2

NAVAL POSTGRADUATE SCHOOL
Monterey, California

DISSERTATION

RADAR DATA PROCESSING USING A DISTRIBUTED COMPUTATIONAL SYSTEM

by

Gilberto Ferreira Mota

June, 1992

Dissertation Co-Supervisor:	U. R. Kodres
Dissertation Co-Supervisor:	M. L. Nelson

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School; 6b. OFFICE SYMBOL *(If applicable)* CS	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS *(City, State, and ZIP Code)* Monterey, CA 93943-5000	7b. ADDRESS *(City, State, and ZIP Code)* Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS *(City, State, and ZIP Code)*	10. SOURCE OF FUNDING NUMBERS: Program Element No. / Project No. / Task No. / Work Unit Accession Number

11. TITLE *(Include Security Classification)*
RADAR DATA PROCESSING USING A DISTRIBUTED COMPUTATIONAL SYSTEM

12. PERSONAL AUTHOR(S) Gilberto Ferreira Mota

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(year, month, day)*	15. PAGE COUNT
Doctoral Dissertation	From July 1990 To June 1992	June 1992	228

16. SUPPLEMENTARY NOTATION
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.

17. COSATI CODES

FIELD	GROUP	SUBGROUP

18. SUBJECT TERMS *(continue on reverse if necessary and identify by block number)*
Radar Data Processing, Multiple-Target Tracking, Distributed-Memory Systems, Static Scheduling, Dynamic Load Balancing, Object-Oriented Decomposition, Program and System Partitioning

19. ABSTRACT *(continue on reverse if necessary and identify by block number)*

This research specifies and validates a new concurrent decomposition scheme, called Confined Space Search Decomposition (CSSD), to exploit parallelism of Radar Data Processing algorithms using a Distributed Computational System. To formalize the specification we propose and apply an object-oriented methodology called Decomposition Cost Evaluation Model (DCEM). To reduce the penalties of load imbalance we propose a distributed dynamic load balance heuristic called Object Reincarnation (OR). To validate the research we first compare our decomposition with an identified alternative using the proposed DCEM model and then develop a theoretical prediction of selected parameters. We also develop a simulation to check the Object Reincarnation Concept.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS REPORT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Advisor's Name U. R. Kodres and M. L. Nelson	22b. TELEPHONE *(Include Area code)* (408)646-2197	22c. OFFICE SYMBOL CSkr

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED

Radar Data Processing Using a Distributed Computational System

by

Gilberto Ferreira Mota
Commander, Brazilian Navy
B.S.,University of Sao Paulo,1979

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1992

Author: [illegible]
Gilberto Ferreira Mota

Approved by:

[illegible]
Arthur L. Schoenstadt
Professor of Mathematics

[illegible]
Shridhar Shukla
Assistant Professor of
Electrical and Computer Engineering

[illegible]
Thomas C. Wu
Associate Professor of
Computer Science

[illegible]
Michael L. Nelson
Assistant Professor of Computer Science
Dissertation Co-Supervisor

[illegible]
Uno R. Kodres
Professor of Computer Science
Dissertation Co-Supervisor

Approved by: [illegible]
Robert B. McGhee, Chairman, Department of Computer Science

Approved by: [illegible]
Richard S. Elster, Dean of Instruction

ABSTRACT

This research specifies and validates a new concurrent decomposition scheme, called Confined Space Search Decomposition (CSSD), to exploit parallelism of Radar Data Processing algorithms using a Distributed Computational System. To formalize the specification we propose and apply an object-oriented methodology called Decomposition Cost Evaluation Model (DCEM). To reduce the penalties of load imbalance we propose a distributed dynamic load balance heuristic called Object Reincarnation (OR). To validate the research we first compare our decomposition with an identified alternative using the proposed DCEM model and then develop a theoretical prediction of selected parameters. We also develop a simulation to check the Object Reincarnation concept.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

TABLE OF SYMBOLS AND/OR ABBREVIATIONS

AFAP:	All Functions to all Processors
ANDU:	Angular Nonuniform Distribution and Distance Uniform Distribution
AUDN:	Angular Uniform Distribution and Distance Nonuniform Distribution
AH:	Application Hierarchy
BM/C^3:	Ballistic Missile / Command, Control, and Communications
C_p:	Number of Correlation Processors
C_{RT}:	Retransmit Cost
C_1:	Average Correlation Time per Target
CCC:	Class Computation Cost
CH:	Correlation Hierarchy
CP:	Correlation Processor
CSSD:	Confined Space Search Decomposition
CXC:	Class Communication Cost
DCEM:	Decomposition Cost Evaluation Model
EMSE:	Estimation Mean Square Error
FG:	First Gate
FI:	Filter Improvement
GL_{AVG}:	Global Average Load
GLI:	Global Load Imbalance Factor
HCC:	Hierarchy Computation Cost

HXC:	Hierarchy Communication Cost
IG:	Initiation Gate
IH:	Interface Hierarchy
IMM:	Interacting Multiple Model
IP:	Interface Processor
LL_{AVG}:	Local Average Load
LLI:	Local Load Imbalance Factor
MCC:	Method Computation Cost
MG:	Merge Gate
MMSE:	Measure Mean Square Error
MS:	Milliseconds
N_t:	Maximum Number of Expected Targets
OCC:	Object Computation Cost
OO:	Object-Oriented
OOD:	Object-Oriented Design
OOP:	Object-Oriented Programming
OOPL:	Object-Oriented Programming Language
OR:	Object Reincarnation
OXC:	Object Communication Cost
PCC:	Processor Computation Cost
PXC:	Processor Communication Cost
RC:	Result Cost
RDPP:	Radar Data Processing Processor
RID:	Receiver Initiated Diffusion
SFSP:	Some Functions to Some Processors
SG:	Second Gate

SPCH:	Single Processor Class Hierarchy
SRC:	Service Request Cost
$T_{conc}(N)$:	Time to Compute on N nodes
T_p:	Number of Tracking Processors
T_{seq}:	Time to Compute on One Node
T_1:	Average Tracking Time per Target
TH:	Tracking Hierarchy
TP:	Tracking Processor
VA:	Viterbi Algorithm

ACKNOWLEDGMENTS

I would like to thank my wife Tania for her encouragement and love.

I would like to express my sincere gratitude to Mario J. F. Braga, Director of the Brazilian Navy Research Institute, for his unfaltering support.

I would like to thank my dissertation Co-Supervisors for their guidance in the conception and preparation of this dissertation.

I. INTRODUCTION

A. HISTORICAL OVERVIEW

In the early days, Radar Data Processing (RDP) aboard surface ships was maintained by operators transcribing information reported by radar operators onto plexiglass status boards. This method requires a continuous high level of accuracy and vigilance from the people doing the tracking and, moreover, it has been recognized that this manpower-intensive process is too slow to support modern day requirements. [Ref. 1]

With the introduction of digital computers in Combat Systems, a man/machine partnership operation was created where the extraction of radar data was confined to small areas surrounding the predicted position of targets. The operator working on a plan position display had the responsibility of target initiation through an analysis of the radar picture. During operation, the man in control indicates the position of an intended target via rolling a ball marker. The computer evaluates an extraction area around the target, and from this point on the tracking is done under computer control. [Ref. 1]

In recent years, modern surveillance systems use automatic extraction of radar data. Target initiation, target correlation, target estimation, target prediction, and target termination may automatically be done under computer control reducing, to a minimum, the human participation portion of the man/machine partnership. [Ref. 2,3]

B. MOTIVATION

In military applications, the lower is the processing time, the higher is the time available for human decisions, the lower is the system reaction time to existing threats, and the higher is the system capability to shield an entire task force against an increasing number of threats.

Combat Systems Architectures are evolving from a single computer processing data from several sensors to several computers processing data from a single sensor.

In the US Navy, studies have been developed which set forth computer systems architecture concepts for the combat systems of the 2010-2030 timeframe that satisfy the needs of the next generation of surface combatants [Ref. 4]. Some of these concepts serve as motivation for this research:

> *Based on current capabilities, and anticipated future developments, the microprocessors used will be five times to one thousand times more powerful than the AN/UYK-43 of today. [Ref.4:p.50]*

> *Microprocessors should be chosen from among those widely used in the commercial world. These microprocessors should be militarized, but should continue to use commercial Instruction Set Architecture (ISA). One advantage to this is that commercially developed executive programs and run time systems will exist. [Ref.4:p.50]*

The proposed concept is to spread computers throughout the ship; Individual computers are located in or near the function they support. Concentrating computers in one place creates a geographical single point of failure susceptible to a single missile, torpedo, or shell. [Ref.4:p.51]

Projected requirements for future combat systems show a substantial need for increased computing capacity. Improvements in microprocessors are fueled by commercial demands, whereas development costs for AN/UYK-43 improvements are borne directly and entirely by the Navy. [Ref.4:p.52]

C. PREVIOUS RESEARCH

The California Institute of Technology Concurrent Computation Project, as part of a larger investigation of concurrent BM/C^3 computations requested by the U.S. Air Force Electronic Systems Division, has developed a multiple target tracker [Ref. 5,6]. Targets of interest are thrusting rockets being launched at regular time intervals from multiple sites. The sensor is geostationary with a fixed time interval between successive scans of data. Target initiation, target termination, target extension to a predicted position, target to measure association, and target filtering are automatically executed using a hypercube architecture. [Ref. 6]

The tracking algorithm is a Kalman filter adding system noise to allow the filter to respond to the actual accelerations of the target, and the correlation is executed using a single correlation gate [Ref. 6]. All sensor reports lying within the correlation gate are paired with the extended target [Ref. 6], which means that no ambiguity resolution is supported. Also, it appears that in the designed proposal every rocket is always detected after launch. That is, no probability of detection is being considered as well as its reflections in the size of association gates, target initiation, and target termination algorithms. Any target which has no association in its correlation gate is deleted from the target file [Ref. 6], which means that the target termination is decided in a single step. Two targets are merged if they are paired to the same sensor report during the last four scans of data [Ref. 6]. As we can see, in this proposed decomposition the merge requirement is a consequence of the absence of an ambiguity resolution algorithm to prevent the same measure being paired to more than one target.

The target file is distributed among the nodes of the hypercube, with each node having access to the full set of sensor reports at each scan, correlation and tracking algorithms are executed in the same node, and load balance is obtained by a redistribution of the target file. To avoid time consuming cube wide searches to apply the merge criterion, the assignment of targets to nodes must be such that all targets paired to the same measure must be assigned to the same node of the hypercube. To the extent possible, an algorithm was proposed to minimize the actual transfers of targets between nodes. [Ref. 6]

D. RESEARCH DESCRIPTION

This research specifies and validates a new concurrent decomposition scheme, called Confined Space Search Decomposition (CSSD), to exploit parallelism of radar data processing algorithms (correlation and tracking) using a distributed computational system.

1. The Application

The tracking algorithm is a Kalman filter, adding system noise to avoid filter divergence during target maneuver. The correlation algorithm uses two correlation gates to reduce the ambiguity resolution overhead, and the computation cost to evaluate probabilities of association. All sensor reports lying within the first correlation gate are paired with the extended target, and all residual measures are paired with all residual targets using a second correlation gate, applying an ambiguity resolution algorithm to avoid the same measure being paired to more than one target. After the two correlation stages, residual measures are classified as tentative targets, and residual targets are classified as terminating targets. Tentative targets must be submitted to a validation test to be confirmed as firm, and terminating targets must be submitted to a deletion test to be considered as lost.

Targets of interest are located within a circle in the X-Y plane centered at the radar site. All targets are detected with a fixed time interval between successive radar scans and have an associated probability of detection. Target initiation, target termination, target extension to a predicted position, target to measure association, and target filtering are automatically executed.

2. Object-Oriented Decomposition

An object-oriented decomposition of correlation and tracking algorithms is proposed. Client and server objects are identified, and a contract is specified which details the responsibilities of all identified objects.

3. The Decomposition Cost Evaluation Model (DCEM)

To formalize the specification we propose and apply an object-oriented methodology called Decomposition Cost Evaluation Model (DCEM). This methodology can be viewed as an extension to the object-oriented design process and produces as output a hint of the 'best' class hierarchy decomposition and topology for use in some application. This hint is obtained through conceptual and analytical comparisons among user identified options.

4. The Confined Space Search Decomposition (CSSD)

The Confined Space Search Decomposition proposal exploits parallelism of Radar Data Processing algorithms by:

1. Reducing the communication cost to transfer data among processors;

2. Overlapping correlation and tracking algorithms to avoid the traditional approach of all functions to all processors; and

3. Decomposing the total correlation problem into independent correlation problems of smaller size.

It uses a tree topology ($(1-C_p-T_p)$-tree) with an interface processor at the root node, C_p correlation processors at level 1, and T_p processors (for each correlation processor) at level 2.

To reduce the communication cost, the search space is divided into fixed size sections and each correlation processor is executing the target to measure association, working with all measures detected within some assigned number of successive sections in the tactical scene. Global load balance is obtained by adjusting the number of correlation sections assigned to correlation processors, and local load balance is obtained by a redistribution of the targets detected within the search space of some correlation processor among child tracking processors.

To support a smooth transition of a target when it crosses the correlation processor visibility space boundary an overlap space is defined. When the target is located within the overlap space of correlation processors, its estimation is reported by more than one correlation processor. This means that a merge algorithm is needed to support the compression of equivalent targets.

5. The Object Reincarnation Proposal

To reduce the penalties of load imbalance we propose a distributed dynamic load balance heuristic called Object Reincarnation (OR). In this proposal, objects viewed as computation sinks die in some processor site reducing its load and are reincarnated in another site increasing its load.

6. Research Validation

To validate the research we first compare the CSSD proposal with an identified alternative using the proposed DCEM model and then develop a theoretical prediction of selected parameters.

E. ORGANIZATION OF CHAPTERS

This dissertation is organized as follows:

Chapters II and III are dedicated to the analysis of tracking and correlation algorithms. The Kalman filter response and the size and shape of initiation, correlation, and merge gates are evaluated to support their use in the research.

In Chapter IV an object-oriented design, categorized as responsibility-driven, is applied to the software decomposition of correlation and tracking algorithms. Client and server objects are identified and specified, and a contract is specified which details the responsibilities of servers, clients, and required resources needed to execute the contract services. The single processor class hierarchy is also specified.

In Chapter V we specify the Confined Space Search Decomposition proposal. To formalize the specification we propose and apply an object-oriented methodology called Decomposition Cost Evaluation Model (DCEM). To reduce the penalties of load imbalance we propose a distributed dynamic load balance heuristic called Object Reincarnation (OR). The Confined Space Search Decomposition is compared with an identified alternative using the proposed DCEM model.

In Chapter VI we develop a theoretical prediction of performance, tracking capacity, and system reaction time of the Confined Space Search Decomposition. Best and worst cases are analyzed. To verify that the application supports a division of the search space in correlation sections and the object reincarnation in another processor site, we develop a simulation to check the tracking filter capability to reduce the measurement errors when targets cross the space search boundary of CPs.

Finally, Chapter VII concludes the research with a summary of significant results, strengths, weaknesses detected, and suggests future research directions.

II. TRACKING ALGORITHMS

This chapter provides an introduction to the Kalman filter and evaluates the filter response when system noise is added. We begin with a brief overview of α and $\alpha-\beta$ filters which provide a basis for the Kalman filter.

A. α AND $\alpha-\beta$ FILTERS

Radar measurements are represented by a discrete sequence of bearing and ranges. Those measurements carry, in addition to their inaccuracy, an associated uncertainty which is usually represented by additive noise. [Ref. 7]

Tracking algorithms process radar measurements to accomplish the following purposes [Ref. 2]:

1. Reduce the measurement errors by means of time averaging;
2. Estimate the position and velocity of the target; and
3. Predict future target position.

These algorithms can be implemented as ***Digital Filters***. A digital filter can be defined as a linear combination of an input and previous output sequence of values. As a particular case, the input sequence $x_n=x(t=n)$ represents measurements equally spaced in time and the output sequence $\hat{x}_n=\hat{x}(t=n)$ can be represented as:

$$\hat{x}_n = \sum_{k=1}^{+\infty} a_k \hat{x}_{n-k} + \sum_{k=-\infty}^{+\infty} b_k x_{n-k} \tag{2.1}$$

The filter coefficients are a_k and b_k.

If the input coefficients b_k are defined for k in the interval $[0,\infty)$ the filter is said to be ***Causal*** since the output does not depend on future input measures.

If all the previous output coefficients $a_k=0$, the filter is said to be ***Nonrecursive*** which means that there is no feedback from the output to the input. A ***Causal Recursive Filter***, used for position estimation can be represented as:

$$\hat{x}_n = \sum_{k=1}^{\infty} a_k \hat{x}_{n-k} + \sum_{k=0}^{\infty} b_k x_{n-k} \tag{2.2}$$

Of special interest is the filter represented as:

$$\hat{x}_n = (1-\alpha)\hat{x}_{n-1} + \alpha x_n$$

Or alternatively as:

$$\hat{x}_n = \hat{x}_{n-1} + \alpha(x_n - \hat{x}_{n-1})$$

If $\alpha=1$ then $\hat{x}_n=x_n$, if $\alpha=0$ then $\hat{x}_n=\hat{x}_{n-1}$, and if α is between 0 and 1 then the filter output is a weighted average of x_n and $\hat{x}_{n-1}$. That is, α reflects the confidence that we have in our measurement. With $\alpha=1$ we strongly believe in the measured value x_n and with $\alpha=0$ we will completely disregard the measure x_n and use the previous estimation $\hat{x}_{n-1}$.

This is called the α filter, where the coefficient α is the ***Position Filter Gain*** and $(x_n - \hat{x}_{n-1})$ is the ***Filter Residue***. This filter would be useful if applied to stationary targets for which measures of one coordinate are being taken.

When a target is stationary, its prediction is equal to its estimation in the previous iteration, thus:

$$x_p = \hat{x}_{n-1}$$

In this case, the position estimation is:

$$\hat{x}_n = x_p + \alpha(x_n - x_p)$$

Consider now a target moving with constant velocity. Along with the position equation to estimate the target velocity, we have:

$$\hat{v}_{xn} = v_{xp} + \beta(\, v_{xn} - v_{xp}\,)$$

Since most radars do not measure the velocity, we have to obtain this information from the position measurements assumed to be available every Δt, therefore:

$$v_{xn} = \frac{1}{\Delta t}(x_n - \hat{x}_{n-1})$$

$$v_{xp} = \frac{1}{\Delta t}(x_p - \hat{x}_{n-1})$$

$$v_{xn} - v_{xp} = \frac{1}{\Delta t}(\, x_n - x_p\,)$$

In summary, for targets moving with constant velocity we have:

1. *Estimation Equations*

$$\hat{x}_n = x_p + \alpha(x_n - x_p)$$

$$\hat{v}_{xn} = v_{xp} + \frac{\beta}{\Delta t}(x_n - x_p) \tag{2.3}$$

2. *Prediction Equations*

$$x_p = \hat{x}_{n-1} + \hat{v}_{x(n-1)}\Delta t$$

$$v_{xp} = \hat{v}_{x(n-1)} \tag{2.4}$$

If the measure is ahead of the prediction (i.e., $x_n > x_p$) then the velocity receives a positive correction. If the measure is behind the prediction (i.e., $x_n < x_p$) then the velocity receives a negative correction. If the measure is equal to the prediction then the velocity receives no correction (see Equations 2.3, 2.4).

This is called the α–β filter, where the new coefficient β is the ***Velocity Filter Gain***. The values of α and β must be selected to properly assess the attributes of:

1. Filter instability;

2. Noise reduction; and

3. Transient performance.

Figures (1,2, and 3) show the operational limits of (α, β).

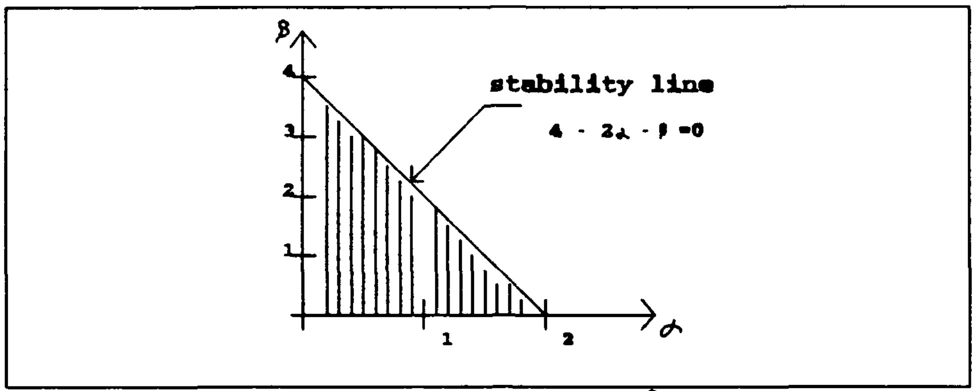


Figure 1 Allowed Values of α and β [Ref.2:p.182]

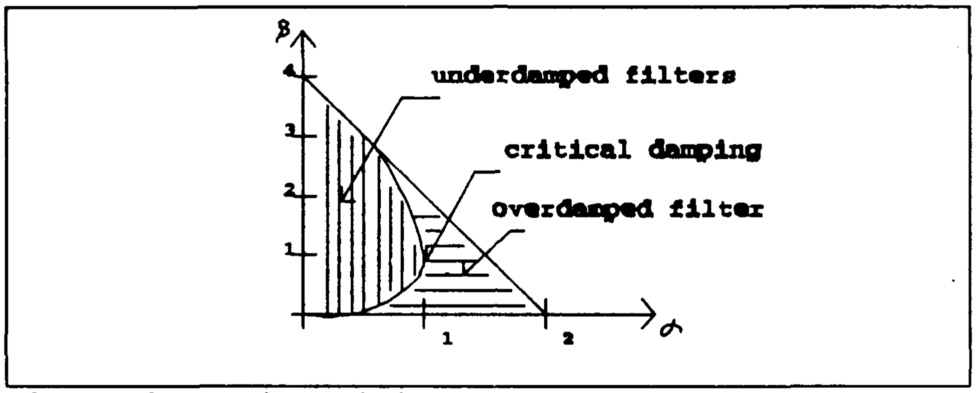


Figure 2 Damping Limits [Ref.2:p.183]

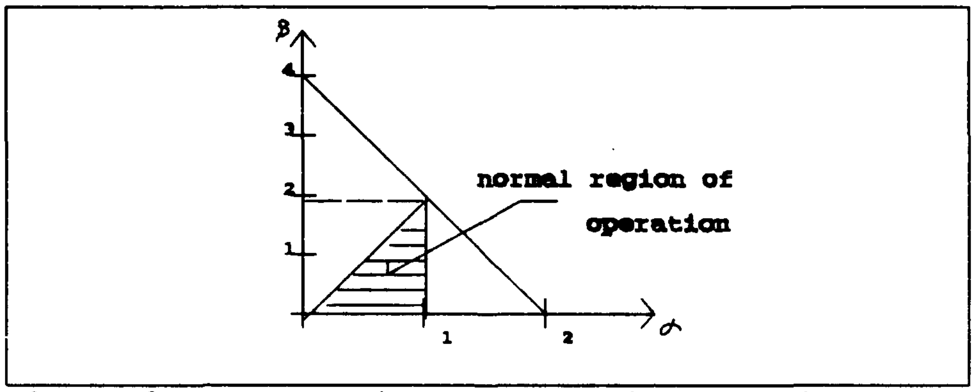


Figure 3 Normal Region of Operation [Ref.2:p.183]

But what is the meaning of 'properly assess' the attributes? Can we calculate some 'best' gain pair (α,B) to weigh the pair (measure,prediction) at each measurement? To answer these questions we need to introduce the Kalman filter.

B. THE KALMAN FILTER

The criteria used in the Kalman filter to select the best (α,β) pair are:

1. *Minimum Average Error; and*

2. *Minimum Square Error.*

Additionally, the errors in the observations of the targets are normally distributed. The filter equations may be summarized as follows:

1. Evaluate the filter gains (α,β) using the uncertainty of the prediction and measurement;

2. Find the estimated position and velocity of the target;

3. Compute the uncertainty of the estimated position and velocity of the target;

4. Find the predicted position and velocity of the target to be used in the next iteration; and

5. Compute the uncertainty of the predicted position and velocity of the target to allow the computation of the filter gains in the next iteration.

The actual equations can be derived as [Ref. 8, and 9]:

1. **The Filter Gain**

The best gain (α, β) , at the iteration **n**, can be expressed as:

$$\alpha_{xn} = \frac{(\sigma_{xp})_n^2}{(\sigma_{xp})_n^2 + (\sigma_x)_n^2}$$

(2.5)

$$\beta_{xn} = \frac{\Delta t * (\sigma_{xvxp})_n^2}{(\sigma_{xp})_n^2 + (\sigma_x)_n^2}$$

where:

$(\sigma_{xp})_n^2$ is the variance of position after prediction;

$(\sigma_{vxp})_n^2$ is the variance of velocity after prediction;

$(\sigma_{xvxp})_n^2$ is the covariance between position and velocity after prediction; and

$(\sigma_x)_n^2$ is the variance of the measure assumed to be 1% of the measure value.

2. **Estimated Position and Velocity**

As in the α-β filter:

$$\hat{x}_n = x_p + \alpha(x_n - x_p)$$

(2.6)

$$\hat{v}_{xn} = v_{xp} + \frac{\beta}{\Delta t}(x_n - x_p)$$

3. **Uncertainty in the Estimated Position and Velocity**

The uncertainty in the estimated position and velocity can be computed as:

$$(\sigma_{\hat{x}})^2_n = (1 - \alpha)(\sigma_{xp})^2_n$$

$$(\sigma_{\hat{v}x})^2_n = (\sigma_{vxp})^2_n - \frac{\beta}{\Delta t}(\sigma_{xvxp})^2_n \tag{2.7}$$

$$(\sigma_{\hat{x}\hat{v}x})^2_n = (1 - \alpha)(\sigma_{xvxp})^2_n$$

where:

$(\sigma_{\hat{x}})^2_n$ is the variance of position after estimation;

$(\sigma_{\hat{v}x})^2_n$ is the variance of velocity after estimation; and

$(\sigma_{\hat{x}\hat{v}x})^2_n$ is the covariance between position and velocity after estimation.

4. Predicted Position and Velocity

In a first order system (i.e., constant velocity):

$$x_{p(n+1)} = \hat{x}_n + \hat{v}_{xn} * \Delta t$$

$$\tag{2.8}$$

$$v_{xp(n+1)} = \hat{v}_{xn}$$

5. Uncertainty in the Predicted Position and Velocity

The uncertainty in the predicted position and velocity can be expressed as:

$$(\sigma_{xp})^2_{n+1} = (\sigma_{\hat{x}})^2_n + 2*\Delta t(\sigma_{\hat{x}\hat{v}x})^2_n + \Delta t^2(\sigma_{\hat{v}x})^2_n$$

$$(\sigma_{vxp})^2_{n+1} = (\sigma_{\hat{v}x})^2_n \tag{2.9}$$

$$(\sigma_{xvxp})^2_{n+1} = (\sigma_{\hat{x}\hat{v}x})^2_n + \Delta t(\sigma_{\hat{v}x})^2_n$$

6. Filter Initiation

When a new target starts the tracking phase, the filter needs to be initiated with the following initial values:

1. Predicted position and velocity of the target; and
2. Uncertainty in the predicted position and velocity of the target.

The predicted position and velocity (x_p, v_{xp}) are transferred by the initiation algorithm (which is described in the next chapter).

The uncertainty in the predicted position can be computed as:

$$(\sigma_{xp})_1^2 = (0.01*x_p)^2$$

$$(\sigma_{vxp})_1^2 = \frac{(2*v_{xp})^2}{x_p}\ , \ x_p \geq 12 \qquad (2.10)$$

$$(\sigma_{vxp})_1^2 = \frac{(2*v_{xp})^2}{12}\ , \ x_p < 12$$

$$(\sigma_{xvxp})_1^2 = 0.0$$

7. Maneuver Detection

The Kalman filter, as introduced in this section, works for targets with constant course and speed. If the target maneuver after the filter has settled with an optimal gain, the measures will not affect the gain and the filter will not follow the maneuver. To prevent this 'disconnection from reality,' we can assume that targets undergo random accelerations and simulate this behavior by adding system noise. The effect of the system noise is to increase the prediction uncertainty; the position filter gain (α) is driven towards unity, thus improving the maneuver response.

It is important not to confuse the system noise with the measurement noise. The measurement noise appears due to the inaccuracy and uncertainty in the measurement and is simulated by a random number generator. The system noise is placed in the model to simulate targets undergoing random accelerations.

This new model is expressed as [Ref. 8]:

a. *Random Acceleration Coefficient*

$$a_{n+1} = \frac{x_n - x_p}{\sqrt{(\sigma_{xp})^2_n + (\sigma_x)^2_n}} \qquad (2.11)$$

b. *Predicted Position and Velocity*

$$x_{p(n+1)} = \hat{x}_n + \hat{v}_{xn} * \Delta t + 0.5 * a_{n+1} * \Delta t^2$$

$$(2.12)$$

$$v_{xp(n+1)} = \hat{v}_{xn} + a_{n+1} * \Delta t$$

c. *Uncertainty in the Prediction*

$$(\sigma_{xp})^2_{n+1} = (\sigma_{\hat{x}})^2_n + 2\Delta t(\sigma_{\hat{x}\hat{v}x})^2_n + \Delta t^2(\sigma_{\hat{v}x})^2_n + 0.25 * a^2_{n+1} * \Delta t^4$$

$$(\sigma_{vxp})^2_{n+1} = (\sigma_{\hat{v}x})^2_n + a^2_{n+1} * \Delta t^2 \qquad (2.13)$$

$$(\sigma_{xvxp})^2_{n+1} = (\sigma_{\hat{x}\hat{v}x})^2_n + \Delta t(\sigma_{\hat{v}x})^2_n + 0.5 * a^2_{n+1} * \Delta t^3$$

8. Tracking in the X-Y Plane

As introduced in Chapter I, our targets of interest are located within a circle in the X-Y plane centered at the radar site. Radar measurements are represented by a discrete sequence of bearing and ranges, and targets are detected with a fixed time interval.

The equations that relate range and bearing with the X-Y coordinates are:

$$X = R\sin(\theta)$$
$$Y = R\cos(\theta) \tag{2.14}$$

If we assume as a reasonable hypothesis that the errors in polar coordinates are small compared with the true target coordinates, then the filtering process is independent for the coordinates X and Y. Thus, the computation can be done with two filters one for each coordinate. [Ref. 2,8]

C. FILTER RESPONSE ANALYSIS

In this section we evaluate the Kalman filter response to validate its use in this research.

1. Methodology

To analyze the filter response, a program was developed to filter simulated radar plots. Radar plots represent a time evolution of noisy measurements. Radar measurements carry an associated uncertainty which is usually represented by additive noise. The errors in the observations of the targets are normally distributed. Figure 4 depicts a schematic diagram of the structure used.

A pair of independent Gaussian variables $g_{1,2} = N(0,\sigma_{1,2})$ can be obtained from two independent uniformly distributed variables $u_{1,2}$ using the Box-Muller algorithm [Ref. 2]:

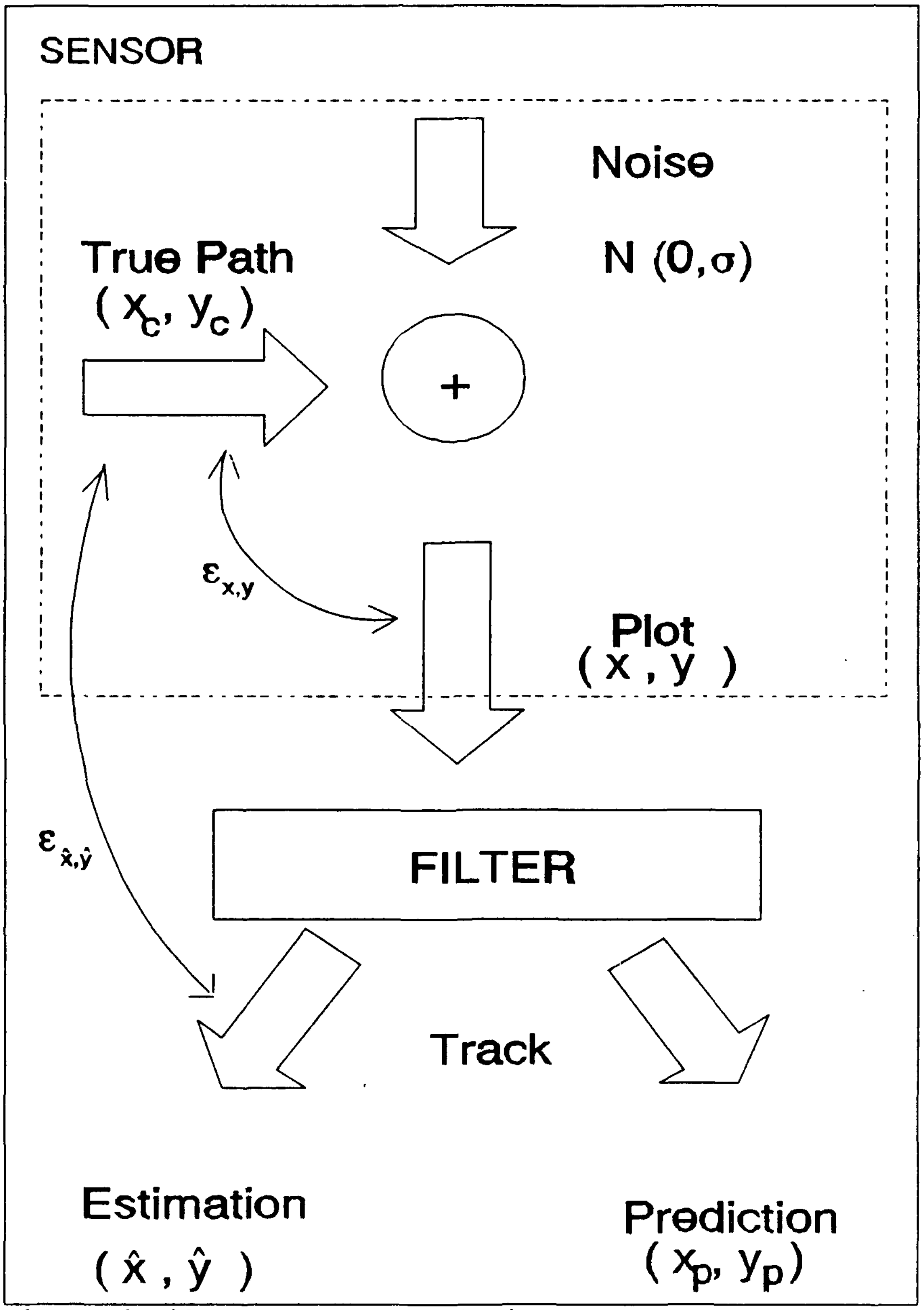


Figure 4 Filter Response Analysis

$$g_1 = \sigma_1 * \sqrt{-2 \ln (1 - u_1)} \cos 2\pi u_2$$
$$g_2 = \sigma_2 * \sqrt{-2 \ln (1 - u_1)} \sin 2\pi u_2 \tag{2.15}$$

The sensor detection can be simulated as:

$$x = x_c + \sigma_1 * g_1$$
$$y = y_c + \sigma_2 * g_2$$
$$\sigma_1 = 0.01 * x_c \tag{2.16}$$
$$\sigma_2 = 0.01 * y_c$$

The evaluation program was organized into two loops. The inner loop refers to the time progression of the target motion along its path. The outer loop performs a set of statistically independent trials. The output obtained in each trial is not by itself representative of the system behavior. For each trial, average square errors along the path are computed. The mean value of the errors over **N** trials are computed by averaging the errors over the trials.

The following square errors were considered in the evaluation:

$$\epsilon_{\hat{x}}^2 = (\hat{x} - x_c)^2$$

$$\epsilon_{\hat{y}}^2 = (\hat{y} - y_c)^2 \qquad (2.17)$$

$$\epsilon_x^2 = (x - x_c)^2$$

$$\epsilon_y^2 = (y - y_c)^2$$

where:

$\varepsilon_{\hat{x}}^2$, $\varepsilon_{\hat{y}}^2$ are the estimation square errors;

ε_x^2 , ε_y^2 are the measurement square errors;

x_c , y_c are the true target coordinates;

x , y are the detection coordinates; and

$\hat{x}$, $\hat{y}$ are the estimation coordinates.

The Measure Mean Square Error (MMSE) and the Estimation Mean Square Error (EMSE) can be computed as:

$$(MMSE)_{x,y} = \frac{1}{N} \sum_{j=1}^{j=N} [\frac{1}{S} \sum_{i=1}^{i=S} (\epsilon_{x,y}^2)_i]_j$$

$$(2.18)$$

$$(EMSE)_{x,y} = \frac{1}{N} \sum_{j=1}^{j=N} [\frac{1}{S} \sum_{i=1}^{i=S} (\epsilon_{\hat{x},\hat{y}}^2)_i]_j$$

where:

N is the number of trials;

S is the number of radar detections in each trial;

i is the inner loop index; and

j is the outer loop index.

To measure the efficacy of the filter we defined filter improvement as:

$$(FI)_{x,y} = \left(\left(\frac{\sqrt{(MMSE)_{x,y}}}{\sqrt{(EMSE)_{x,y}}} \right) - 1 \right) * 100\ \% \qquad \textbf{(2.19)}$$

2. Assumptions

During the analysis the following assumptions were used:

1. A single target is being tracked;

2. For each trial, the target is detected with 100% certainty at all iterations of the loop along the path; and

3. The single target path is detected with a fixed time interval and the number of observations in each trial is S=112.

It was observed that after N = 100 trials there was no significant difference in the mean value of the errors being computed for each path, which means that an increase in the external loop size would only affect the validation processing time. However, we used N = 200 trials to achieve accurate evaluation with safety.

3. Target Motion Model

The analysis of the filter response was implemented using the following motion models:

1. Outbound helicopter with constant course and speed;

2. Outbound helicopter with constant course and acceleration; and

3. Aircraft in circular flight with constant angular speed.

4. Conclusions

Table 1 depicts the results of the evaluation and the main conclusion can be expressed as:

The Kalman filter is decreasing the measurement errors in the motion models used in the analysis.

D. MULTIFILTER ALGORITHMS

The Kalman filter as introduced in this chapter is a simplified algorithm. Multifilter algorithms can also be used to solve the maneuvering target problem; However, these algorithms are too computation intensive to be used with SISD architectures [Ref. 10]. Table 2 depicts the serial execution time of the ***Interacting Multiple Model*** (IMM) and ***Viterbi Algorithms*** (VA) running in the National Semiconductors DB32332 boards [Ref. 10].

TABLE 1 FILTER RESPONSE EVALUATION

Path	X-MMSE	X-EMSE	X-FI	Y-MMSE	Y-EMSE	Y-FI
Outbound $C=135^\circ$ $V_r=60$ m/s	737	153	119%	744	155	119%
Outbound $C=315^\circ$ $V_r=20$ m/s $A_r=0.5 m/s^2$	1034	229	112%	1047	234	112%
Circular (0,15000) $V_\theta=0.01 r/s^2$	7174	2354	75%	15059	6851	48%

TABLE 2 RUN TIME (SECONDS)(100 ITERATIONS) [REF.10:P.251]

Algorithm	Serial Time
IMM	66.54
VA	57.71

III. CORRELATION ALGORITHMS

This chapter is included in the dissertation to help readers not familiar with correlation algorithms. It also analyzes the size and shape of initiation, correlation, and merge gates.

A. INTRODUCTION

Radar measurements (plots) may be represented by a discrete sequence of X-Y coordinates assumed to be detected within a fixed time interval and with an associated probability of detection. Target predictions are computed using the Kalman filter as discussed in Chapter II. Correlation algorithms associate a set of plots with a set of target predictions to accomplish the following objectives [Ref. 2]:

1. Decide on the optimal assignment pair (plot,target);
2. Decide on the initiation of residual plots; and
3. Decide on the termination of existing targets.

When an optimal pair (plot,target) is made, the plot is used as a new input to the tracking filter to produce refined estimates of the target position and velocity, and to predict the target position in the next detection.

When a residual plot is found, a ***Tentative Target*** is created. Tentative targets are either confirmed as ***Firm***

Targets or destroyed as ***False Targets*** depending on the results of a confirmation algorithm.

When a residual target is found, a ***Terminating Target*** is created. Terminating targets are either reconfirmed as ***Firm Targets*** or destroyed as ***Lost Targets*** depending on the results of a confirmation algorithm.

B. THE PLOT TO TARGET CORRELATION PROBLEM

The simplest selection of plot-target pairing uses the smallest distance criterion in the association [Ref. 2]. The algorithm uses two correlation gates. The first gate makes no allowance for maneuver, which means that its size is 'small enough' to avoid ***Ambiguity Resolution***. Plots which are still left without association are tested against any remaining targets using a second gate, this time allowing target maneuvering. Because of the large size of this second correlation gate, ambiguity resolution might be required. However, this would normally only happen with maneuvering targets near each other.

The correlation gate is a region in the space centered on the predicted target position. The shape and size of the gate are determined so as to provide a high probability that the actual measurement, if detected, will lie within the gate. The detailed formulae involved in the calculations are quite complex and are as such unattractive for use in a Real-Time

System. One technique for computation load reduction is to use an approximate rectangular gate (XY-Plane) [Ref. 2].

Figure 5 depicts a schematic view of the plot to target correlation problem. The ambiguity resolution problem arises when more than one plot lies within the gate of one or more targets. The correlation gate of several targets can be overlapped. This means that the same plot can be the closest association of different targets. Those ambiguous situations occur when a target passes through a cluttered area or when several targets are in the same neighborhood, as in the tracking of a formation of aircraft. Plots falling within each target correlation gate are stored in ascending order of distance from the target predicted position.

One proposed solution to solve the ambiguity problem is the n-step closest association algorithm [Ref. 2]. In each step, a correlation table is constructed to mark the closest plot association of each residual target. All correlated plots and targets are identified to prevent wrong associations in future steps. When the same plot correlates with more than one target the closest association is chosen. This algorithm may be improved by adding a tag to each association with the probability of pairing correctness [Ref. 2]. This algorithm ends when [Ref. 2]:

1. All targets are correlated; or
2. All plots are correlated; or

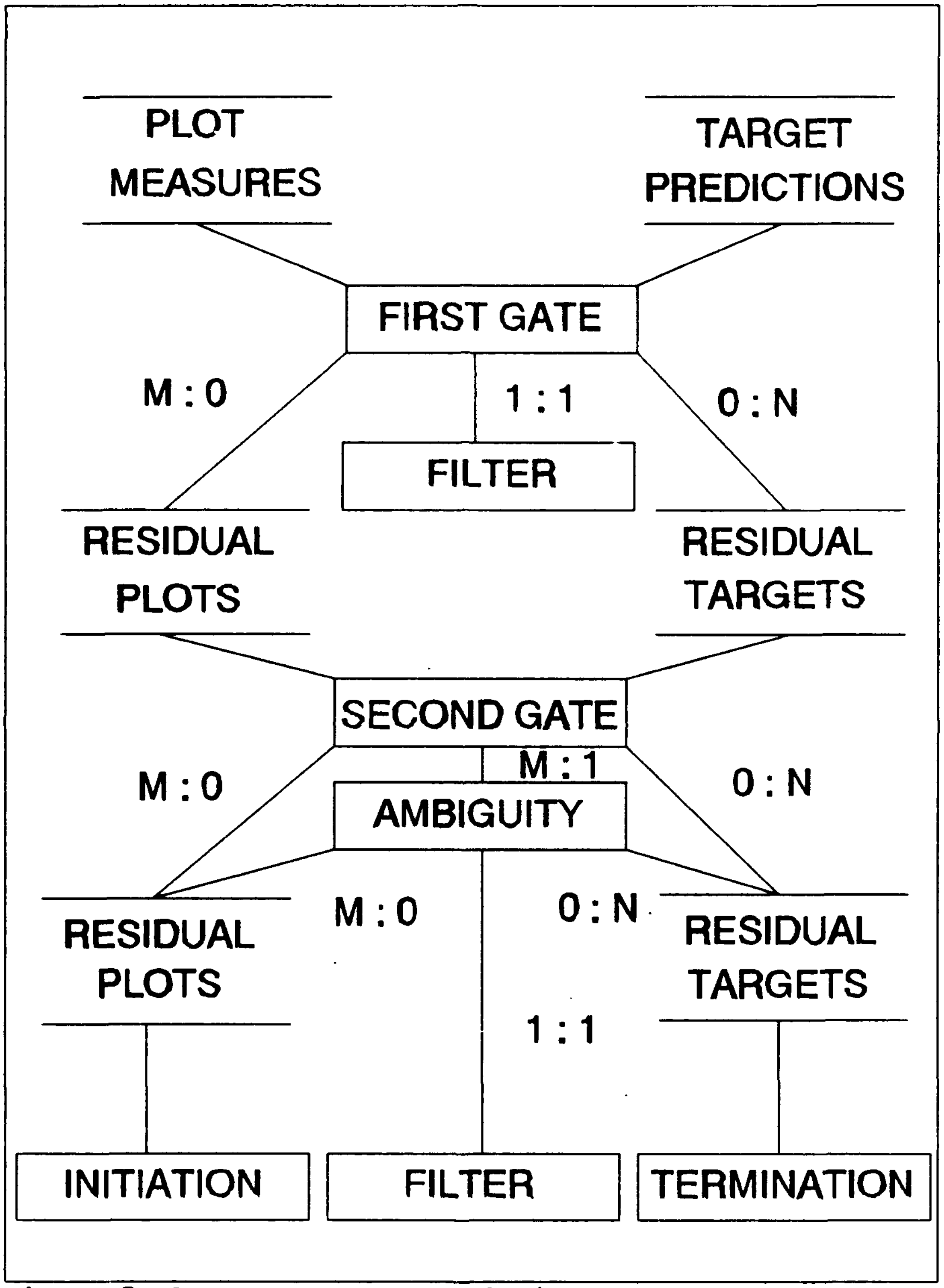


Figure 5 Plot to Target Correlation

3. All residual targets are with the correlation buffer empty.

To help the visualization of the termination conditions, we will simulate all the steps of the algorithm execution using three distinct examples, one for each condition.

Example A - All Targets Correlated (Figure 6):

Step 1 - The pairs (T_1,P_1), (T_3,P_2) and (T_5,P_3) are made because $(d_1<d_2)$, $(d_3<d_4)$, and $(d_5<d_6)$;

Step 2 - The pairs (T_2,P_4) and (T_4,P_5) are made because $(d_2<d_6)$ and P_5 correlates only with T_4;

Step 3 - The pair (T_6,P_6) is made; and

Step 4 - Correlation ends because all targets are correlated.

Example B - All Plots Correlated (Figure 7):

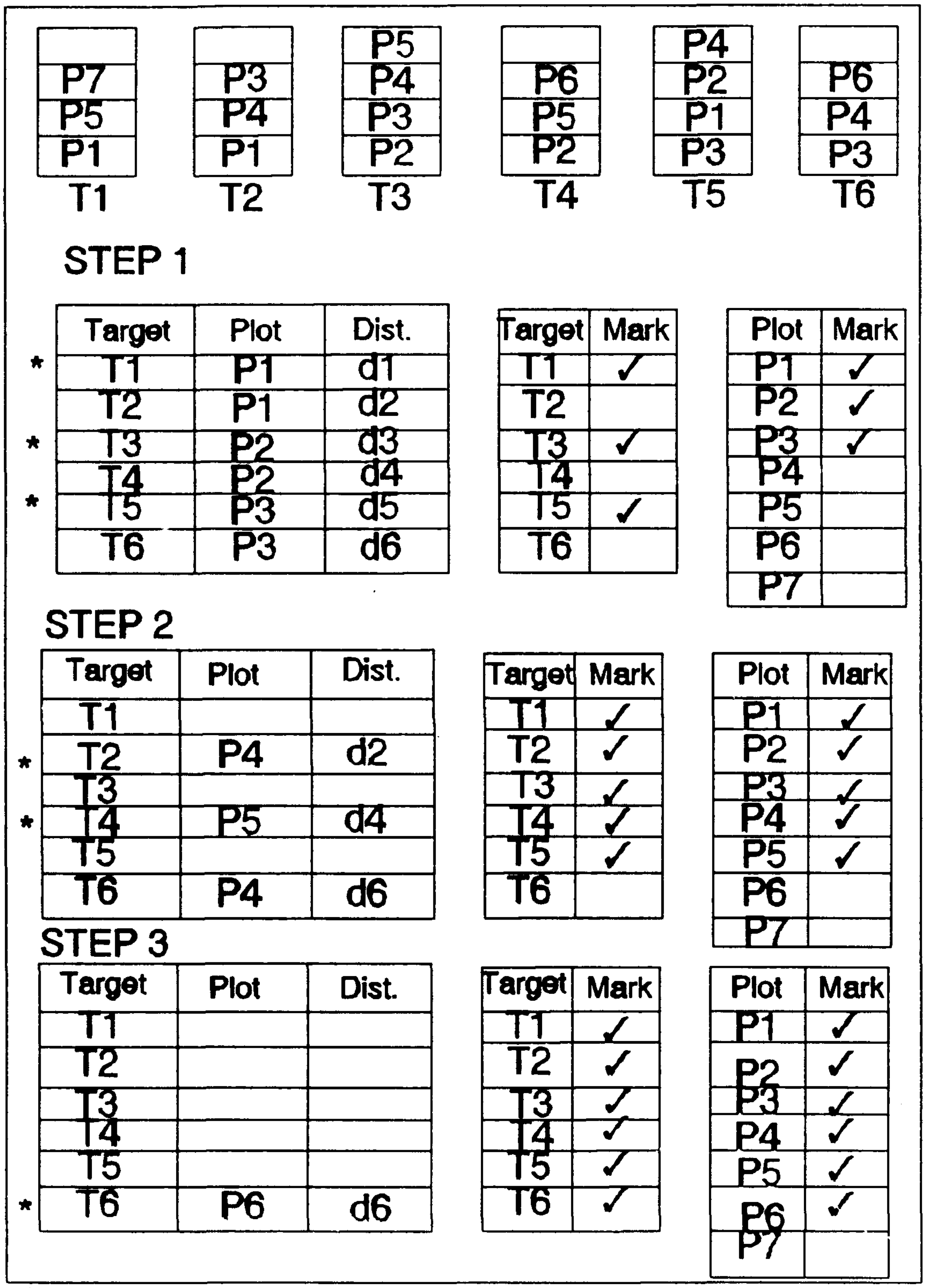
Step 1 - The pairs (T_1,P_1), (T_4,P_2) and (T_6,P_3) are made because $(d_1<d_2<d_3)$, $(d_6<d_5)$, and P_2 correlates only with T_4;

Step 2 - The pair (T_5,P_4) is made because P_4 correlates only with T_5. P_3 does not correlate with T_2 neither with T_3 because it was correlated with T_6 in step 1; and

Step 3 - Correlation ends because all plots are already correlated.

Example C - All Residual Targets with Empty Buffer (Figure 8):

Step 1 - The pairs (T_1,P_1), (T_3,P_2), (T_4,P_3), (T_5,P_4), and (T_6,P_6) are made because $(d_1<d_2)$ and P_2,P_3,P_4,P_6 correlate only with T_3,T_4,T_5, and T_6, respectively;



T1	T2	T3	T4	T5	T6
		P5		P4	
P7	P3	P4	P6	P2	P6
P5	P4	P3	P5	P1	P4
P1	P1	P2	P2	P3	P3

STEP 1

	Target	Plot	Dist.
*	~~T1~~	~~P1~~	~~d1~~
	T2	P1	d2
*	~~T3~~	~~P2~~	~~d3~~
	~~T4~~	~~P2~~	~~d4~~
*	~~T5~~	~~P3~~	~~d5~~
	T6	P3	d6

Target	Mark
~~T1~~	✓
T2	
~~T3~~	✓
~~T4~~	
~~T5~~	✓
T6	

Plot	Mark
~~P1~~	✓
P2	✓
~~P3~~	✓
P4	
P5	
P6	
P7	

STEP 2

	Target	Plot	Dist.
	~~T1~~		
*	T2	P4	d2
	~~T3~~		
*	~~T4~~	P5	d4
	~~T5~~		
	~~T6~~	P4	d6

Target	Mark
~~T1~~	✓
T2	✓
~~T3~~	✓
~~T4~~	✓
~~T5~~	✓
~~T6~~	

Plot	Mark
~~P1~~	✓
P2	✓
~~P3~~	✓
P4	✓
P5	✓
P6	
~~P7~~	

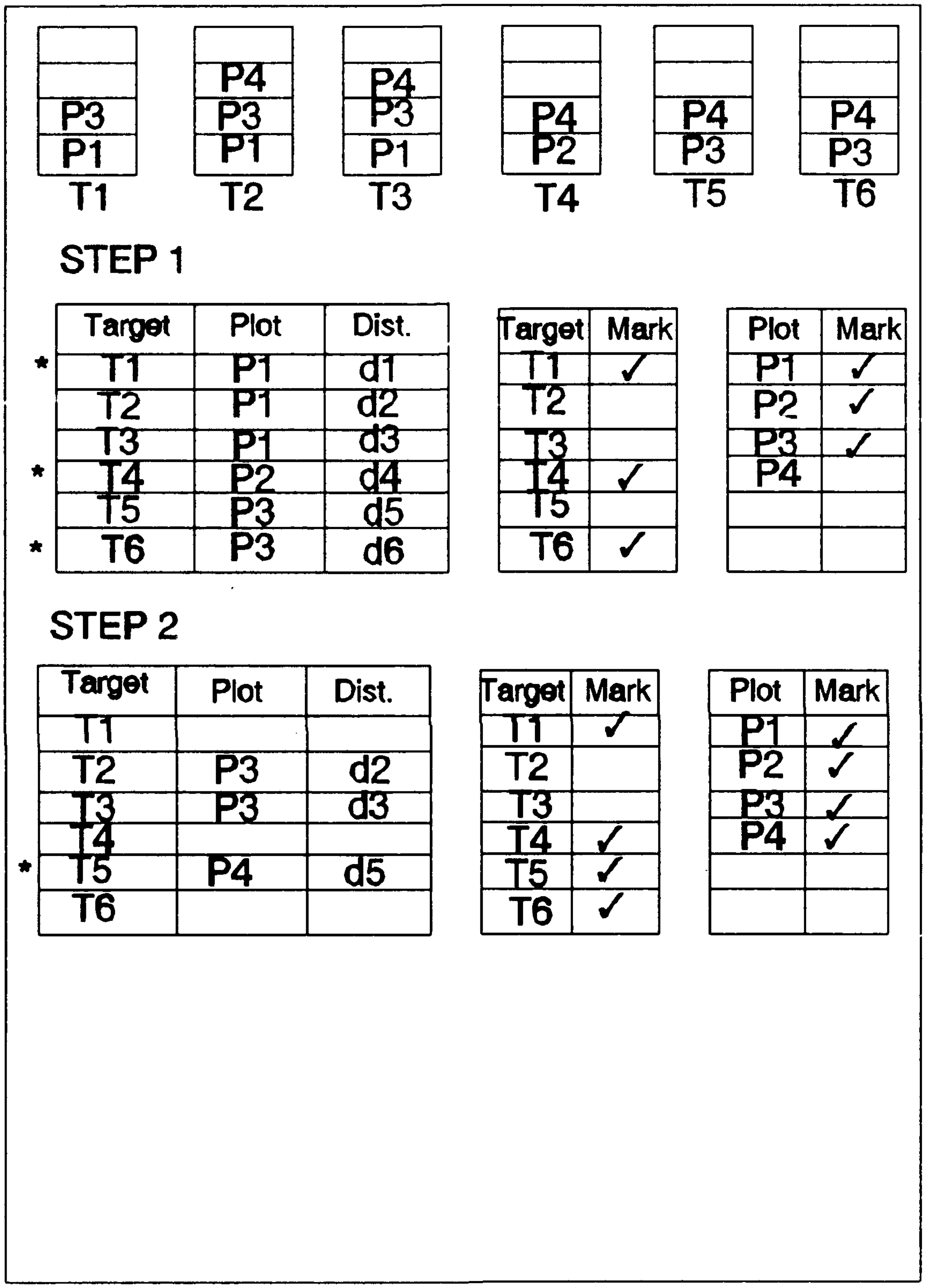
STEP 3

	Target	Plot	Dist.
	~~T1~~		
	~~T2~~		
	~~T3~~		
	~~T4~~		
	~~T5~~		
*	~~T6~~	P6	d6

Target	Mark
~~T1~~	✓
T2	✓
~~T3~~	✓
~~T4~~	✓
~~T5~~	✓
~~T6~~	✓

Plot	Mark
~~P1~~	✓
~~P2~~	✓
~~P3~~	✓
P4	✓
P5	✓
~~P6~~	✓
~~P7~~	

Figure 6 All Targets Correlated



T1	T2	T3	T4	T5	T6
	P4	P4			
P3	P3	P3	P4	P4	P4
P1	P1	P1	P2	P3	P3

STEP 1

	Target	Plot	Dist.
*	T1	P1	d1
	T2	P1	d2
	T3	P1	d3
*	T4	P2	d4
	T5	P3	d5
*	T6	P3	d6

Target	Mark
T1	✓
T2	
T3	
T4	✓
T5	
T6	✓

Plot	Mark
P1	✓
P2	✓
P3	✓
P4	

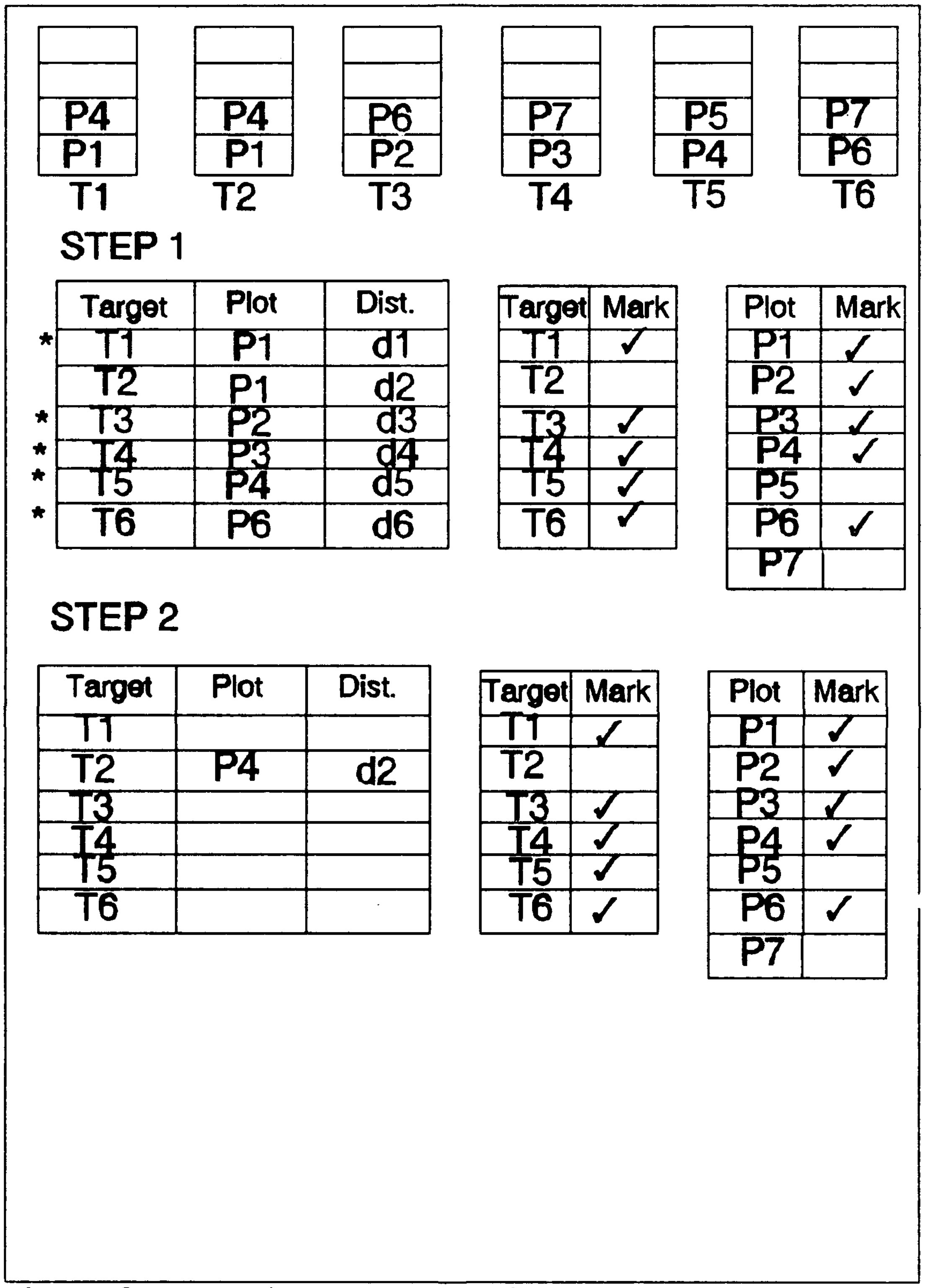
STEP 2

	Target	Plot	Dist.
	T1		
	T2	P3	d2
	T3	P3	d3
	T4		
*	T5	P4	d5
	T6		

Target	Mark
T1	✓
T2	
T3	
T4	✓
T5	✓
T6	✓

Plot	Mark
P1	✓
P2	✓
P3	✓
P4	✓

Figure 7 All Plots Correlated



T1	T2	T3	T4	T5	T6
P4	P4	P6	P7	P5	P7
P1	P1	P2	P3	P4	P6

STEP 1

	Target	Plot	Dist.
*	~~T1~~	~~P1~~	~~d1~~
	T2	P1	d2
*	~~T3~~	~~P2~~	~~d3~~
*	~~T4~~	~~P3~~	~~d4~~
*	~~T5~~	~~P4~~	~~d5~~
*	T6	P6	d6

Target	Mark
~~T1~~	✓
T2	
~~T3~~	✓
~~T4~~	✓
~~T5~~	✓
T6	✓

Plot	Mark
~~P1~~	✓
P2	✓
~~P3~~	✓
P4	✓
~~P5~~	
P6	✓
P7	

STEP 2

Target	Plot	Dist.
~~T1~~		
T2	P4	d2
~~T3~~		
~~T4~~		
~~T5~~		
T6		

Target	Mark
~~T1~~	✓
T2	
~~T3~~	✓
~~T4~~	✓
~~T5~~	✓
T6	✓

Plot	Mark
~~P1~~	✓
P2	✓
~~P3~~	✓
~~P4~~	✓
~~P5~~	
P6	✓
P7	

Figure 8 All Residual Targets With Empty Buffer

Step 2 - No pair is made because P_4 was correlated with T_5 in step 1; and

Step 3 - Correlation ends because all residual targets (T_2) are with the correlation buffer empty.

In Chapter IV an object-oriented implementation of the n-step closest association algorithm is proposed.

C. TARGET CONFIRMATION, INITIATION, AND TERMINATION

1. Target Confirmation

Target confirmation is the procedure used to verify if a tentative target should be confirmed as a firm target or destroyed as a false target, and to verify if a terminating target should be reconfirmed as a firm target or destroyed as a lost target. The confirmation algorithm is based on the ***Sequential Test*** [Ref. 11]. In the sequential test the decision is based on the target observation of the ratio of actually correlated echoes to demanded echoes.

Figures 9 and 10 depict the initiation and termination decision algorithms, where:

A - decision threshold for target acceptance;

B - decision threshold for target rejection;

M - demanded echoes;

M_{Lim} - maximum allowed number of samples;

k - good echoes returned; and

u - sample function.

$M < M_{Lim}$			$M \geq M_{Lim}$	
$u \geq A$	$u \leq B$	$B < u < A$	$k/m \geq 0.5$	$k/m < 0.5$
Firm Target	False Target	Uncertainty Wait Next Sample	Firm Target	False Target

Figure 9 Initiation Decision Algorithm

$M < M_{Lim}$			$M \geq M_{Lim}$	
$u \geq A$	$u \leq B$	$B < u < A$	$k/m \geq 0.5$	$k/m < 0.5$
Firm Target	Lost Target	Uncertainty Wait Next Sample	Firm Target	Lost Target

Figure 10 Termination Decision Algorithm

The sample function can be computed as:

$$u = \left(\frac{p_{d1}}{p_{d0}}\right)^{k} * \left(\frac{1 - p_{d1}}{1 - p_{d0}}\right)^{m - k} \qquad (3.1)$$

where:

p_{d1} is the lower limit of the probability of detection of a real target; and

p_{d0} is the upper limit of the rate of false returns.

Both p_{d1}, and p_{d0} are imposed by environmental conditions. Typical values are [Ref. 11]: $p_{d1} = 0.9$, $p_{d0} = 0.06$, $A = 239.2$, and $B = 0.01131$.

To avoid the uncertainty condition during a large number of samples, we will use $M_{Lim} = 5$.

2. Target Initiation

Target initiation is the procedure by which a new target entering the radar coverage is acquired by the tracking system of the sensor. During target initiation, tentative targets are created and submitted to the sequential test to decide upon true or false targets. The main objective of any automatic initiation procedure is to initiate targets shortly after detection. On the other hand, the procedure should prevent initiation of false targets to avoid an overload of the tracking filter in environments with low probability of detection.

The association of residual plots with tentative targets uses an ***Initiation Gate***. The initiation gate is a square gate centered on the tentative target's predicted position. The initiation algorithm may be outlined as follow:

a. New Tentative Target

The prediction and estimation can be computed as:

$$(x_p)_2 = x_1 \qquad (\hat{x})_1 = x_1 \tag{3.2}$$

b. Tentative Target with Correlated Measure

During the initiation phase the measure is considered more reliable than the prediction, therefore:

(1) Target Confirmed as Firm. Transfer to the tracking filter x_n and $(v_{xp})_n$ to allow the computation of $(x_p)_{n+1}$ and $(\hat{x})_n$. Also, destroy the tentative target.

(2) Uncertainty. The prediction and estimation can be expressed as:

$$(x_p)_{n+1} = x_n + (x_n - x_{n-1})$$

$$(v_{xp})_{n+1} = \frac{x_n - x_{n-1}}{\Delta t} \tag{3.3}$$

$$(\hat{x})_n = x_n$$

c. ***Tentative Target without Correlated Measure***

In this situation the prediction is the best information available, therefore:

(1) *Target Confirmed as False*. The required action is to destroy the tentative target.

(2) *Uncertainty*. The prediction and estimation can be expressed as:

$$(x_p)_{n+1} = (x_p)_n + [(x_p)_n - x_{n-1}]$$

$$(v_{xp})_{n+1} = \frac{(x_p)_n - x_{n-1}}{\Delta t} \qquad (3.4)$$

$$(\hat{x})_n = (x_p)_n$$

3. Target Termination

Target termination is the procedure used to determine if the target was lost owing to lack of subsequent plots. During target termination, all terminating targets are submitted to the sequential test to decide upon reconfirmation as firm or target lost. Terminating targets are transferred to the tracking filter and when no correlation measure is found the prediction replaces the measure in the tracking algorithm. The termination algorithm may be outlined as follow:

a. New Terminating Target

The required actions are:

1. Modify the target status from firm to terminating; and

2. Transfer the target prediction as a replacement to the measure in the tracking algorithm.

b. Terminating Target with Correlated Measure

(1) Target Reconfirmed as Firm. Modify the target status from terminating to firm and transfer the target correlated measure to the tracking algorithm.

(2) Uncertainty. Transfer the correlated measure to the tracking algorithm.

c. Terminating Target without Correlated Measure

(1) Target Lost. The required action is to destroy the target.

(2) Uncertainty. Transfer the target prediction as a replacement to the measure in the tracking algorithm.

D. THE MERGE PROBLEM

Target estimation reports may be represented by a discrete sequence of X-Y coordinates assumed to be available every radar scanning.

Parallel processing architectures splits the radar data processing functions among several processors. When any target estimation is reported by more than one processor a ***Merge Gate*** is used to support the identification of equivalent targets.

The merge gate is a square gate centered on target estimation reports.

E. GATES SPECIFICATIONS ANALYSIS

In this section we analyze the size and shape of initiation, correlation and merge gates to validate theirs use in the research.

1. Methodology

The methodology used in the analysis is equivalent to the one used in the analysis of the Kalman filter in Chapter II, with the following differences:

1. Analysis of initiation and correlation gates: For each trial, the number of hits within the gate is computed, averaged over the trials, and compared with the maximum number of possible hits; and

2. Analysis of the merge gate: All target estimations reported during a fixed time interval are compared with each other to see if they lie within the merge gate. For each trial, the number of hits inside of the gate is computed, averaged over the trials, and compared with the maximum number of possible hits.

2. Correlation Gates: Dimensions and Shape

The gates dimensions and shape are specified as a function of the prediction standard deviation, which is assumed to be 1% of the prediction value. Therefore:

$$\sigma_{xp} = 0.01 * x_p \tag{3.5}$$

Table 3 depicts the dimensions and shape of initiation, correlation and merge gates used during the validation tests.

TABLE 3 CORRELATION GATES: DIMENSIONS AND SHAPE

Gate	Size	$Size_{min}$	Shape
Initiation	$g = 2*12\sigma$	3000 m	Square
First	$g = 2*3\sigma$	40 m	Rectangular
Second	$g = 2*12\sigma$	2000 m	Rectangular
Merge	$g = 2*4\sigma$	40 m	Square

3. Assumptions

During the analysis the following assumptions were used:

1. A single target is being tracked by two processors;

2. Number of trials: N = 200 trials;

3. Number of samples/trial: S = 112 samples/trial;

4. Processor 1, samples/trial: S_1 = 56 samples/trial;

5. Processor 2, samples/trial: S_2 = 66 samples/trial;

6. Overlap between processors: O = 10 samples;

7. Minimum number of samples to allow initiation: $(S_{min})_i$=4 samples; and

8. Probability of detection: variable (P_d=1.0, P_d=0.95, and P_d=0.9).

From the assumptions used, we would like to highlight the following topics:

1. Minimum number of hits in the initiation gate of each processor: Min(hits initiation) = 200 trials * 4 hits/trial = 800 hits;

2. When the number of hits in the initiation gate of each processor is greater than 800, this means that more than 4 hits were necessary in the initiation algorithm and/or a firm target had the contact lost and a new tentative target was created;

3. Maximum number of hits (Processor 1): Max(hits initiation+first+second) = 56 hits/trial * 200 trials = 11200 hits;

4. Maximum number of hits (Processor 2): Max(hits initiation+first+second) = 66 hits/trial * 200 trials = 13200 hits;

5. When the number of hits within the initiation gate plus the number of hits within the first gate plus the number of hits within the second gate is less than the maximum number of hits for each processor, this means that a tentative target missed a hit in the initiation gate during the confirmation phase and/or a firm target missed a hit in the first and second gates (modifying its status to terminating);

6. Maximum number of hits in the merge gate: Max(hits merge) = 200 trials * 10 hits/trial = 2000 hits; and

7. When the number of hits in the merge gate is less than 2000 , this means that the merge gate was not able to identify the dual report of the single target as equivalent in some trials.

4. Target Motion Model

The single target motion model can be expressed as:

1. Initial position: $(x,y) = (1500,0)$;

2. Course: $C = 000°$;

3. Motion $t \in [0,50s]$: $V_y = 350$ m/s, $A_y = 0$; and

4. Motion $t \in [51s,111s]$: $V_{0y} = 350$ m/s, $A_y = 70m/s^2$ (7g).

5. Results

Table 4 depicts the results obtained during the gate integration tests. The hits within the merge(MG), initiation(IG), first(FG) and second gates(SG) of the processors 1(P1) and 2(P2) are evaluated as a function of the probability of detection P_d.

From these results, we can conclude that when the probability of detection decreases:

1. The number of hits in the initiation gate is greater than 800. That is, the number of hits needed to modify the status of a tentative target to firm target is increasing and/or firm targets had the contact lost and a new tentative

target was created due to the arrival of new noncorrelated measures;

2. The number of hits in the first gate is decreasing and the number of hits in the second gate is increasing. That is, the computed predictions are getting worse, bringing, as a consequence, an increase in the ambiguity resolution overhead;

TABLE 4 GATES INTEGRATION TESTS

P_d	Hit MG	Hit IG P1	Hit FG P1	Hit SG P1	Hit IG P2	Hit FG P2	Hit SG P2
1.0	1998	800	9563	837	800	11477	923
.95	1970	864	9477	859	816	11390	994
.90	1938	993	9246	961	897	11256	1047

3. The number of hits in the merge gate is decreasing. That is, the computed estimations are getting worse and the capability of the merge gate to identify equivalent targets is decreasing. However, the hit rate is always greater than 95%, which we consider to be a good compromise between the requirement of small gates to avoid different targets being reported by different processors to be considered as a single one and the requirement of large gates to avoid the same

target being reported by different processors to be considered as different ones.

In general, we can also say that:

1. The sum of hits within the initiation, first, and second gates is equal to the maximum number of hits for each processor. That is, for all detected measures we always have a hit in one of the gates; and

2. The number of hits in the first gate is greater than 90% of the hits in the first and second gates, which we consider to be a good balance between the requirement to reduce the ambiguity resolution overhead and the requirement to avoid different plots to be paired to the same target.

6. Conclusions

The results obtained endorses the following conclusions:

1. The initiation, correlation and merge gates as specified in Table 3 are well balanced to be used in this research.

2. A hit ratio greater than 90% is achieved within the first correlation gate for the target motion model used in the analysis.

F. TARGET SPLITTING ALGORITHMS

An alternative proposal to correlation algorithms is known as ***target splitting***. Under this scheme, the tracking system does not have to commit itself immediately or irrevocably to a single assignment of each report. If a plot is highly correlated with more than one target, ***hypothesis targets*** can be created. Subsequent reports can be used to determine which assignment is correct. [Ref. 12]

One worrisome consequence of the target splitting algorithm is known as ***target explosion*** (i.e, a proliferation in the number of hypothesis targets that a program must keep tabs on). The proliferation can be controlled with the same target-deletion mechanism used in the nearest-neighbor algorithm, which scans through all the targets from time to time and eliminates those that have a low probability of association with recent plots. However, since two hypothesis targets may lock onto the trajectory of a single target the standard target-deletion mechanism has to be modified to detect redundant targets. [Ref. 12]

Two-phase gating algorithms may also be used. However, in the target splitting algorithm the correlation gate is a region in the space centered on plots (the number of hypothesis targets is expected to be greater than the number of plots). All candidates located within the first correlation gate are committed as hypothesis targets. Plots which are still left without association are tested against any remaining targets using a second larger gate. Because of the large size of this second correlation gate, pruning might be required to reduce the penalties imposed by the target explosion problem. A simple pruning strategy is to select the *n* hypothesis candidates with the highest probability of association, where *n* is the maximum number of hypotheses that computational resource constraints will allow [Ref. 12].

G. COMPUTATION COST OF GATING OPERATIONS

The most obvious proposals for multiple-target correlation makes the difficulty of an n-target problem proportional to n^2. Experiments developed at the Naval Research Laboratory with thousands of targets produced encouraging results. In these experiments target predictions are stored as points in a multidimensional tree data structure. Then for each plot a gating range is defined, and the tree is searched for all target predictions falling within the range. Each such search requires at most $n^{2/3}+k$ operations, and in many instances the actual performance is appreciably better. [Ref. 12]

The execution time of tree-based association algorithms on a personal workstation for 128,000 targets is a little more than 10 minutes [Ref.12:p.141]. That is, the average cost per target is about 5 ms.

The computation cost of gating operations can be reduced by:

1. The use of two association gates. This avoids the calculation of probabilities of association for pairs obtained during the first gate correlation phase;

2. The use of tree-based association algorithms; and

3. The decomposition of the total correlation problem of size N-target into C_p independent correlation problems of size N/C_p, which can be processed in parallel.

Our research exploits parallelism of Radar Data Processing algorithms (correlation and tracking) by:

1. Reducing the communication cost to transfer data among processors;

2. Overlapping correlation and tracking algorithms; and

3. Decomposing the total gating problem into independent gating problems of smaller size.

IV. OBJECT-ORIENTED DECOMPOSITION

In this chapter an object-oriented design is applied to the decomposition of correlation and tracking algorithms. Client and server objects are identified, a contract is specified which details the responsibilities of all identified objects, the object structures are specified, and the initial class hierarchy is defined. However, the initial design is conceived without regard for the hardware architecture. Later, in Chapter V, the decomposition of objects to processors is proposed and the design is then adjusted to fit the proposed architecture.

A. INTRODUCTION

It is assumed that the reader is at least somewhat familiar with object-oriented (OO) terminology. However, a brief introduction is included as the terminology often varies greatly from one system to another. For a more complete introduction, the reader should refer to [Ref. 13,14, and 15].

An ***object*** can be defined as an entity with a self-contained set of variables (representing the object's state) which can only be manipulated by a set of ***methods*** (procedures) defined exclusively for that purpose. A ***message*** is sent to an object to tell it to execute one of its methods. A ***class*** can then be defined as a description of similar objects. It can be likened to a template or a cookie cutter [Ref. 15]. An object is sometimes referred as an ***instance*** of a class.

The variables making up an object can be either class variables or instance variables. A ***class variable*** is one which is shared both in name and value by all instances of a class (i.e., changing the value of a class variable for an object causes the value to be changed for all instances of that class). A ***instance variable*** is shared in name only by all instances of a class (i.e., changing the value of an instance variable for an object has no affect on any other instance of that class).

Methods can also be categorized as either class methods or instance methods. A ***class method*** is executed when a message is sent to a class, while an ***instance method*** is executed when a message is sent to an instance of a class (note that this is quite different from the concept of class and instance variables).

Inheritance can be defined simply as a code sharing mechanism. It allows a new class to be defined based upon the definition of an existing class without having to manually copy all of the existing code. A ***subclass*** inherits all of the variables and methods defined for its ***superclass***.

A ***class hierarchy*** can be represented as a tree structure which indicates the inheritance relationship between the various classes. If a class is allowed to have multiple superclasses, then the system is said to support ***multiple inheritance*** (MI); the inheritance relationship diagram developed in a MI environment is technically a ***lattice***, but is commonly referred to more simply as a hierarchy.

Objects can also be composed of other objects, in which case they are called ***composite objects*** (or ***aggregate objects***). That is, some of the variables making up the object are themselves objects, called ***dependent objects***. Composition and inheritance are the primary building blocks used in constructing object-oriented systems.

Object-oriented programming (OOP) is becoming widely accepted as a viable approach to nearly any programming project. In a concurrent OOP system, many forms of concurrency are possible [Ref. 16]. We may send messages to different objects, causing several objects doing things concurrently. We may send several messages to the same object, causing it to perform several methods concurrently. We may also be able to have a single method for an object do several things concurrently.

Object-oriented design (OOD) has been categorized as being either data-driven or responsibility-driven [Ref. 17]. With a ***data-driven approach***, it is the structure of the data which drives the design. It is claimed that this approach violates encapsulation in that it makes the structure of the object a part of its definition, and that this leads to operations which reflect the given structure [Ref. 17].

The ***responsibility-driven approach*** is based upon the ***client/server model*** where the interaction between the client and the server is described in a contract specifying ***what*** the server does for the client rather than ***how*** the server does it [Ref. 17].

However, it has also been argued that these two approaches are essentially one and the same, as long as the designer strives for a high degree of encapsulation. That is, the structure does not have to drive the design, it simply indicates that the object is responsible for providing that information on demand. [Ref. 18]

To date, there is no design methodology that is universally accepted by the object-oriented community. In this research we apply the responsibility-driven approach to the decomposition of correlation and tracking algorithms. Object specification and the class hierarchy definition will use the language-independent syntax as presented in [Ref. 18]. Class names are written in CAPITALS, variable and method names are written in SMALL CAPITALS, variable types are written in normal type, and variable values are written in *italics*. Figure 11 depicts an example of the used syntax notation.

B. OBJECT IDENTIFICATION

An object is both an encapsulation and an abstraction: an encapsulation of attributes and exclusive services on those attributes; and an abstraction of the problem space,

CLASS: CLASS NAME	
Superclasses:	SUPERCLASS 1, SUPERCLASS 2, ...
Class Variables:	CLASS VARIABLE 1: type [*default value*]
	CLASS VARIABLE 2: type [*default value*]
	⋮
Instance Variables:	INSTANCE VARIABLE 1: type [*default value*]
	INSTANCE VARIABLE 2: type [*default value*]
	⋮
Methods:	METHOD 1
	METHOD 2
	⋮

Figure 11 Language-Independent Class Definition [Ref.18:p.3]

representing an occurrence of ***something*** in the problem space. Identification of objects requires a considerable knowledge of the problem space. Chapters II and III introduced the dissertation application area, and they are used as the basis for the object identification problem.

1. The Servers

Servers provide a set of services upon request. The services to be supported are classified into the following phases:

a. *Detection Phase*

During the detection phase, radar measurements must be simulated. This means that a sensor object must provide the service ***Report Radar Detections*** (see Equation 2.16).

b. *Correlation Phase*

During the correlation phase, the n-step closest association algorithm, the initiation algorithm, and the termination algorithm must be executed. Each firm, terminating, or tentative target must provide the service

Report your best Correlation Proposal (see Figures 6,7, and 8). Firm and terminating targets must support the same services and herein will be considered as objects of the same class.

c. *Tracking Phase*

Firm and tentative targets must support the service ***Execute Tracking***. Firm targets execute the request using the Kalman filter (see Equations 2.6, 2.10, and 2.12) and tentative targets using the initiation algorithm (see Equations 3.2, 3.3, and 3.4).

d. *Compression Phase*

During the compression phase equivalent targets must be identified. In this application, equivalence is a user interface filter which prevents a single target from being presented to the user as multiple targets when its estimation is reported by more than one processor in a distributed computational system. Firm and tentative targets provide the service ***Are you Equivalent to the Object Passed as Parameter?***. This question may be rephrased as: ***Is the Object Passed as Parameter Lying Within your Merge Gate?***.

e. *Presentation Phase*

Targets not recognized as equivalents must support the service ***Store yourself in Secondary Memory***.

f. Server Object Summary

In summary, we identified ***sensor***, ***firm***, and ***tentative*** targets as the server objects of our problem space.

2. The Client

Again, looking at the problem space we identify the requirement to create and destroy objects dynamically. During each data scanning, ***something*** needs to ask the sensor object to report data detections. These detections need to be visible by firm and tentative objects during correlation report requests. Firm objects renewed with the best correlation report are asked to execute tracking using the Kalman filter. Tentative objects renewed with the best correlation report are asked to execute tracking using the initiation algorithm. Firm and tentative objects updated with estimations are paired to check equivalence. Firm and tentative objects not considered as equivalents are stored in secondary memory to support data analysis.

The ***controller*** of those operations will be our client, herein named the ***scheduler*** object.

C. CONTRACT SPECIFICATION

A contract specifies the responsibilities of the identified objects (sensor, firm, tentative, and scheduler) and the required resources to execute the services. During the specification of responsibilities we will also specify the input/output interface. If desired, the reader may refer to

Sections D and E to foresee the details (such as the data types) involved associated with the class definitions.

1. Responsibility of the Sensor Object

A single responsibility was identified:

a. *Simulate Radar Detections*

Targets of interest are located within a circle (radius=200 NM) in the X-Y plane centered at the radar site. All targets are detected with a fixed time interval between successive detections (ΔT=3s) and with an associated probability of detection ($0.9 \leq P_d \leq 1.0$). The motion models of interest are inbound/outbound targets with constant radial velocity ($v_{x,y} \leq 700$ m/s) or acceleration ($a_{x,y} \leq 70$ m/s^2), and targets in circular movement with constant angular speed.

(1) Interface Specification.

Method: SIMRADET.

Input: None.

Output: DETECTIONS: buffer of DETECTION.

2. Responsibilities of Firm Objects

With the problem space in mind, we identify the following responsibilities of firm objects:

a. *Sort Correlation Buffer*

Measures to be paired with firm objects must be grouped in ascending order of distance from the object itself (see Figures 6, 7, and 8). Correlation proposals are presented in order of distances, beginning with the closest.

(1) Interface Specification

Method: SOCOBU.

Input: COBU: buffer of PAIRING.

Output: SOBU: buffer of PAIRING.

b. Report Correlation Proposal: First Gate

During the first gate correlation phase a single request is made by the scheduler because all detections lying within the first gate are considered equivalent. If no proposal is available, an invalid measure is reported to advise the scheduler.

(1) Interface Specification.

Method: RECOPRFG.

Input: None.

Output: PROPFG: PAIRING.

c. Report Correlation Proposal: Second Gate

Firm objects that are not correlated during the first gate correlation phase are asked by the scheduler to propose the best correlation obtained with a larger gate. Proposals presented during the second gate correlation phase (M-Flag) may need several request iterations (i.e., additional request for proposals may be needed) due to the possibility of ambiguity. If no proposal is available the present prediction is used as a replacement to the best correlated measure report (P-Flag).

(1) Interface Specification.

Method: RECOPRSG.

Input: None.

Output: PROPSG: PAIRING;

PROPFLAG: {*M*,*P*}

d. Report Current Status

The status of firm objects is one of the following: Firm (FI), Terminating (TE), or Dead (DE) (see Chapter III).

(1) Interface Specification.

Method: RECUST.

Input: None.

Output: MYST: {*FI*,*TE*,*DE*}.

e. Set Status

New firm objects are created with status firm. Objects with status firm are asked to set status as terminating when no correlation proposal is confirmed. Firm objects with status terminating are asked to modify their status depending on the results of the sequential test (see Figure 9).

(1) Interface Specification.

Method: SEST.

Input: NEWST: {*FI*,*TE*,*DE*}.

Output: None.

f. Execute Sequential Test

When a correlation proposal is confirmed (CC), a firm object (status terminating) must be asked by the scheduler to execute the sequential test to decide whether to remain terminating (U) or modify the status to firm (F). When no correlation is confirmed (CN), firm objects must be asked by the scheduler to execute the sequential test to decide whether to remain firm with status terminating (U) or dead due to contact lost (L). It is up to the scheduler object to destroy dead objects dynamically.

(1) Interface Specification.

Method: EXSEQTE.

Input: CORST: $\{CC, CN\}$.

Output: RESEQTE: $\{F, L, U\}$.

g. Execute Tracking

Firm objects must support the request to execute tracking using the Kalman filter. The internal state is updated with new predictions (see Equation 2.12) and current estimations (see Equation 2.16).

(1) Interface Specification.

Method: EXTR.

Input: COME: PAIRING.

Output: None.

h. *Check Equivalence*

Firm and tentative objects updated with estimations must be paired with each other to check equivalence. Objects are considered equivalent if they lie within the same merge gate. Equivalence is determined via a user interface filter which is used to prevent a single target from being presented to the user as multiple targets when its estimation is reported by more than one processor. Objects to be checked for equivalence must be paired by the scheduler. The pairing sequence is irrelevant and implementation dependent. It is important to emphasize that the object identified as equivalent is not to be destroyed because this would mean the destruction of a firm or tentative object and not just its external representation. Thus, the equivalence designation ensures that equivalent objects are displayed only one time.

(1) Interface Specification.

Method: CHKEQ.

Input: CHKOBJ: {*FIRM, TENTATIVE*}.

Output: ANSWER: {*Y, N*}.

i. *Store Yourself*

Firm objects not considered as equivalents must be stored in secondary memory to support data analysis.

(1) Interface Specification.

Method: STYOU.

Input: DRIVERID: integer.

Output: None.

3. Responsibilities of Tentative Objects

In the application description (see Chapters II and III), the following responsibilities were identified:

a. Sort Correlation Buffer

Measures to be paired with a tentative object must be grouped in ascending order of distance from the object itself (see Figures 6, 7, and 8). Correlation proposals are presented in order of distances, beginning with the closest.

(1) Interface Specification

Method: SOCOBU.

Input: COBU: buffer of PAIRING.

Output: SOBU: buffer of PAIRING.

b. Report Correlation Proposal: Initiation Gate

Tentative objects are asked by the scheduler to propose the best correlation obtained using the initiation gate. Proposals presented during the initiation gate correlation phase (M-Flag) may need several request iterations (i.e., additional request for proposals may be needed) due to the possibility of ambiguity. If no proposal is available, the present prediction is used as a replacement of the best correlated measure report (P-Flag).

(1) Interface Specification.

Method: RECOPRIG.

Input: None.

Output: PROPIG: PAIRING;

PROPFLAG: $\{M,P\}$.

c. *Report Current Status*

The status of tentative objects is one of the following: Confirming (CI), Confirmed as Firm (CF), or Dead (DE) (see Chapter III).

(1) Interface Specification.

Method: RECUST.

Input: None.

Output: MYST: $\{CI,CF,DE\}$.

d. *Set Status*

New tentative objects are created with status confirming (CI). They are then asked to modify their status depending on the results of the sequential test (see Figure 10).

(1) Interface Specification.

Method: SEST.

Input: NEWST: $\{CI,CF,DE\}$.

Output: None.

e. *Execute Sequential Test*

When a co-relation proposal is confirmed (GC), a tentative object must be asked by the scheduler to execute the sequential test to determine whether to keep the status as confirming (U) or modify the status to confirmed (F). When no correlation is confirmed (CN), a tentative object must be asked by the scheduler to execute the sequential test to decide whether to keep the status as confirming (U) or modify the status to dead (L). It is up to the scheduler object to dynamically create firm objects when tentative objects are confirmed as firm (F), and to destroy tentative objects either when they are confirmed as firm (F) or dead (L).

(1) Interface Specification.

Method: EXSEQTE.

Input: CORST: {*CC*, *CN*}

Output: RESEQTE: {*F*, *L*, *U*}.

f. Execute Tracking

Tentative objects must support the request to execute tracking using the initiation algorithm. The internal state is updated with new predictions and current estimations (see Equations 3.3 and 3.4).

(1) Interface Specification.

Method: EXTR.

Input: COME: PAIRING.

Output: None.

g. Check Equivalence

This service was described during the identification of the responsibilities of firm objects (see Section C2h).

(1) Interface Specification.

Method: CHKEQ.

Input: CHKOBJ: {*FIRM, TENTATIVE*}.

Output: ANSWER: {*Y,N*}.

h. Store Yourself

Tentative objects not considered as equivalents must be stored in secondary memory to support data analysis.

(1) Interface Specification.

Method: STYOU.

Input: DRIVERID: integer.

Output: None.

4. Responsibilities of the Scheduler Object

As previously discussed (see Section B2), the scheduler controls the operations of sensor, firm and tentative objects. Its responsibilities are implemented using a single method, RUN, started at the beginning of the execution phase.

a. Create the Sensor Object

At the beginning of execution the scheduler object is running alone. To start the simulation of radar detections, the sensor object is created.

*b. **Ask Report of Radar Detections***

During each data scanning the scheduler object must ask the sensor object to report radar detection. These reported detections are tagged with a measure identification number.

*c. **Set Correlation Buffer: First Gate***

During the first gate correlation phase, radar detections reported by the sensor object are set to be visible by firm objects. To execute this service the scheduler object must keep track of the Object Id of all firm objects.

*d. **Ask Correlation Proposal: First Gate***

Correlation proposals coming from firm objects during the first gate correlation phase must be accepted. It is responsibility of the scheduler object to update the set of measures still available for correlation and the list of Object Ids that did not submit any correlation proposal (wrong measure flag).

*e. **Set Correlation Buffer: Second Gate***

Measures not correlated during the first gate correlation phase are set to be visible by firm objects not yet correlated.

*f. **Ask Correlation Proposal: Second Gate***

Correlation proposals coming from firm objects during the second gate correlation phase are tagged with the correlation distance. When the same measure is reported to be the correlation proposal of more than one firm object it is the responsibility of the scheduler object to request another proposal. This cycle of request iterations will end when all firm objects have had one proposal accepted or when no additional proposals are available (in this situation, the present prediction is used as a replacement for the best correlated measure report). It is the responsibility of the scheduler object to update the set of measures still available for correlation and the list of Object Ids with no accepted correlation proposal.

g. Ask Status Report: Firm Objects

Firm objects with status dead must be destroyed, firm objects with status firm must execute the sequential test when no correlation proposal is accepted, and firm objects with status terminating must always execute the sequential test to decide whether to keep the status as terminating or modify it to dead or firm.

h. Destroy Firm Objects

Firm objects reporting the status dead must be destroyed. The Object Id is removed from the list of valid firm objects, the unused space is returned to the heap, and the Object Id may be reused.

i. Ask Sequential Test: Firm Objects

Request execution of the sequential test of firm objects with status terminating when a correlation proposal is confirmed. Request execution of the sequential test of firm objects when no correlation proposal is confirmed.

j. *Ask Set Status: Firm Objects*

Modify the status of firm objects either due to the result of the sequential test, or when the status of a firm object needs to be changed to terminating.

k. *Ask Tracking Execution: Firm Objects*

Firm objects renewed with the best correlation report are asked to update their internal state with new predictions and current estimations using the Kalman filter algorithm.

l. *Set Correlation Buffer: Initiation Gate*

Measures that could not be correlated with firm objects are set to visible for tentative objects. To execute this service, the scheduler must keep track of the Object Id of all tentative objects.

m. *Ask Correlation Proposal: Initiation Gate*

Correlation proposals coming from tentative objects during the initiation gate correlation phase are tagged with the correlation distance. When the same measure is reported to be the correlation proposal of more than one tentative object, it is the responsibility of the scheduler object to ask another proposal of the proposing object with the denied proposal. This cycle of request iterations will end when all tentative objects have had one proposal accepted, or when no additional proposals are available (in this situation, the present prediction is used as a replacement to the best correlated measure report). It is the responsibility of the scheduler object to update the set of measures still available for correlation and the list of Object Ids with no accepted correlation proposal.

*n. **Ask Status Report: Tentative Objects***

Tentative objects with status dead or confirmed must be destroyed and tentative objects with status confirming must always execute the sequential test to decide wheter to keep the status as confirming or modify it to dead or confirmed.

*o. **Create Firm Objects***

Firm objects are created when tentative objects report their status as confirmed. The Object Id is then inserted into the list of valid firm objects.

*p. **Destroy Tentative Objects***

Tentative objects are destroyed when reporting their status as dead or confirmed. The Object Id is removed from the list of valid tentative objects and the unused space is returned to the heap.

*q. **Ask Sequential Test: Tentative Objects***

Request execution of the sequential test of tentative objects with status confirming when a correlation proposal is confirmed to decide wheter to keep the status as confirming or modify it to confirmed or when no correlation proposal is accepted to decide wheter to keep the status as confirming or modify it to dead.

r. Ask Set Status: Tentative Objects

Modify the status of tentative objects due to the result of the sequential test. New tentative objects are created with status confirming.

s. Ask Tracking Execution: Tentative Objects

Tentative objects renewed with the best correlation report are asked to update their internal state with new predictions and current estimations executing the initiation algorithm.

t. Create New Tentative Objects

Residual measures not correlated in any of the previous requests become tentative objects. The Object Id of each newly created object is inserted into the list of valid tentative objects. New tentative Objects are initiated with status confirming.

u. Ask Equivalence Check

Firm and tentative objects renewed with estimations are paired with each other to check for equivalence. The request receiver and the object being checked are selected arbitrarily. This request needs several iterations and will end when all firm and tentative objects are paired.

v. Storage Request

Firm and tentative objects not considered as equivalents must be stored in secondary memory to support data analysis.

D. OBJECT STRUCTURE

Since the ***Object Behavior*** has been specified, we can now specify the ***Object Structure***.

1. Object Structure of the Sensor Object

The object structure of the sensor object includes instance variables to encapsulate the state of a simulated radar and its detections.

SP: (*3.0..5.0*) [*3.0*]. /* Sample Period */

PD: (*0.9..1.0*) [*1.0*]. /* Probability of Detection */

DETECTIONS: buffer of *DETECTION*. /* Detections */

2. Object Structure of Firm Objects

The object structure of firm objects includes all instance variables needed during the correlation, tracking compression, and presentation phases of this application. The correlation buffer will be declared as an inherited class variable since it must be visible by all firm objects.

TN: integer. /* Target Number */

COBU: buffer of PAIRING. /* Pairing Buffer */

SOBU: buffer of PAIRING. /* Sorted Pairing Buffer */

MYPROP: PAIRING. /* Correlation Proposal */

MYPR: PREDICTION. /* Position after Prediction */

MYES: ESTIMATION. /* Position after Estimation */

MYST: {*FI,TE,DE*} [*FI*]. /* Status */

FGSZ: SIZE. /* First gate dimensions */

SGSZ: SIZE. /* Second gate dimensions */

MGSZ: SIZE. /* Merge gate dimensions */

3. Object Structure of Tentative Objects

The object structure of tentative objects includes all instance variables needed during the correlation, tracking compression, and presentation phases of this application. The correlation buffer will be declared as an inherited class variable since it must be visible by all tentative objects.

TN: integer. /* Target Number */

COBU: buffer of PAIRING. /* Pairing Buffer */

SOBU: buffer of PAIRING. /* Sorted Pairing Buffer */

MYPROP: PAIRING. /* Correlation Proposal */

MYPR: PREDICTION. /* Position after Prediction */

MYES: ESTIMATION. /* Position after Estimation */

MYST: {*CI,CF,DE*} [*CI*]. /* Status */

IGSZ: SIZE. /* Initiation gate dimensions */

MGSZ: SIZE. /* Merge gate dimensions */

4. Object Structure of the Scheduler Object

The object structure of the scheduler object includes all instance variables needed to control existing firm and tentative objects. The correlation buffer will be declared as an inherited class variable since it must be set with radar detections needed by firm and tentative objects during the correlation phase.

```
COBU: buffer of PAIRING. /* Pairing Buffer */
COFI: buffer of CONTROL. /* Control Firm Alive */
COTE: buffer of CONTROL. /* Control Tent. Alive */
```

E. CLASS HIERARCHY

The result of an object-oriented design is a hierarchy of classes [Ref. 19]. Since the ***Object Structures*** have been specified we can now specify the ***Class Hierarchy***. Definitions for the classes discussed in this section are contained in Appendix A.

1. Component Classes

A ***composite object*** is an object which consists of other objects called ***component objects***. Instances of component classes may be implemented as dependent objects or subobjects. A ***dependent object*** is completely dependent upon its aggregate. ***Subobjects***, on the other hand, may exist as stand-alone objects in their own right [Ref. 13]. Since the four main classes of objects (sensor, firm, tentative, and scheduler) are all composite objects, their various components will be discussed first.

a. Class Prediction

Instances of the class prediction are dependent objects encapsulating the predicted position and velocity of firm or tentative objects.

b. Class Estimation

Instances of the class estimation are dependent objects encapsulating the estimated position and velocity of firm or tentative objects.

c. Class Detection

Instances of the class detection are dependent objects encapsulating simulated radar measurements of the sensor object.

d. Class Size

Instances of the class size are dependent objects encapsulating the dimensions of correlation gates of firm or tentative objects.

e. Class Pairing

Instances of the class pairing are subobjects visible to firm, tentative, and scheduler objects used during the association of firm and tentative objects with detections reported by the sensor object to the scheduler object.

f. *Class Control*

Instances of the class control are dependent objects of the scheduler object being used to control which firm and tentative objects are alive or dead.

2. Abstract Classes

An ***abstract class*** is a class which does not have any instances. It generally exists to be used only as an ancestor to other classes which may have instances [Ref. 13].

a. *Class Target*

In this application, the correlation buffer (COBU) is set by the scheduler and must be visible for read operations by firm and tentative objects during the execution of the sort method (SOCOBU). The class TARGET will be defined as a superclass of the concrete classes FIRM, TENTATIVE, and SCHEDULER.

3. Concrete Classes

A ***concrete class*** is one which does have instances, although it may also be used as an ancestor to other classes [Ref. 13]. Each concrete class is associated with one of the identified objects of our application (see Section B).

a. *Class Sensor*

The class sensor has a single instance (sensor object) responsible to simulate radar detections.

b. Class Firm

The class firm will have as many instances (firm objects) as the number of firm targets being tracked.

c. Class Tentative

The class tentative will have as many instances (tentative objects) as the number of tentative targets awaiting confirmation.

d. Class Scheduler

The class scheduler has a single instance (scheduler object) responsible to control the operations of sensor, firm and tentative objects.

4. Single Processor Class Hierarchy (SPCH)

Figure 12 depicts the single processor class hierarchy, and figure 13 depicts the client-server relationship. TARGET is an abstract class; SENSOR, FIRM, and TENTATIVE are concrete classes used to instantiate server objects; and SCHEDULER is a concrete class used to instantiate the client object (i.e., the controller).

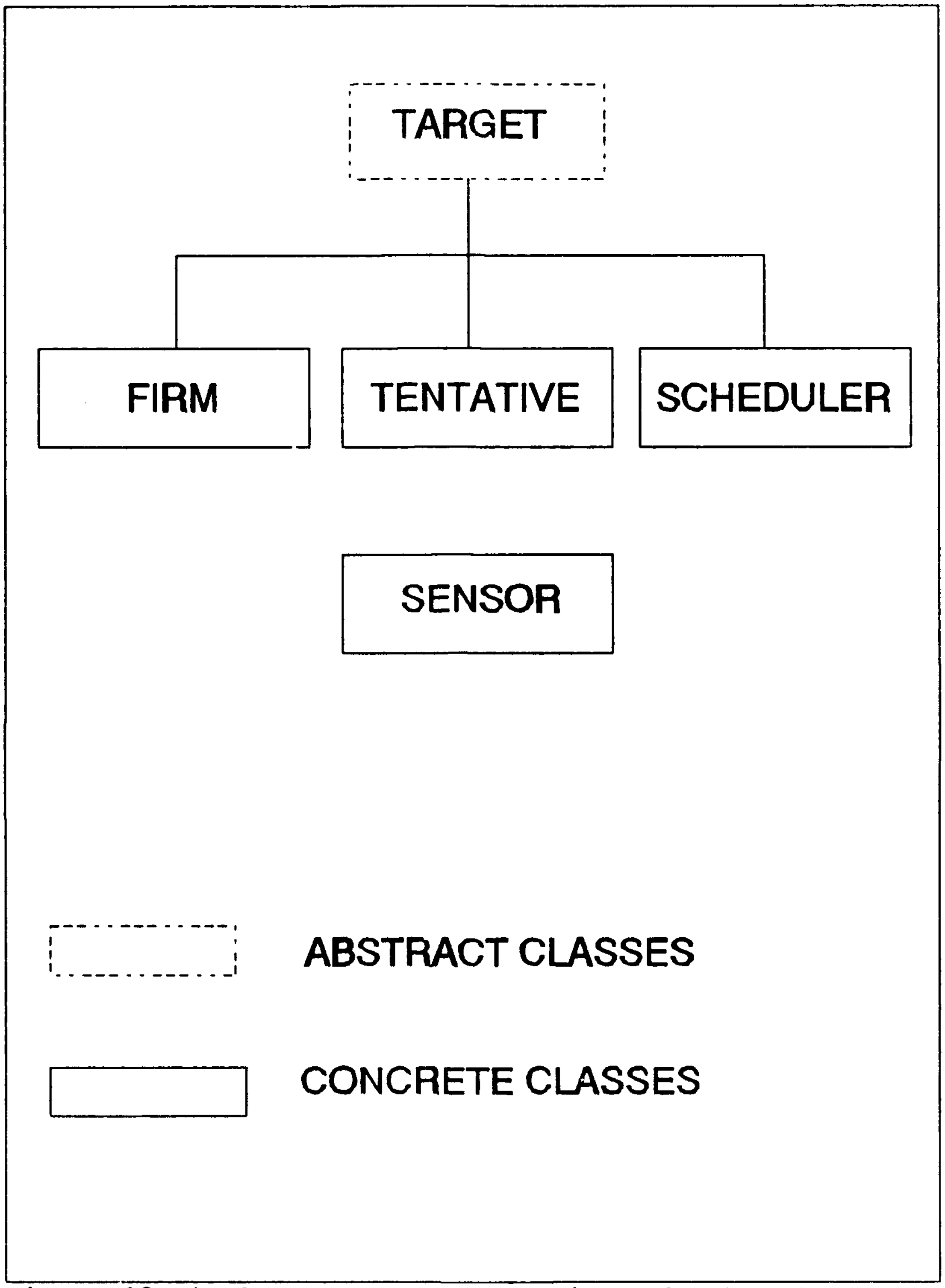


Figure 12 Single Processor Class Hierarchy (SPCH)

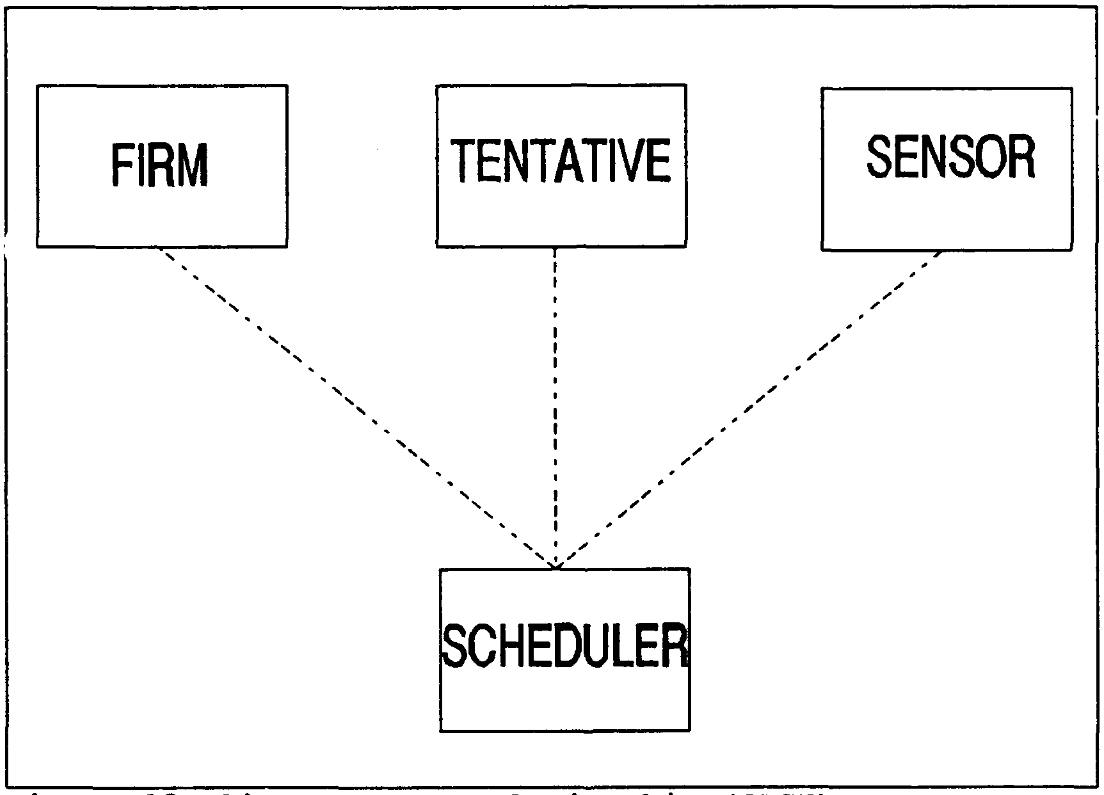


Figure 13 Client-Server Relationship (SPCH)

V. ARCHITECTURE SPECIFICATION

In this chapter we specify the ***Confined Space Search Decomposition*** (CSSD) proposal. To formalize the specification process we propose and apply to our case study an object-oriented methodology called ***Decomposition Cost Evaluation Model*** (DCEM). To reduce the penalties of load imbalance we propose a distributed dynamic load balancing heuristic called ***Object Reincarnation*** (OR).

A. INTRODUCTION

1. The Problem

a. *Scheduling*

An optimal solution to the task scheduling problem has been proven to be computationally hard (i.e., NP-complete) [Ref. 20,21, and 22]. Thus, obtaining optimal schedules is not practical [Ref. 23]. As a result, many of the research efforts in this area have focused on heuristic methods [Ref. 24,25]. Task distribution is important not only for the execution of application programs on distributed computational systems, but also for the design stage to determine a computer architecture specification which will perform better for a type of application [Ref. 26]. In general, when the number of computational sinks (tasks and concurrent objects) is greater than the number of processors, then some contraction steps are needed during the mapping specification [Ref. 27].

b. *Specification*

To formalize the specification we propose and apply to our case study an object-oriented methodology, called Decomposition Cost Evaluation Model (DCEM). In the Decomposition Cost Evaluation Model, the mapping problem is brought to a higher level of abstraction where the question is which classes should be loaded on which processors, and not which tasks (sometimes not well related) should be loaded on which processors. To support this decision we define communication and computation cost functions of class hierarchies.

c. *Analysis of Options*

The output of the Decomposition Cost Evaluation Model is a hint of the 'best' mapping proposal and of the 'best' interconnection among processors (topology) for use in the application being analyzed. This hint is obtained through conceptual and analytical comparisons among user identified options. Analytical comparisons are made considering that the efficiency (E) of a Distributed Computational System can be expressed as [Ref. 28]:

$$E = \frac{T_{seq}}{T_{conc}(N) * N}$$

where:

T_{seq} is the time to compute on one node; and

$T_{conc}(N)$ is the time to compute on N nodes.

That is, when we keep N and T_{seq} with the same value we can compare two proposals doing an evaluation of $T_{conc}(N)$ for each proposal.

d. *Confined Space Search Decomposition*

The Confined Space Search Decomposition (CSSD) proposal intends to exploit parallelism of radar data processing algorithms by:

1. Reducing the communication cost to transfer data among processors;

2. Overlapping correlation and tracking algorithms to avoid the traditional approach of all functions to all processors; and

3. Decomposing the total gating problem into independent gating problems of smaller size.

e. *Load Imbalance*

Distributed computational systems have been shown to be very efficient in solving problems that can be partitioned into tasks with uniform computation and communication patterns [Ref. 28,29]. Dynamic load balancing schemes are needed to efficiently solve non-uniform problems on distributed computational systems [Ref. 30]. Many load balancing techniques have been proposed and reviewed in the literature [Ref. 31, and 32].

(1) Receiver Initiated Diffusion (RID). A distributed load balancing strategy for improving the performance of a highly parallel multicomputer system, called The Receiver Initiated Diffusion (RID) method, was proposed in [Ref. 31]. In this approach, underloaded processors request proportionate amounts of load from overloaded neighbors which then dispatch a portion of their load to the requesting processor [Ref. 31]. Task migration is necessary in this approach, however. The eligibility of tasks for transfer is restricted to those tasks which have not yet begun execution. This procedure is intended to reduce the communication cost because the transfer of a task which has begun execution is expensive since it requires the storing of the processor's state [Ref. 31].

(2) Object Reincarnation (OR). Ideally, during the execution of any load balancing strategy, no communication costs should be incurred at all. In Radar Data Processing (RDP) applications, firm and tentative targets are objects which need to see correlated detections to remain alive. To reduce the penalties of load imbalance, a distributed dynamic load balancing heuristic, called Object Reincarnation (OR), is proposed herein. In this strategy we adjust the visibility space of correlation processors. Objects viewed as computation

sinks die in one processor site (reducing its load) and are reincarnated in another site (increasing its load). Our proposal also supports fault recovery as an extension to the load balance problem. It should be remembered, however, that all realistic decomposition methods are approximate as the load-balancing problem has not been proven to be solvable [Ref. 28].

2. Organization of Sections

In section B, we introduce the Decomposition Cost Evaluation Model to formalize the architecture specification.

In Section C, we summarize the results obtained when we apply the Decomposition Cost Evaluation Model to our application. The analysis developed to support the results is detailed in Appendix B.

In Section D, we detail the Confined Space Search Decomposition. The division of the search space into sections and space regions (triple of sections) as well as the mapping of space regions to processors is specified.

In Section E, we introduce the Object Reincarnation proposal to reduce the penalties of load imbalance. Global and local load imbalance algorithms are proposed.

In section F, we propose an algorithm for fault recovery using the Object Reincarnation approach.

B. THE DECOMPOSITION COST EVALUATION MODEL

In this section we propose a model founded upon four basic principles and nine guidelines to help determine what is the 'best' approach for use in the decomposition of objects to processors. Any decision made during the application of the guidelines must respect the basic principles.

The specification as a whole is an iterative process. Each decision taken may need to be reviewed several times before the class hierarchy is considered satisfactory. During each review we must:

1. Refine contract responsibilities;
2. Refine objects structures; and
3. Refine class hierarchies.

The four basic principles are:

1. ***Specialization Decomposition***: Concrete classes and their ancestors must be loaded on the same processor. This means that inheritance requires neither communication nor synchronization among processors.

2. ***Locality***: A single object cannot be loaded on more than one processor.

3. ***Load Balancing***: The decomposition of objects to processors must ensure that, as far as possible, each node is performing the same amount of work [Ref. 28].

4. ***Communication Cost***: The decomposition of objects to processors must reduce $T_{conc}(N)$ (i.e., the time to compute on

N nodes) [Ref. 28]. Time spent communicating can represent a degradation of performance compared to a sequential node.

The nine guidelines are:

1. Identification of concrete classes;
2. Identification of interface functions;
3. Identification of high cost functions;
4. Divide and conquer;
5. Identification of options;
6. Conceptual comparisons among options;
7. Evaluation of the communication cost;
8. Evaluation of the computation cost; and
9. Analytical comparisons among options.

1. Identification of Concrete Classes

Concrete classes can be viewed as a sink of computation time. We will shortly see that this is relevant information in the decomposition of objects to processors.

2. Identification of Interface Functions

Interface functions are natural candidates to be loaded on interface processors. Interface processors are responsible for delivering external data as well as for collecting results. In this step, the single processor hierarchy is divided into two hierarchies: the ***Interface Hierarchy*** which encapsulates interface functions in classes assigned to interface processors; and the ***Application***

Hierarchy which encapsulates application functions in classes assigned to application processors.

3. Identification of High Cost Functions

High cost functions are natural candidates to be executed in several processors. When a class definition includes computation intensive methods, a review of the application hierarchy may be advised. New classes are defined to encapsulate those methods in different processors; in this alternative the application hierarchy is partitioned into ***Function Hierarchies***.

4. Divide and Conquer

The decomposition of objects to processors must ensure that, as far as possible, each node is performing the same amount of work. The workload division may be implemented in either of two ways:

a. All Functions to All Processors (AFAP)

In this option the ***Application Hierarchy*** is replicated throughout the network. Classes loaded on different processors are considered different classes despite having the same properties. Work division is obtained by balancing the number of objects among processors. The main issue in this approach is how to support communicating objects while keeping a low communication overhead.

b. *Some Functions to Some Processors (SFSP)*

In this option ***Function Hierarchies*** are assigned to different processors of the network. Classes being used for inheritance should be replicated. Work division is designed to reduce the communication overhead. Communicating objects are assigned, whenever possible, to neighboring processors. The main issue in this approach is how to overlap the execution of methods to improve performance.

5. Identification of Options

So far, we have identified application and function hierarchies. The application hierarchy is a natural candidate for the AFAP division of work since the application hierarchy encapsulates all identified application functions in classes replicated throughout the network, while function hierarchies are natural candidates for the SFSP division of work since function hierarchies are partitions of the application hierarchy encapsulating some application functions in classes assigned to partitions of the network of processors. The decomposition of objects to processors is a domain decomposition problem from the interconnection network of objects to the interconnection network of processors.

A topology is usually characterized by its diameter, degree of each node, connectivity, and average distance. The diameter is the maximum distance that a message must travel from one node to another. The degree of a node is the number of ports provided for a processor to connect with other processors. The connectivity provides a measure of the number of 'independent' paths connecting a pair of nodes. The average distance is the distance that messages must travel, on average, in the network. An ideal interconnection network of processors is thus a network with a short diameter, small degree, high connectivity, and a short average distance.

In this step we try to identify options to map the interconnection network of objects to the interconnection network of processors.

6. Conceptual Comparisons Among Options

For each identified option a qualitative approach is used to list the expected strengths and weaknesses of the proposed solutions. Any 'well accepted' concept of the architecture community may be used in this analysis.

7. Evaluation of the Communication Cost

When any two communicating objects are loaded on different processors, the communication cost can be divided into three components:

*1. **The service request cost** (SRC):* Computed as the communication cost needed to send T bytes in the sender object or to receive T bytes in the receiver object;

*2. **The result cost** (RC):* Computed as the communication cost needed to send R bytes in the receiver object or to receive R bytes in the sender object; and

*3. **The retransmit cost** (C_{RT}):* Computed as the communication cost needed to retransmit T bytes or to retransmit R bytes.

Components (1) and (2) represent communication overhead in the processors where the communicating objects are loaded, while component (3) represents communication overhead in processors used to route the request and return results. In general, communication cost includes queuing time, reception and/or transmission time, and propagation time.

For each identified option the processor communication cost may be evaluated using the following sequence:

a. The Object Communication Cost (OXC)

The communication cost function of object j (either sender or receiver) loaded on processor ***P*** can be evaluated by:

$$OXC_j = \sum_{k=1}^{N_{co}} c_{jk} \tag{5.1}$$

where:

N_{co} is the number of communicating objects not loaded on the processor ***P***; and

c_{jk} is the communication cost of object j with object k, which are loaded on different processors (note: this communication cost includes both the service request and result costs).

The object cost function is defined during some application dependent time interval. The cost function of objects that do not exist during the entire interval are evaluated during the intersection of their existence with the selected interval.

b. The Class Communication Cost (CXC)

The communication cost function of a class i can be evaluated by:

$$CXC_i = \sum_{j=1}^{N_i} OXC_j \tag{5.2}$$

where:

N_i is the number of instances of class i;

OXC_j is the communication cost function of object j.

The cost function of abstract classes is assumed to be zero as these classes have no instances.

c. The Hierarchy Communication Cost (HXC)

The communication cost function of some hierarchy ***h*** can be evaluated by:

$$HXC_h = \sum_{i=1}^{N_c} CXC_i \tag{5.3}$$

where:

N_c is the number of classes in hierarchy h; and

CXC_i is the communication cost function of class i.

d. The Processor Communication Cost (PXC)

The processor communication cost function can be evaluated by:

$$PXC = \left(\sum_{h=1}^{N_h} HXC_h \right) + C_{RT} \tag{5.4}$$

where:

N_h is the number of hierarchies to be loaded on processor P;

HXC_h is the communication cost function of hierarchy h; and

C_{RT} is the retransmit cost of processor P.

8. Evaluation of the Computation Cost

For each identified option, the processor computation cost may be evaluated using the following sequence:

a. The Method Computation Cost (MCC)

The method cost function (***$MCC_{P,k}$***) is defined as the computation time of some method ***k*** in processor ***P***. It can be manually estimated using the processor instruction performance information or by counting the number of processor ticks needed to execute the method.

b. The Object Computation Cost (OCC)

The computation cost function of any object j of some class i can be evaluated by:

$$OCC_{i,j} = \sum_{k=1}^{N_m} N_k * MCC_{P,k} \qquad (5.5)$$

where:

K is the method number index;

j is the object number index;

i is the class number index;

N_m is the number of visible methods for object j;

$MCC_{P,k}$ is the cost function of the method k when executed in processor P; and

N_k is the number of messages sent to object j to execute the method k.

The object cost function is defined during some application dependent time interval. The cost function of objects that do not exist during the entire interval are evaluated during the intersection of their existence with the selected interval.

c. The Class Computation Cost (CCC)

The computation cost function of a class i can be evaluated by:

$$CCC_i = \sum_{j=1}^{N_i} OCC_{i,j} \qquad (5.6)$$

where:

N_i is the number of instances of class i; and

$OCC_{i,j}$ is the computation cost function of the object j of the class i.

Once again, the cost function of abstract classes is assumed to be zero as these classes have no instances.

d. The Hierarchy Computation Cost (HCC)

The computation cost function of some hierarchy ***h*** can be evaluated by:

$$HCC_h = \sum_{i=1}^{N_c} CCC_i \tag{5.7}$$

where:

h is the hierarchy number index;

N_c is the number of classes in hierarchy h; and

CCC_i is the computation cost function of class i.

e. The Processor Computation Cost (PCC)

The processor computation cost function can be evaluated by:

$$PCC = \sum_{h=1}^{N_h} HCC_h \tag{5.8}$$

where:

N_h is the number of hierarchies to be loaded on processor P; and

HCC_h is the computation cost function of the hierarchy h.

9. Analytical Comparisons Among Options

The expected efficiency of the identified options is compared after an evaluation of $T_{conc}(N)$ for each proposal.

C. APPLYING THE DECOMPOSITION COST EVALUATION MODEL

In this section we summarize the results obtained when we apply the Decomposition Cost Evaluation Model to our application. The analysis developed to support the results is detailed in Appendix B.

Two options are identified: the hypercube topology (d-cube) to implement the all functions to all processors design, and the tree topology $(1-C_p-T_p)$-tree to implement the some functions to some processors design.

The tree topology is built with an interface processor as the root node, C_p correlation processors at level 1, and T_p tracking processors (for each correlation processor) at level 2.

The single processor class hierarchy designed in Chapter IV is decomposed into an interface hierarchy, and an application hierarchy. The application hierarchy is decomposed into two function hierarchies: the correlation hierarchy and the tracking hierarchy.

In the hypercube option we load the interface hierarchy on the interface processor (IP), and replicate the application hierarchy on the remaining processors.

In the tree option we load the interface hierarchy on the interface processor (IP), the correlation hierarchy on correlation processors (CPs), and the tracking hierarchy on tracking processors (TPs).

Figure 14 depicts a comparative analysis of the expected efficiency of the identified options. *r* is the ratio between the average correlation time per target and the average tracking time per target, and Δ_{com} is the hypercube communication cost minus the tree communication cost.

Table 25 (Appendix B) depicts the communication cost of the identified options. To support a continuous flow of (plot,target) pairs from correlation to tracking processors, and to avoid idle time on tracking processors we must have r less than $1/T_p$. The main conclusion can be expressed as:

The tree proposal should be more efficient than the hypercube proposal when we overlap correlation and tracking algorithms, and reduce the communication cost by avoiding the broadcast of all measures to all processors.

Figure 15 depicts a conceptual view of the selected option. Plots coming from the radar subsystem are routed to correlation processors, assignment pairs (plot,target) are routed to tracking processors, predictions are routed back to correlation processors, and estimations are routed to the display subsystem.

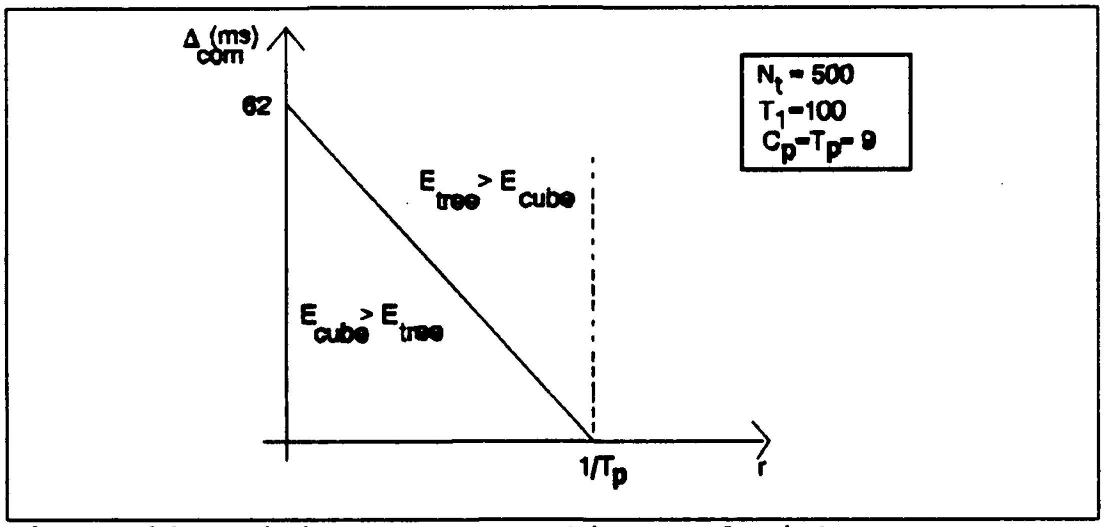


Figure 14 Efficiency (Comparative Analysis)

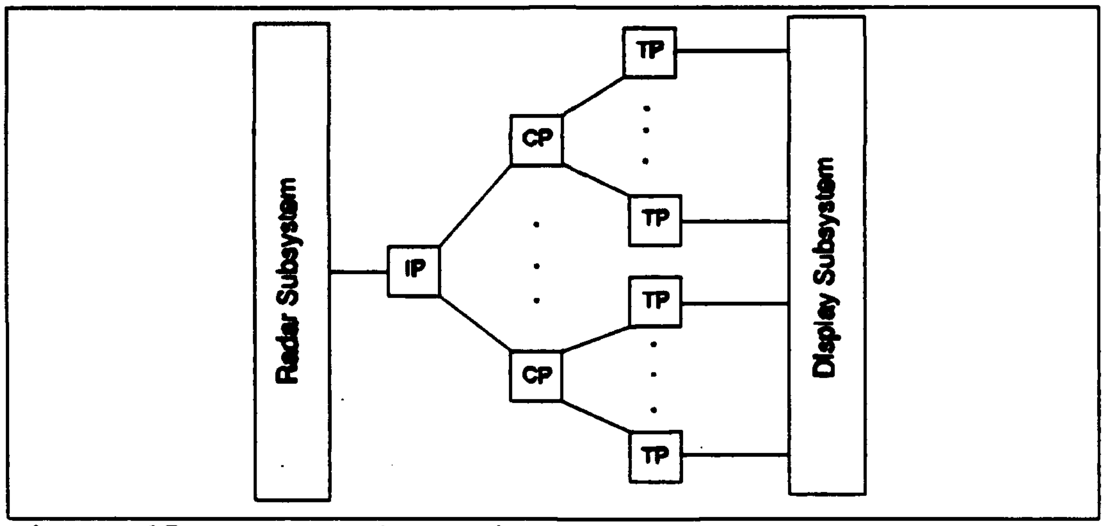


Figure 15 Conceptual Architecture

D. THE CONFINED SPACE SEARCH DECOMPOSITION

During the application of the decomposition cost evaluation model we concluded that the division of the search space into sections reduces the communication cost of the tree option. In our proposal, called Confined Space Search Decomposition (CSSD), the search space is divided into fixed size sections and each CP is executing the target to plot correlation with all plots detected within some assigned number of successive sections in the tactical scene. An overlap space between CPs is defined to support a smooth target transition when the target crosses the visibility space boundary of some CP. At the interface processor we split the set of all detected plots and route plots subsets knowing which sections are assigned to which CPs. When the target is located within the overlap space of CPs, its estimation is computed by more than one tracking processor. In this case, equivalent targets are merged.

Figure 16 depicts the transition of a generic target through the overlap space between CPs. During its course the target starts being tracked by processor I (point A), crosses the overlap space being tracked by processors I and II (point B), and ends its path being tracked by processor II (point C).

Targets crossing the overlap space are terminated in the old processor and initiated in the new processor. This happens because in the old CP no reported plot will be associated with an existing firm target and in the new CP a new reported plot without association with existing targets will become a tentative target. When the target modifies its status to terminating in the old processor it has already been confirmed as firm in the new processor.

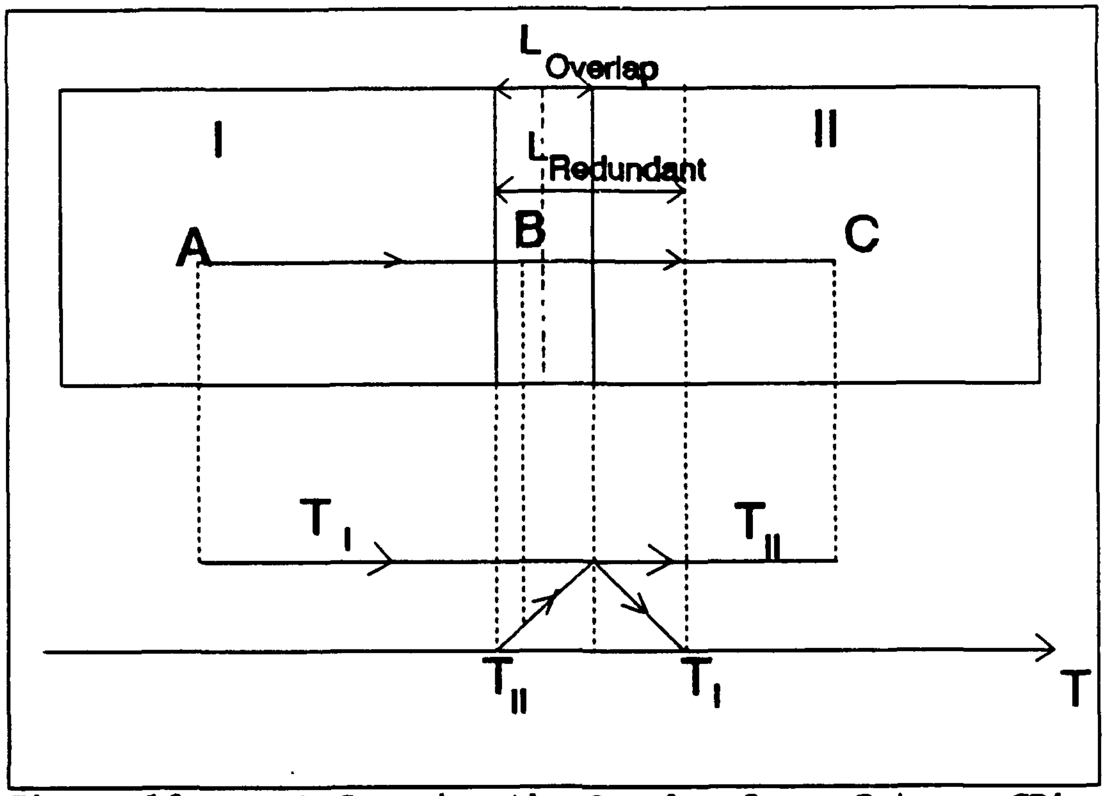


Figure 16 Target Crossing the Overlap Space Between CP's

1. Section Specification

Now, we face the problem associated with the 'best performance' division of the search space into fixed size sections. Ideally, the number of visible targets for any correlation processor should be the same to avoid global load imbalance. However, radar data processing is a non-uniform problem so, the 'best performance' division of the search space is a function of the expected distribution of targets within the surveillance environment (Environment Model). During our discussions, we will assume an environment model with uniform angular distribution in $[0,2\pi]$ and non-uniform distance distribution in [0,200] NM (*AUDN*). This assumption leads to a division of the search space into angular sections (see Figure 17). Targets detected within the inner circle, viewed as a danger area, are broadcast to all correlation processors. That is, redundancy is used as a safety procedure for targets detected in close range of the radar site.

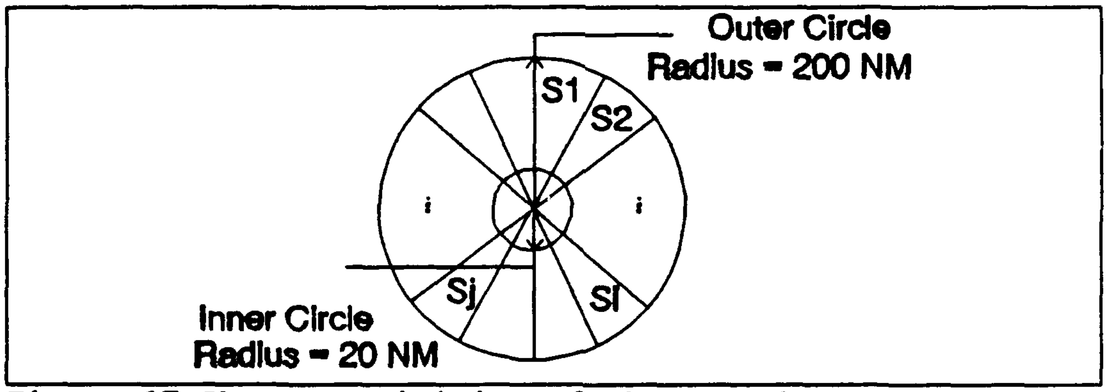


Figure 17 The AUDN Division of the Search Space

When the real target distribution does not match the expected target distribution, load imbalance can occur. In the next section, we propose a distributed dynamic load balancing heuristic, called ***Object Reincarnation*** (OR), to be used in RDP applications. In the worst case, when all detected targets are located within a single section then a single branch of the $(1\text{-}C_p\text{-}T_p)$-tree will be responsible to execute the RDP functions for all detected targets. In this situation, a degradation in the system reaction time is expected. That is, for a particular application, the computational power of any branch must be specified to support the worst acceptable system reaction time for the maximum number of expected targets.

However, this does not mean that the architecture design should be conceived to favor the infrequent case [Ref. 33].

Sections are used as the overlap unit between CPs, thus its size must be compatible with:

1. The worst time required to terminate an old target and to initiate a new one; and

2. The velocity of the fastest target (assumed as 700 m/s).

The initiation and termination algorithms requires at most five samples to initiate a new target or to terminate an old target (see Figures 9 and 10), therefore:

$$T_{i,t} = 3s/sample * 5samples = 15s$$

$$\Delta\theta \geq \frac{V_t * T_{i,t}}{R} = \frac{700*15*180}{20*1852*3.141516} = 16^{\circ}.2$$

where:

R is the radius of the inner circle;

V_t is the tangential velocity of the fastest target; and

$T_{i,t}$ is the worst time required during the initiation and termination algorithms.

As a consequence of the previous analysis, we decided to divide the search space into 18 fixed size sections of 20 degrees in each section. In general, sections can be specified as:

$S_k = \sphericalangle\ ((k - 1)*20^\circ, K*20^\circ),\ k = 1...18$ **(5.9)**

2. Mapping Space Regions to Processors

During the execution of correlation algorithms all CPs must receive the report of plots detected within some assigned number of successive sections and an overlap section should be defined between the visibility space of some pair of CPs. To support those requirements, we defined the ***Space Region*** as a triple of space sections, therefore given:

$$R_{i/2} = (S_{i-1}, S_i, S_{(i+1)\ MOD\ 18})$$

$$R_{j/2} = (S_{j-1}, S_j, S_{(j+1)\ MOD\ 18})$$

we have:

$$R_{i/2} \cap R_{j/2} = S_{j-1}\ \textit{if}\ j = (i + 2)\ MOD^*\ 18$$

$$R_{i/2} \cap R_{j/2} = S_{i-1}\ \textit{if}\ i = (j + 2)\ MOD^*\ 18 \qquad \textbf{(5.10)}$$

$$R_{i/2} \cap R_{j/2} = \Phi\ ,\ \textit{otherwise}$$

where:

i,j = 2,4,6,8,10,12,14,16,and 18; and

N mod* N = N.

The initial distribution of regions among processors is defined as:

1. #R is the number of regions; and

2. #P is the number of processors.

where #R ≥ #P.

If:

1. #R DIV #P = M; and

2. #R MOD #P = N.

then:

1. When N=0, $P_1 \dots P_{\#P}$ receive M sequential regions;

2. Otherwise, $P_1 \dots P_N$ receive (M+1) sequential regions and $P_{N+1} \dots P_{\#P}$ receive M sequential regions.

3. Results Summary

The 'best performance' division of the search space is a function of the environment model. The ***AUDN*** assumption leads to a division of the search space into angular sections. If we had assumed non-uniform angular distribution in $[0,2\pi]$ and uniform distance distribution in [0,200] NM (***ANDU***), then we would divide the search space into ring sections. In this situation, we would have 20 fixed size sections of 10 NM in each section; however, the definition of space regions and their mapping to processors would follow an equivalent procedure.

E. OBJECT REINCARNATION

Dynamic load balancing schemes are needed to efficiently solve non-uniform problems on distributed computational systems. Ideally, during the execution of any load balancing strategy no communication costs should be incurred to transfer load among processors. In the (1-C_p-T_p)-tree (SFSP) option, load balance may be needed either because detections are not evenly distributed among CPs (***Global Load Balance***) or because existing tracks are not evenly distributed among TPs (***Local Load Balance***).

In our proposal, called ***Object Reincarnation***, we see objects as computation sinks. Load balance is obtained when objects die in one processor site (reducing its load) and are reincarnated in another site (increasing its load). The correlation hierarchy is replicated throughout CPs and the tracking hierarchy is replicated throughout TPs. Therefore, object migration is not needed as the internal state of dying objects is re-created in another site. The issue is: Does the application support the re-creation of objects in another site with 'acceptable' tracking penalty? The procedures used to motivate the reincarnation can be described as:

1. ***Global Load Balance***: The IP adjusts the visibility space of CPs. Firm and tentative objects must see correlated detections to remain alive (see Chapter III). When the IP reduces the visibility space of an overloaded CP some existing firm objects can start to loose association with reported detections and as a consequence will die reducing the load of the overloaded CP. On the other hand, when the IP increases the visibility space of an underloaded CP some new tentative objects will reincarnate dying objects increasing the load of

the underloaded CP. Since the number of samples needed in the initiation and termination algorithms is the same then the reincarnation will happen without loss of tracking.

2. ***Local Load Balance***: CPs cancel some input tracking reports of overloaded TPs and route those reports to underloaded TPs. Existing tracks loosing tracking reports will die, reducing the load of overloaded TPs. Similarly, new tracking reports will reincarnate dying objects, increasing the load of underloaded TPs.

1. Global Load Balance Algorithm

The algorithm consists of the following steps to be executed by the IP during each data sample.

a. *Evaluate the Global Average Load (GL_{AVG})*

The global average load is the average number of targets controlled by CPs.

$$GL_{AVG} = \frac{\sum_{i=1}^{C_p} GL_i}{C_p} \tag{5.11}$$

where:

GL_i is the number of estimations reported by correlation processor i with distance from the radar site greater than the radius of the inner circle (10 NM); and

C_p is the number of CPs.

b. Compute the Global Load Imbalance Factor (GLI_i)

The global load imbalance factor of correlation processor i is used as a heuristic measure of the extra work being executed in each branch of the $(1-C_p-T_p)$-tree.

$$GLI_i = \frac{GL_i - GL_{AVG}}{GL_{AVG}} \tag{5.12}$$

c. Adjust the Visibility Space of CPs

The visibility space of CPs is adjusted to motivate object reincarnation as follows:

IF $[(GLI_i \geq 0.75)$ AND $((GL_i - GL_{AVG}) > (0.1N_t/C_p))]$, then the IP removes at most one region from the CP_i and transfers this region to the CP_{i+1};

otherwise, no region is removed from the CP_i;

where:

N_t is the maximum number of expected targets in the search space.

d. Algorithm Remarks

The following remarks apply to the algorithm execution:

1. Correlation processors are numbered as specified in the initial distribution of regions among processors (see Section D2);

2. Each CP must control at least one region;

3. Regions are numbered (see Equations 5.9 and 5.10);

4. Regions to be removed from any CP are the ones with higher indices (i.e., if the CP_1 controls regions R_1,R_2,R_3, and due to load imbalance two regions are going to be removed from the CP_1 by the IP, then the selected regions to be removed are the regions R_2 and R_3); and

5. The proposed heuristic to support the decision: 'Should we adjust the visibility space?' is a logical AND operation between an absolute and a relative criterion. The absolute criterion intends to avoid execution of load balance when the load deviation from the average (GL_i-GL_{AVG}) is small (less than 10%) compared with the expected load in each branch. The relative criterion is a tradeoff between performance and filtering degradation (we will return to this point in Chapter VI). Since each branch must be specified to support the worst acceptable system reaction time for the maximum number of expected targets we decided to favor the filter improvement using a high index value (0.75).

2. Local Load Balance Algorithm

The algorithm consists of the following steps to be executed by each CP during each data sample.

a. *Evaluate the Local Average Load (LL_{AVG})*

The local average load is the average number of targets controlled by TPs.

$$LL_{AVG} = \frac{\sum_{j=1}^{T_p} LL_j}{T_p} \qquad (5.13)$$

where:

LL_j is the number of estimations reported by the tracking processor j; and

T_p is the number of TPs.

b. Compute the Local Load Imbalance Factor (LLI_j)

The local load imbalance factor of the tracking processor j is used as a heuristic measure of its extra work.

$$LLI_j = \frac{LL_j - LL_{AVG}}{LL_{AVG}} \qquad (5.14)$$

c. Adjust Tracking Reports

Tracking reports sent to TPs are canceled to motivate object reincarnation when:

IF $[(LLI_j \geq 0.25)$ AND $((LL_j - LL_{AVG}) > (0.1N_t/(C_pT_p)))]$, then the CP cancels $\lfloor (LLI_j * LL_{AVG}) \rfloor$ (i.e., the surplus load) tracking reports from the TP_j and redistributes these reports uniformly among the tracking processor siblings;

otherwise, no tracking report of the TP_j is canceled.

*d. **Algorithm Remarks***

The following remarks apply to the algorithm execution:

1. In each branch of the $(1-C_p-T_p)$-tree, tracking processors are numbered from left to right;

2. Tracking reports to be canceled from overloaded TPs are selected arbitrarily by the CP;

3. CPs use a circular allocation policy when new firm targets are assigned to TPs; and

4. The proposed heuristic to support the decision: 'Should we cancel tracking reports?' is a logical AND operation between an absolute and a relative criterion. The absolute criterion intends to avoid execution of load balance when the load deviation from the average (LL_i-LL_{AVG}) is small (less than 10%) compared with the expected load in each TP. The relative criterion is a tradeoff between performance and filtering degradation (we will return to this point in Chapter VI). Since TPs represent a large percentage of the computational power we decided to favor performance using a small index value (0.25). Using a circular allocation policy for each new firm target then local load imbalance is expected only as a consequence of arbitrary contact losses.

3. Results Summary

The object reincarnation proposal is a distributed load balancing strategy without any additional communication cost either because as a natural consequence of its functions the IP must know how many estimations are being reported by each CP or because as a natural consequence of its function CPs must know how many output tracking updates are being reported by each TP.

F. FAULT RECOVERY

During the last five years, the problem of routing messages on hypercubes with faulty components has motivated an intense research effort, resulting in several proposals [Ref. 34,35, and 36] being presented.

During the conceptual comparisons between the d-cube (AFAP) and (1-C_p-T_p)-tree (SFSP) options (see Appendix B), we discussed the expected consequences for the application when the (1-C_p-T_p)-tree operates with faulty components (nodes or links). The (1-C_p-T_p)-tree offers low connectivity. That is, failure of any of its links creates two subsets of processors that cannot communicate with each other. Any link failure isolates one parent node (IP or CP) from its child node (CP or TP), so link or processor failures requires load transfer from some child node to its siblings.

1. **Fault Recovery Algorithm: Isolated CP**

This algorithm is executed by the IP during each data sample and consists of the following steps:

1. Send a check message to all existing CPs;

2. Mark CPs unable to answer as dead; and

3. Transfer load of any CP marked as dead to the next alive CP.

All detections lying within regions previously assigned to some dead CP will be transferred to the next CP (remember, CPs are numbered) recognized as alive.

2. **Fault Recovery Algorithm: Isolated TP**

This algorithm is executed by each CP during each data sample and consists of the following steps:

1. Send a check message to all existing child TPs;

2. Mark TPs unable to answer as dead; and

3. Redistribute load of any TP marked as dead uniformly among live siblings.

All tracking reports previously assigned to some dead TP will be uniformly distributed among alive siblings.

3. **Results Summary**

The fault recovery procedures in $(1-C_p-T_p)$-trees can be viewed as an extension to the distributed load balancing strategy presented in the previous section (Section D). Object reincarnation is used either to transfer load from dead CPs to their siblings (Global Load Balance) or to transfer load from dead TPs to their siblings (Local Load Balance). Dynamic adjustment of the routing software is not required and each node needs to know only the status (dead or alive) of its own links.

VI. ARCHITECTURE VALIDATION

In this chapter we validate the architecture specified in Chapter V. To keep our validation independent of a particular processor we develop mathematical expressions to evaluate performance, to compute the expected tracking capacity, to estimate the system reaction time, and to check the tracking filter capability to reduce the measurement errors when targets cross the space search boundary of CPs. The developed expressions are used in the computation of upper and lower limit values, and the results are analyzed.

A. INTRODUCTION

During our reasoning about how we should validate our proposal we asked ourselves:

1. Which parameters should we select to validate our specified architecture?

2. Are the selected parameters sufficient?

The answers to these questions should be based on the primary motivation for our work, that is:

In military applications, the lower is the processing time, the higher is the time available for human decisions, the lower is the system reaction time to existing threats, the higher is the system capability to shield a task force against an increasing number of threats.

The main goal of any distributed computational system is to improve performance. Performance is a standard metric accepted by the architecture community to check how well a network of processors is being used to accomplish a job when compared with a single processor. However, we found that performance alone is not enough to answer all desirable questions. Looking to the motivation of our research the following topics can be emphasized:

1. ***Tracking Capacity***: maximum number of targets that our system is able to track with real-time response; and

2. ***System Reaction Time***: average time lag between target detection and target estimation report of all targets being tracked.

These two points suggest that we should define metrics to evaluate the expected tracking capacity and system reaction time.

In Chapter IV, we introduced the following questions:

1. Does the application support a division of the search space in correlation sections?

2. Does the application support the re-creation of objects in another site with 'acceptable' tracking penalty?

To answer these questions we must implement a simulation to check the tracking filter capability to reduce the measurement errors when targets cross the space search boundary of CPs.

B. PERFORMANCE EVALUATION

An important measure of the performance of a concurrent computer is the ***speedup*** factor S associated with a particular calculation. The speedup is defined as the ratio of the time required to complete a given calculation on a single-node processor to the equivalent calculation performed on a concurrent processor with N nodes [Ref. 28]. It follows that the speedup S depends upon N (the number of nodes), and is given by:

$$S(N) = \frac{T_{seq}}{T_{conc}(N)} \qquad (6.1)$$

where:

T_{seq} is the time to compute on one node; and

$T_{conc}(N)$ is the time to compute on N nodes.

As we will find, it is sometimes useful to introduce the concurrent ***efficiency*** factor E, defined by:

$$E = \frac{S(N)}{N} \qquad (6.2)$$

Inefficiency in the system is introduced by:

1. Additional control and communication involved in distributing the problem over the N processing nodes; and

2. Load Balancing: the speedup is generally limited by the speed of the slowest node.

1. Identifying Sources of Inefficiency

a. Additional Control

In Chapter IV we introduced our single processor design. The ***controller*** of all identified functions was called the ***scheduler*** object. During the contract specification we identified the responsibilities of the scheduler object (see Section C4).

In Chapter V we introduced our $(1\text{-}C_p\text{-}T_p)$-tree proposal where interface functions are loaded on interface processors, correlation functions are loaded on correlation processors, and tracking functions are loaded on tracking processors. In Appendix B new controllers are identified (interface_scheduler, correlation_scheduler, and tracking_scheduler). The existence of new controllers requires reallocation of responsibilities and definition of new responsibilities. Reallocation of responsibilities is not expected to add to the overhead. However, definition of new responsibilities is a source of inefficiency.

b. Additional Communication

In Appendix B we evaluate the communication cost of the IP(IPXC), CPs(CPXC), and TPs(TPXC). Any time spent in communication constitutes a penalty on the overall performance as compared with the sequential case. In some processors, communication can be overlapped with computation to decrease its influence as a factor of inefficiency.

c. *Load Balancing*

In a distributed computational system we must ensure that, as far as possible, each node is performing the same amount of work.

In Chapter V we introduced the object reincarnation proposal to improve load balance using a distributed strategy without any additional communication cost. This proposal covers load imbalance among CPs and load imbalance among TPs. It does not however, improve load balance between the CP and TPs of each branch. To attack this problem we overlap the execution of correlation and tracking algorithms.

2. Upper Limit Efficiency

In the $(1-C_p-T_p)$-tree proposal the upper limit efficiency is achieved when global and local load balance operations are not required. This is because the workload in each branch is the same, and firm targets are evenly distributed among TPs.

Applying Equations 6.1 and 6.2, we have:

a. *Time to Compute on One Node (T_{seq})*

$$T_{seq} = T_{if} + T_{cor} + T_{tr}$$

where:

T_{if} is the time expended in interface functions;

T_{cor} is the time expended in correlation functions;

and

T_{tr} is the time expended in tracking functions.

Assuming:

1. $(T_{cor}+T_{tr}) >> T_{if}$ (i.e., correlation and tracking are the dominating costs); and

2. T_{cor} and T_{tr} are proportional to the number of targets (N_t) (i.e., in steady state correlation and tracking is executed for each firm target).

then:

$$T_{cor} = C_1 N_t$$

$$T_{tr} = T_1 N_t \tag{6.3}$$

$$T_{seq} = (C_1 + T_1) N_t$$

where:

C_1 is the average correlation time per target; and

T_1 is the average tracking time per target.

b. Time to Compute on N Nodes ($T_{comc}(N)$)

As introduced in this section, we identified additional control, additional communication and load balancing as our sources of inefficiency. In the absence of global and local load imbalance we will assume that the load imbalance between the CP and TPs of each branch is our dominating factor of inefficiency (later we will verify this assumption), therefore $T_{conc}(N)$ can be evaluated as:

$$T_{conc}(N) = fC_1T_p + MAX(T_1(W_{tp}-1), C_1W_{cp}-fC_1T_p) + T_1$$

where:

W_{cp} is the workload of correlation processors (i.e., the number of targets loaded on each CP);

W_{tp} is the workload of tracking processors (i.e., the number of targets loaded on each TP); and

f is a fraction of the average correlation time per target expended during the first gate correlation phase (i.e., fC_1 is the average first gate correlation time per target).

When $r < (1/T_p)$ the expression to evaluate $T_{conc}(N)$ can be rewritten as:

$$T_{conc}(N) = fC_1T_p + T_1W_{tp} \tag{6.4}$$

The correlation operation $(C_1W_{cp}-fC_1T_p)$ is executed in CPs in parallel with the tracking operation $(T_1(W_{tp}-1))$ in TPs.

Using the assumption of global and local load imbalance we have:

$$W_{cp} = \frac{N_t}{C_p};$$

$$W_{tp} = \frac{N_t}{C_p T_p}.$$

If $(T_1 W_{tp} >> f C_1 T_p)$ and $(T_1 W_{tp} > C_1 W_{cp})$, that is:

$$W_{tp} \gg f r T_p; \ (f<1)$$

(6.5)

$$\frac{T_1}{T_p} > C_1$$

then $T_{conc}(N)$ can be evaluated by (see Equation 6.4):

$$T_{conc}(N) = f C_1 T_p + T_1 W_{tp} \approx \frac{T_1 N_t}{T_p C_p} \tag{6.6}$$

c. *Efficiency Evaluation*

The number of nodes (N) in the $(1-C_p-T_p)$-tree proposal can be evaluated by:

$$N = C_p T_p + C_p + 1$$

Assuming $(C_pT_p+C_p) \gg 1$, then:

$$N = C_p(T_p+1) \tag{6.7}$$

Using equations 6.2 - 6.7, we have:

$$E = \frac{(C_1+T_1)N_t}{(fC_1T_p+T_1W_{tp})C_p(T_p+1)} \tag{6.8}$$

Load imbalance between the CP and TPs of each branch is removed when:

1. $W_{cp}C_1 = W_{tp}T_1$ (i.e., $C_1 = T_1/T_p$);

2. $f=0$ (i.e., the cost of the first gate correlation phase is null).

In this situation, we have:

$$E = \frac{\frac{T_1}{T_p}+T_1}{\frac{T_1}{T_p}(T_p+1)} = 1$$

This is an expected result because our analysis is being done considering the load imbalance (CP,TP) as the single source of inefficiency. That is, when we remove all sources of inefficiency, then E=1.

Assuming:

1. Tracking as the dominating cost between tracking and correlation operations (i.e., $C_1=rT_1$, $r < (1/T_p)$); and

2. $W_{tp}T_1 >> fC_1T_p$ (i.e, the tracking cost of all targets loaded on TPs is very high when compared with the first gate correlation cost of T_p targets).

then we can rewrite Equation 6.8 as:

$$E = \frac{(1+r)T_1}{\frac{T_1}{T_p}(T_p+1)} = \frac{(1+r)T_p}{T_p+1} \tag{6.9}$$

Table 5 depicts the upper limit efficiency of the $(1-C_p-T_p)$-tree proposal $(r=(0.9/T_p))$.

TABLE 5 UPPER LIMIT EFFICIENCY

Topology	Efficiency
(1-3-3)-tree	97%
(1-9-9)-tree	99%

3. Lower Limit Efficiency

In the $(1-C_p-T_p)$-tree proposal the lower limit efficiency is obtained when:

1. All detected targets are located within the visibility space of a single tree branch; and

2. Local load balance operation is required in the overloaded branch.

Again, applying Equations 6.1 and 6.2 we have:

a. ***Time to Compute on One Node*** ***(T_{seq})***

The time to compute on one node was evaluated in Equation 6.3.

b. ***Time to Compute on N Nodes*** ***($T_{conc}(N)$)***

With all detections lying within a single branch we have:

1. $W_{op} = N_t$; and

2. $W_{tp} = 1.25(N_t/T_p)$. That is, to start the execution of the local load imbalance algorithm specified in Chapter V, Section D we accept a maximum local load imbalance factor (LLI=0.25).

When $T_1W_{tp} > C_1W_{op}$, that is:

$$1.25T_1\frac{N_t}{T_p} > C_1N_t$$

(6.10)

$$\frac{T_1}{T_p} > \frac{C_1}{1.25}$$

then $T_{conc}(N)$ can be evaluated by (see Equation 6.4):

$$T_{conc}(N) = fC_1T_p + \frac{1.25N_t}{T_p}T_1$$

Again, assuming $T_1 W_{tp} >> fC_1 T_p$ we have:

$$T_{conc}(N) = 1.25 \frac{T_1}{T_p} N_t \tag{6.11}$$

c. *Efficiency Evaluation*

Using Equations 6.2, 6.3, 6.7, and 6.11 we have:

$$E = \frac{(1+r)\, T_p}{1.25 C_p (T_p+1)}$$

(6.12)

$$E = \frac{1}{1.25 C_p} E_{max}$$

where:

E_{max} is the upper limit efficiency (see Equation 6.9).

Relative criterions must be taken with care, since for instance Equation 6.12 indicates that when we increase C_p we decrease the lower limit efficiency. However, it does not show that when we increase C_p the probability to detect all targets within the visibility space of a single tree branch is expected to decrease because increasing C_p decreases the number of space regions being controlled by each CP (see Chapter V).

Table 6 depicts the lower limit efficiency of the $(1-C_p-T_p)$-tree proposal.

TABLE 6 LOWER LIMIT EFFICIENCY

Topology	Efficiency
(1-3-3)-tree	25%
(1-9-9)-tree	9%

4. Efficiency with Load Imbalance

After the evaluation of the best and worst case conditions we need to predict the expected efficiency with load imbalance operation. This prediction is made with the following assumptions:

1. To start the execution of the local load imbalance algorithm as specified in Chapter V, Section D we accept a maximum local load imbalance factor (LLI=0.25);

2. To start the execution of the global load imbalance algorithm as specified in Chapter V, Section D we accept a maximum global load imbalance factor (GLI=0.75); and

3. Maximum global and local load imbalance are present in the same branch of the $(1-C_p-T_p)$-tree.

Again, applying Equations 6.1 and 6.2 we have:

a. *Time to Compute on One Node (T_{seq})*

Equation 6.3 can be rewritten as:

$$T_{seq} = (1+r)T_1 N_t \quad (r<1) \tag{6.13}$$

In this equation $C_1 = rT_1$. That is, r is the ratio between the average correlation time per target (C_1) and the average tracking time per target (T_1).

b. *Time to compute on N Nodes ($T_{conc}(N)$)*

With global and local load imbalance we have:

$$W_{cp} = (1.75)\frac{N_t}{C_p};$$

$$W_{tp} = (1.75)(1.25)\frac{N_t}{C_p T_p}.$$

When $T_1 W_{tp} > C_1 W_{cp}$, that is:

$$(1.75)(1.25)T_1\frac{N_t}{C_p T_p} > (1.75)C_1\frac{N_t}{C_p}$$

$$\frac{T_1}{T_p} > 0.8C_1 \tag{6.14}$$

Then $T_{conc}(N)$ can be evaluated by (see Equation 6.4):

$$T_{conc}(N) = fC_1T_p+(1.75)(1.25)T_1\frac{N_t}{C_pT_p}$$

$$T_{conc}(N) = (frT_p+(1.75)(1.25)\frac{N_t}{C_pT_p})T_1 \tag{6.15}$$

c. Efficiency Evaluation

Using Equations 6.2, 6.7, 6.13, and 6.15, we have:

$$E = \frac{(1+r)T_pN_t}{(frT_p+(1.75)(1.25)\frac{N_t}{C_pT_p})T_1C_p(T_p+1)} \tag{6.16}$$

The strongest tracking dominating condition (see Equations 6.5, 6.10, and 6.14) can be expressed as:

$$\frac{T_1}{T_p} > rT_1$$

$$r \leq \frac{1}{T_p}$$

If $(T_p \leq 9)$ then $r \leq 0.11$.

Table 7 depicts the relationship among the average tracking time per target (T_1), average correlation time per target (C_1), and the average first gate correlation time per target (fC_1) considering an average tracking time per target of hundreds of milliseconds (see Table 2), and the average first gate correlation time per target a tenth part of the average correlation time per target($fC_1=0.1C_1$).

TABLE 7 CORRELATION AND TRACKING COSTS (MS)

T_1	$(C_1)_{max}$	fC_1
100	11	1.1
250	27	2.7
500	55	5.5

Again, assuming $T_1W_{tp} >> fC_1T_p$ we have:

$$E = \frac{(1+r)T_p}{(1.75)(1.25)(T_p+1)}$$

$$E = \frac{1}{(1.75)(1.25)}E_{max} \qquad (6.17)$$

Where:

E_{max} is the upper limit efficiency (see Equation 6.9).

Load imbalance is a major source of inefficiency in a distributed computational system [Ref. 28]. In our proposal, global load imbalance happens when the real target distribution does not match with the environment model (AUDN) (see Chapter V), and local load imbalance is expected only as a consequence of arbitrary contact losses (see Chapter V). The object reincarnation proposal reduces the effects of load imbalance with no additional communication cost.

Table 8 depicts the expected values of the efficiency with load imbalance in the $(1-C_p-T_p)$-tree proposal.

TABLE 8 EFFICIENCY WITH LOAD IMBALANCE

Topology	Efficiency
(1-3-3)-tree	44%
(1-9-9)-tree	45%

5. Verifying the Inefficiency Assumption

During the evaluation of the upper limit efficiency we assumed that the load imbalance between the CP and TPs of each branch was our dominating factor of inefficiency in the absence of global and local load imbalance. We then analyzed the consequences of global and local load imbalance. All of these surveys were based on the assumption that the inefficiency introduced by additional control and additional communication are small when compared with the inefficiency introduced by load imbalance. We now verify this assumption.

In the worst case, the inefficiency introduced by additional control and additional communication is not overlapped with the execution of correlation and tracking algorithms. In this hypothesis, $T_{conc}(N)$ can be rewritten as:

$$T_{conc}(N) = R_1 N_t + X_1 N_t + f C_1 T_p + T_1 W_{tp}$$

where:

R_1 is the average additional control time per target; and

X_1 is the average additional communication time per target.

To keep the previous efficiency evaluation with an error less than 10% we must have:

$$(R_1 + X_1) N_t \leq 0.1 [f C_1 T_p + T_1 W_{tp}]_{\min}$$

This Expression can be rewritten as (see Equations 6.6, 6.11, and 6.15):

$$(R_1 + X_1) N_t \leq 0.1 \frac{T_1 N_t}{T_p C_p} \qquad (6.18)$$

In Appendix B we evaluate $X_1 N_t$ as:

$$X_1 N_t = IPXC + CPXC + TPXC \approx 10ms \ (N_t = 500)$$

then $X_1 \approx 0.02$ ms.

Equation 6.18 can be rewritten as ($T_p = 9, C_p = 9$):

$$R_1 \leq \frac{T_1}{810} - 0.02$$

Table 9 depicts upper limit values for the additional control overhead to keep the previous efficiency evaluation with an error less than 10%.

TABLE 9 ADDITIONAL CONTROL OVERHEAD (N_t=500) (MS)

T_1	$(R_1)_{max}$	R_1N_t
100	0.1	50
250	0.28	140
500	0.59	295

C. TRACKING CAPACITY

In the introduction of this chapter we defined tracking capacity as the maximum number of targets that our system is able to track with real-time response. In this research we are assuming that the real-time response is constrained by a fixed time interval between successive radar scans (ΔT=3s). That is, the real time constraint can be expressed as:

$$T_{conc}(N) \leq \Delta T \tag{6.19}$$

In Chapter V we introduced the confined space search proposal. This proposal defines an overlap space between CPs to support a target smooth transition when the target crosses the visibility space boundary of some CP. The overlap space introduces redundant work (i.e., some targets may be reported by more than one correlation processor).

The fraction of the tactical scene used as overlap space among all CPs (F_{all}) can be computed as $F_{all}=0.1$. This is because the inner radius is 20NM and the outer radius is 200NM.

The fraction of the tactical scene used as overlap space between two successive CPs (F_2) can be computed as:

$$F_2 = (1-F_{all})\frac{20C_p}{360}$$

This is because we divided the search space into 18 fixed size sections of 20 degrees in each section (see Appendix B).

When a target is located within the inner circle it is reported by C_p processors. When a target is located within the overlap section (but outside of the inner circle) it is reported by two CPs. Otherwise, it is reported by a single CP (see Figure 17).

1. **Upper Limit Capacity**

During the computation of the upper limit efficiency $T_{conc}(N)$ was evaluated by Equation 6.6 therefore, the real-time constraint (see Equation 6.19) can be expressed as:

$$N_c \le (\frac{T_p C_p}{T_1})\Delta T \qquad (6.20)$$

Also, if all targets are evenly distributed within the tactical scene then the number of distinct target reports (N_d) can be computed as:

$$N_d = N_t(1-F_{all}-F_2) + \frac{1}{2}N_t(F_2) + \frac{1}{C_p}N_t(F_{all}) \qquad \textbf{(6.21)}$$

where:

$N_t(1-F_{all}-F_2)$ is the number of targets reported by a single CP;

$N_t(F_2)$ is the number of targets reported by two CPs; and

$N_t(F_{all})$ is the number of targets reported by C_p CPs.

Table 10 depicts the expected upper limit capacity of the (1-9-9)-tree.

TABLE 10 UPPER LIMIT CAPACITY (1-9-9)-TREE

T_1	N_t	N_d
100	2,430	1,667
250	972	666
500	486	333

In the best case, targets are not located within overlap sections. This assumption leads to N_d=N_t.

2. **Lower Limit Capacity**

During the computation of the lower limit efficiency $T_{conc}(N)$ was evaluated by Equation 6.11 thus, the real time constraint (see Equation 6.19) can be expressed as:

$$N_t \leq \left(\frac{T_p}{1.25T_1}\right)\Delta T \tag{6.22}$$

When all detected targets are located within the visibility space of a single tree branch the number of distinct target reports (N_d) can be computed as:

$$N_d = N_t \tag{6.23}$$

Table 11 depicts the expected lower limit capacity of the (1-9-9)-tree.

TABLE 11 LOWER LIMIT CAPACITY (1-9-9)-TREE

T_1	N_t	N_d
100	216	216
250	86	86
500	43	43

3. Capacity with Load Imbalance

During the computation of the efficiency with load imbalance $T_{conc}(N)$ was evaluated by Equation 6.15 thus, the real time constraint (see Equation 6.19) can be expressed as:

$$N_t \leq \left(\frac{C_p T_p}{(1.25)(1.75)T_1}\right)\Delta T \tag{6.24}$$

The number of distinct target reports (N_d) is computed as in Equation 6.21.

Table 12 depicts the expected capacity with load imbalance of the (1-9-9)-tree.

TABLE 12 CAPACITY WITH LOAD IMBALANCE (1-9-9)-TREE

T_1	N_t	N_d
100	1110	761
250	444	304
500	222	152

Again, in the best case targets are not located within overlap sections (i.e., N_d=N_t).

D. SYSTEM REACTION TIME

In the introduction of this chapter we defined system reaction time (SRT) as the average time lag between target detection and target estimation report of all targets being tracked. In this analysis we assume that:

1. Communication can take place simultaneously on all of the incident links of a node and in both directions; and

2. Communication resources are sufficiently plentiful so that there is never a need for queuing communication packets.

1. Best Reaction Time

The best system reaction time ($(SRT)_{bc}$) is expected to happen when all branches of the ($1-C_p-T_p$)-tree are working with the same load and each branch works without local load imbalance.

The time lag (TL) between target detection and target estimation report of the first target being tracked by any branch can be expressed as:

$$TL_1 = R_1+X_1+fC_1+T_1 \tag{6.25}$$

However, $T_1 >> (R_1+X_1+fC_1)$. For instance, with T_1=100ms we have:

1. $R_1 \approx$ 0.1ms (see Table 9);
2. $X_1 \approx$ 0.02ms (see Section B5); and
3. $fC_1 \approx$ 1.0ms (see Table 7).

then, Equation 6.25 can be rewritten as:

$$TL_1 = T_1 \tag{6.26}$$

The time lag between target detection and target estimation report of the n-th target being tracked by the same tracking processor of any branch can be expressed as:

$$TL_n = nT_1 \tag{6.27}$$

Using the assumption that the IP receives C_p reports in parallel, we have:

$$(SRT)_{bc} = \frac{T_1}{W_{tp}} \sum_{n=1}^{W_{tp}} n \tag{6.28}$$

where:

$$W_{tp} = \frac{N_t}{C_p T_p};$$

$$\sum_{n=1}^{W_{tp}} n = \frac{W_{tp}}{2} (W_{tp}+1) .$$

Equation 6.28 can be rewritten as:

$$(SRT)_{bc} = \frac{T_1}{2} \left(\frac{N_t}{C_p T_p}+1\right) \tag{6.29}$$

When we increase (T_1, N_t) we increase SRT because we are increasing the computation demand, and when we increase (C_p, T_p) we decrease SRT because we are increasing the number of computational resources available.

2. Worst Reaction Time

The worst system reaction time ($(SRT)_{wc}$) is expected to happen when all detected targets are located within the visibility space of a single tree branch and local load balance is required in the overloaded branch.

In this case, we have:

1. (T_p-1) tracking processors working with the average workload W_{tp}; and

2. One single tracking processor working with a workload of (1.25) W_{tp}.

then, Equation 6.28 can be rewritten as:

$$(SRT)_{wc} = \left(\frac{T_p-1}{T_p}\right)\left(\frac{T_1}{W_{tp}}\right)\sum_{n=1}^{W_{tp}} n + \left(\frac{1}{T_p}\right)\left(\frac{T_1}{1.25W_{tp}}\right)\sum_{n=1}^{1.25W_{tp}} n \tag{6.30}$$

where:

$$W_{tp} = \frac{N_t}{T_p};$$

$$\sum_{n=1}^{1.25W_{tp}} n = \frac{1.25W_{tp}}{2}(1.25W_{tp}+1).$$

Equation 6.30 can be rewritten as:

$$(SRT)_{wc} = \left(\frac{T_1}{2T_p}\right)\left[(T_p-1)\left(\frac{N_t}{T_p}+1\right)+\left(1.25\frac{N_t}{T_p}+1\right)\right] \tag{6.31}$$

3. **Reaction Time with Load Imbalance**

The system reaction time with load imbalance ($(SRT)_{li}$) is expected to happen when maximum global and local load imbalance are present in the same branch of the $(1-C_p-T_p)$-tree and the remaining branches are working neither with global nor with local load imbalance.

In this case, we have:

1. $(C_p-1)T_p$ tracking processors working with the average workload W_{tp};

2. (T_p-1) tracking processors working with a workload of $(1.75)W_{tp}$; and

3. One single tracking processor working with a workload of $(1.75)(1.25)W_{tp}$.

then, Equation 6.28 can be rewritten as:

$$(SRT)_{li} = \left(\frac{T_p-1}{C_pT_p}\right)\left(\frac{T_1}{1.75W_{tp}}\right)\sum_{n=1}^{1.75W_{tp}} n \; +$$

$$\left(\frac{1}{C_pT_p}\right)\left(\frac{T_1}{(1.25)(1.75)W_{tp}}\right)\sum_{n=1}^{(1.25)(1.75)W_{tp}} n \; + \qquad \mathbf{(6.32)}$$

$$\left(\frac{C_p-1}{C_p}\right)\left(\frac{T_1}{W_{tp}}\right)\sum_{n=1}^{W_{tp}} n$$

where:

$$W_{tp} = \frac{N_t}{C_p T_p};$$

$$\sum_{n=1}^{cW_{tp}} = (\frac{cW_{tp}}{2})(cW_{tp}+1).$$

Equation 6.32 can be rewritten as:

$$(SRT)_{1i} = (\frac{T_1(T_p-1)}{2C_pT_p})(1.75\frac{N_t}{C_pT_p}+1) +$$

$$(\frac{T_1}{2C_pT_p})((1.25)(1.75)\frac{N_t}{C_pT_p}+1) + \qquad (6.33)$$

$$(\frac{C_p-1}{C_p})(\frac{T_1}{2})(\cdot\frac{N_t}{C_pT_p}+1)$$

4. Results Summary

Table 13 depicts a comparative analysis of the system reaction time as a function of the number of targets using the (1-9-9)-tree and a tracking algorithm with T_1=100ms (see Equations 6.29, 6.31, and 6.33)

TABLE 13 SYSTEM REACTION TIME (MS) (T_1=100MS) (1-9-9)-TREE

N_t	$(SRT)_{bc}$	$(SRT)_{wc}$	$(SRT)_{li}$
50	81	335	84
100	112	621	117
150	143	906	151

E. CONFINED SPACE SEARCH VALIDATION

In Chapter V, we introduced the following questions:

1. Does the application support a division of the search space in correlation sections?

2. Does the application support the re-creation of objects in another site with 'acceptable' tracking penalty?

To answer these questions we must implement a simulation to check the tracking filter capability to reduce the measurement errors when targets cross the space search boundary of CPs. The worst case condition happens when we need re-create targets located within the visibility space of a single tree branch. In this case the target termination in one site will happen in parallel with the target initiation in another site (see Figure 16). However, targets without correlated detection (status terminating) are still tracked with the prediction replacing the detection as an input to the tracking algorithm.

1. **Methodology**

The methodology used in the analysis is equivalent to the one used in the analysis of the Kalman filter in Chapter II, with the following differences:

1. The filter improvement of a target tracked by a single processor is compared with the filter improvement when the same target is tracked by two processors; and

2. No overlap section is defined between tree branches.

2. **Target Motion Model**

In this analysis we used the following motion models:

1. Path A: crossing target with constant velocity (v=560 m/s) (foreseeable behavior); and

2. Path B: crossing target with variable acceleration (a=7g when crossing) (unforeseeable behavior).

3. **Implementation Language**

Object-oriented design (OOD) can be used regardless of whether or not the implementation language is object-oriented (OO). Although it may be claimed that implementing an OOD in an object-oriented programming language (OOPL) is the natural way to proceed there are several 'problem areas' that must be resolved before the full potential of concurrent and distributed object-oriented programming (OOP) systems can be realized. [Ref. 37]

Unfortunately, the only OOPL that was available that would execute on a Transputer [Ref. 38] when this research started was Classic-Ada [Ref. 39]. The use of Classic-Ada as an OOPL was examined in [Ref. 40]. Also, its use as a concurrent OOPL was surveyed in [Ref. 41]. However, there were two problems with choosing this language:

1. Running Classic-Ada on our transputers is rather awkward in that the Classic-Ada processor runs only on our Unix-based systems, which means running the Classic-Ada programs through the Classic-Ada processor on the Unix system to produce Ada code which is then ported to the Transputer for compilation; and

2. We only had one Transputer system upon which we could run Ada programs. That is, failure of that system would be catastrophic for this research schedule.

We also investigated the use of C++ [Ref. 42], but it took us so long to locate, order, and receive a version of C++ that ran on the Transputer that we decided to go ahead and work with Logical C [Ref. 43]. However, we made extensive use of OOD, and all C programs were written as much as possible in an 'OO manner'.

4. Conclusions

Table 14 depicts the simulation results. The results obtained support the following conclusions:

1. Tracking degradation does happen when targets cross the border without overlap sections with foreseeable behavior; and

2. Tracking improvement may happen when the target modify its behavior close of the crossing border.

TABLE 14 CONFINED SPACE SEARCH VALIDATION

Path	FI-1P	FI-2P	FI-2P/FI-1P
A	130.3%	110.2%	0.85
B	123.3%	124.1%	1.01

Conclusion 2 can be better understood if we remember that the output of recursive filters is a function of the input and previous output. The previous output acts as the filter memory and when the target modify its assumed behavior we may have tracking improvement because the filter memory is playing against a good estimation report.

No tracking degradation is expected when targets are located within the space defined as ***danger area*** (see Figure 17).

VII. CONCLUSIONS

A. SUMMARY OF SIGNIFICANT RESULTS

The primary goal of this research was to specify and validate a new concurrent decomposition scheme to exploit parallelism of Radar Data Processing algorithms in distributed systems using a tree topology. We avoided the traditional approach of all functions to all processors assigning the execution of correlation algorithms to correlation processors and tracking algorithms to tracking processors. To improve efficiency by reducing the communication cost, and to decrease the gating complexity, we divided the search space into fixed size sections, and distributed the surveillance within the tactical scene among processors located on different branches of the tree. The overlap of correlation and tracking computation, and a reduction of the computation load to evaluate probabilities of association were ensured by using two correlation gates.

During the specification of our proposal we introduced the Decomposition Cost Evaluation Model (DCEM) to support the following decision: Given a single processor class hierarchy design, how should we efficiently decompose this hierarchy into interface and function(s) hierarchies to load on distributed systems? To make possible analytical comparisons among user identified options we defined communication and computation cost functions of objects, classes, hierarchies, and processors. We applied this model to compare our proposal with the hypercube alternative when the application hierarchy is replicated throughout the network.

Load imbalance is a major source of performance degradation in distributed systems. Ideally, during the execution of any load balance strategy no communication costs should be incurred at all. In our research, objects are viewed as computation sinks. That is, load balance can be obtained by transferring objects from one processor site to another. In the Object Reincarnation (OR) proposal we replicate visible methods and rebuild the object state with minor application penalty and without additional communication costs rather than physically transfer objects from one site to another moving visible methods and the object state. That is, with Object Reincarnation, objects die in one processor site (reducing its load) and are reincarnated in another site (increasing its load). This also supports fault recovery as an extension to the load balance problem.

B. STRENGTHS AND WEAKNESSES

1. Strengths

The Confined Space Search Decomposition (CSSD) enhances parallel operations by reducing the communication cost of transferring data among processors and by overlapping the execution of correlation and tracking algorithms. Also, the complexity of multiple-target gating is reduced from $O(N_t^2)$ to $O((N_t/C_p)^2)$ when targets are evenly distributed within the tactical scene.

We have taken an object-oriented approach to the problem of decomposition in distributed systems. In the Decomposition Cost Evaluation Model, the mapping problem is brought to a higher level of abstraction where the question is which classes should be loaded on which processors, not which tasks should be loaded on which processors.

The Object Reincarnation approach supports load balance without extra communication among processors, and with minor application penalty.

2. Weaknesses

The tree topology offers low connectivity. That is, failure of any of its links creates two subsets of processors that cannot communicate with each other. The solution to this weakness was the proposal of a fault recovery algorithm as an extension to the load balance problem using the Object Reincarnation approach.

The performance of the Confined Space Search Decomposition is sensitive to the environment model. Here the question is: What is the frequent case? If the real target distribution matches the expected target distribution then we get the benefits of the Confined Space Search Decomposition.

Otherwise, load imbalance can occur. In the worst case a single branch may be responsible for executing radar data processing functions for all detected targets. That is, for a particular application, the computational power of any branch must be specified to support the worst acceptable system reaction time for the maximum number of expected targets. However, as discussed in Appendix B it is our view that the architecture design should not be conceived to favor the infrequent case.

C. SUGGESTED FUTURE DIRECTIONS

We have taken a theoretical path in the validation of our research. Thus, we recommend the implementation of the Confined Space Search Decomposition to evaluate performance, tracking capacity, and system reaction time of our proposal using a particular processor (such as the Inmos T9000 Transputer, as discussed in Appendix B).

Extensions to the Ada language such as Classic-Ada brings the power of concurrent object-oriented programming to the Ada developer. C++ is now available as a preprocessor to produce INMOS ANSI C or 3LC code to run on networks of Transputers. We identify these options as good language candidates to use in the implementation.

We used the Decomposition Cost Evaluation Model to compare the hypercube alternative with the tree alternative. We also recommend the application of the model to compare our proposal with other distributed systems using different topologies and/or code allocation policies.

D. CONCLUDING REMARKS

In this research we reviewed possible implementations of radar data processing algorithms, developed a single processor object-oriented design for this application, decomposed this design into a distributed computer system using two user identified alternatives, compared these alternatives, and then developed a theoretical prediction of selected parameters for the selected architecture. To accomplish this, we faced two main problems: The decomposition of a single processor software design into a distributed computer system, and the load balance issue. As a solution to the decomposition of the software design we introduced the Decomposition Cost Evaluation Model from which we derived the Confined Space Search Decomposition, and to reduce the penalties of load imbalance we proposed the Object Reincarnation Heuristic. Thus, we have proposed and validated a distributed computational system that will increase the computing capacity of future combat systems.

APPENDIX A

1. Component Classes

CLASS: PREDICTION

Superclasses:	None.
Class Variables:	None.
Instance Variables:	XP: real.
	VXP: real.
	YP: real.
	VYP: real.
Methods:	GET & SET XP.
	GET & SET VXP.
	GET & SET YP.
	GET & SET VYP.

CLASS: ESTIMATION	
Superclasses:	None.
Class Variables:	None.
Instance Variables:	XE: real.
	VXE: real.
	YE: real.
	VYE: real.
Methods:	GET & SET XE.
	GET & SET VXE.
	GET & SET YE.
	GET & SET VYE.

CLASS: DETECTION	
Superclasses:	None.
Class Variables:	None.
Instance Variables:	XD: real.
	YD: real.
Methods:	GET & SET XD.
	GET & SET YD.

CLASS: SIZE	
Superclasses:	None.
Class Variables:	None.
Instance Variables:	XS: real.
	YS: real.
Methods:	GET & SET XS.
	GET & SET YS.

CLASS: PAIRING	
Superclasses:	None.
Class Variables:	None.
Instance Variables:	MID: integer.
	MEASURE: DETECTION.
	DISTANCE: real [*0.0*].
	TN: integer.
Methods:	GET & SET MID.
	GET & SET MEASURE.
	GET & SET DISTANCE.
	GET & SET TN.

CLASS: CONTROL	
Superclasses:	None.
Class Variables:	None.
Instance Variables:	TN: integer.
	STATUS: {*AL,DE*} [*DE*].
Methods:	GET & SET TN.
	GET & SET STATUS.

2. Abstract Classes

CLASS: TARGET	
Superclasses:	None.
Class Variables:	COBU: buffer of PAIRING.
Instance Variables:	None.
Methods:	None.

3. Concrete Classes

CLASS: SENSOR	
Superclasses:	None.
Class Variables:	None.
Instance Variables:	SP: (*3.0..5.0*) [*3.0*].
	PD: (*0.9..1.0*) [*1.0*].
	DETECTIONS: buffer of DETECTION.
Methods:	SIMRADET

CLASS: FIRM	
Superclasses:	TARGET.
Class Variables:	None.
Instance Variables:	TN: integer.
	SOBU: buffer of PAIRING.
	MYPROP: PAIRING.
	MYPR: PREDICTION.
	MYES: ESTIMATION.
	MYST: {*FI*, *TE*, *DE*} [*FI*].
	FGSZ: SIZE.
	SGSZ: SIZE.
	MGSZ: SIZE.
Methods:	SOCOBU
	RECOPRFG
	RECOPRSG
	RECUST
	SEST
	EXSEQTE
	EXTR
	CHKEQ
	STYOU

CLASS: TENTATIVE	
Superclasses:	TARGET.
Class Variables:	None.
Instance Variables:	TN: integer.
	SOBU: buffer of PAIRING.
	MYPROP: PAIRING.
	MYPR: PREDICTION.
	MYES: ESTIMATION.
	MYST: {*CI*,*CF*,*DE*} [*CI*].
	IGSZ: SIZE.
	MGSZ: SIZE.
Methods:	SOCOBU
	RECOPRIG
	RECUST
	SEST
	EXSEQTE
	EXTR
	CHKEQ
	STYOU

CLASS: SCHEDULER

Superclasses:	TARGET.
Class Variables:	None.
Instance Variables:	COFI: buffer of CONTROL.
	COTE: buffer of CONTROL.
Methods:	RUN

APPENDIX B

APPLYING THE DECOMPOSITION COST EVALUATION MODEL

1. Identification of Concrete Classes

In Chapter IV we identified SENSOR, FIRM, TENTATIVE, and SCHEDULER as the concrete classes of our application. Thus, these are the preliminary classes to consider during the evaluation of communication and computation costs.

2. Identification of Interface Functions

During the initial design in Chapter IV, we assigned interface and application functions to a single processor. Here we want to build an *Interface Hierarchy* to encapsulate interface functions in classes assigned to Interface Processors (***IPs***), and an *Application Hierarchy* to encapsulate application functions in classes assigned to Radar Data Processing Processors (***RDPPs***). The following interface functions were identified:

1. The method SIMRADET in class SENSOR is responsible to simulate radar detections. This suggests that the class SENSOR should be assigned to the interface hierarchy.

2. Methods CHKEQ and STYOU in classes FIRM and TENTATIVE are responsible for executing the check equivalence and store

yourself services. This suggests that a new class INTERFACE should be created in the interface hierarchy to encapsulate those services.

3. A new client called INTERFACE_SCHEDULER must be created in the interface hierarchy. This new client will be the communication interface with the old SCHEDULER renamed as RDP_SCHEDULER.

a. Interface Hierarchy (IH)

(1) Refining Contract Responsibilities. The existence of a new client requires refinement of responsibilities. The following reallocation of responsibilities from the rdp_scheduler (single instance of the class RDP_SCHEDULER in the application hierarchy) to the interface_scheduler (single instance of the class INTERFACE_SCHEDULER in the interface hierarchy) are identified:

1. Create the sensor object;
2. Ask report of radar detections;
3. Ask equivalence check; and
4. Storage request.

Also, the following new responsibilities were allocated to the interface_scheduler:

1. Transmit correlation buffer: During each data scanning the interface_scheduler object must transmit radar detections to rdp_schedulers using some communication server;

2. Ask report of estimations: During each data scanning the interface_scheduler must request the delivery of positions after estimation from rdp_schedulers;

3. Create interface objects: Each estimation received becomes an interface object;

4. Destroy interface objects: During each data scanning all created interface objects are destroyed.

(2) **Refining Objects Structures**. The objects structures are defined as:

1. Sensor object (Chapter IV);

2. Interface object:

ESRE: ESTREPORT. /* Estimation Report */

ESBU: buffer of ESTREPORT.

MGSZ: SIZE.

3. Interface_scheduler:

DEBU: buffer of DETECTION.

ESBU: buffer of ESTREPORT.

COIN: buffer of CONTROL.

(3) **Refining Class Hierarchies**. The classes in the hierarchy are defined as:

1. Class ESTREPORT (Component):

CLASS: ESTREPORT	
Superclasses:	None.
Class Variables:	None.
Instance Variables:	TN: integer.
	MYES: ESTIMATION.
Methods:	GET & SET TN.
	GET & SET MYES.

2. Class ESTIMATION (see Appendix A);

3. Class DETECTION (see Appendix A);

4. Class CONTROL (see Appendix A);

5. Class SIZE (see Appendix A);

6. Class ESTBUFFER (Abstract):

CLASS: ESTBUFFER	
Superclasses:	None.
Class Variables:	ESBU: buffer of ESTREPORT.
Instance Variables:	None.
Methods:	None.

The estimation buffer (ESBU) is set by the interface_scheduler and must be visible for read operations by interface objects during the execution of the CHKEQ method. The class ESTBUFFER is defined as a superclass of the concrete classes INTERFACE and INTERFACE_SCHEDULER;

7. Class SENSOR (see Appendix A);

8. Class INTERFACE (Concrete):

CLASS: INTERFACE

Superclasses:	ESTBUFFER.
Class Variables:	None.
Instance Variables:	ESRE: ESTREPORT.
	MGSZ: SIZE.
Methods:	CHKEQ.
	STYOU.

9. Class INTERFACE_SCHEDULER (Concrete):

CLASS: INTERFACE_SCHEDULER

Superclasses:	ESTBUFFER.
Class Variables:	None.
Instance Variables:	COIN: buffer of CONTROL.
	DEBU: buffer of DETECTION.
Methods:	RUN.

Figure 18 depicts the interface hierarchy, and Figure 19 depicts the client-server relationship. ESTBUFFER is an abstract class, SENSOR and INTERFACE are concrete classes used to instantiate server objects, and INTERFACE_SCHEDULER is a concrete class used to instantiate the client object (i.e., the controller).

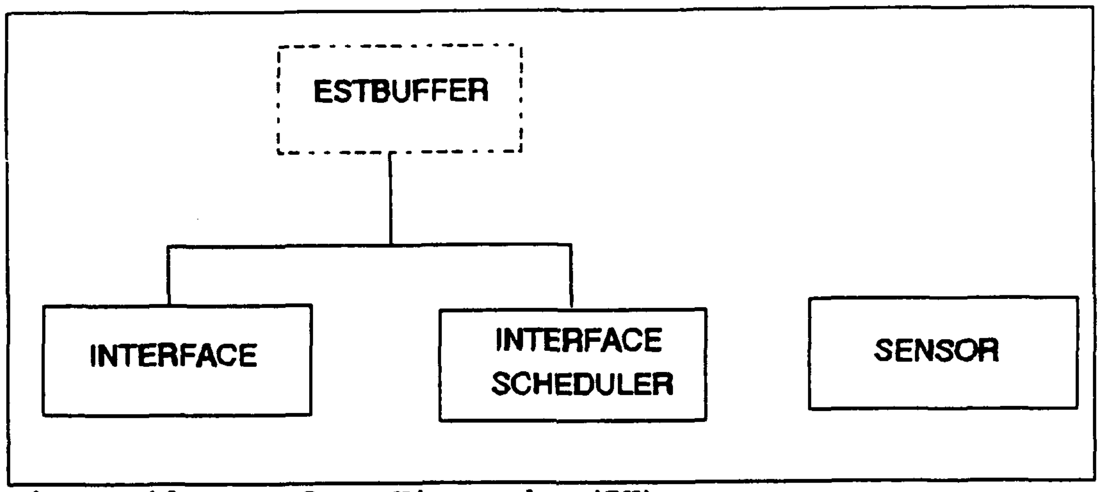


Figure 18 Interface Hierarchy (IH)

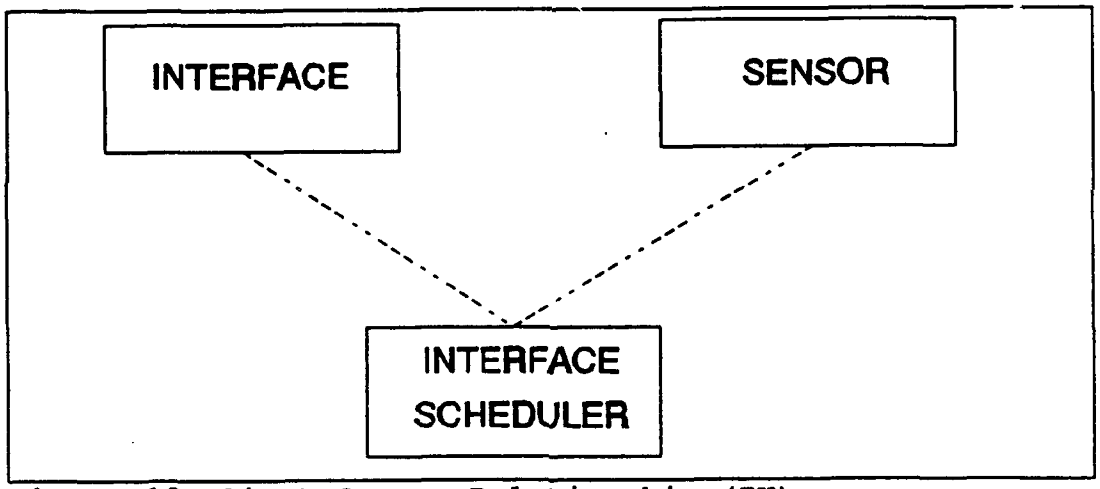


Figure 19 Client-Server Relationship (IH)

b. Application Hierarchy (AH)

(1) **Refining Contract Responsibilities.** New responsibilities are allocated to the rdp_scheduler:

1. Ask estimation to firm objects;

2. Ask estimation to tentative objects;

3. Report estimations to the Interface_Scheduler.

(2) **Refining Objects Structures**. The objects structures are redefined as:

1. Firm object: (Remove MGSZ, Chapter IV);

2. Tentative object: (Remove MGSZ, Chapter IV);

3. Rdp_scheduler:

ESBU: buffer of ESTREPORT. /* New */

(3) **Refining Class Hierarchies**. The classes in the hierarchy are defined as:

1. Class PREDICTION (see Appendix A);
2. Class ESTIMATION (see Appendix A);
3. Class PAIRING (see Appendix A);
4. Class CONTROL (see Appendix A);
5. Class SIZE (see Appendix A);
6. Class TARGET (see Appendix A);
7. Class ESTREPORT (Interface Hierarchy);

8. Class FIRM (Concrete):

CLASS: FIRM	
Superclasses:	TARGET.
Class Variables:	None.
Instance Variables:	/* Remove MGSZ */
Methods:	SOCOBU
	RECOPRFG
	RECOPRSG
	RECUST
	SEST
	EXSEQTE
	EXTR
	GETTN /* New */
	GETMYES /* New */

Methods GETN and GETMYES are created to support the responsibility report estimations of the rdp_scheduler.

9. Class TENTATIVE (Concrete):

CLASS: TENTATIVE

Superclasses:	TARGET.
Class Variables:	None.
Instance Variables:	/* Remove MGSZ */
Methods:	SOCOBU
	RECOPRIG
	RECUST
	SEST
	EXSEQTE
	EXTR
	GETTN /* New */
	GETMYES /* New */

10. Class RDP_SCHEDULER (Concrete):

CLASS: RDP_SCHEDULER

Superclasses:	TARGET.
Class Variables:	None.
Instance Variables:	COFI: buffer of CONTROL.
	COTE: buffer of CONTROL.
	ESBU: buffer of ESTREPORT.
Methods:	RUN

Figure 20 depicts the application hierarchy, and Figure 21 depicts the client-server relationship. TARGET is an abstract class, FIRM and TENTATIVE are concrete classes used to instantiate server objects, and RDP_SCHEDULER is a concrete class used to instantiate the client object (i.e., the controller).

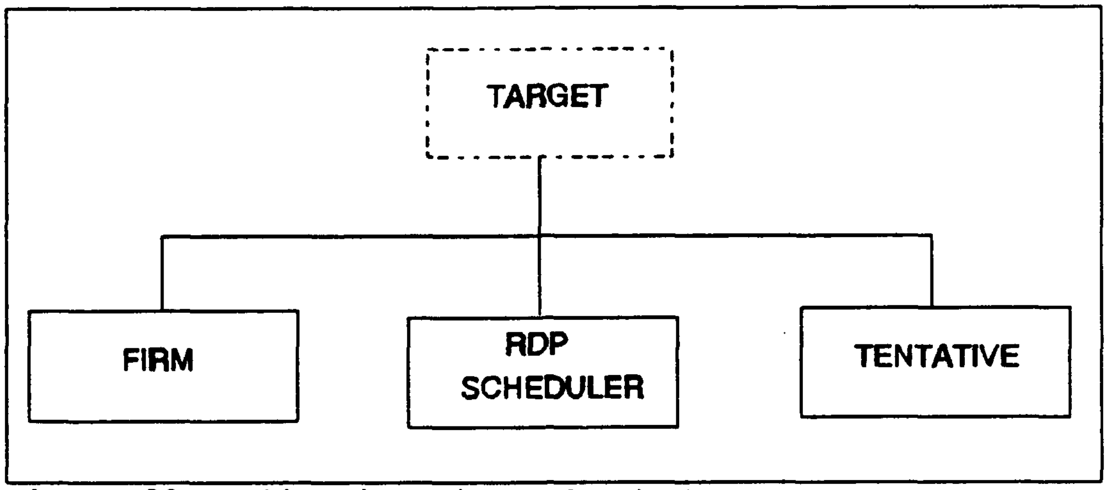


Figure 20 Application Hierarchy (AH)

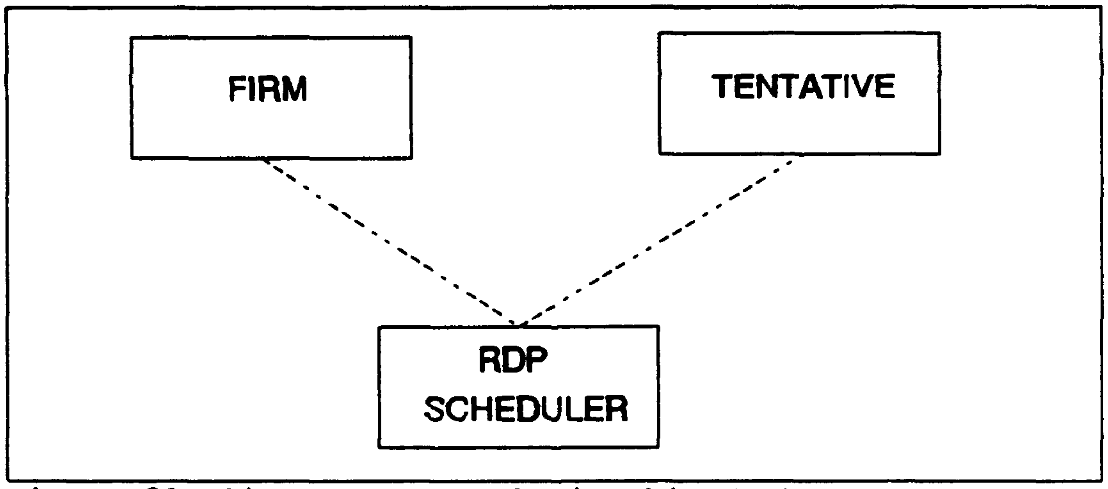


Figure 21 Client-Server Relationship (AH)

3. Identification of High Cost Functions

As introduced in Chapter II, tracking functions using multifilter algorithms such as the ***Interacting Multiple Model***

(IMM) and ***Viterbi Algorithm*** (VA) are too computation intensive (≈600 ms) to be used in real applications with SISD architectures.

In the application hierarchy, we identify the method EXTR (Execute Tracking) in the classes FIRM and TENTATIVE. The initiation algorithm used to implement this method in the class TENTATIVE (Equations 3.2, 3.3, and 3.4) is very simple and does not deserve any special attention; however, improved implementations of tracking algorithms are high cost functions, and high cost functions are natural candidates to be executed in several processors. This suggests:

1. Partition of the application hierarchy into two function hierarchies: a *Tracking Hierarchy* to encapsulate tracking functions in classes assigned to Tracking Processors (***TPs***) and a *Correlation Hierarchy* to encapsulate correlation functions in classes assigned to Correlation Processors (***CPs***);

2. Migration of the method EXTR from the class FIRM in the application hierarchy to the class TRACKING in the tracking hierarchy;

3. Creation of two new clients: TRACKING_SCHEDULER (controller in tracking processors); and CORRELATION_SCHEDULER (controller in correlation processors).

a. Tracking Hierarchy (TH)

(1) Refining Contract Responsibilities. The existence of a new client requires refinement of responsibilities. The function ask tracking execution to firm objects is transferred from the old rdp_scheduler renamed as correlation_scheduler (single instance of the class CORRELATION_SCHEDULER in the correlation hierarchy) to the tracking_scheduler (single instance of the class TRACKING_SCHEDULER in the tracking hierarchy).

New responsibilities are allocated to the tracking_scheduler:

1. Ask tracking report: For each track (instance of the class TRACKING), during each data scanning, the correlation_scheduler must deliver to the tracking_scheduler a tracking report including the target number, correlated detection, and initial prediction (needed only when a new track is going to be created);

2. Transmit tracking update: For each track, during each data scanning, tracking processors must deliver to correlation processors a tracking update including the target number new prediction and current estimation;

3. Create tracking objects: New firm targets are reported with an encoded target number to signal the requirement of a new track; and

4. Destroy tracking objects: For each track, during each data scanning, a tracking report must be received. The absence of this report is used as a signal to destroy the track.

(2) **Refining Objects Structures**. The objects structures are defined as:

1. Tracking object:

TN: integer.

MYDE: DETECTION.

MYPR: PREDICTION.

MYES: ESTIMATION.

2. Tracking_scheduler:

COTR: buffer of CONTROL.

(3) **Refining Class Hierarchies**. The classes in the hierarchy are defined as:

1. Class TRREP (Component):

CLASS: TRREP

Superclasses:	None.
Class Variables:	None.
Instance Variables:	TN: integer.
	MYDE: DETECTION.
	MYPR: PREDICTION.
Methods:	GET & SET TN.
	GET & SET MYDE.
	GET & SET MYPR.

2. Class TRUPD (Component):

CLASS: TRUPD

Superclasses:	None.
Class Variables:	None.
Instance Variables:	TN: integer.
	MYES: ESTIMATION.
	MYPR: PREDICTION.
Methods:	GET & SET TN.
	GET & SET MYES.
	GET & SET MYPR.

3. Class DETECTION (see Appendix A);

4. Class PREDICTION (see Appendix A);

5. Class ESTIMATION (see Appendix A);

6. Class CONTROL (see Appendix A);

7. Class TRACKING (Concrete):

CLASS: TRACKING

Superclasses:	None.
Class Variables:	None.
Instance Variables:	TN: integer.
	MYDE: DETECTION.
	MYES: ESTIMATION.
	MYPR: PREDICTION.
Methods:	EXTR.
	GET & SET TN.
	GET & SET MYDE.
	GET & SET MYES.
	GET & SET MYPR.

8. Class TRACKING_SCHEDULER (Concrete):

CLASS: TRACKING_SCHEDULER

Superclasses:	None.
Class Variables:	None.
Instance Variables:	COTR: buffer of CONTROL;
Methods:	RUN.

Figure 22 depicts the tracking hierarchy, and Figure 23 depicts the client-server relationship. TRACKING is a concrete class used to instantiate server objects, and TRACKING_SCHEDULER is a concrete class used to instantiate the client object (i.e., the controller).

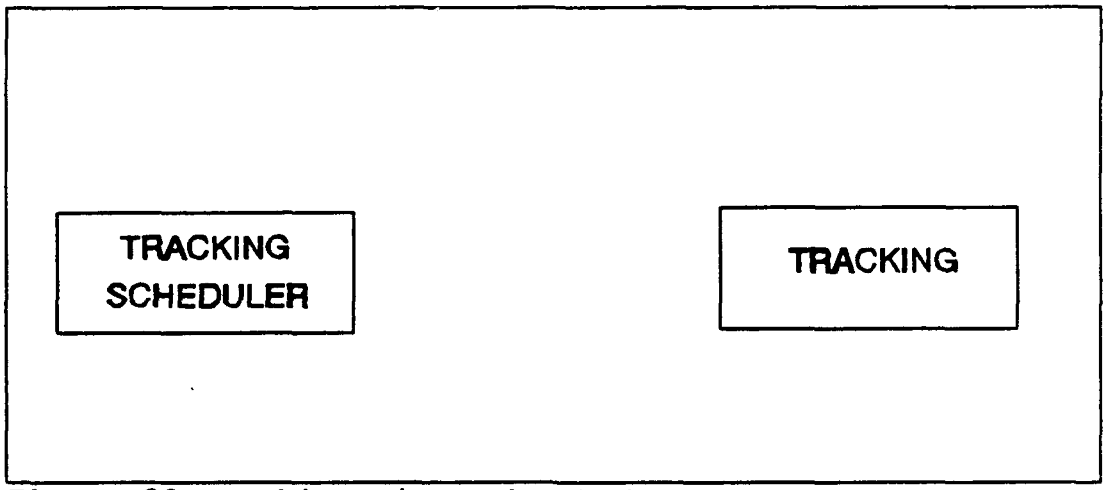


Figure 22 Tracking Hierarchy (TH)

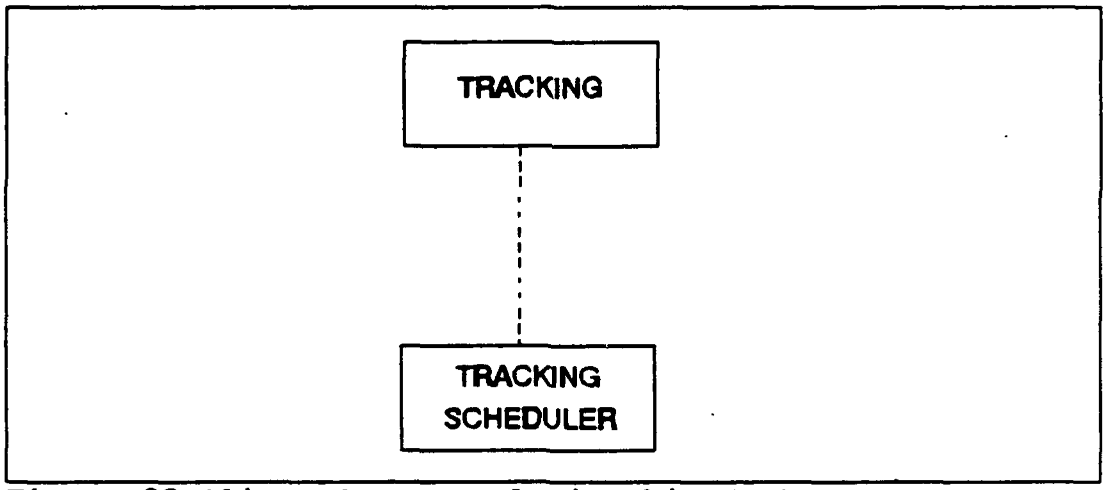


Figure 23 Client-Server Relationship (TH)

a. Correlation Hierarchy (CH)

(1) **Refining Contract Responsibilities.** New responsibilities are created and some canceled in the rdp_scheduler renamed as correlation_scheduler:

1. Ask estimation to firm objects (canceled);

2. Transmit tracking report (New);

3. Ask tracking update (New).

(2) **Refining Objects Structures**. The objects structures are redefined as:

1. Firm object: (Remove MYES, Application Hierarchy);

2. Tentative object: (Application Hierarchy);

3. Correlation_scheduler: (Application Hierarchy).

(3) **Refining Class Hierarchies**. The classes in the hierarchy are defined as:

1. Class PREDICTION (see Appendix A);
2. Class ESTIMATION (see Appendix A);
3. Class PAIRING (see Appendix A);
4. Class CONTROL (see Appendix A);
5. Class SIZE (see Appendix A);
6. Class TARGET (see Appendix A);
7. Class ESTREPORT (Interface Hierarchy);

8. Class FIRM (Concrete):

CLASS: FIRM	
Superclasses:	TARGET.
Class Variables:	None.
Instance Variables:	/* Remove MYES */
Methods:	SOCOBU
	RECOPRFG
	RECOPRSG
	RECUST
	SEST
	EXSEQTE
	GETTN

9. Class TENTATIVE (Application Hierarchy);

10. Class CORRELATION_SCHEDULER (Application Hierarchy).

Figure 24 depicts the correlation hierarchy, and Figure 25 depicts the client-server relationship. TARGET is an abstract class, FIRM and TENTATIVE are concrete classes used to instantiate server objects, and CORRELATION_SCHEDULER is a concrete class used to instantiate the client object (i.e., the controller).

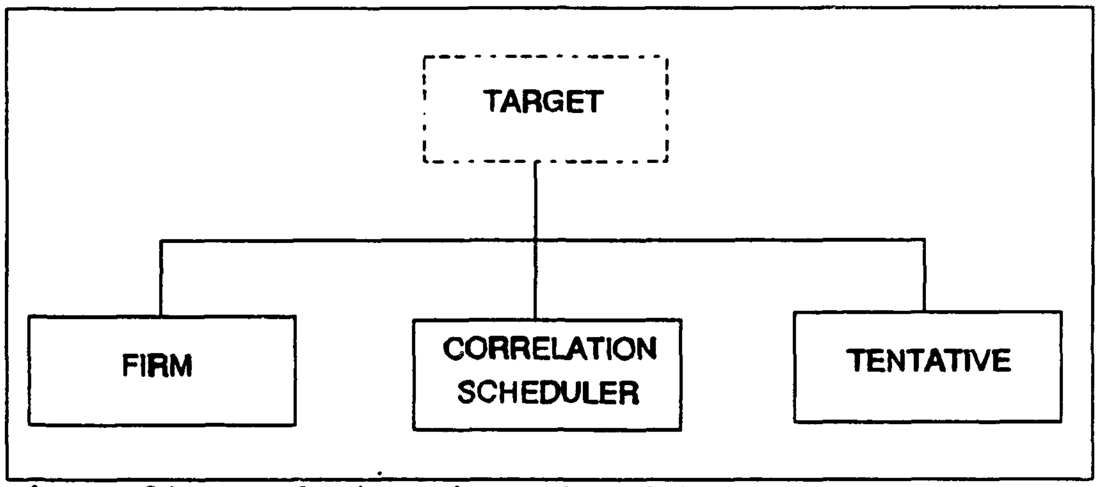


Figure 24 Correlation Hierarchy (CH)

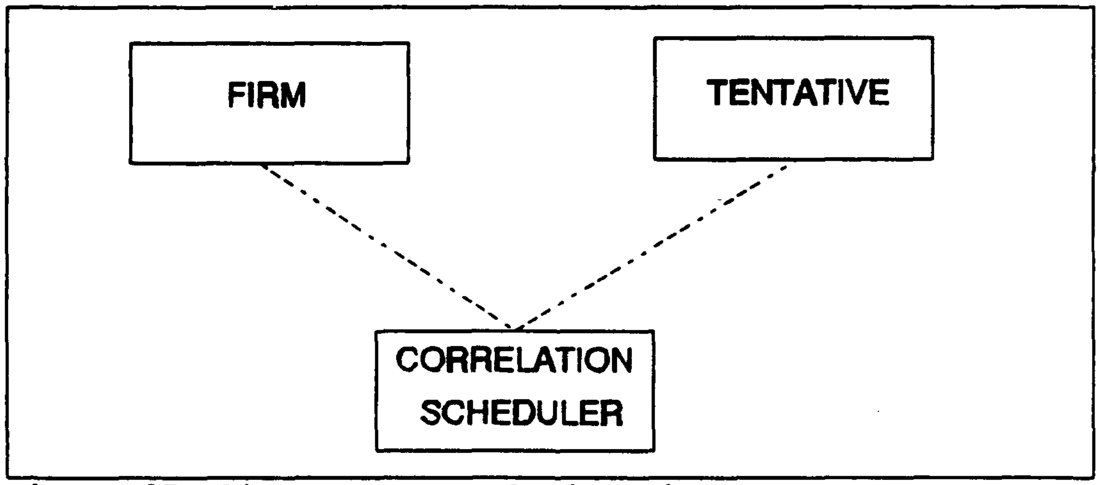


Figure 25 Client-Server Relationship (CH)

4. Divide and Conquer

a. All Functions to All Processors (AFAP)

During the identification of interface functions the single processor was replaced by an interface processor (*IP*) and radar data processing processors (***RDPPs***). In this option the application hierarchy, assigned to RDPPs, is replicated throughout the network. The buffer of detection (DEBU) is transmitted to RDPPs by the IP. The buffer of estimation reports (ESBU) is transmitted to the IP by RDPPs. The main issue is to select an architecture topology to support communicating objects with low communication overhead.

b. Some Functions to Some Processors (SFSP)

During the identification of high cost functions the RDPPs were replaced by correlation processors (***CPs***) and tracking processors (***TPs***). In this option, correlation hierarchies are assigned to CPs and tracking hierarchies are assigned to TPs. The buffer of detection (DEBU) is transmitted to CPs by the IP. Input tracking reports (TRINP) are transmitted from CPs to TPs. Output tracking updates (TROUT) are transmitted from TPs to CPs and the buffer of estimation reports (ESBU) is transmitted from CPs to the IP. The communication pattern: IP => CPs => TPs => CPs => IP suggests that (IP,CPs) and (CPs,TPs) should be neighbors. Also, to reduce the communication cost of the buffer of detections not all detections should be transmitted to all correlation processors. This means that the 'search space' should be divided in correlation sections.

5. Identification of Options

The hypercube topology offers high connectivity and short diameter; it was identified as OPTION I to implement the *Application Hierarchy Design*. We will use the notation **d-cube** to represent a cube with diameter **d**.

The tree topology offers low connectivity because the failure of any one of its links creates two subsets of processors that cannot communicate with one another. However, the interconnection network of objects is also a tree with the IP as a root at level 0, CPs at level 1, and TPs at level 2. This optimize the communication cost, so the tree topology was identified as OPTION II to implement the *Function Hierarchies Design*. We will use the notation **$(1-C_p-T_p)$-tree** to represent a tree with the interface processor as the root node, C_p correlation processors at level 1, and T_p processors (for each correlation processor) at level 2.

6. Conceptual Comparisons Among Options

Since the options have been identified (d-cube (AFAP) and $(1-C_p-T_p)$-tree (SFSP)) we can now start the conceptual comparisons.

a. Topology Comparisons

Tables 15 and 16 depict topology data associated with the two selected options. Table 17 depicts some typical values.

TABLE 15 TOPOLOGY DATA D-CUBE

Characteristic	Value
Diameter	d
Connectivity	d
Node Degree	d
Average Distance	$[2^d d]/[2(2^d-1)]$

TABLE 16 TOPOLOGY DATA $(1-C_p-T_p)$-TREE

Characteristic	Value
Diameter of Communicating Objects	2
Connectivity	1
Node Degree	IP - C_p CPs - T_p+1 TPs - 1
Average Distance of Communicating Objects	$(C_p+2T_pC_p)/(C_pT_p+C_p+1)$

TABLE 17 TOPOLOGY DATA (COMPARATIVE ANALYSIS)

Topology	Diameter	Connectivity	Average Degree	Average Distance
6-Cube	6	6	6	3.05
7-Cube	7	7	7	3.53
(1-8-8)-Tree	2	1	1.97	1.86
(1-9-9)-Tree	2	1	1.98	1.88

The average degree of the (1-C_p-T_p)-tree can be evaluated as:

$$(Degree)_{Avg} = \frac{C_p + (T_p + 1)C_p + T_pC_p}{C_pT_p + C_p + 1} \tag{B.1}$$

When $C_p = T_p$, we have:

$$(Degree)_{Avg} = \frac{2C_p^2 + 2C_p}{C_p^2 + C_p + 1} \approx 2 \text{ when } C_p \geq 3 \tag{B.2}$$

In Tables 15,16, and 17 we would like to emphasize the following:

1. The (1-C_p-T_p)-tree offers low connectivity. Failure of any of its links creates two subsets of processors that cannot communicate with one another. Any link failure isolates one parent node (IP or CP) from its child node (CP or TP), so link failures requires load transfer from some child node to

its siblings; we will return to this point later. The connectivity of the d-cube increases with the cube dimension.

2. When $C_p=T_p$ and $C_p \geq 3$, the average degree of the $(1-C_p-T_p)$-tree is not sensitive to the number of processors (≈ 2, Equation B.2). however, the degree of the IP is C_p and the degree of CPs is (T_p+1). The degree of the d-cube increases with the cube dimension. Nodes with high degree require hardware support (in each node) to increase the parallelism between computation and communication. During the Transputing'91 Conference, Inmos introduced the T9000 Transputer with a dedicated communication processor which operates concurrently with the main processor and a packet routing switch connecting 32 links to each other via a 32 by 32 way, non-blocking crossbar switch with sub-microsecond latency. The goal is to remain with maximum degree four in each node of a distributed computational system. To avoid network hot spots the routing switch can optionally implement a two phase routing algorithm (Universal Routing).

3. When the cube dimension is greater then or equal to four, the average distance (e.g., for d=4, $(\text{Distance})_{avg}$ = 2.13) of the d-cube is greater than the maximum distance that a message must travel in the $(1-C_p-T_p)$-tree (diameter=2). This increases the communication cost of the d-cube when compared with the $(1-C_p-T_p)$-tree.

b. Principles of Computer Design

Perhaps the most important and pervasive principle of computer design is to handle the common case fast. In making a design tradeoff, favor the frequent case over the infrequent case. This principle also applies when determining how to spend resources. We have identified tracking as the high cost function of our application. In the $(1-C_p-T_p)$-tree architecture the computational power is mainly concentrated in TPs (Table 18).

TABLE 18 COMPUTATIONAL POWER DISTRIBUTION

Topology	% TPs
(1-3-3)-tree	69%
(1-9-9)-tree	89%
(1-9-2)-tree	64%

To avoid the penalty of the Amdahl's Law (i.e., the performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used), we need to overlap tracking with correlation and enhance correlation. The correlation algorithm is implemented in phases: first gate, second gate with ambiguity resolution, and initiation/ termination (see Chapter III). For each target, tracking can start as soon as a pair (plot,target) is obtained. To enhance correlation we need to divide the 'search space' into correlation sections. This means that we have different CPs working in different space sections; therefore, not all detections need to be transmitted to all correlation processors. Again, the issue is: Does the application support a division of the search space in correlation sections?.

c. Fault Tolerance Comparisons

Tables 19 and 20 depict the expected consequences for the application when a selected option operates with faulty components (nodes or links). Tables 21 and 22 depicts possible actions needed during the recovery of a faulty condition.

TABLE 19 FAULT CONSEQUENCES D-CUBE

Component	Consequence
Node	Root - Lose all processing power; Otherwise - Lose one processor.
Link	Processing power remains the same.

TABLE 20 FAULT CONSEQUENCES $(1-C_p-T_p)$-TREE

Component	Consequence
Node/Link	IP - Lose all processing power; CPs - Lose 1 CP and T_p TPs; TPs - Lose one processor.

TABLE 21 FAULT RECOVERY D-CUBE

Component	Recovery
Node	Root - Radar switch; Otherwise - Dynamic adjust of the routing software.
Link	Dynamic adjust of the routing software.

TABLE 22 FAULT RECOVERY $(1\text{-}C_p\text{-}T_p)$-TREE

Component	Recovery
Node/Link	Root - Radar switch; CPs - Transfer of load to siblings; TPs - Transfer of load to siblings.

During the last five years, the problem of routing messages on hypercubes with faulty components has motivated an intense research effort, resulting in several proposals being presented. Algorithms were proposed which require only knowledge of the status of a processor's immediate neighbors or which requires each node to know only the information of its own links.

In this research we study the fault tolerance problem (Chapter V) in $(1\text{-}C_p\text{-}T_p)$-trees as an extension to the load balance problem where: the IP needs to know the status of IP-CPs links to decide if load transfer from the isolated CP to its siblings is required (***Global Load Balance***); and each CP needs to know the status of CP-TPs links to decide if load transfer from the isolated TP to its siblings is required (***Local Load Balance***).

7. Evaluation of the Communication Cost

During the evaluation of the communication cost components we will assume:

1. Communication can take place simultaneously on all of the incident links of a node and in both directions;

2. Communication resources are sufficiently plentiful so that there is never a need for queuing communication packets;

3. The physical distance between transmitter and receiver is so small that propagation delay is negligible;

4. Link speed: b bits/second;

5. Three bits of overhead for each byte of any communication packet; and

6. Absence of load imbalance (i.e., there is a uniform distribution of targets among nodes);

In the previous assumptions we would like to emphasize the following:

1. Assumptions 1 and 2 are expected to benefit the d-cube option due to the retransmit cost (see Section B7). No retransmission overhead is needed in the $(1\text{-}C_p\text{-}T_p)$-tree option;

2. Assumptions 3, 4, and 5 are expected to equally affect both selected options; and

3. Assumption 6 concerns both options. In the AFAP (d-cube) option the IP controls the load balance (centralized policy), while in the SFSP ($(1-C_p-T_p)$-tree) option the IP controls the global load balance and CPs control the local load balance (distributed policy).

a. Communication Cost: d-Cube (AFAP)

In this option the application hierarchy (assigned to RDPPs) is replicated throughout the network. The buffer of detection (DEBU) is transmitted to RDPPs by the IP. The buffer of estimation reports (ESBU) is transmitted to the IP by RDPPs. The following communication objects were identified:

1. Interface_scheduler running in the IP; and

2. Rdp_scheduler running in RDPPs.

The communication cost components were identified as:

*1. **Service request cost** (SRC):* Overhead in the IP to transmit and in RDPPs to receive the buffer of detections;

*2. **Result cost** (RC):* Overhead in RDPPs to transmit and in the IP to receive the buffer of estimation reports;

*3. **Retransmit cost** (C_{RT}):* Overhead in RDPPs used to route buffer of detections and return buffer of estimation reports.

Table 23 depicts the required packet sizes (detection and estimation reports).

TABLE 23 PACKET SIZES D-CUBE

Class	Bytes (Info)	Bytes (Overhead)	Bytes (Total)
DETECTION	16	6	22
ESTREPORT	36	13.5	49.5

Applying the proposed methodology (Chapter V, Section B1h), we have:

(1) **IP Communication Cost.** The communication cost of the IP may be evaluated using the following sequence:

1. Object communication cost (OXC)

$$OXC_{interface\_scheduler} = \sum_{k=1}^{(2^d - 1)} c_{interface\_scheduler,k}$$

where:

k-objects are rdp_schedulers running in RDPPs.

Assuming single node broadcast (same detection buffer from the IP to every RDPP) with parallel transmission in d-links of N_t detections and single node accumulation (send to IP estimation reports from every RDPP) with parallel reception in each link of N_t/d distinct estimation reports, we have:

$$OXC_{interface\_scheduler} = SRC + RC$$

$$SRC = N_t(targets)*22(bytes/target)*(8/b)(s/byte)$$

$$RC = (N_t/d)(targets)*49.5(bytes/target)*(8/b)(s/byte)$$

$$OXC_{interface\_scheduler} = \frac{176N_t}{b}(1 + \frac{2.25}{d})$$

2. Class communication cost (CXC)

The interface_scheduler is the single instance of the class INTERFACE_SCHEDULER, thus:

$$CXC_{INTERFACE\_SCHEDULER} = OXC_{interface\_scheduler}$$

3. Hierarchy communication cost (HXC)

The INTERFACE_SCHEDULER is the single class of the interface hierarchy loaded on the interface processor where a single communication object (interface_scheduler) is created, thus:

$$HXC_{INTERFACE} = CXC_{INTERFACE\_SCHEDULER}$$

4. Processor communication cost (PXC)

The interface hierarchy is the single hierarchy loaded on the interface processor, thus:

$$IPXC = \frac{176N_t}{b}(1 + \frac{2.25}{d}) \qquad \textbf{(B.3)}$$

The retransmit cost (C_{RT}) of the IP is zero.

(2) RDPP Communication cost. The communication cost of RDPPs may be evaluated using the following sequence:

1. Object Communication Cost (OXC)

$$OXC_{rdp\text{-}schedule^-} = \sum_{k=1}^{1} c_{rdp\text{-}scheduler,k}$$

The single k-object is the interface_scheduler running in the IP.

Each RDPP receives N_t detections and transmits $(N_t/(2^d-1))$ estimation reports (absence of load imbalance), thus:

$$OXC_{rdp\text{-}scheduler} = SRC + RC$$

$$SRC = N_t(targets)*22(bytes/target)*(8/b)(s/byte)$$

$$RC=(N_t/(2^d - 1))(targets)*49.5(bytes/target)*(8/b)(s/byte)$$

$$OXC_{rdp\text{-}scheduler} = \frac{176N_t}{b}(1 + \frac{2.25}{2^d - 1})$$

2. Class communication cost (CXC)

The rdp_scheduler is the single instance of the class RDP_SCHEDULER, therefore:

$$CXC_{RDP\text{-}SCHEDULER} = OXC_{rdp\text{-}scheduler}$$

3. Hierarchy communication cost (HXC)

The RDP_SCHEDULER is the single class of the application hierarchy loaded on RDPPs where a single communication object (rdp_scheduler) is created, thus:

$$HXC_{APPLICATION} = CXC_{RDP\_SCHEDULER}$$

4. Processor communication cost (PXC)

The application hierarchy is the single hierarchy loaded on RDPPs, thus:

$$RDPPXC = \frac{176N_t}{b}\left(1 + \frac{2.25}{2^d - 1}\right) +$$

$$\left(\frac{d2^d}{2(2^d-1)} - 1\right)\left(\frac{396N_t}{b(2^d-1)} + \frac{176N_t}{b}\right)$$

where:

$((d2^d/(2(2^d-1))) - 1)$ is the average number of retransmission nodes evaluated as the average distance minus one;

$(176N_t/b)$ is the cost to route the buffer of detections; and

($396N_t/(b(2^d-1))$) is the cost to return the buffer of estimation reports.

Assuming $2^d \approx (2^d-1)$, RDPPXC can be rewritten as:

$$RDPPXC = \frac{88dN_t}{b}(1+\frac{2.25}{2^d}) \qquad \textbf{(B.4)}$$

b. Communication Cost: $(1-C_p-T_p)$-Tree (SFSP)

In this option, correlation hierarchies are assigned to CPs and tracking hierarchies are assigned to TPs. The buffer of detection (DEBU) is transmitted to CPs by the IP. Input tracking reports (TRINP) are transmitted from CPs to TPs. Output tracking updates (TROUT) are transmitted from TPs to CPs and the buffer of estimation reports (ESBU) is transmitted from CPs to the IP. The following communication objects were identified:

1. Interface_scheduler running in the IP;
2. Correlation_scheduler running in CPs; and
3. Tracking_scheduler running in TPs.

The communication cost components were identified as:

*1. **Service request cost (SRC):*** Overhead in the IP to transmit and in CPs to receive the buffer of detections; and Overhead in CPs to transmit and in TPs to receive tracking reports;

*2. **Result cost (RC):*** Overhead in TPs to transmit and in CPs to receive tracking updates; and Overhead in CPs to transmit and in the IP to receive the buffer of estimation reports;

*3. **Retransmit cost*** *(C_{RT})*: No retransmission overhead is needed.

Table 24 depicts the required packet sizes (detection, tracking reports, tracking updates, and estimation reports).

TABLE 24 PACKET SIZES (1-C_T-T_T)-TREE

Class	Bytes (Info)	Bytes (Overhead)	Bytes (Total)
DETECTION	16	6	22
TRREP	52	19.5	71.5
TRUPD	68	25.5	93.5
ESTREPORT	36	13.5	49.5

Applying the proposed methodology (Chapter V, Section B1h), we have:

(1) IP Communication Cost. The communication cost of the IP may be evaluated using the following sequence:

*1. **Object communication cost** (OXC)*

$$OXC_{interface\_scheduler} = \sum_{k=1}^{C_p} c_{interface\_scheduler,k}$$

where:

k-objects are correlation_schedulers running in CPs.

Assuming parallel transmission in C_p-links of N_t/C_p detections in each link and parallel reception in C_p-links of N_t/C_p estimation reports, we have:

$$OXC_{interface\_scheduler} = SRC + RC$$

$$SRC = (N_t/C_p)\,(targets)*22\,(bytes/target)*(8/b)\,(s/byte)$$

$$RC = (N_t/C_p)\,(targets)*49.5\,(bytes/target)*(8/b)\,(s/byte)$$

$$OXC_{interface\_scheduler} = \frac{572N_t}{bC_p}$$

2. Class communication cost (CXC)

The interface_scheduler is the single instance of the class INTERFACE_SCHEDULER, thus:

$$CXC_{INTERFACE\_SCHEDULER} = OXC_{interface\_scheduler}$$

3. Hierarchy communication cost (HXC)

The INTERFACE_SCHEDULER is the single class of the interface hierarchy loaded on the interface processor where a single communication object (interface_scheduler) is created, thus:

$$HXC_{INTERFACE} = CXC_{INTERFACE\_SCHEDULER}$$

4. Processor communication cost (PXC)

The interface hierarchy is the single hierarchy loaded on the interface processor, thus:

$$IPXC = \frac{572N_t}{bC_p} \qquad \textbf{(B.5)}$$

The retransmit cost (C_{RT}) of the IP is zero.

(2) TP Communication Cost. The communication cost of TPs may be evaluated using the following sequence:

1. Object communication cost (OXC)

$$OXC_{tracking\_scheduler} = \sum_{k=1}^{1} C_{tracking\_scheduler.k}$$

The single k-object is the correlation_scheduler running in the CP of the same tree branch (parent of TP).

Each TP receives ($N_t/(C_pT_p)$) tracking reports and transmits ($N_t/(C_pT_p)$) tracking updates (absence of load imbalance), therefore:

$$OXC_{tracking\_scheduler} = SRC + RC$$

$$SRC = N_t/(C_pT_p)\,(targets)*71.5\,(bytes/target)*(8/b)\,(s/byte)$$

$$RC = N_t/(C_pT_p)\,(targets)*93.5\,(bytes/target)*(8/b)\,(s/byte)$$

$$OXC_{tracking\_scheduler} = \frac{1320N_t}{bC_pT_p}$$

2. Class communication cost (CXC)

The tracking_scheduler is the single instance of the class TRACKING_SCHEDULER, thus:

$$CXC_{TRACKING\_SCHEDULER} = OXC_{tracking\_scheduler}$$

3. Hierarchy communication cost (HXC)

The TRACKING_SCHEDULER is the single class of the tracking hierarchy loaded on TPs where a single communication object (tracking_scheduler) is created, thus:

$$HXC_{TRACKING} = CXC_{TRACKING\_SCHEDULER}$$

4. Processor communication cost (PXC)

The tracking hierarchy is the single hierarchy loaded on TPs, therefore:

$$TPXC = \frac{1320N_t}{bC_pT_p} \tag{B.6}$$

The retransmit cost (C_{RT}) of the TP is zero.

(3) CP Communication cost. The communication cost of CPs may be evaluated using the following sequence:

1. Object communication cost (OXC)

$$OXC_{correlation\_scheduler} = \sum_{k=1}^{(T_p+1)} C_{correlation\_scheduler,k}$$

Where:

k-objects are tracking_schedulers (T_p) and the interface_scheduler.

Each CP receives (N_t/C_p) detections and ($N_t/(C_pT_p)$) tracking updates and transmits (N_t/C_p) estimation reports and ($N_t/(C_pT_p)$) tracking reports (absence of load imbalance), thus:

$$OXC_{correlation\_scheduler} = SRC + RC$$

$$SRC = \frac{176N_t}{bC_p} + \frac{572N_t}{bC_pT_p}$$

$$RC = \frac{396N_t}{bC_p} + \frac{748N_t}{bC_pT_p}$$

$$OXC_{correlation\_scheduler} = \frac{572N_t}{bC_p}(1+\frac{2.31}{T_p})$$

2. Class communication cost (CXC)

The correlation_scheduler is the single instance of the class CORRELATION_SCHEDULER, thus:

$$CXC_{CORRELATION\_SCHEDULER} = OXC_{correlation\_scheduler}$$

3. Hierarchy communication cost (HXC)

The CORRELATION_SCHEDULER is the single class of the correlation hierarchy loaded on CPs where a single communication object (correlation_scheduler) is created, thus:

$$HXC_{CORRELATION} = CXC_{CORRELATION\_SCHEDULER}$$

4. Processor communication cost (PXC)

The correlation hierarchy is the single hierarchy loaded on CPs, thus:

$$CPXC = \frac{572N_t}{bC_p}(1+\frac{2.31}{T_p}) \qquad \textbf{(B.7)}$$

The retransmit cost (C_{RT}) of the CP is zero.

c. Comparative Analysis

Table 25 depicts a comparative analysis of communication costs (N_t=500, b=10Mbits/s) (see Equations B.3 - B.7). As expected, in this application the communication cost of the d-cube architecture is greater than the cost of the (1-C_p-T_p)-tree architecture. This is mainly because:

1. The interconnection network of objects is also a tree with communicating objects assigned to neighboring processors; and

2. The search space was divided into correlation sections reducing the cost to transmit the buffer of detections.

TABLE 25 COMPARATIVE ANALYSIS OF COMMUNICATION COSTS (MS)

Topology	IPXC	RDPPXC	CPXC	TPXC
6-cube	12.1	31.3	-	-
7-cube	11.6	27.3	-	-
(1-8-8)-tree	3.6	-	4.6	1.0
(1-9-9)-tree	3.2	-	4.0	0.8

8. Evaluation of the Computation Cost

A strict analysis of the computation cost requires the definition of the processor being used. However, to keep our evaluation independent of any particular processor we will use the following assumptions:

1. Absence of load imbalance (i.e., there is a uniform distribution of targets among nodes);

2. Tracking (method EXTR) and correlation (first gate: method RECOPRFG; second gate: RECOPRSG) are the main computation costs. That is, in steady state initiation and termination algorithms are not executed.

3. The computation cost is proportional to the number of firm targets. That is, tracking and correlation are algorithms executed for each firm target;

4. The average correlation time per target (C_1) can be expressed as a fraction of the average tracking time per

target (T_1). That is, $C_1 = rT_1$ ($r < 1$) (tracking was identified as our high cost function); and

5. The average first gate correlation time per target (FG_1) can be expressed as a fraction of the average correlation time per target (C_1). $FG_1 = fC_1$ ($f < 1$) (the first gate correlation phase does not require ambiguity resolution).

a. Computation Cost: d-Cube (AFAP)

In this option the application hierarchy is assigned to RDPPs. The RDPP computation cost (RDPPCC) can be evaluated by:

$$RDPPCC = W_{rdpp}(C_1+T_1) = (1+r)\,W_{rdpp}T_1 \qquad \text{(B.8)}$$

W_{rdpp} is the workload of RDPPs (i.e., the number of targets loaded on each RDPP) and can be evaluated by:

$$W_{rdpp} = \frac{N_t}{2^d-1}$$

When all functions are loaded on all processors we have no overlap between correlation and tracking operations.

b. Computation Cost: $(1-C_p-T_p)$-Tree (SFSP)

In this option the correlation hierarchy is assigned to CPs and the tracking hierarchy is assigned to TPs. The CP and TP computation costs (CPCC, and TPCC) can be evaluated by:

$$CPCC = W_{cp}C_1 = W_{cp}IT_1$$

$$TPCC = W_{tp}T_1 \tag{B.9}$$

where:

W_{cp} is the worklcad of CPs; and

W_{tp} is the workload of TP_s.

When correlation algorithms are loaded on CPs and tracking algorithms are loaded on TPs we may overlap correlation and tracking operations. The overlapped (CP:TP) computation cost (CPTPCC) can be evaluated by:

$$CPTPCC = fC_1T_p+W_{tp}T_1 \tag{B.10}$$

Tracking can start after the correlation phase. Also, $W_{tp}T_1 >> fC_1T_p$ (i.e., the tracking cost of all targets loaded on TPs is very high when compared with the first gate correlation cost of T_p targets). Therefore, we can rewrite Equation B.10 as:

$$CPTPCC \approx W_{tp}T_1 \tag{B.11}$$

The workload of TPs can be evaluated by:

$$W_{tp} = \frac{N_t}{C_pT_p}$$

9. Analytical Comparisons Among Options

As introduced in Chapter V (Section A), the expected efficiency of two proposals can be compared by evaluating the time to compute on N nodes ($T_{conc}(N)$) when:

1. The time to compute on one node is the same. That is, we must develop the application using the same software design and the same resources; and

2. The number of nodes N is the same. That is, to compare the efficiency of the d-cube (AFAP) proposal with the efficiency of the $(1-C_p-T_p)$-tree proposal we must have:

$$(2^d-1) = C_p(T_p+1) \tag{B.12}$$

In the d-cube (AFAP) proposal the time to compute on N nodes ($(T_{conc}(N))_{cube}$) can be evaluated by:

$$(T_{conc}(N))_{cube} = RDPPXC+RDPPCC \tag{B.13}$$

where:

RDPPXC is the communication cost of RDPPs (see Equation B.4); and

RDPPCC is the computation cost of RDPPs (see Equation B.8).

In the $(1-C_p-T_p)$-tree proposal the time to compute on N nodes ($(T_{conc}(N))_{tree}$) can be evaluated by:

$$(T_{conc}(N))_{tree} = TPXC + CPXC + CPTPCC \qquad \text{(B.14)}$$

where:

TPXC is the communication cost of TPs (see Equation B.6);

CPXC is the communication cost of CPs (see Equation B.7); and

CPTPCC is the overlapped CP:TP computation cost (see Equation B.11).

In Equation B.14 we assumed no communication overlap between CPs and TPs (worst case condition).

The efficiency of the $(1-C_p-T_p)$-tree proposal is expected to be greater than the efficiency of the d-cube when $(T_{conc}(N))_{tree} < (T_{conc}(N))_{cube}$. That is:

$$TPXC + CPXC + CPTPCC < RDPPXC + RDPPCC \qquad \text{(B.15)}$$

When $r < (1/T_p)$ Equation B.15 can be rewritten as:

$$\Delta_{com} \geq \left(\frac{1 - rT_p}{T_p(T_p+1)}\right) \frac{N_t T_1}{C_p} \quad (r>0) \qquad \text{(B.16)}$$

$$\Delta_{com} \geq \frac{1}{T_p(T_p+1)} \frac{N_t T_1}{C_p} \quad (r=0) \qquad \text{(B.17)}$$

RDPPCC was rewritten using the condition imposed by Equation B.12, Δ_{com} is the communication cost of the d-cube option minus the communication cost of the $(1-C_p-T_p)$-tree option.

LIST OF REFERENCES

[1] Langston M. "Computers in Combat Systems", EASCON '82, pp 415-418, 1982.

[2] Farina A., Studer F.A. "Radar Data Processing, Vol. I - Introduction and Tracking", RSP Press ISBN 0-86380-026-2, 1985.

[3] Farina A., Studer F.A. "Radar Data Processing, Vol. II - Advanced Topic and Applications", RSP Press ISBN 0-86380-038-6, 1985.

[4] Zitzman, Lewis H., Falatko, Stephen M. "Computer System Architecture Concept for Future Combat Systems", Naval Engineers Journal, pp 43-62, May 1990.

[5] Baillie C.F., Gottschalk T.D. and Kolawa A. "Comparisons of Concurrent Tracking on various Hypercubes", The third conference on hypercube concurrent computers and applications, ACM Press ISBN 0-89791-278-0, Vol. I, pp 155-166, January/1988.

[6] Gottschalk T.D. "Concurrent Multiple Target Tracking", The third conference on hypercube concurrent computers and applications, ACM Press ISBN 0-89791-278-0, Vol. II, pp 1247-1268, January/1988.

[7] Bar-Shalom Y. and Fortmann T.E. "Tracking and Data Association", Academic Press, 1988.

[8] Vilhena R. "Comparação de Filtros para Processamento de Medições Sonar", COPPE,M.Sc., Engenharia de Sistemas e Computação, s/ data.

[9] Vilhena R. "Introdução aos Algoritmos para Processamento de Marcações e Distâncias", Escola Naval - Notas de Aula - Automação de Sistemas Navais, s/ data.

[10] Averbuch A., Itzikowitz S., and Kapon T. "Parallel Implementation of Multiple Model Tracking Algorithms", IEEE Transactions on Parallel and Distributed Systems, pp 242-251, April 1991.

[11] Guenter B. "Automatic Track Initiation with Phased Array Radar", IEEE International Radar Conference, pp 423-428, April 1975.

[12] Uhlmann, Jeffrey k. "Algorithms for Multiple-Target Tracking", American Scientist, Vol 80, pp 128-141, March-April 1992.

[13] Nelson M. L. "An Introduction to Object-Oriented Programming", NPS Report No. NPS52-90-024, Apr 1990.

[14] Stefik M. and Bobrow D. G. "Object-Oriented Programming: Themes and Variations", The AI Magazine, Vol 6, No 4, pp 40-62, winter 1986.

[15] Wegner P. "Dimensions of Object-Based Language Design", OOPSLA'87 Proceedings, October 1987; special issue of of SIGPLAN Notices, Vol 22, No 12, pp 168-182, December 1987.

[16] Nelson M. L. "Concurrency & Object-Oriented Programming", SIGPLAN Notices, Vol 26, No 10, pp 63-72, October 1991.

[17] Wirfs B. R. and Wilkerson B. "Object-Oriented Design: A Responsibility-Driven Approach", OOPSLA '89 Proceedings, pp 71-75, October 1989.

[18] de Paula E. G. and Nelson M. L. "An Object-Oriented Design Methodology", NPS, Report No. NPSCS-91-007, January 1991.

[19] Winblad A. L., Edwards S. D. and King D. R. "Object-Oriented Software", Addison-Wesley Publishing Co, Reading, Mass, 1990.

[20] Coffman E. G. et al. "Computer and Job-Shop Scheduling Theory", Wiley-Interscience, New York, 1976.

[21] Garey M. R. and Johnson D. S. "Computers and Intractability: a Guide to the Theory of NP-completeness", Bell Laboratories Tech. Rep., Murray Hill, NJ.

[22] Miklosko J. and Kotov V. E. "Algorithms, Software and Hardware of Parallel Computers", VEDA, Publishing House of the Slovak Academy of Sciences, Bratislava, 1984.

[23] Shirazi B., Wang M. and Pathak G. "Analysis and Evaluation of Heuristic Methods for Static Task Scheduling", Journal of Parallel and Distributed Computing, Academic Press, Vol 10, No 3, pp 222-232, November 1990.

[24] Chen W. K. and Gehringer E. F. "A Graph-Oriented Mapping Strategy for a Hypercube", The 3rd Conference on Hypercube Concurrent Computers and Applications, ACM Press ISBN 0-89791-278-0, Vol I, pp 200-209, January 1988.

[25] Ercal F., Ramanujam J. and Sadayappan P. "Task Allocation onto a Hypercube by Recursive Mincut Bipartitioning", The 3rd Conference on Hypercube Concurrent Computers and Applications, ACM Press ISBN 0-89791-278-0, Vol I, pp 210-221, January 1988.

[26] Covington R. C. et al. "The Rice Parallel Processing Testbed", Proc. 1988 Sigmetrics Conf. on Measurement and Modeling Computer Systems, Santa Fe, NM, p. 4, May 1988.

[27] Berman F. "Experience with an automatic solution to the mapping problem" in The Characteristics of Parallel Algorithms, MIT Press 1987.

[28] Fox G. et al. "Solving Problems on Concurrent Processors", Prentice Hall, N.J., ISBN 0-13-823022-6, Vol I, 1988.

[29] Bertsekas D. P., Tsitsiklis J. N. "Parallel and Distributed Computation", Prentice Hall, N.J., ISBN 0-13-648700-9, 1989.

[30] Willebeek-LeMair M. and Reeves P. A., "Region Growing on a Hypercube Multiprocessor", The 3rd Conference on Hypercube Concurrent Computers and Applications, ACM Press ISBN 0-89791-278-0, Vol II, pp 1033-1042, January 1988.

[31] Willebeek-LeMair M. and Reeves P. A., "A Distributed Dynamic Load Balancing Strategy for Highly Parallel Multicomputer Systems", Proceedings of the Fourth Siam Conference, Siam Proceedings Series List, ISBN 0-89871-262-9, pp 351-356, December 1989.

[32] Salmon J. "A Mathematical Analysis of the Scattered Decomposition", The 3rd Conference on Hypercube Concurrent Computers and Applications, ACM Press ISBN 0-89791-278-0, Vol I, pp 239-240, January 1988.

[33] Hennessy J. L. and Patterson D. A. "Computer Architecture a Quantitative Approach", Morgan Publishers, ISBN 1-55860-069-8, 1990.

[34] Gordon J. M. and Stout Q. F. "Hypercube Message Routing in the Presence of Faults", The 3rd Conference on Hypercube Concurrent Computers and Applications, ACM Press ISBN 0-89791-278-0, Vol I, pp 318-327, January 1988.

[35] Chen M. S. and Shin K. G. "Message Routing in an Injured Hypercube", The 3rd Conference on Hypercube Concurrent Computers and Applications, ACM Press ISBN 0-89791-278-0, Vol I, pp 312-317, January 1988.

[36] Kandlur D. D. and Shin K. G. "Hypercube Management in the Presence of Node Failures", The 3rd Conference on Hypercube Concurrent Computers and Applications, ACM Press ISBN 0-89791-278-0, Vol I, pp 328-336, January 1988.

[37] Nelson M. L. "Concurrent and Distributed Object-Oriented Languages: Promises and Pitfalls", NATO Workshop on Object-Oriented Modelling of Distributed Systems, May 1992.

[38] "The T9000 Transputer", Inmos Databook Series, 1st Edition, 1991.

[39] "Classic-Ada Users Manual", Software Productivity Solutions, Inc., Indiatlantic, FL, 1989.

[40] Nelson M. L. and Mota G. F. "Object-Oriented Programming in Classic-Ada", Ada Letters, Vol XII, No 2, pp 102-110, Mar/Apr 1992.

[41] Nelson M. L., Mota G. F. and Theologitis V. " Concurrent Object-Oriented Programming in Classic-Ada", Draft, Apr 1992.

[42] Stroustrup B. "The C++ Programming Language", Addison-Wesley, Reading, MA, 1986.

[43] "Logical Systems C Users Manual", Logical Systems, Corvallis, OR, 1989.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Chairman, Code CS Computer Science Department Naval Postgraduate School Monterey, CA 93943-5101	1
4.	Professor Uno R. Kodres, Code CSKr Computer Science Department Naval Postgraduate School Monterey, CA 93943	1
5.	Professor Michael L. Nelson, Code CSNe Computer Science Department Naval Postgraduate School Monterey, CA 93943	1
6.	Professor Thomas C. Wu, Code CSWq Computer Science Department Naval Postgraduate School Monterey, CA 93943	1
7.	Professor Shridhar Shukla, Code EC Electrical and Computer Engineering Department Naval Postgraduate School Monterey, CA 93943	1
8.	Professor Arthur L. Schoenstadt, Code MA Mathematics Department Naval Postgraduate School Monterey, CA 93943	1
9.	Mr. Grosche Jurgen Research Institute for Mathematics and Electronics (FFM) Neuenahrer St. 20 D-5307 Wachtberg Germany	1

10. Instituto de Pesquisas da Marinha 1
Brazilian Naval Commission
4706 Wisconsin Ave.,N.W.
Washington, D.C. 20016

11. Centro de Analise de Sistemas Navais 1
Brazilian Naval Commission
4706 Wisconsin Ave.,N.W.
Washington, D.C. 20016

12. Centro de Apoio aos Sistemas Operativos 1
Brazilian Naval Commission
4706 Wisconsin Ave.,N.W.
Washington, D.C. 20016

13. CDR Gilberto F. Mota 1
Brazilian Naval Commission
4706 Wisconsin Ave.,N.W.
Washington, D.C. 20016