(p + 1));
        }
        System.out.println("\n");
        for (int c = 0; c < scenario.getNumberOfCluster(); c++) {
            System.out.print(formatIdentifier.format(c + 1));
            for (int k = 0; k < scenario.getNumberOfPackages(); k++) {
                if (gams.getAllocationTable(c, k) != 0) {
                    System.out.print(formatIdentifier.format(gams.getAllocationTable(c,
                                     k)));
                } else {
                    System.out.print("-- ");
                }
            }
            System.out.print("\n");
        }

    }

    public void printAllocationTable() {
      ArrayList arrayList =  new ArrayList();
      for (int k = 0; k < scenario.getNumberOfPackages(); k++) {
         for (int c = 0; c < scenario.getNumberOfCluster(); c++) {
            if (gams.getAllocationTable(c, k) != 0 && !arrayList.contains(new Integer(k))
                ) {
               arrayList.add(new Integer(k));
            }
         }
      }

        System.out.print("\n");
        for (int a = 0; a < (arrayList.size()-2)/2; a++) {
            System.out.print("----");
        }
        System.out.print("- Allocations -");
        for (int a = 0; a < (arrayList.size()-2)/2; a++) {
            System.out.print("----");
        }
        System.out.print("\n    ");
        for (int a = 0; a < arrayList.size(); a++) {
            System.out.print("P" +
                    formatIdentifier.format(((Integer)arrayList.get(a)).intValue() + 1));
        }
        System.out.print("\n");
        for (int c = 0; c < scenario.getNumberOfCluster(); c++) {
            System.out.print("C" + formatIdentifier.format(c+1));
            for (int a = 0; a < arrayList.size(); a++) {

                if (gams.getAllocationTable(c, ((Integer)arrayList.get(a)).intValue())
                        != 0 &&
```

```
                        gams.getAllocationTable
                        (c,((Integer)arrayList.get(a)).intValue()) < 10) {
                   System.out.print(formatNumber2.format(gams.getAllocationTable(c,
                        ((Integer)arrayList.get(a)).intValue())));
               } else if (gams.getAllocationTable(c,
                        ((Integer)arrayList.get(a)).intValue())
                        != 0 && gams.getAllocationTable(c,
                        ((Integer)arrayList.get(a)).intValue()) >= 10) {
               System.out.print(formatNumber3.format(gams.getAllocationTable(c,
                             ((Integer)arrayList.get(a)).intValue())));
               }else {
                   System.out.print("--- ");
               }

           }
           System.out.print("\n");
       }
   }
} // end Output_Command
```

INITIAL DISTRIBUTION LIST

1. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

2. LTC Thomas Cioppa
   TRADOC Analysis Center Monterey
   Monterey, California

3. Maj Donovan Phillips
   TRADOC Analysis Center Monterey
   Monterey, California

4. Professor W. Matthew Carlyle
   Department of Operations Research
   Monterey, California