Draco Domain Analysis for a Real Time Application: The Analysis

by
Sigmund Sundfor
June 1983

Reuse Project RTP 015

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92717

This research was supported by NSF grant MCS-81-03718 with the cooperation of the Air Force Office of Scientific Research, and grants from A/S Kongsberg Vapenfabrikk and Royal Norwegian Council for Scientific and Industrial Research.

Table of Contents

List of Figures

1. Acknowledgments

I would like to thank everybody who has helped me with this work. In particular, I would like to thank Peter Freeman who invited me to come to Irvine and suggested this work. He has guided and encouraged me along the way and reviewed the work as it has progressed. Jim Neighbors whose thesis is the basis for this work, has patiently explained the principles of Draco, helped sort out my problems with the mechanism and given me a lot of useful suggestions. Ira Baxter has helped me evaluate several of the ideas. Tom Marlin and Tore Aavatsmark helped review part of the SADT models. Martin Katz has helped me out of the numerous "fights" I have had with the computer system and the tools.

2. Introduction

This domain analysis has been done to gain more experience in applying the concepts proposed by James M. Neighbors [Neighbors 80]. In particular, the intention has been to try out these principles on real-time, embedded systems [Freeman 82]. This report presents the actual domain analysis itself. The results are discussed in a second paper [Sundfor 83].

The Draco system was developed to demonstrate the ideas and principles developed by James M. Neighbors. It has been used to develop some systems by Dr. Neighbors himself. In addition there has been one more domain analysis done [Gonzales 81].

Draco is an experimental system. It does not have all the user friendliness that one would expect of a production system. However, it does work sufficiently well that a newcomer (the author) could use it to define the domain language resulting from the analysis.

The researcher doing this domain analysis, comes from industry with several years of experience in developing software for real-time, embedded systems. This includes the type of system chosen for this analysis, namely ship borne gun control systems. The domain chosen is a subsystem of this, namely the tactical plot.

The domain analysis done includes the definition of the domain language syntax and a prettyprinter to print out the internal form generated by the parser. In addition, a few sample specifications of systems have been made using the domain language. The work does not include generating actual systems. The intention has only been to do the analysis itself. To generate actual systems from specifications in this domain language, transformations and refinements will have to be defined. (This is the domain design part of using Draco.) In addition, some underlying domains that has been assumed, will have to be developed.

This paper assumes that the reader is familiar with Draco.

3. The Application Area - Ship Borne Gun Control Systems

Originally the intention was to do the analysis on real-time embedded systems in general. There are some common characteristics of such systems that may be well worth analyzing. There is the real-time constraint, the "embeddedness" which generally involves an intimate interaction with special purpose hardware. In addition, many of them are what Belady and Lehman describe as large systems [Belady & Lehman 79]. Some of these characteristics are discussed in the other report on this research. The intention of the domain analysis is to develop a domain description captured in the form of a domain language. This domain language will serve as a tool for specifying particular instances (systems) within that domain. The domain must therefore be a concrete one and specific enough so that we can actually have a specification language.

Based on this reasoning, it was found that real-time, embedded systems was too general to be regarded as one domain. Instead it was decided to analyze a particular type of system, namely ship borne gun control systems. They are certainly real-time, and are embedded in the total gun control systems. It is believed that the results on how well Draco works for this domain will be applicable to other real-time, embedded systems.

Ship borne gun control systems use data from several sensors, interact intimately with the operator(s) and control the gun(s). Basically the sensors observe different states of the world. The system uses these to develop a "world model". Radar and other electronic and optical sensors measures positions of targets. The computer system uses these to calculate the targets' motions. If the operator evaluates a target as constituting a threat to the ship, the target data are combined with the ballistics (formulas for the projectile trajectory) to calculate an aim point. In addition, the gun control must take into consideration the motion of the ship itself.

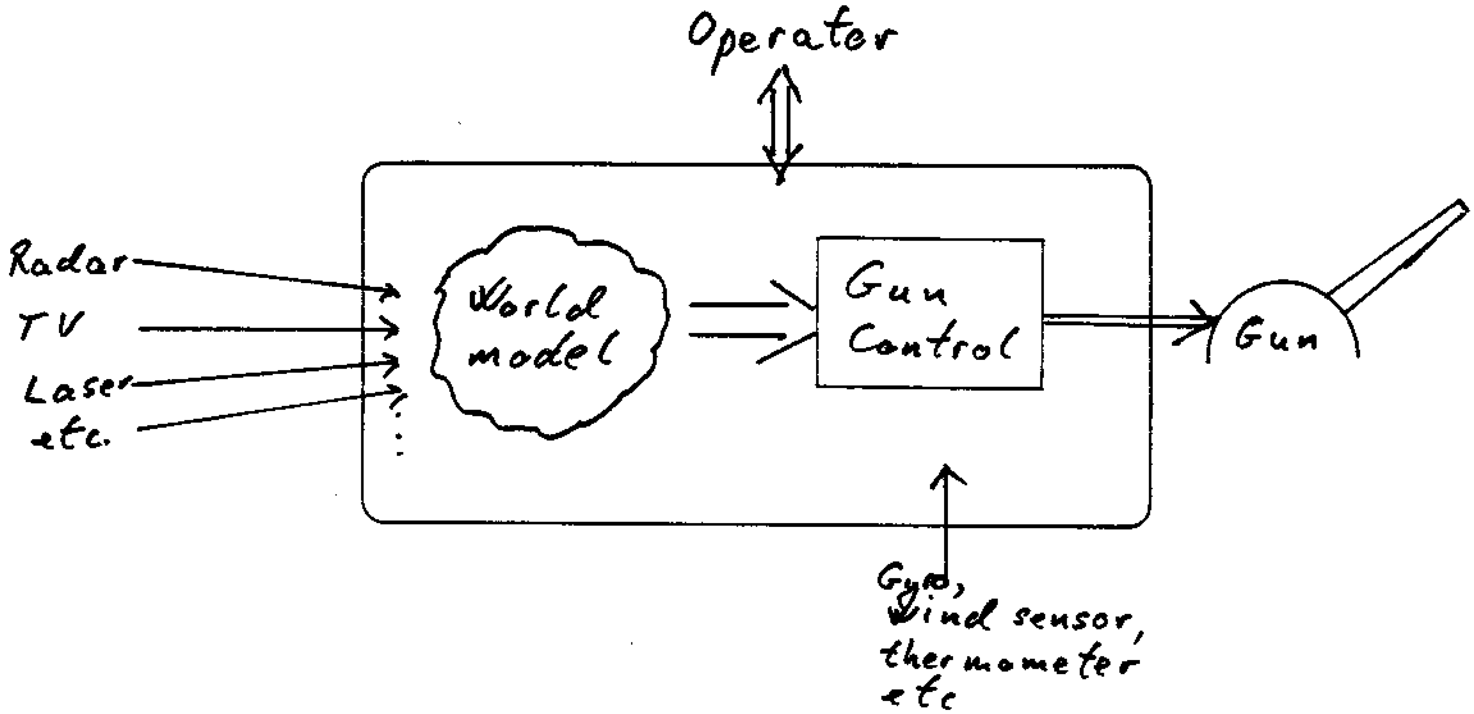


Figure 3-1: Ship borne gun control systems

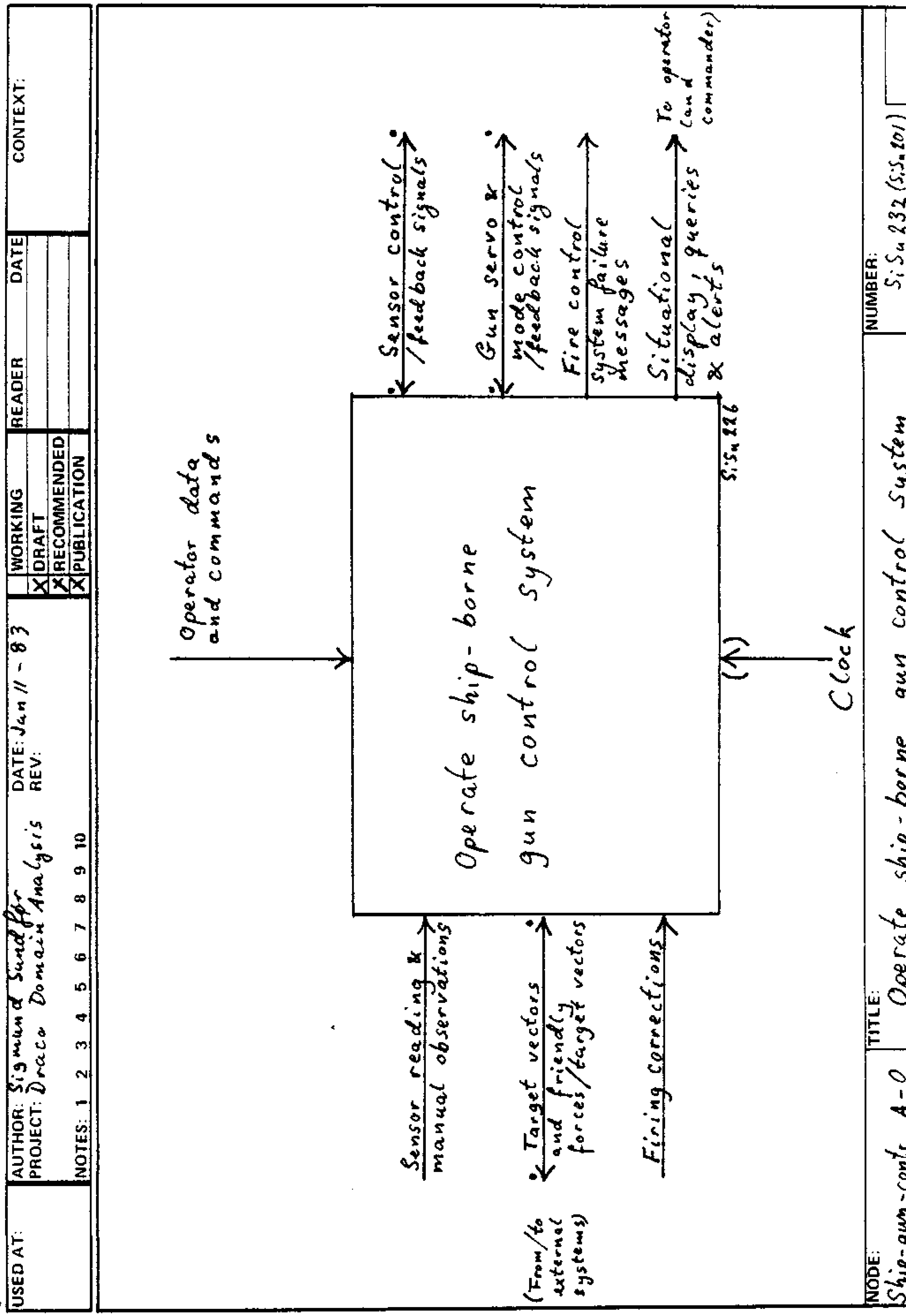
3.1 Model of the Ship Borne Gun Control System Domain

The ship borne gun control system domain has been analysed using SADT™ [Ross 77][1]. The appendix includes a brief guide on how to read SADT diagrams.

The model has the following diagrams:

- A-0: Operate ship-borne gun control system
- FEO -0: Operate ship-borne gun control system (viewpoint, purpose, type of model and a glossary)
- A 0: Operate ship-borne gun control system
- FEO 0,1:Operate ship-borne gun control system (informal description)
- FEO 0,2:Operate ship-borne gun control system (main dataflows)
- A 1: Maintain world model
- A 2: Engage guns on target
- A 3: Display situation
- A 11: Read and control sensors
- A 13: Determine target vectors
- A 32: Generate tactical display picture
- A 321: Generate graphical representation

[1] SADT is a Trademark of SofTech, Inc.


USED AT:
AUTHOR: Sigmund Sundfør
PROJECT: Draco Domain Analysis
DATE: Jan 11 - 83
REV:
NOTES: 1 2 3 4 5 6 7 8 9 10
WORKING
DRAFT
RECOMMENDED
PUBLICATION
READER
DATE
CONTEXT:
SADT™ DIAGRAM FORM ST008 9/75
Form © 1975 SofTech, Inc., 460 Totten Pond Road, Waltham, Mass. 02154, USA
Operator data and commands
Sensor reading & manual observations
Target vectors and friendly forces/target vectors
Firing corrections
(From/to external systems)
Operate ship-borne gun control System
Si Su 226
Sensor control /feedback signals
Gun servo & mode control /feedback signals
Fire control system failure messages
Situational display, queries & alerts
To operator (and commander)
Clock
NODE: Ship-gun-contr A-0
TITLE: Operate ship-borne gun control System
NUMBER: Si Su 232 (Si Su 201)

USED AT:	AUTHOR: Sigmund Sundfor PROJECT: Draco Domain Analysis NOTES: 1 2 3 4 5 6 7 8 9 10	DATE: oct 22-82 REV:	X WORKING X DRAFT X RECOMMENDED X PUBLICATION	READER	DATE	CONTEXT:

Viewpoint: Implementor of software, in the requirement analysis phase.

Purpose: To define the functions (as seen by the user) that the software has to supply. This is to be the first step in a Draco Domain analysis.

Type of model: Model of new syystems, but based on functions from present systems that I know of. The model aims at covering a set of possible systems, i.e. a domain, rather than a particular system.

Note: The model does not include simulation and operator training functions. These are frequently buildt into the same systems. However, they do not realy come under "operate".

Glossary:

Ballistic data - parameters that are used together with the ballistic model to determine the projectile's trajectory. Among these are air pressure, wind, type of ammunition and muzzle velocity.

Sensor - a device that measure real world states. Here some of them are radars, TV trackers, laser rangefinder, gyro, log, thermometer, barometer.

Ship's attitude - ship's roll, pitch and course. This is used to calculate the coordinate transformations between the stationary coordinate reference system and the ship's deck plane.

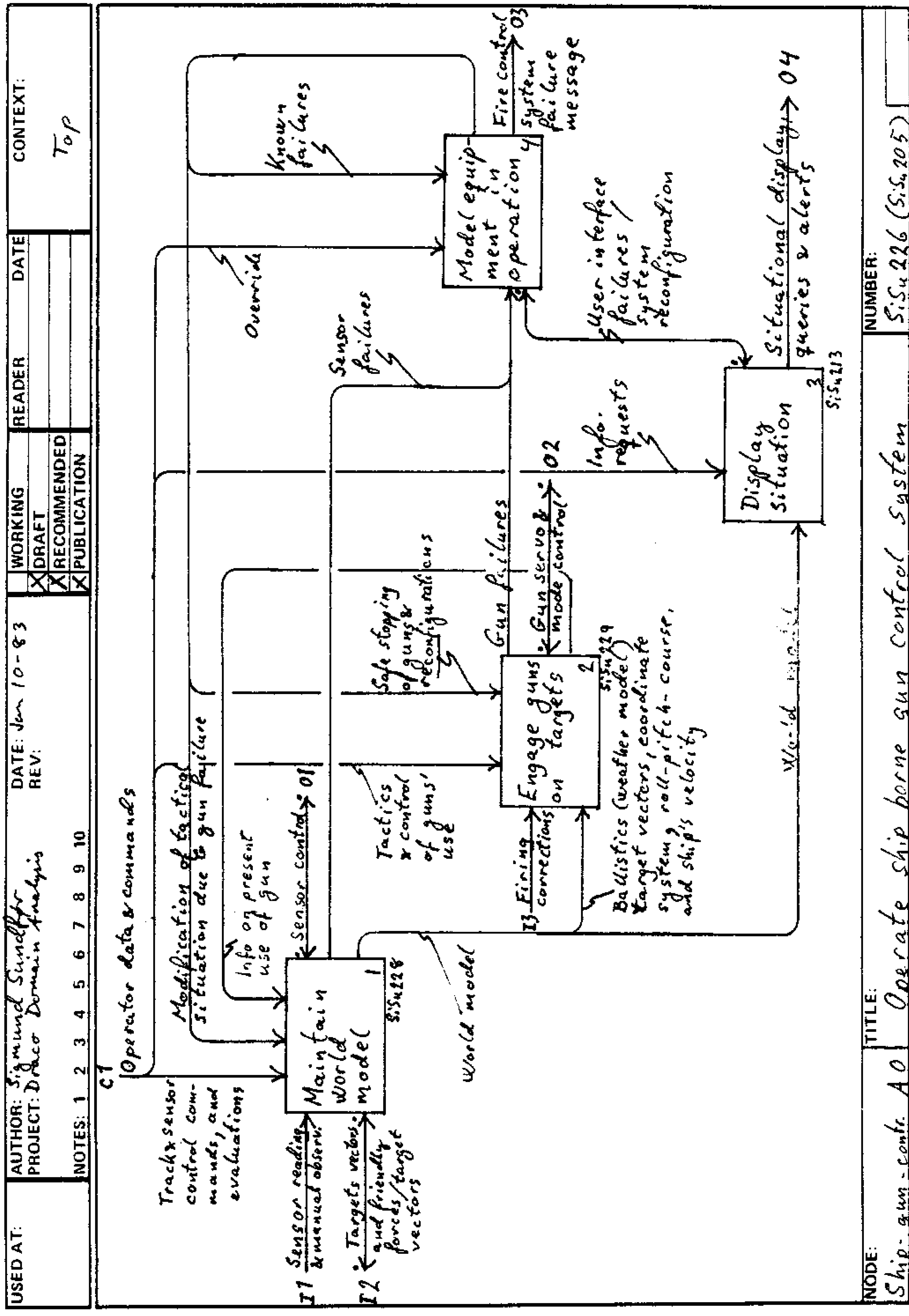
Ship's vector - this is the ship's velocity and position in the coordinate reference system.

Tactics - predetermined reactions to a particular input; e.g. "if it moves, then shoot".

Target - any vehicle; ship, aircraft or submarine, that one wants to gather information on.

Target vector - a description of a target and its state, e.g. hostile, aircraft, identification, position, velocity .

NODE:	TITLE:	NUMBER:
Ship-gun-contr. FEO-0	Operate ship-borne gun control system	S;S4 217 (S;S4 202,3,4)


SADT DIAGRAM FORM SI 098 9/75
Form © 1975 SofTech, Inc., 460 Totten Pond Road, Waltham, Mass. 02154, USA
USED AT:
AUTHOR: Sigmund Sundfør
PROJECT: Draco Domain Analysis
DATE: Jan 10-83
REV:
NOTES: 1 2 3 4 5 6 7 8 9 10
WORKING
X DRAFT
X RECOMMENDED
X PUBLICATION
READER
DATE
CONTEXT: Top
C1 Operator data & commands
I1 Sensor reading & manual observ.
I2 Targets vectors and friendly forces/target vectors
Track & sensor control commands, and evaluations
Maintain World model 1
SiSu228
Modification of tactics situation due to gun failure
Info on present use of gun
Sensor control 01
Tactics & control of guns' use
World model
I3 Firing corrections
Engage guns on targets 2
SiSu229
Safe stopping of guns & reconfigurations
Ballistics (weather model) target vectors, coordinate system, roll-pitch-course, and ship's velocity
Gun failures
Gun servo & mode control 02
Sensor failures
Override
Known failures
Model equipment in operation 4
Fire control 03
system failure message
User interface failures / system reconfiguration
Info. requests
Display situation 3
SiSu213
Situational display 04
queries & alerts
World model
NODE: Ship-gun-contr. A0
TITLE: Operate ship borne gun control System
NUMBER: SiSu226 (SiSu205)

USED AT:	AUTHOR: Sigmund Sundfør PROJECT: Draco Domain Analysis NOTES: 1 2 3 4 5 6 7 8 9 10	DATE: Oct 22-82 REV:	X WORKING X DRAFT X RECOMMENDED X PUBLICATION	READER DATE	CONTEXT: Top

Oct 22 15:32 1982 sgc.feo0 Page 1

Informal description (and incomplete):

Basicly there are 2 functions that are dominant:
   1: The system must determine what goes on around the ship, and then,
      if the user of the system decides it is neccesary:
   2: - the gun must be used to defend the ship.

To find out what goes on, the ship is equipped with sensors. These normally include radars and optical sensors. These measure direction to, and (some of them) distance to other ships, aircrafts and submarines. By repeated measurements, the syystem can estimate position and velocity using some optimal filtering technique.

This is presented to the operator together with maps and other information he/she needs to evaluate the situation.

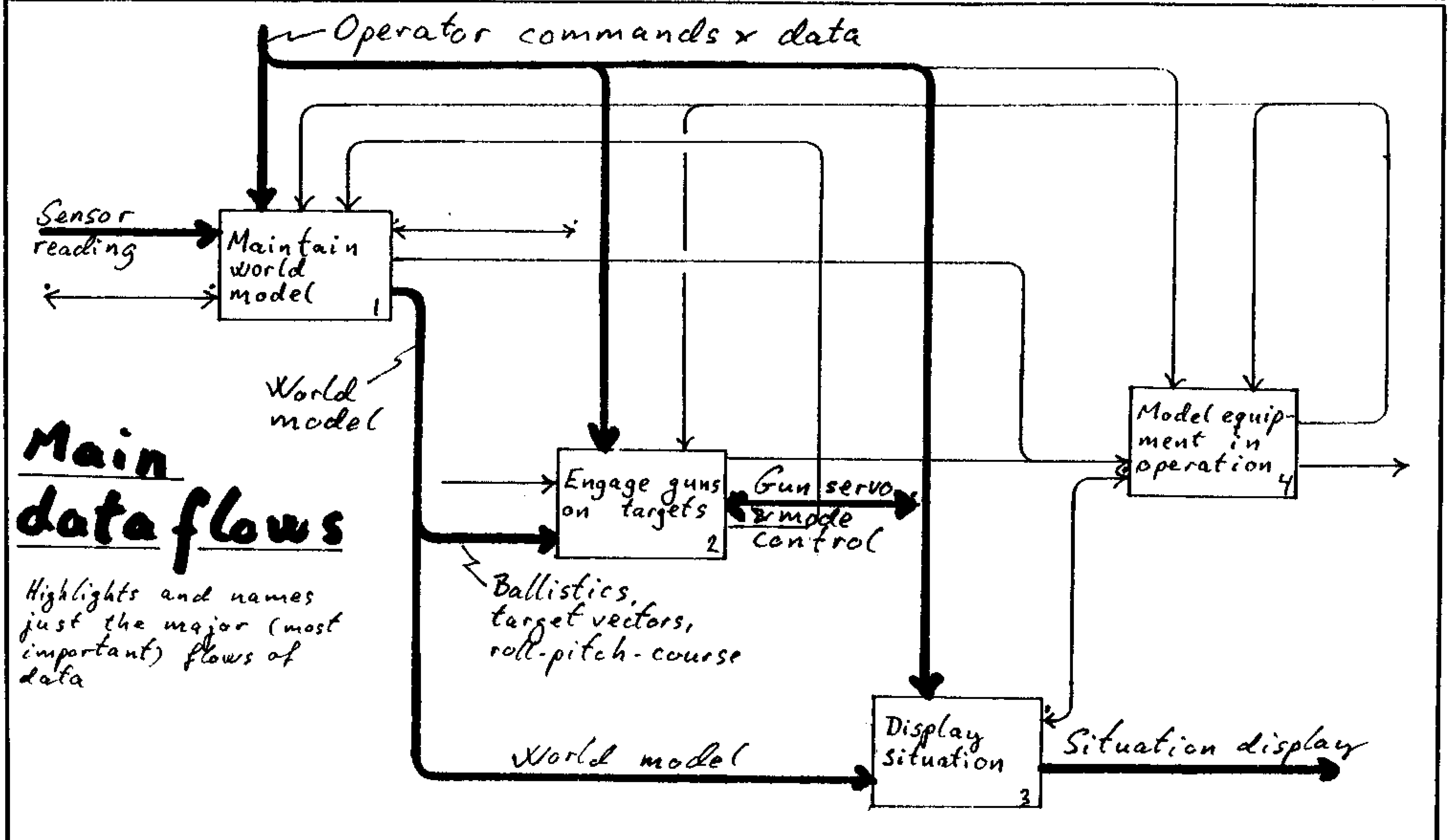
If a threat appears to exist, the operator (or commander) may decide to use the guns. Based upon the target position and velocity together with the ballistic model, the system must predict where it should aim to hit.

All these calculations are carried out in a stationary coordinate system. The ship moves, so finally the data for aiming the guns must be transformed to the ship's reference plane.

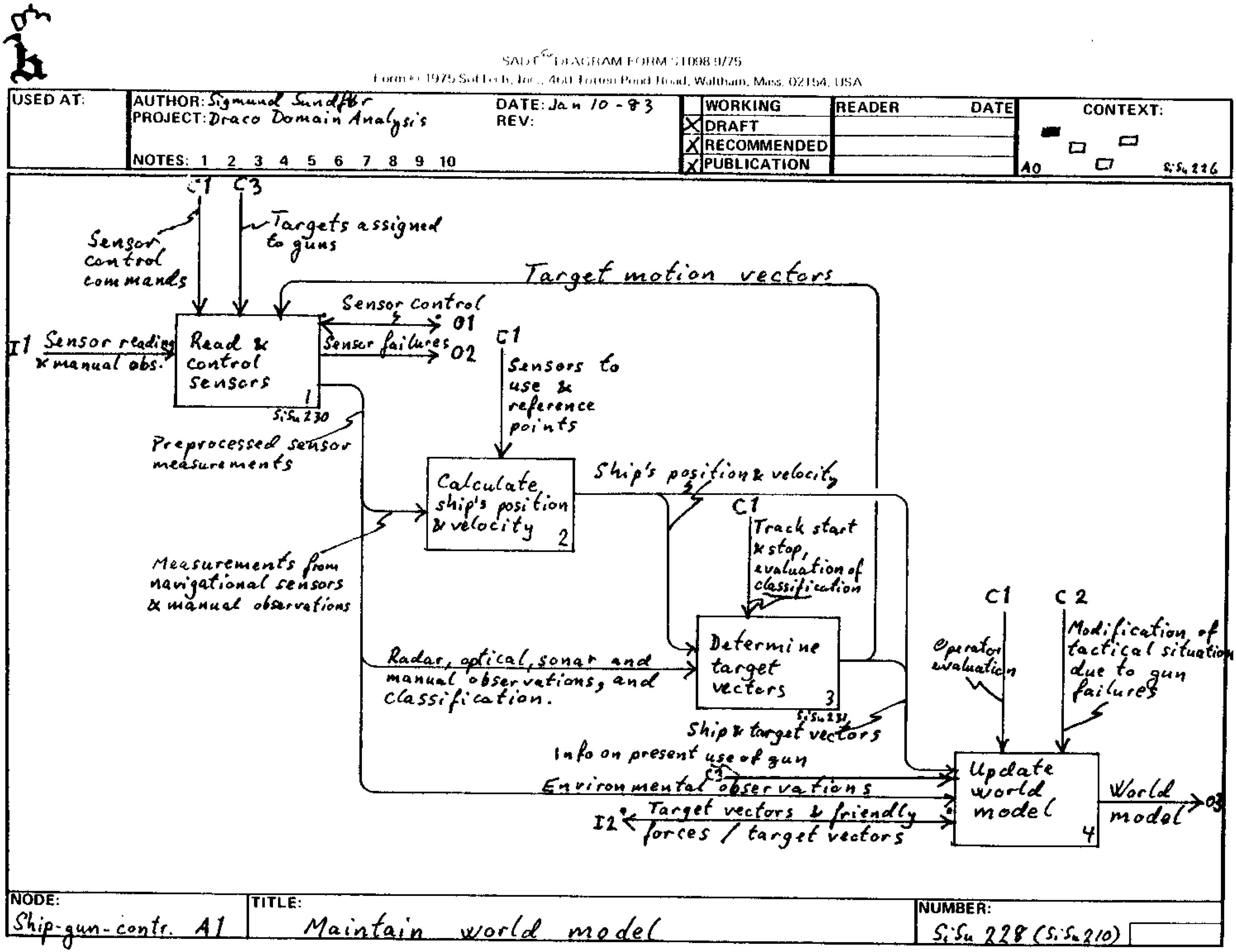
This is just an outline of the main functions to introduce the reader to the type of systems. It is not meant to be complete.

NODE:	TITLE:	NUMBER:
Ship-gun-contr. FEO 0,1	Operate ship-borne gun control System	S. Su 218 (206?)

USED AT:	AUTHOR: Sigmund Sundfør PROJECT: Draco Domain Analysis NOTES: 1 2 3 4 5 6 7 8 9 10	DATE: Jan 10-83 REV:	WORKING X DRAFT X RECOMMENDED X PUBLICATION	READER DATE	CONTEXT: Top

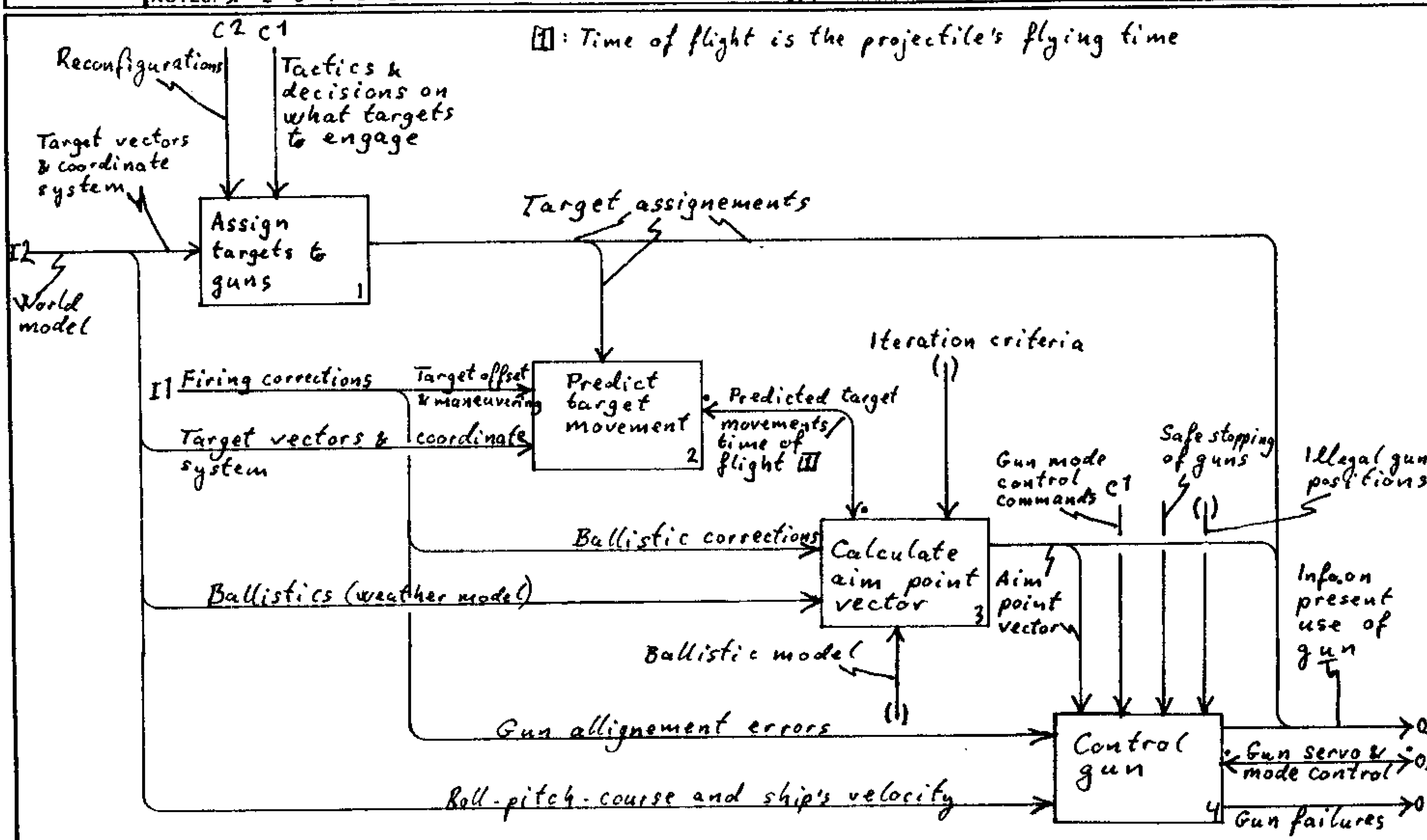


NODE:	TITLE:	NUMBER:
Ship-gun-contr. FEO,2	Operate ship borne gun control system	SiSu 227 (SiSu 209)


SADT DIAGRAM FORM S1098 9/75
Form © 1975 SofTech, Inc., 460 Totten Pond Road, Waltham, Mass. 02154, USA
USED AT:
AUTHOR: Sigmund Sundför
PROJECT: Draco Domain Analysis
DATE: Jan 10 - 83
REV:
NOTES: 1 2 3 4 5 6 7 8 9 10
WORKING
X DRAFT
X RECOMMENDED
X PUBLICATION
READER
DATE
CONTEXT:
A0
S:Su 226
C1
C3
Sensor control commands
Targets assigned to guns
Target motion vectors
I1
Sensor reading & manual obs.
Read & control sensors
1
S:Su 230
Sensor control
O1
Sensor failures
O2
C1
Sensors to use & reference points
Preprocessed sensor measurements
Calculate ship's position & velocity
2
Ship's position & velocity
Measurements from navigational sensors & manual observations
C1
Track start & stop, evaluation of classification
Determine target vectors
3
S:Su 231
Radar, optical, sonar and manual observations, and classification.
Ship & target vectors
C1
C2
Operator evaluation
Modification of tactical situation due to gun failures
Info on present use of gun
C3
Environmental observations
I2
Target vectors & friendly forces / target vectors
Update world model
4
World model
O3
NODE:
Ship-gun-contr. A1
TITLE:
Maintain world model
NUMBER:
S:Su 228 (S:Su 210)

SADT(TM) DIAGRAM FORM S1008 9/75
Form (c) 1975 SofTech, Inc., 460 Totten Pond Road, Waltham, Mass. 02154, USA

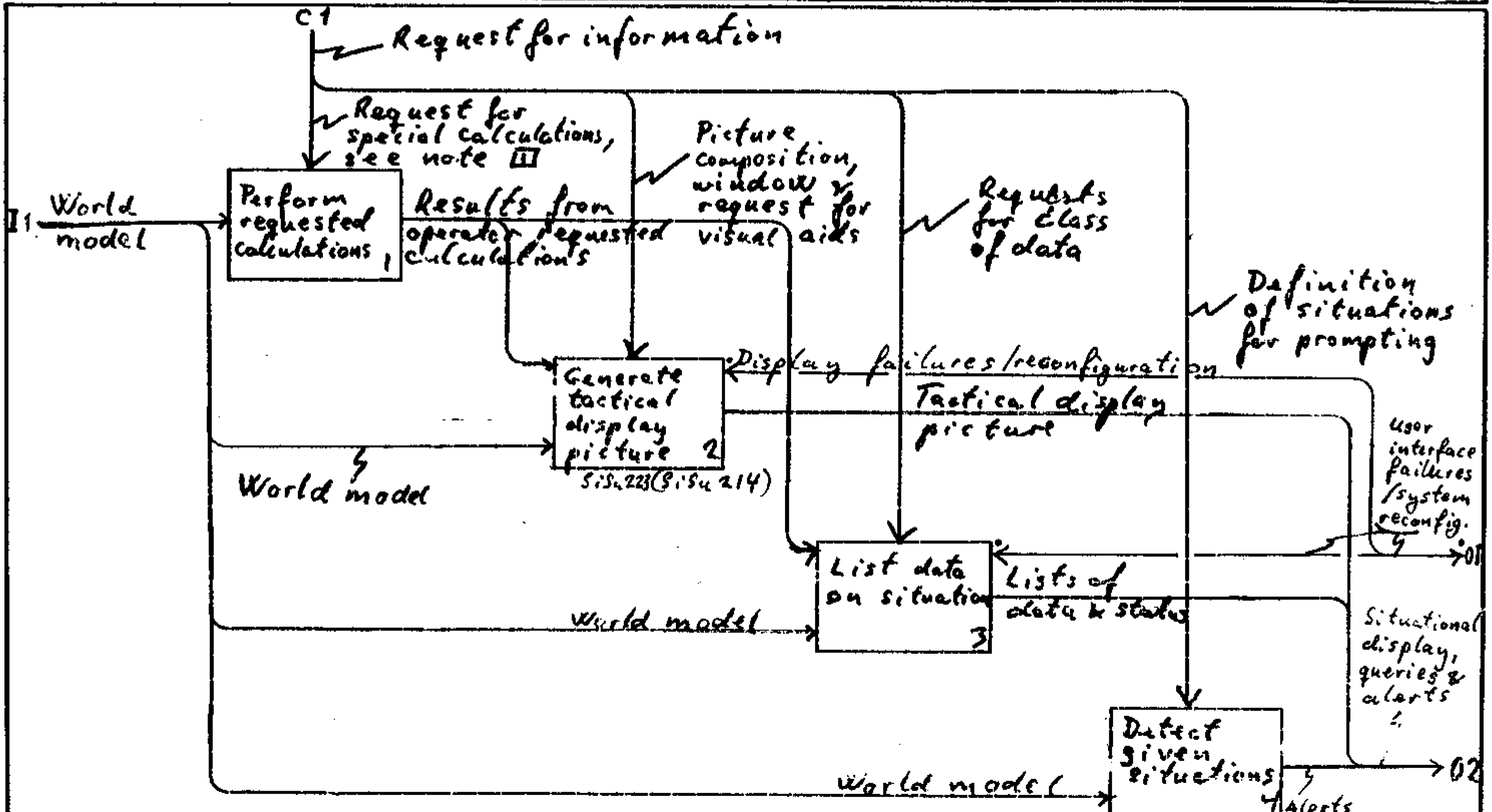
USED AT:	AUTHOR: Sigmund Sundfør PROJECT: Draco Domain Analysis NOTES: ~~1~~ 2 3 4 5 6 7 8 9 10	DATE: Jan 10 - 83 REV:	WORKING X DRAFT X RECOMMENDED X PUBLICATION	READER	DATE	CONTEXT: A0 S:Su 226



NODE: Ship-gun-contr A2	TITLE: Engage guns on targets	NUMBER: S:Su 229 (S:Su 211)

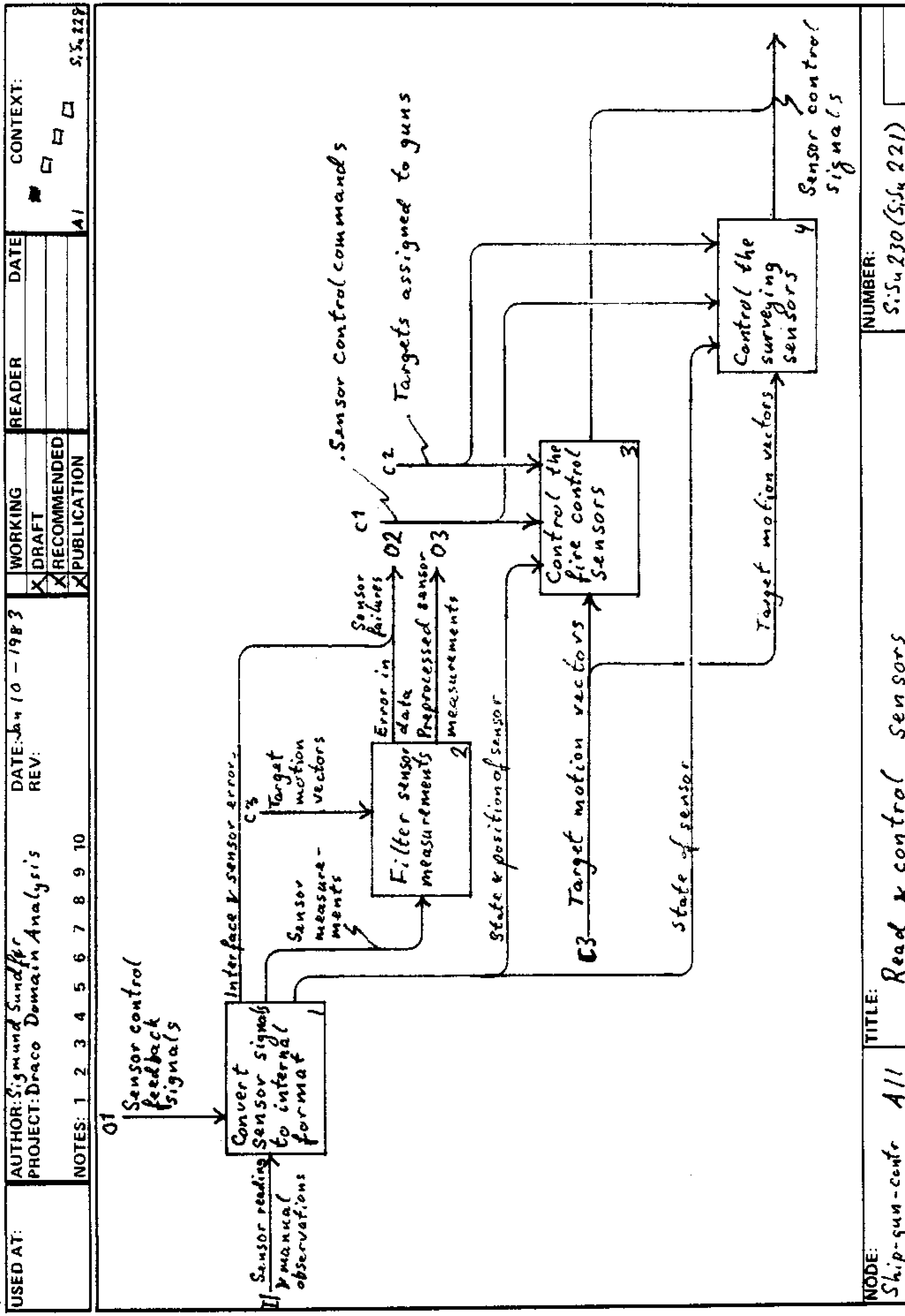
SADT® DIAGRAM FORM ST098 9/75

USED AT:	AUTHOR: Sigmund Sundfor PROJECT: Draco Domain Analysis NOTES: ~~1~~ 2 3 4 5 6 7 8 9 10	DATE: Oct 20 - 82 REV: Jan 10 - 83	☒ WORKING ☒ DRAFT ☒ RECOMMENDED ☒ PUBLICATION	READER	DATE	CONTEXT: AC SiSu 226

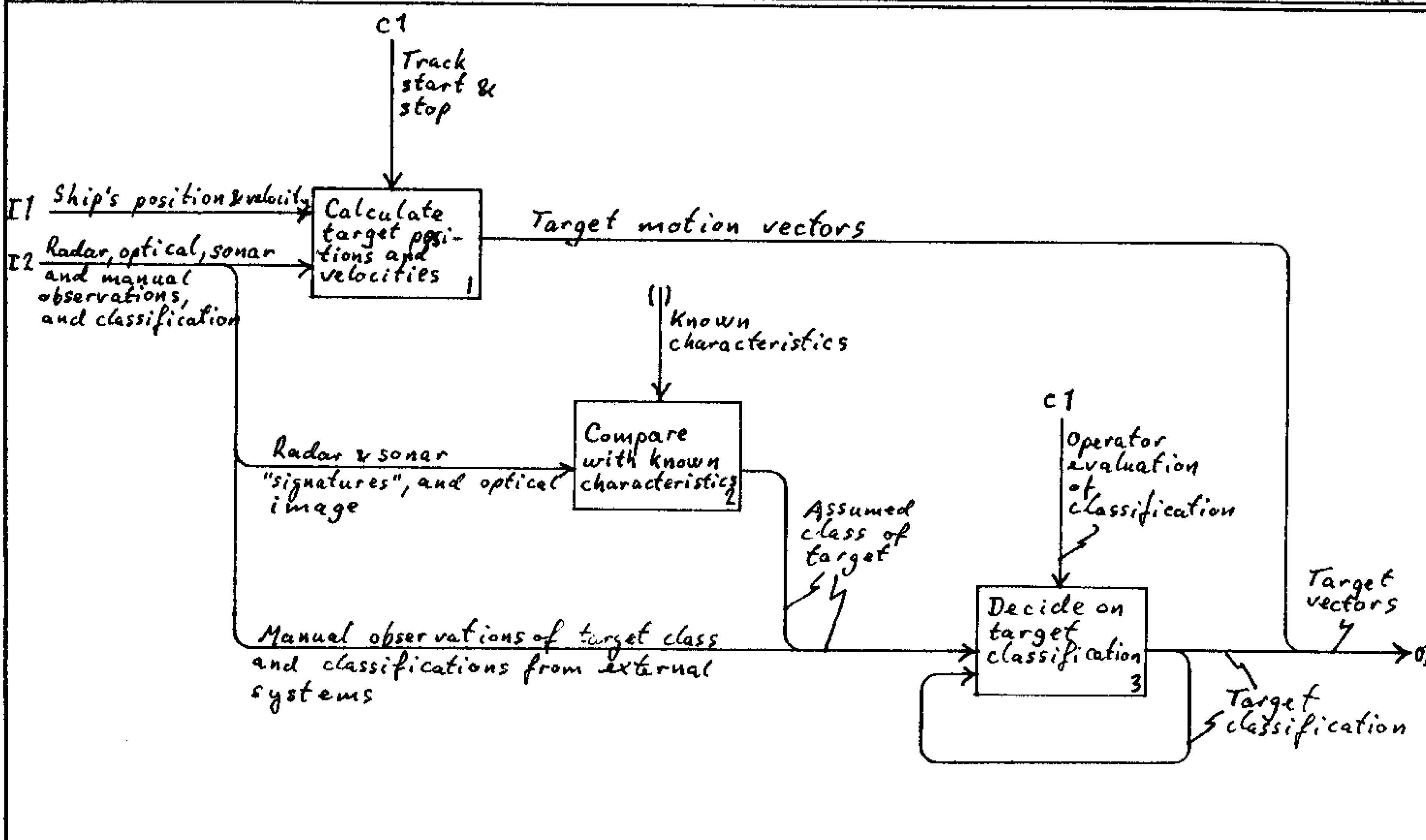


$\boxed{II}$: Special calculations are calculations needed by the operator to evaluate the situation, e.g. "Closest point of approach

NODE: Ship-dynamics A3	TITLE: Display situation	NUMBER: SiSu 213


USED AT:
AUTHOR: Sigmund Sundfor
PROJECT: Draco Domain Analysis
DATE: Jan 10 – 1983
REV:
NOTES: 1 2 3 4 5 6 7 8 9 10
WORKING
DRAFT
RECOMMENDED
PUBLICATION
READER
DATE
CONTEXT:
A1
S:Su 229
SADT(SM) DIAGRAM FORM ST088 9/75
Form © 1975 SofTech, Inc., 460 Totten Pond Road, Waltham, Mass. 02154, USA
I1 Sensor reading & manual observations
O1 Sensor control feedback signals
Convert sensor signals to internal format
1
Interface & sensor error
Sensor measurements
C3 Target motion vectors
Filter sensor measurements
2
Error in data
Sensor failures
O2
Preprocessed sensor measurements
O3
State & position of sensor
C1 Sensor control commands
C2 Targets assigned to guns
Control the fire control sensors
3
C3 Target motion vectors
State of sensor
Target motion vectors
Control the surveying sensors
4
Sensor control signals
NODE: Ship-gun-contr All
TITLE: Read & control Sensors
NUMBER: S:Su 230 (S:Su 221)

USED AT:
AUTHOR: Sigmund Sundfør
PROJECT: Draco Domain Analysis
NOTES: 1 2 3 4 5 6 7 8 9 10
DATE: Jan 11 - 1983
REV:
WORKING
X DRAFT
X RECOMMENDED
X PUBLICATION
READER DATE
CONTEXT: A S:Su 228



NODE: Ship-gun-contr. A13
TITLE: Determine target vectors
NUMBER: S:Su 231

SADT(TM) DIAGRAM FORM ST098 9/75

USED AT:

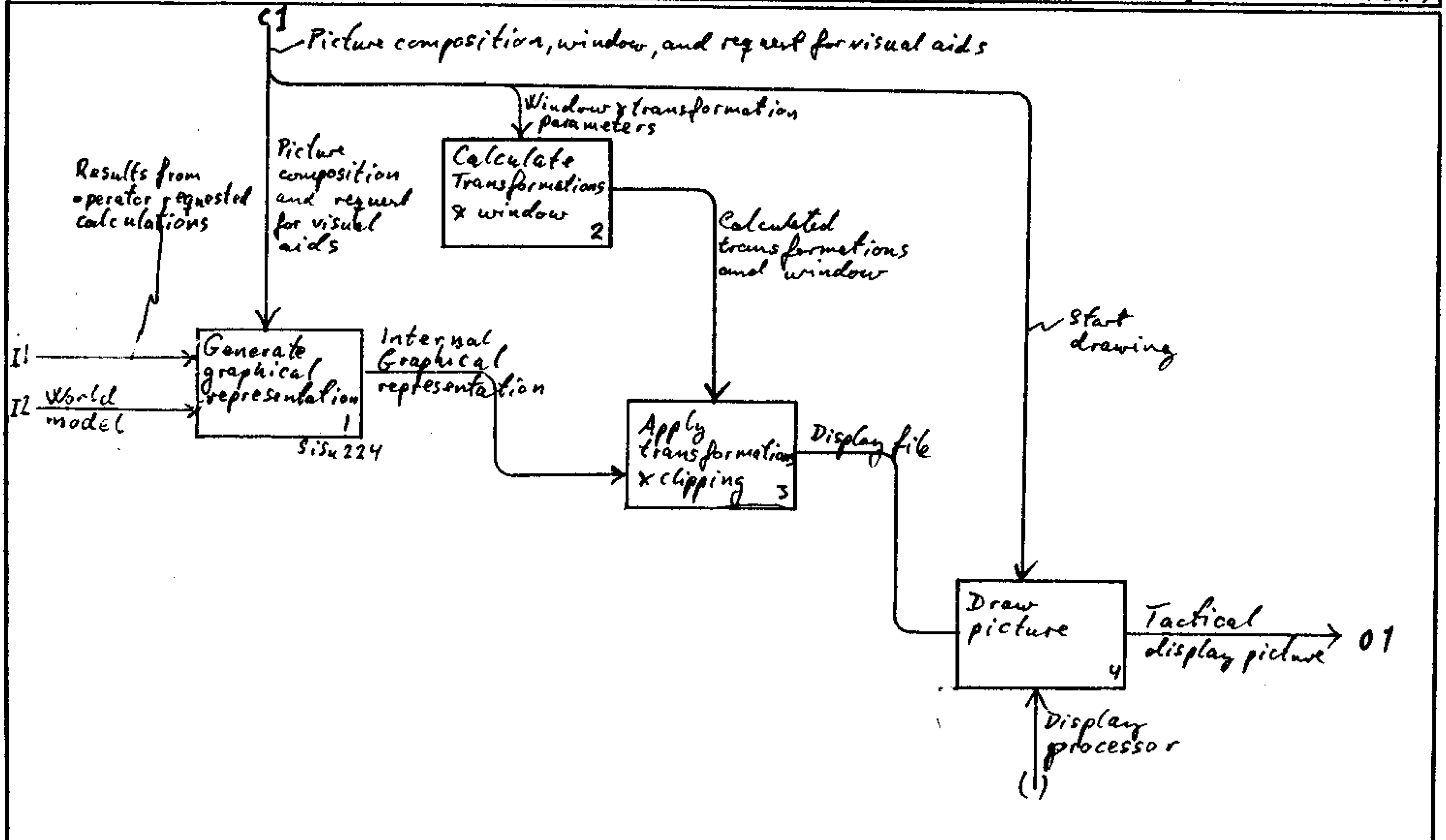
AUTHOR: Sigmund Sundfor
PROJECT: Draco Domain Analysis
DATE: Nov 19-1982
REV: Jan 11 - 1983

NOTES: 1 2 3 4 5 6 7 8 9 10

X WORKING
X DRAFT
X RECOMMENDED
X PUBLICATION

READER | DATE

CONTEXT: A3 SiSu 213



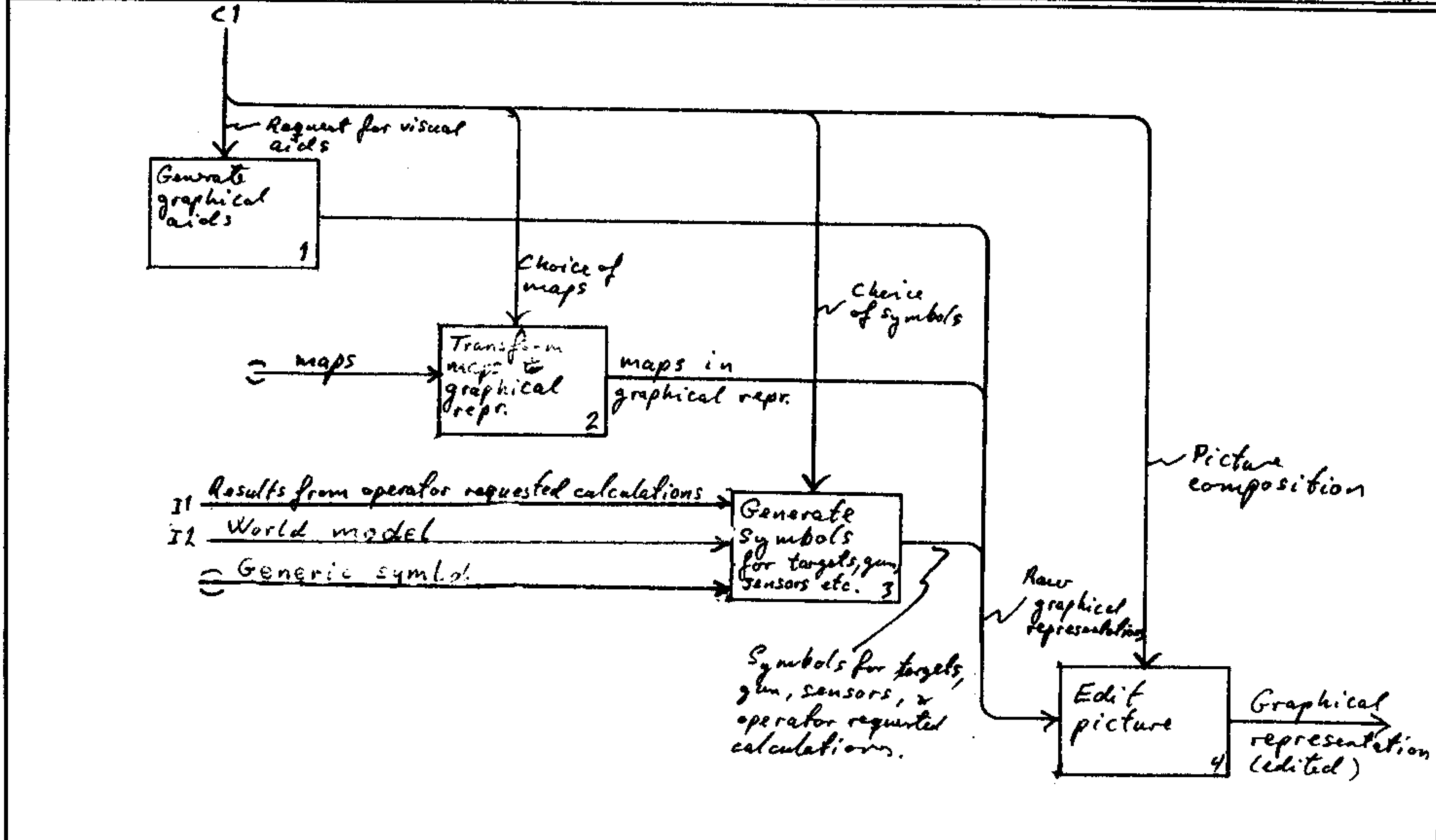
NODE: Ship-gun-contr. A32

TITLE: Generate tactical display picture

NUMBER: SiSu 223 (SiSu 214)

SADT® DIAGRAM FORM ST098 9/75

USED AT:	AUTHOR: Sigmund Sundfor PROJECT: Draco Domain Analysis NOTES: 1 2 3 4 5 6 7 8 9 10	DATE: Nov 19 - 1982 REV: Jan 11 - 1983	☒ WORKING ☒ DRAFT ☒ RECOMMENDED ☒ PUBLICATION	READER DATE	CONTEXT: A32 S;Su 223



NODE:	TITLE:	NUMBER:
Ship-gun-contr A321	Generate graphical representation	S;Su 224

4. Systems consisting of several domains

The preceeding section describes the domain of ship borne gun control systems. It is a model of the domain or anyway a useful subset that covers several possible systems. It is not complete as far as a Draco domain analysis goes.

The question was how well this domain is suited for Draco. In [Neighbors 80] Neighbors argues that one should try to reuse previous development efforts. The Draco system is such a tool for developing many similar systems within a problem domain, reusing the initial analysis and definition of the domain. The initial effort and cost of developing the domain is high, but making new systems within the domain is cheap. Therefore it pays when one is going to make several within a domain. For developing "one-of-a-kind" systems, the "craftsman approach" will still be the most efficient one, i.e. programming in the traditional way.

Today´s and probably also tomorrow´s ship borne gun control systems are quite complex. They provide very many different functions implemented in hardware and software. The size of the software is often in the range of more than 100K bytes. It may be feasible to regard it as one domain. However, this would require the definition of several underlying domains that it could be defined in terms of, i.e. that could be used to define the components of this domain. Even so, it probably would suffer from being too wide a field with too many variations to be a domain suitable for analysis.

<u>The result of a Draco domain analysis should be the definition of a domain language that can be used to specify particular instances (systems) within that domain.</u> A traditional specification for this type of system is typically a sizable book with a large proportion of it giving specifications that affect the software. It must cover what may be regarded as several different areas of knowledge, e.g. ballistics, target tracking, man-machine interface, and hardware interface and control.

Our conclusion was that a ship borne gun control system was too large and spread over too many varying functions to be usefully treated as one domain. One will therefore have to define a complete system in terms of several domains. In addition, experience indicates that some parts of such a system are typically "one-of-a-kind" applications. This appears to be especially true for the control and communication with different subsystems like the sensors. Standardization could change that, but it is not evident that it will be in the interest of the suppliers. It is therefore probably a situation we will have to live with. One-of-a-kind applications are, as Neighbors argues, best suited for the craftsman approach. A system may hence consist of several parts developed using Draco and some by traditional programming.

This leads to having to communicate between different parts of the software constructed using Draco and also between these and other parts programmed in the conventional manner. It is not clear how these interfaces should be designed and how they should be defined for the Draco domains. It seems very difficult to mix programs generated using Draco and other programs in such a way that they may call each other. When defining parts in terms of a Draco domain, one does not know or want to know the low level details of program structure, call mechanisms and parameters. It is probably simpler if they are separated in different tasks. The definition and implementation of the communication between the tasks will have to be developed. The proposed domain definition does suggest a way of doing it. However, how appropriate and feasible this suggestion may be is not investigated any further than the qualitative assessment that it may work.

The other problem that arises is how the total system should be defined. This is not treated further in this research. However, it is important to note the problems that will arise in using the type of methods like Draco for developing complete systems like ship borne gun control systems.

4.1 Methods for Identifying Domains

SADT is a good tool for analyzing an application. The problem can be decomposed into functional parts with the connections between them. It is powerful enough to represent a range of different applications. At the same time, the syntax is sufficiently rigid to minimize ambiguity resulting from different ways of interpreting the models. The model splits the system into functional parts, seen from the applications point of view. I.e. this is an outside in approach.

It was noted earlier that an application like this consists of some parts that remain similar from system to system, while other parts are so different that that they can be thought of as one-of-a-kind parts. These differences are mostly due to differences in interfaces, subsystems, processing power, communication protocols etc. . They are technically based differences, not necessarily differences from the applications point of view. It seems best to define domains that coincides with functions that are related in such a way that they change at the same time, and for the same reasons.

The breakdown of the model into functions (actions) using SADT does not necessarily define the same divisions. This SADT model was made from the applications point of view, while the changes are often technically related. Parnas [Parnas 79] advocates that one should anticipate changes and design with those in mind. He uses the concept of program families. A program family is a collection of component with sufficiently in common to make it worth while to study their commonality before their differences. The components in a family share common secrets that are hidden to the "outside world". Changes may be made in the implementations without affecting other parts of the software. I.e. more of the software is reusable.

Reuse is what we want to achieve using Draco. It appears therefore that the method for defining program families can equally well be used to define Draco domains. This leads to an approach where one tries to group types of changes anticipated from system to system. It is probably best to start at the bottom level and group different parts of the hardware that the software operates upon and interfaces to. Several levels of groups will probably be useful. This is an inside-out approach. At the same time, one must consider changes in technology, e.g. new tracking algorithms may be applied. Lastly it must be matched by changes from the applications point of view, the outside-in approach. In a way it can look like a cake that is cut in 3 different directions.

It may be worth noting that the application, ship borne gun control system, and weapon systems in general, is very much influenced by technology. The basic idea, hitting the target, remains more or less unchanged. The ways one goes about it, however, is very much determined by what is technologically feasible. This may be somewhat simplified, but it does make the point that it is not a simple automation of a manual process. When new technology is used, the application frequently changes. Application changes will therefore often coincide with changes from a technical point of view.

The following two figures illustrates the factors leading to changes; the physical environment and the technology. The third factor, the application from the users point of view is not illustrated. (Notice the amoeba shape of the software - constantly changing.)

Taking these factors into account, one may break the ship borne gun control system into a useful set of domains. Some of these domains will be suitable for definition using Draco, other may be best implemented by conventional programming. Apart from identifying the domains, they should also be organized in a hierarchical fashion showing how one domain is defined in terms of other domains.

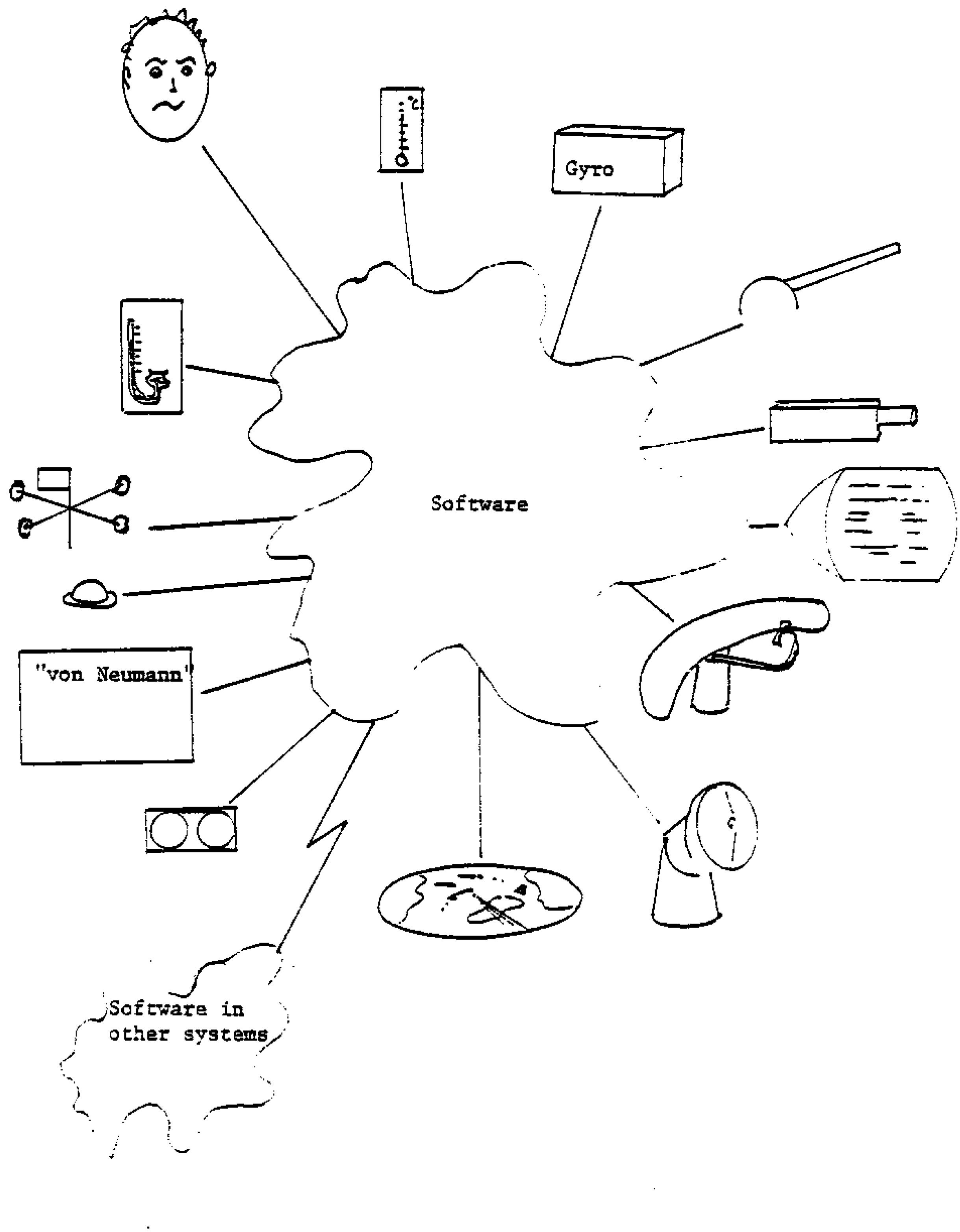


Figure 4-1: The physical environment for the software

Software moulded by:

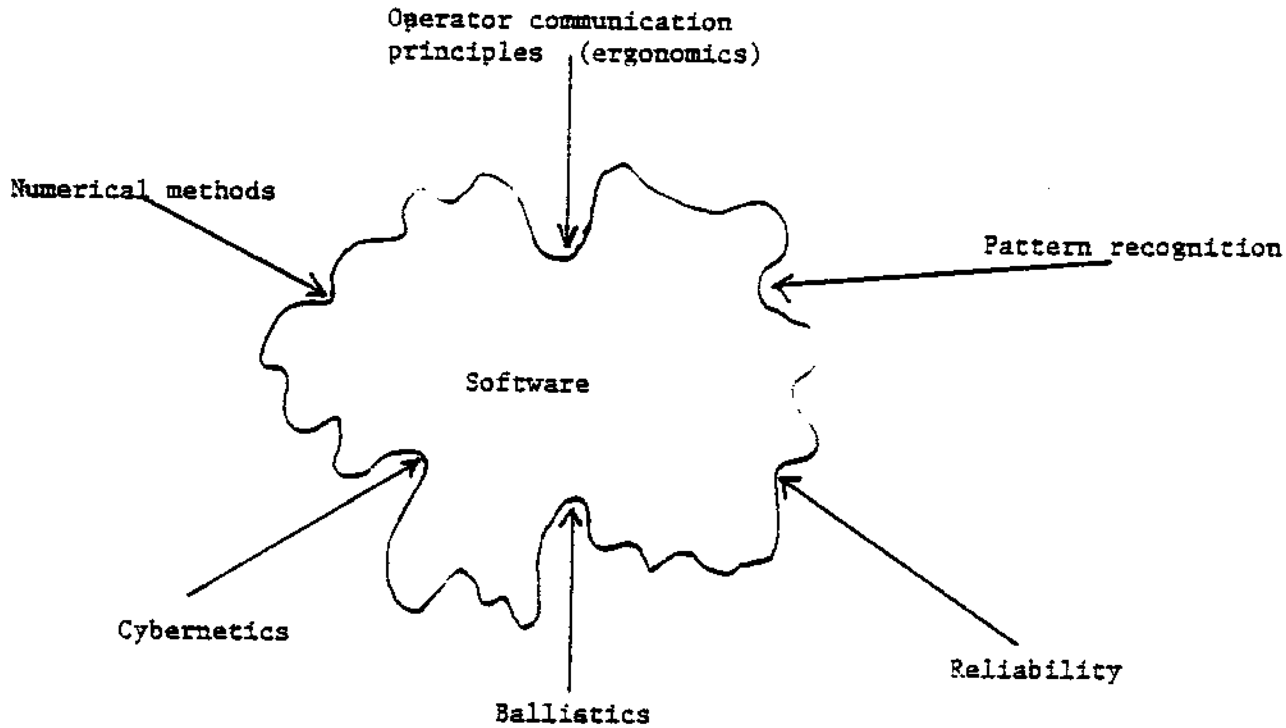


Figure 4-2: The factors moulding the software

5. The Tactical Plot Domain

One of the domains identified using the preceeding reasoning was the tactical plot domain. It is this domain that is analysed fully using Draco.

The tactical display is the CRT display where the user is given a graphical representation of the tactical situation, i.e. the tactical plot. It is normally done by mixing radar video with computer generated graphics. The graphics will highlight information on targets, sensors, gun(s), maps and other information useful for evaluating the situation. There is a fair bit of standardization of the symbols used.

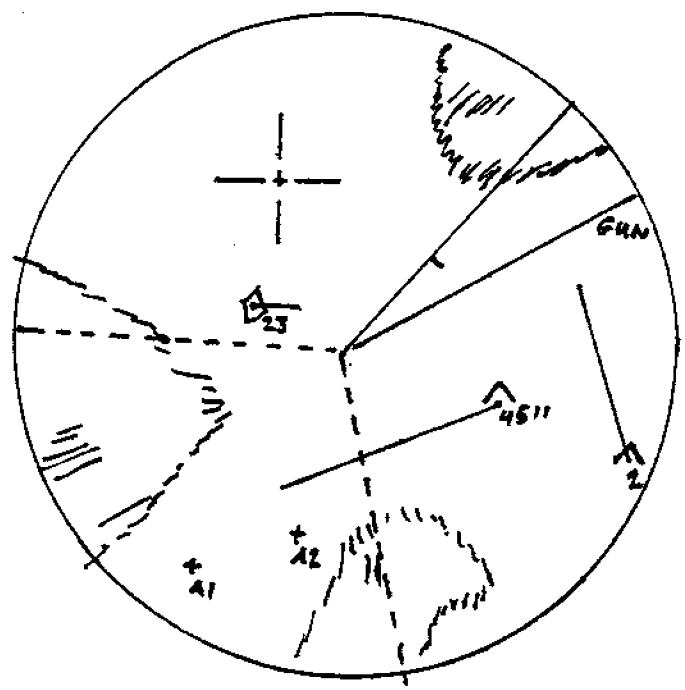


Figure 5-1: A tactical display picture

It is straight-forward two-dimensional graphics, mainly line drawing, both single or multicolour. It may therefore be based on using standardized graphical functions. Different systems require different mixes of symbols and composition of the picture. There may also be slight changes in the symbols used.

There will be several different versions needed, while at the same time the underlying representations do not change too much. It has therefore been a suitable domain for analysis and definition using Draco.

The model used to identify the domain assumes that there is a database of some kind that contains the data on the tactical situation. The function of this domain is to provide a tool (domain language) to specify how this data is to be presented graphically to the user on the tactical display. In other words it covers the mapping from the internal world model in the system to the graphical presentation of this. It does not cover the collection of the data to build and maintain the world model, nor does it deal with the detailed algorithm of how you generate symbols on the particular displays used.

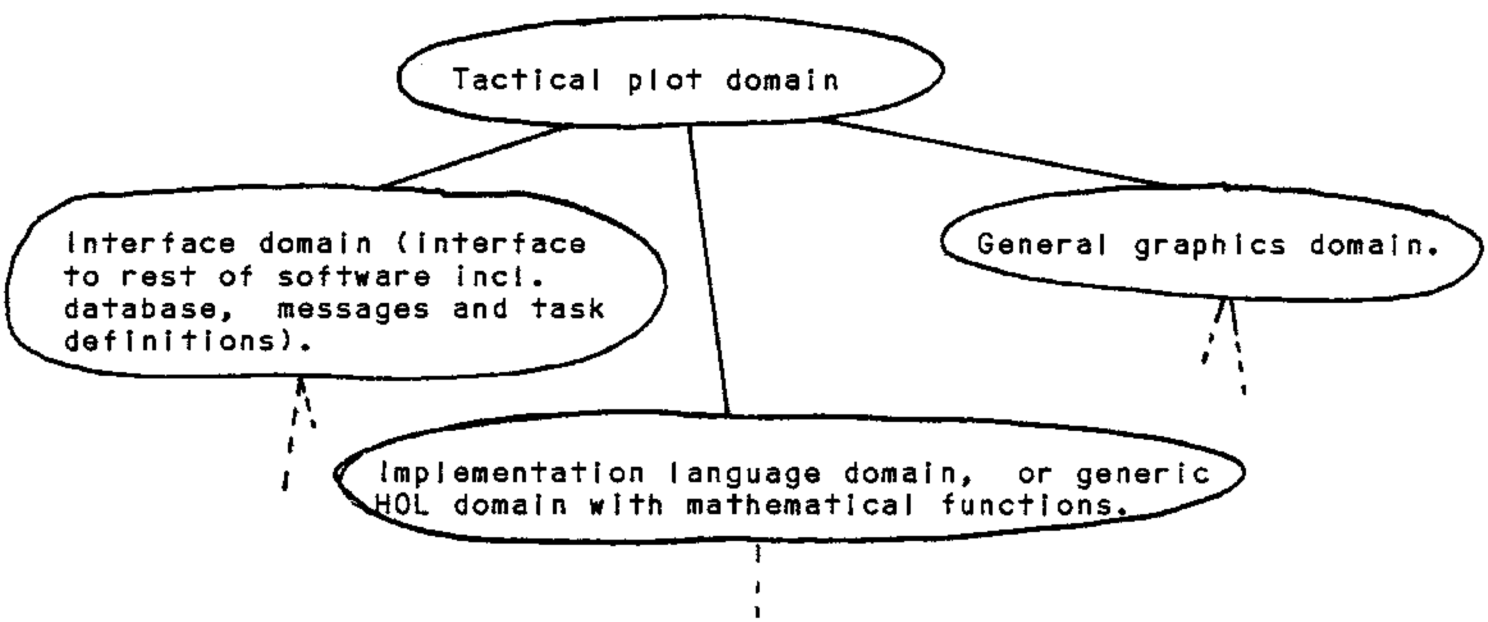


Figure 5-2: The domains that the tactical plot domain builds on

It may be argued that this domain is not the most representative one of real-time, embedded systems. It does not have the very tough timing constraints that may be involved in sensor handling and gun positioning. These latter, however, do belong to the domains that are either the same from system to system, or they change so much due to other sensors or gun that they are best treated as one-of-a-kind jobs.

The CRT may be used for other purposes than displaying the tactical situation like backup for alpha-numeric displays. However, that is not part of the tactical plot picture and is therefore not covered by this domain. These other applications of the CRT will however, probably also use the general graphic domains to implement the language components (refinements).

6. The Analysis of the Tactical Plot Domain

6.1 Model of the Domain

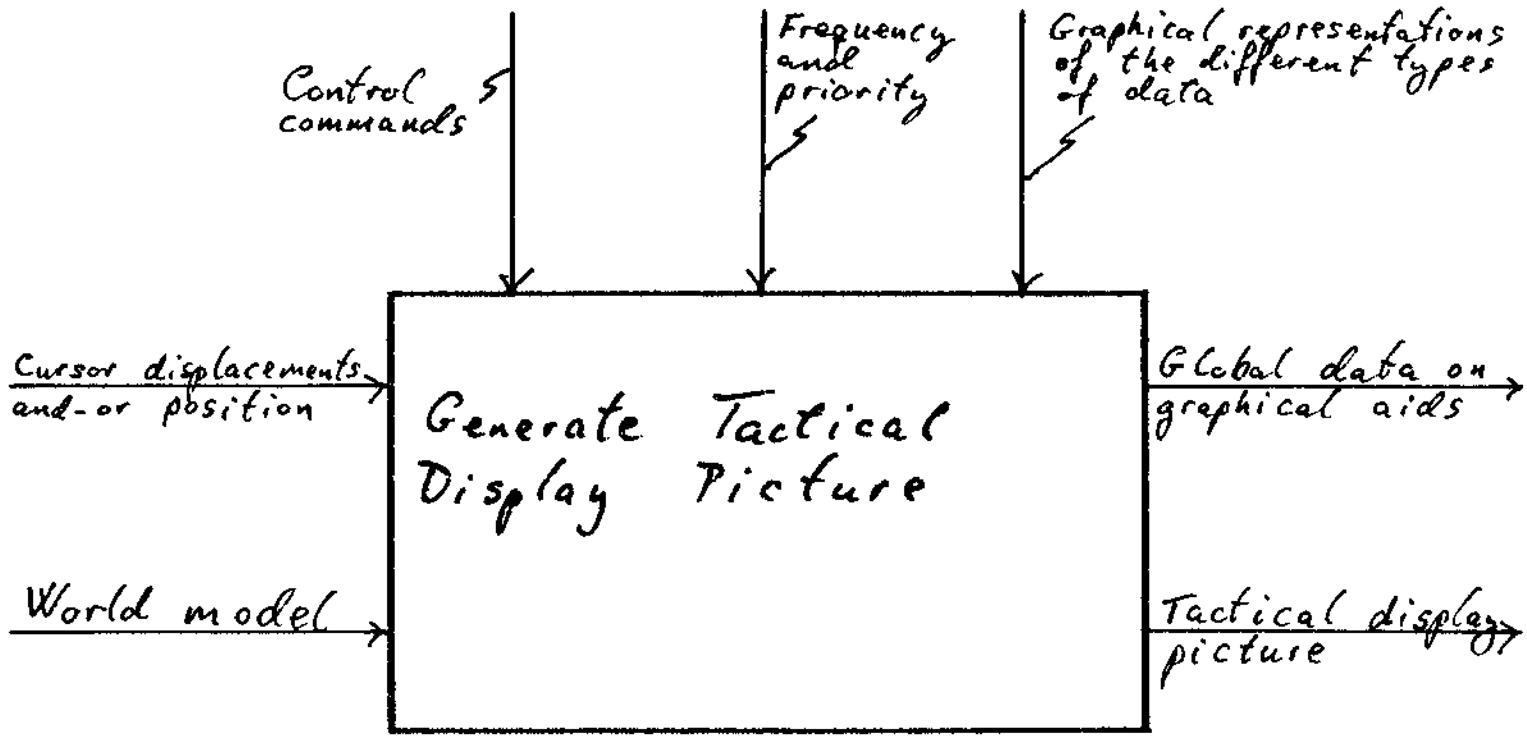
The starting point of the analysis was the modelling of the domain. This was done using SADT. The model is presented on the following pages. It should be noted that the viewpoint (an important parameter that shapes the SADT model) of this model is not the same as for the model of the ship borne gun control system presented in section 2. It cannot therefore be read as an expansion of the activity "Display situation" (node A3) in that model although it covers many of the same functions.

The following diagrams make up the model:

- A -0: Generate Tactical Display Picture
- FEO -0: Generate Tactical Display Picture
- A 0: Generate Tactical Display Picture
- A 1: Generate graphical aids
- A 2: Select data to be displayed
- A 3: Generate symbolic representation
- A 4: Transform to display code
- A 5: Select picture planes, colour and draw
- A 24: Retrieve maps
- A 32: Generate target symbols
- A 34: Generate graphical aids
- A 35: Generate graphical representation for other "world objects"

SADT(R) DIAGRAM FORM ST098 9/75
Form © 1975 SofTech, Inc., 460 Totten Pond Road, Waltham, Mass. 02154, USA

USED AT:	AUTHOR: Sigmund Sunoffer PROJECT: Draco Domain Analysis NOTES: 1 2 3 4 5 6 7 8 9 10	DATE: March 8-83 REV:	X WORKING X DRAFT X RECOMMENDED X PUBLICATION	READER DATE	CONTEXT: None



NODE:	TITLE:	NUMBER:
A-0 Tactical Display	Generate Tactical Display Picture	SiSu 260

USED AT:	AUTHOR: Sigmund Sundfør PROJECT: Draco Domain Analysis NOTES: 1 2 3 4 5 6 7 8 9 10	DATE: March 7-83 REV:	X WORKING X DRAFT X RECOMMENDED X PUBLICATION	READER DATE	CONTEXT: None

Mar 7 10:09 1983 td.feo-0 Page 1

Topic:

Tactical display for a ship borne gun control system.

Viewpoint:

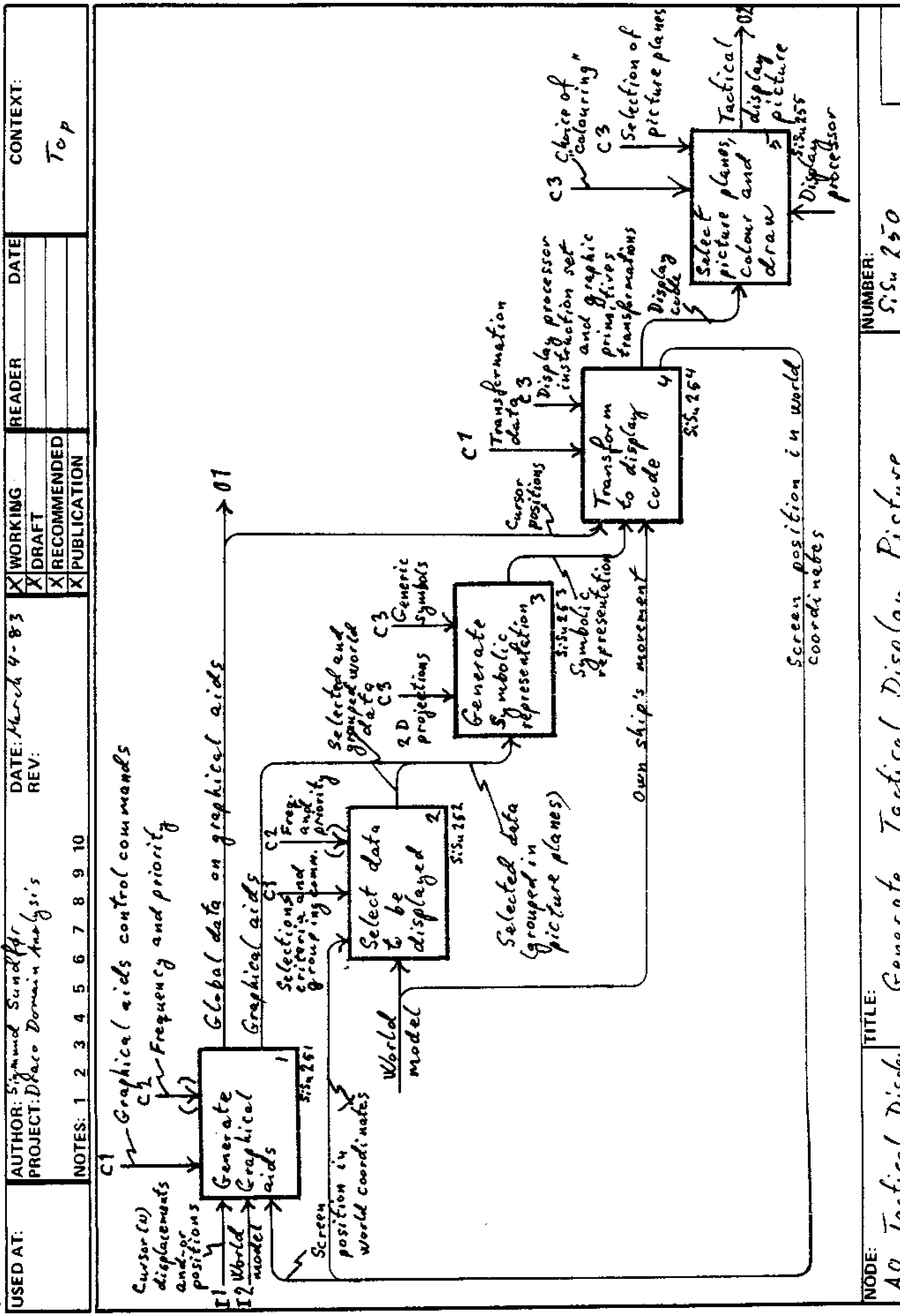
Domain Craftsman.
He/she has a pretty good understanding of what has to be done without bothering too much about the nitty gritty of "how". Some implementation decisions will clearly shine through even so.

Purpose:

To create a maximum model exposing as much as possible of the data and activities in the domain, including some implementation details. It will serve the purpose of a model that can be used to identify what type of information the domain user needs to supply when defining an instance of the domain. There will be a second model showing just this - the domain user's point of view.

In addition, this model does actually define the components (refinements) that the domain builder has to supply.

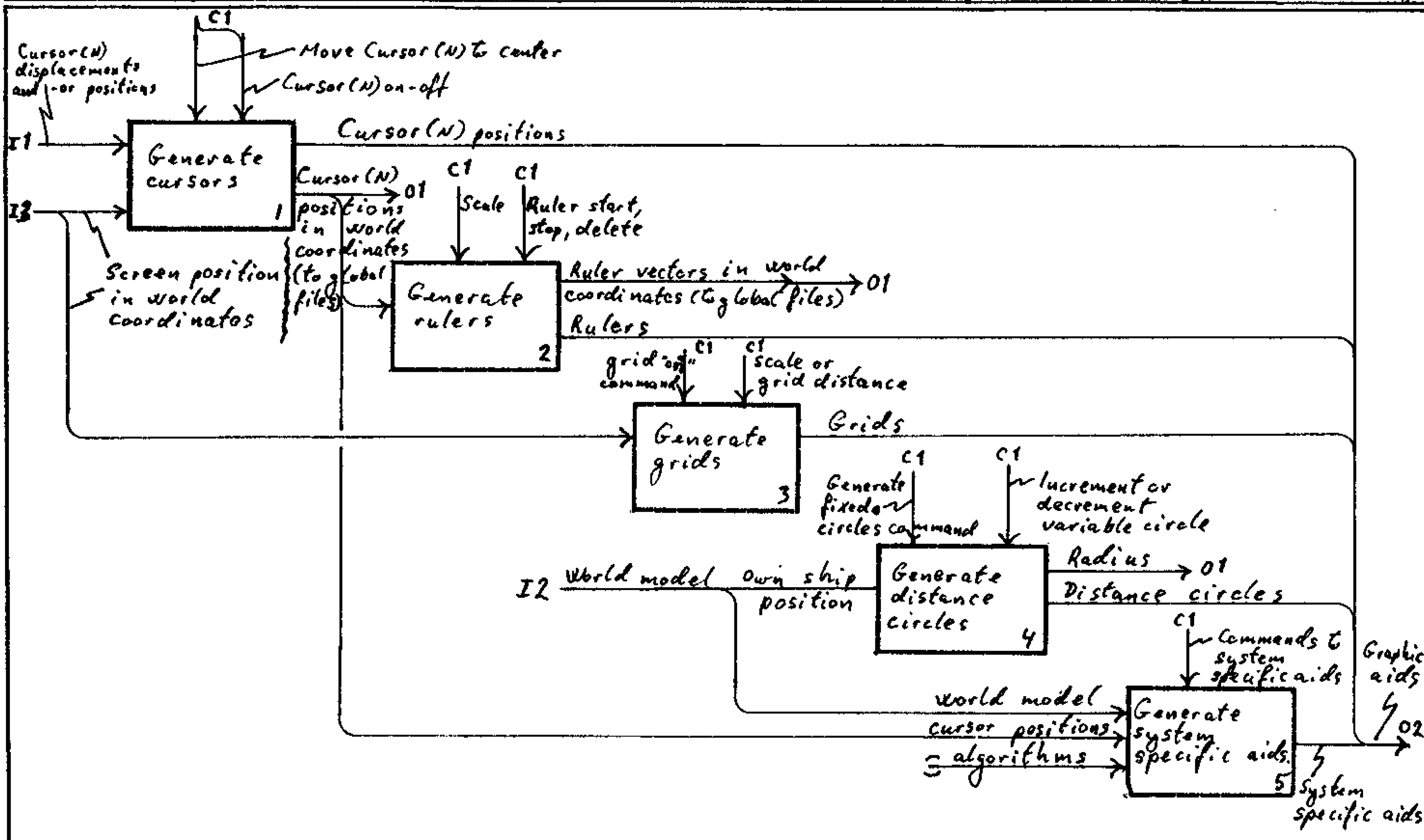
NODE:	TITLE:	NUMBER:
FEO-0 Tactical Display	Generate Tactical Display Picture	SiSu 261



SADT DIAGRAM FORM SТ088 9/75
Form © 1975 SofTech, Inc., 460 Totten Pond Road, Waltham, Mass. 02154, USA
USED AT:
AUTHOR: Sigmund Sundfør
PROJECT: Draco Domain Analysis
DATE: March 4-83
REV:
NOTES: 1 2 3 4 5 6 7 8 9 10
WORKING
DRAFT
RECOMMENDED
PUBLICATION
READER
DATE
CONTEXT:
Top
Generate Graphical aids
Select data to be displayed
Generate Symbolic representation
Transform to display code
Select picture planes, colour and draw
Graphical aids control commands
Frequency and priority
Global data on graphical aids
Graphical aids
Selections criteria and grouping comm.
Freq. and priority
World model
Screen position in world coordinates
Cursor(s) displacements and-or positions
Selected and grouped world data
2D projections
Generic symbols
Selected data (grouped in picture planes)
Symbolic representation
Own ship's movement
Cursor positions
Transformation data
Display processor instruction set and graphic primitives transformations
Display code
Choice of "colouring"
Selection of picture planes
Tactical display picture
Display processor
Screen position in world coordinates
NODE: A0 Tactical Display
TITLE: Generate Tactical Display Picture
NUMBER: SiSu 250

SADT(TM) DIAGRAM FORM ST098 9/75
Form © 1975 SofTech, Inc., 460 Totten Pond Road, Waltham, Mass. 02154, USA

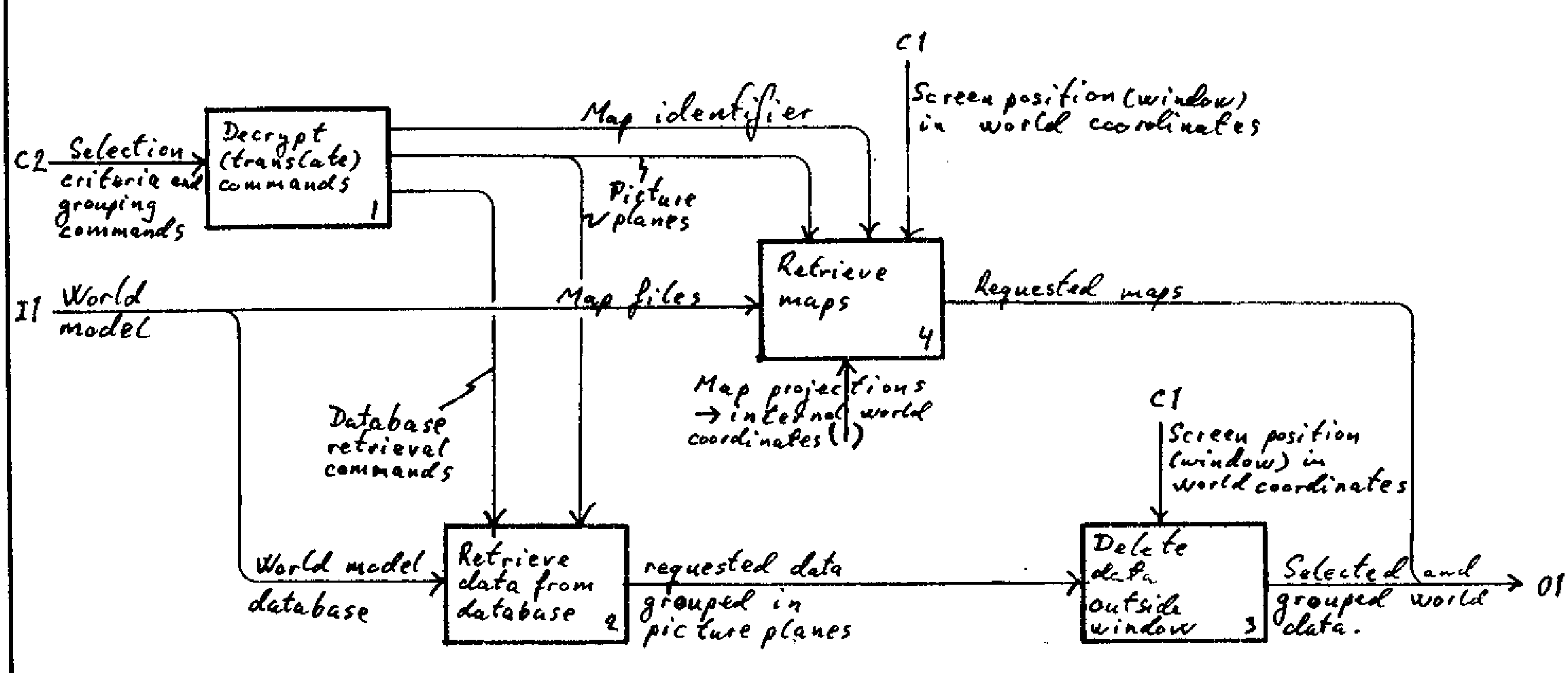
USED AT:	AUTHOR: Sigmund Sundfor PROJECT: Draco Domain Analysis NOTES: 1 2 3 4 5 6 7 8 9 10	DATE: March 4-1983 REV:	X WORKING X DRAFT X RECOMMENDED X PUBLICATION	READER DATE	CONTEXT: 40 S.S. 250



NODE: A1 Tact. Display	TITLE: Generate graphical aids	NUMBER: S.S. 251

SADT® DIAGRAM FORM ST098 9/75
Form © 1975 SofTech, Inc., 460 Totten Pond Road, Waltham, Mass. 02154, USA

USED AT:	AUTHOR: Sigmund Sundfor PROJECT: Draco Domain Analysis NOTES: 1 2 3 4 5 6 7 8 9 10	DATE: March 9-83 REV:	X WORKING X DRAFT X RECOMMENDED X PUBLICATION	READER DATE	CONTEXT: A0 SiSu 250

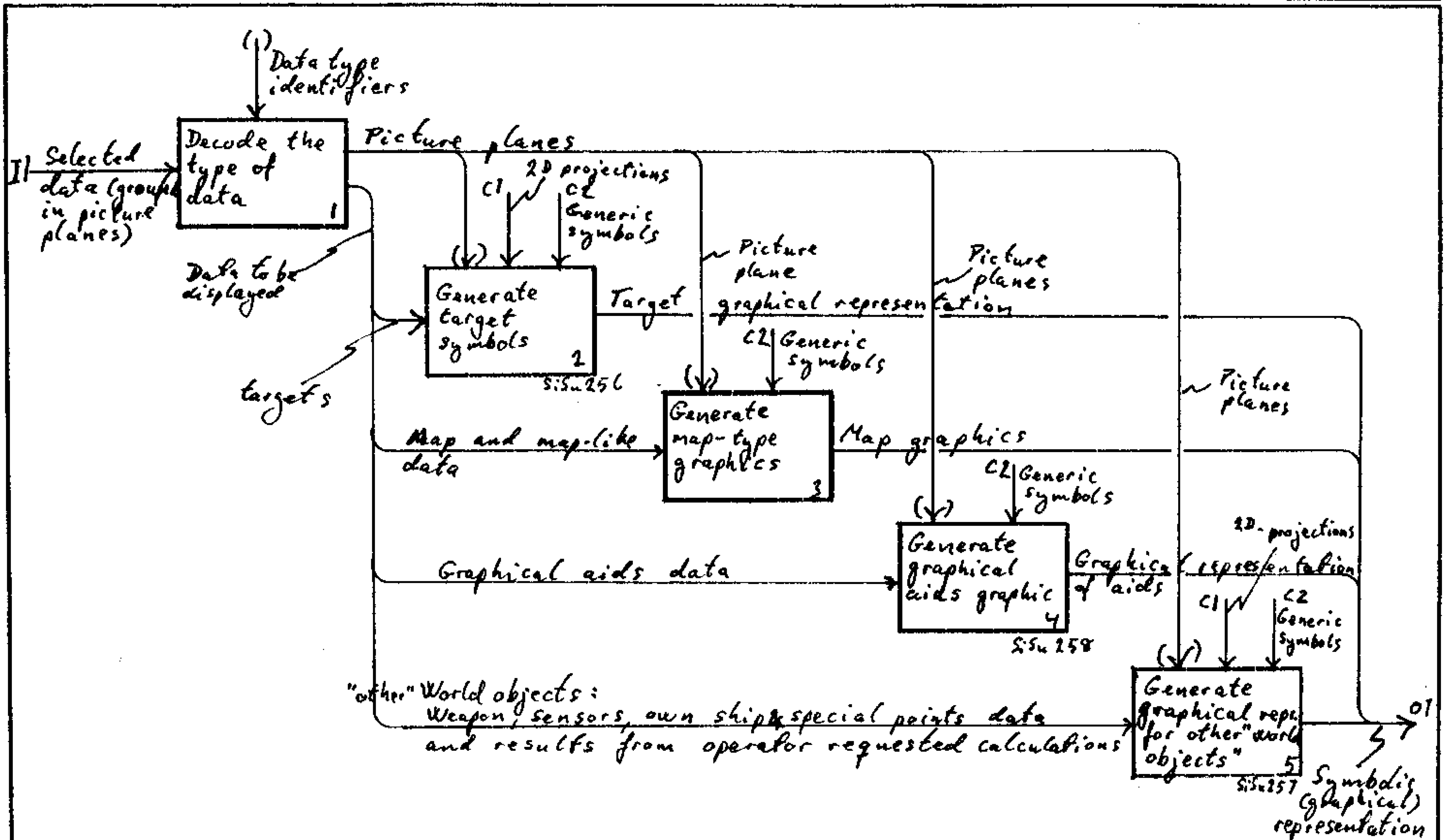


NODE: A2 Tactical Display	TITLE: Select data to be displayed	NUMBER: SiSu 252

SADT® DIAGRAM FORM S1098 9/75

Form © 1975 SofTech, Inc., 460 Totten Pond Road, Waltham, Mass. 02154, USA

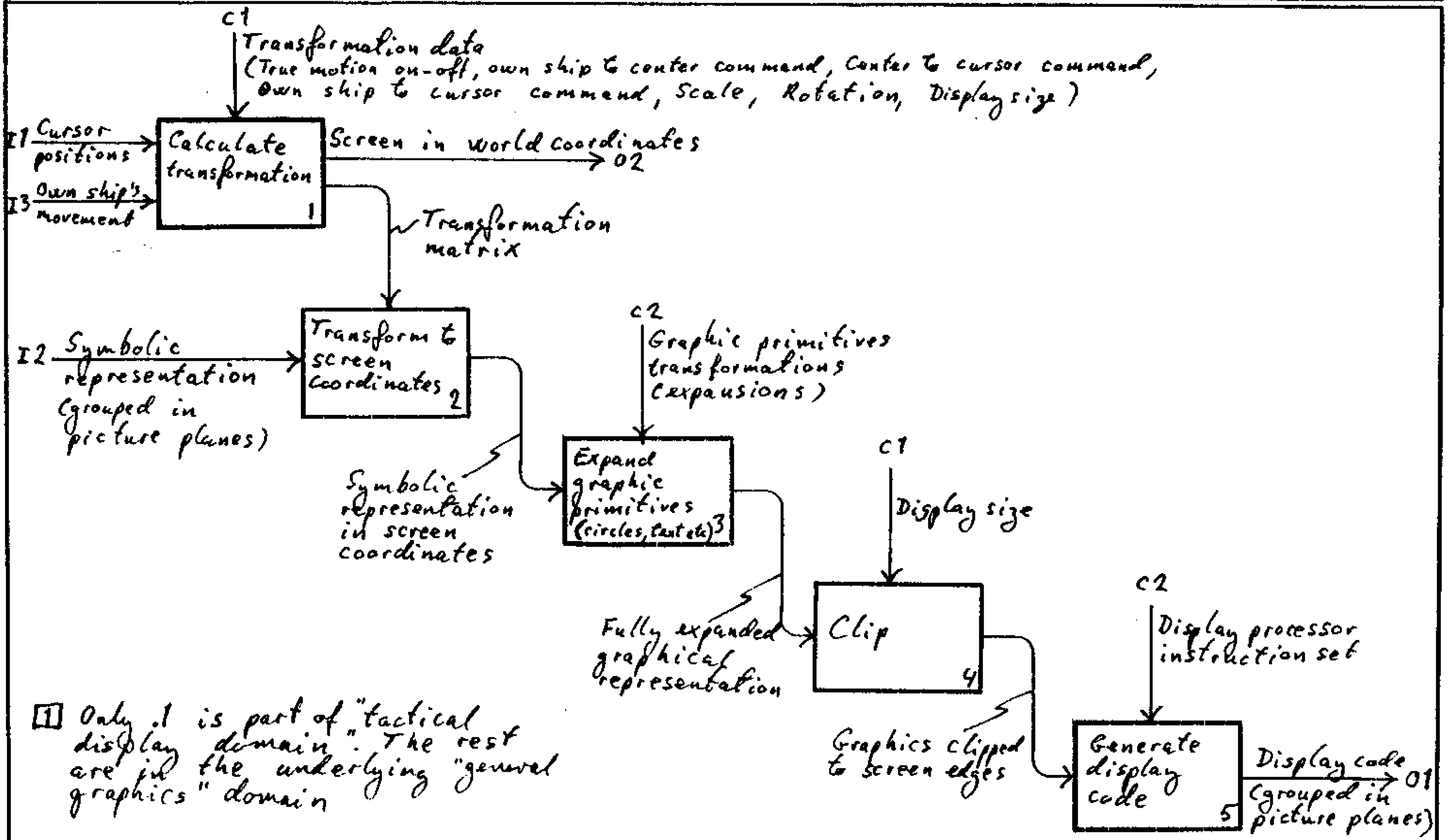
USED AT:	AUTHOR: Sigmund Sundfør PROJECT: Draco Domain Analysis NOTES: 1 2 3 4 5 6 7 8 9 10	DATE: March 4-83 REV: May 27-83	X WORKING X DRAFT X RECOMMENDED X PUBLICATION	READER DATE	CONTEXT: A0 S:S250



NODE: A3 Tactical Display	TITLE: Generate symbolic representation	NUMBER: S:Su 253

SADT® DIAGRAM FORM ST098 9/75

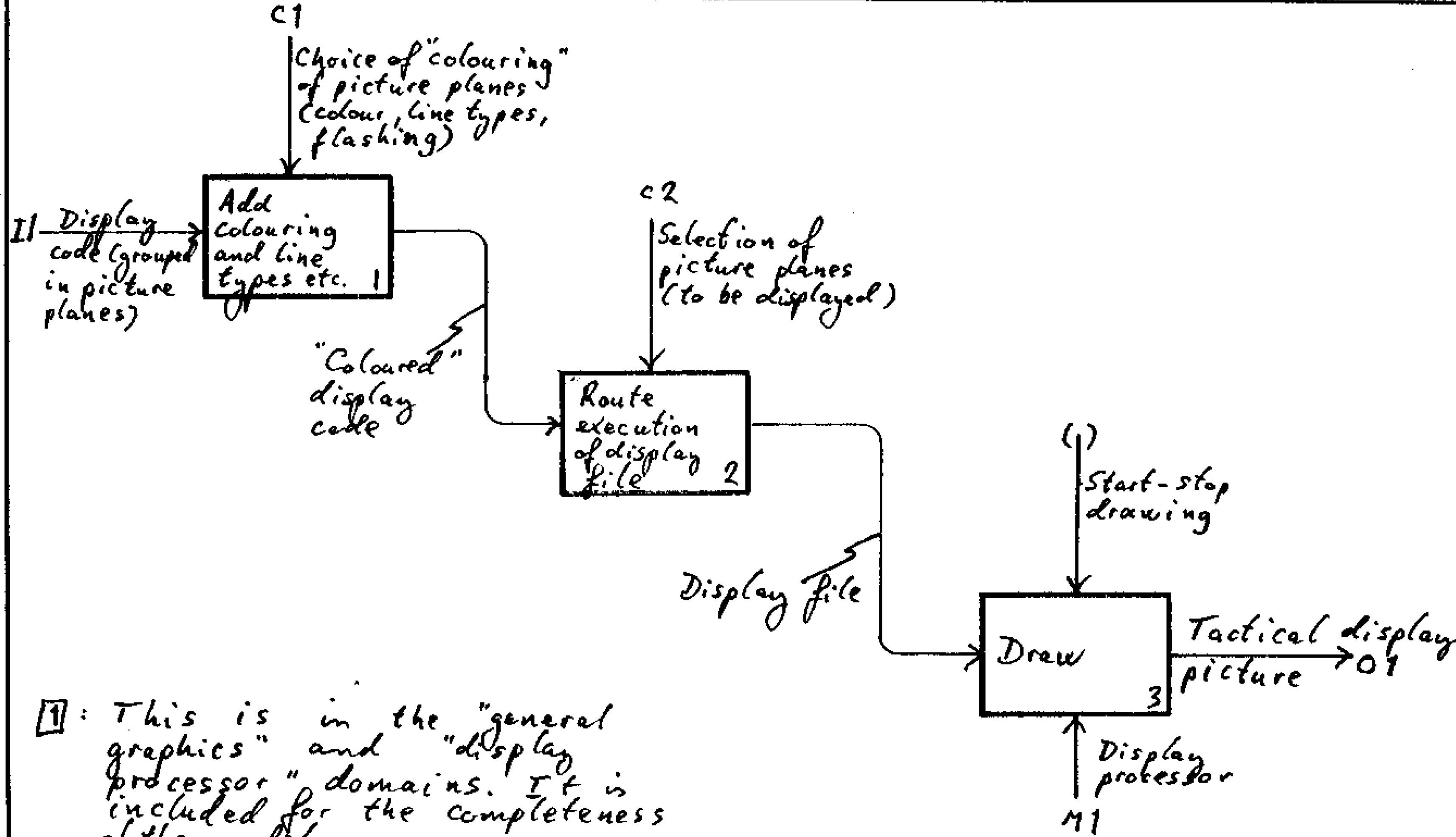
USED AT:	AUTHOR: Sigmund Sundfør PROJECT: Draco Domain Analysis NOTES: ~~1~~ 2 3 4 5 6 7 8 9 10	DATE: March 4 - 83 REV:	X WORKING X DRAFT X RECOMMENDED X PUBLICATION	READER	DATE	CONTEXT: 40 SiSu250



NODE:	TITLE:	NUMBER:
A4 - Tactical Display	Transform to display code	SiSu 254

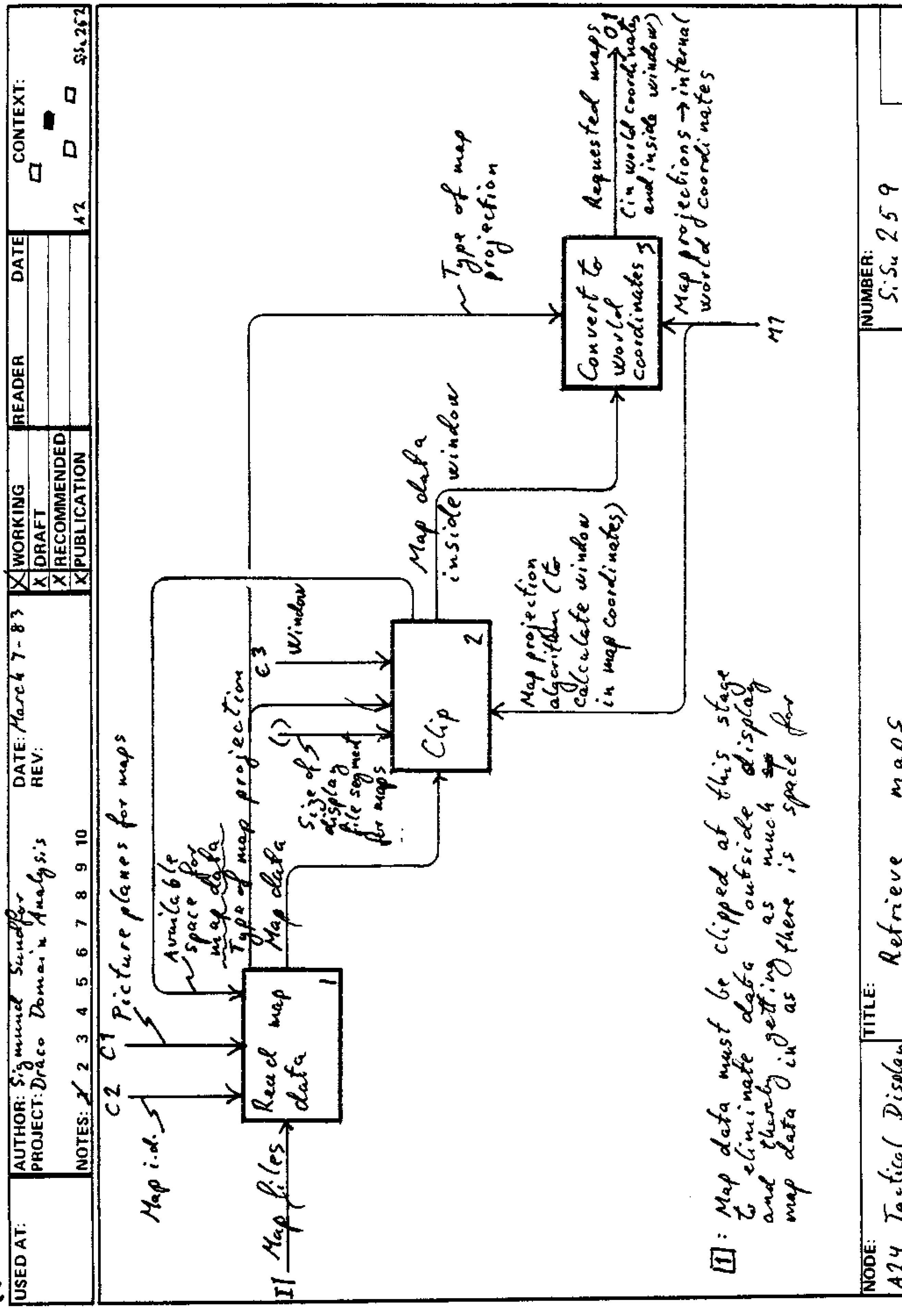
SADT(TM) DIAGRAM FORM ST098 9/75

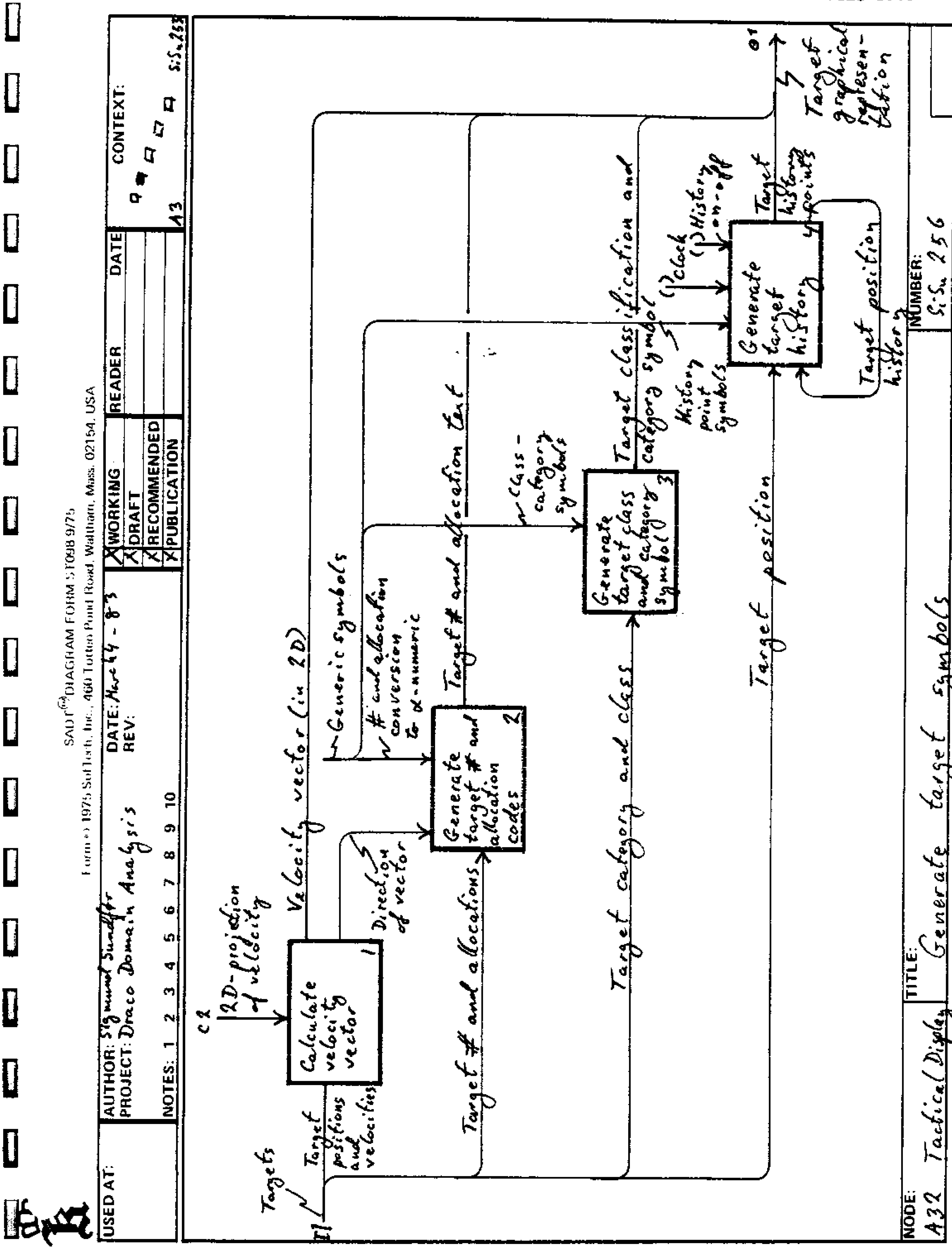
USED AT:	AUTHOR: Sigmund Sundfør PROJECT: Draco Domain Analysis NOTES: ~~1~~ 2 3 4 5 6 7 8 9 10	DATE: March 4-83 REV:	X WORKING X DRAFT X RECOMMENDED X PUBLICATION	READER DATE	CONTEXT: A0 SiSu 250

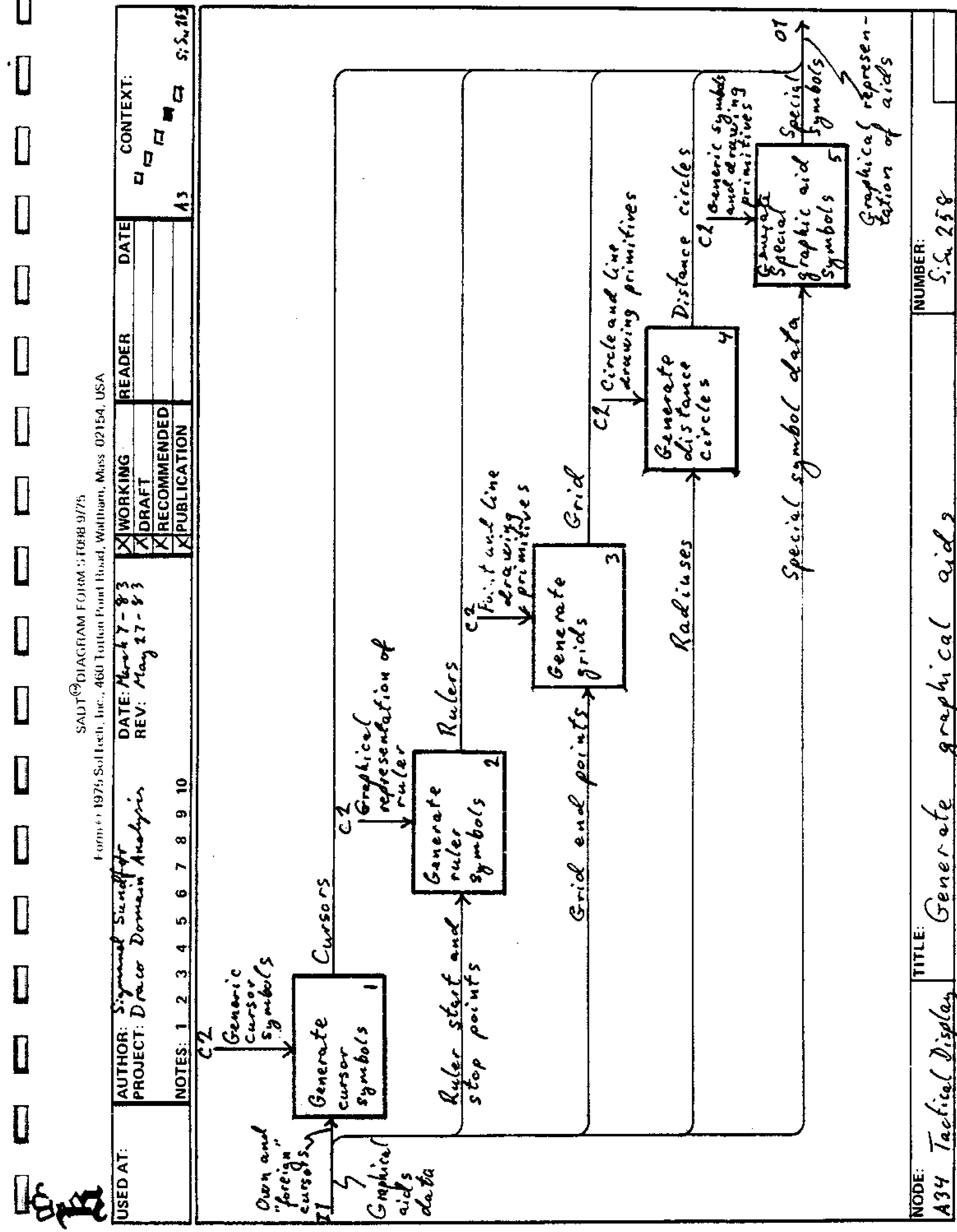


[1]: This is in the "general graphics" and "display processor" domains. It is included for the completeness of the model.

NODE:	TITLE:	NUMBER:
A5 Tactical Display	Select picture planes, colour and draw	SiSu 255


USED AT:
AUTHOR: Sigmund Sundfor
PROJECT: Draco Domain Analysis
DATE: March 7-83
REV:
NOTES: 1 2 3 4 5 6 7 8 9 10
WORKING
DRAFT
RECOMMENDED
PUBLICATION
READER
DATE
CONTEXT:
42
Su 262
SADT DIAGRAM FORM ST098 9/75
Form © 1975 SofTech, Inc., 460 Totten Pond Road, Waltham, Mass 02154, USA
C2
C1
Map i.d.
Picture planes for maps
I1
Map files
Read map data
1
Available space for map data
Type of map projection
Map data
Size of display file segment for maps
(1)
C3
Window
Clip
2
Map data inside window
Map projection algorithm (to calculate window in map coordinates)
Type of map projection
Convert to world coordinates
3
Requested maps (in world coordinates and inside window)
O1
Map projections → internal world coordinates
M1
1: Map data must be clipped at this stage to eliminate data outside display and thereby getting as much map data in as there is space for
NODE: A24 Tactical Display
TITLE: Retrieve maps
NUMBER: S:Su 259


SADT DIAGRAM FORM ST098 9/75
Form © 1975 SofTech, Inc., 460 Totten Pond Road, Waltham, Mass. 02154, USA
USED AT:
AUTHOR: Sigmund Sundfør
PROJECT: Draco Domain Analysis
DATE: March 4 - 83
REV:
NOTES: 1 2 3 4 5 6 7 8 9 10
WORKING
DRAFT
RECOMMENDED
PUBLICATION
READER
DATE
CONTEXT:
A3
S.S. 255
Targets
I1
C2
2D-projection of velocity
Calculate velocity vector
1
Target positions and velocities
Velocity vector (in 2D)
Direction of vector
Target # and allocations
Generic symbols
and allocation conversion to α-numeric
Generate target # and allocation codes
2
Target # and allocation text
Class-category symbols
Target category and class
Generate target class and category symbol
3
Target classification and category symbol
History point symbols
Clock
History on-off
Generate target history
4
Target history points
Target position history
Target position
O1
Target graphical representation
NODE: A32
TITLE: Tactical Display Generate target symbols
NUMBER: S.S. 256


SADT® DIAGRAM FORM ST088 9/75
Form © 1975 SofTech, Inc., 460 Totten Pond Road, Waltham, Mass 02154, USA
USED AT:
AUTHOR: Sigmund Sundfor
PROJECT: Draco Domain Analysis
DATE: March 7-83
REV: May 27-83
NOTES: 1 2 3 4 5 6 7 8 9 10
WORKING
DRAFT
RECOMMENDED
PUBLICATION
READER
DATE
CONTEXT:
A5
S.S. 153
I1
Graphical aids data
Own and "foreign" cursors
C2
Generic cursor symbols
Generate cursor symbols 1
Cursors
Ruler start and stop points
C2
Graphical representation of ruler
Generate ruler symbols 2
Rulers
Grid end points
C2
Font and line drawing primitives
Generate grids 3
Grid
C2
Circle and line drawing primitives
Radiuses
Generate distance circles 4
Distance circles
Special symbol data
C2
Generic symbols and drawing primitives
Generate special graphic aid symbols 5
Special symbols
O1
Graphical representation of aids
NODE: A34
Tactical Display
TITLE: Generate graphical aids
NUMBER: S.S. 258

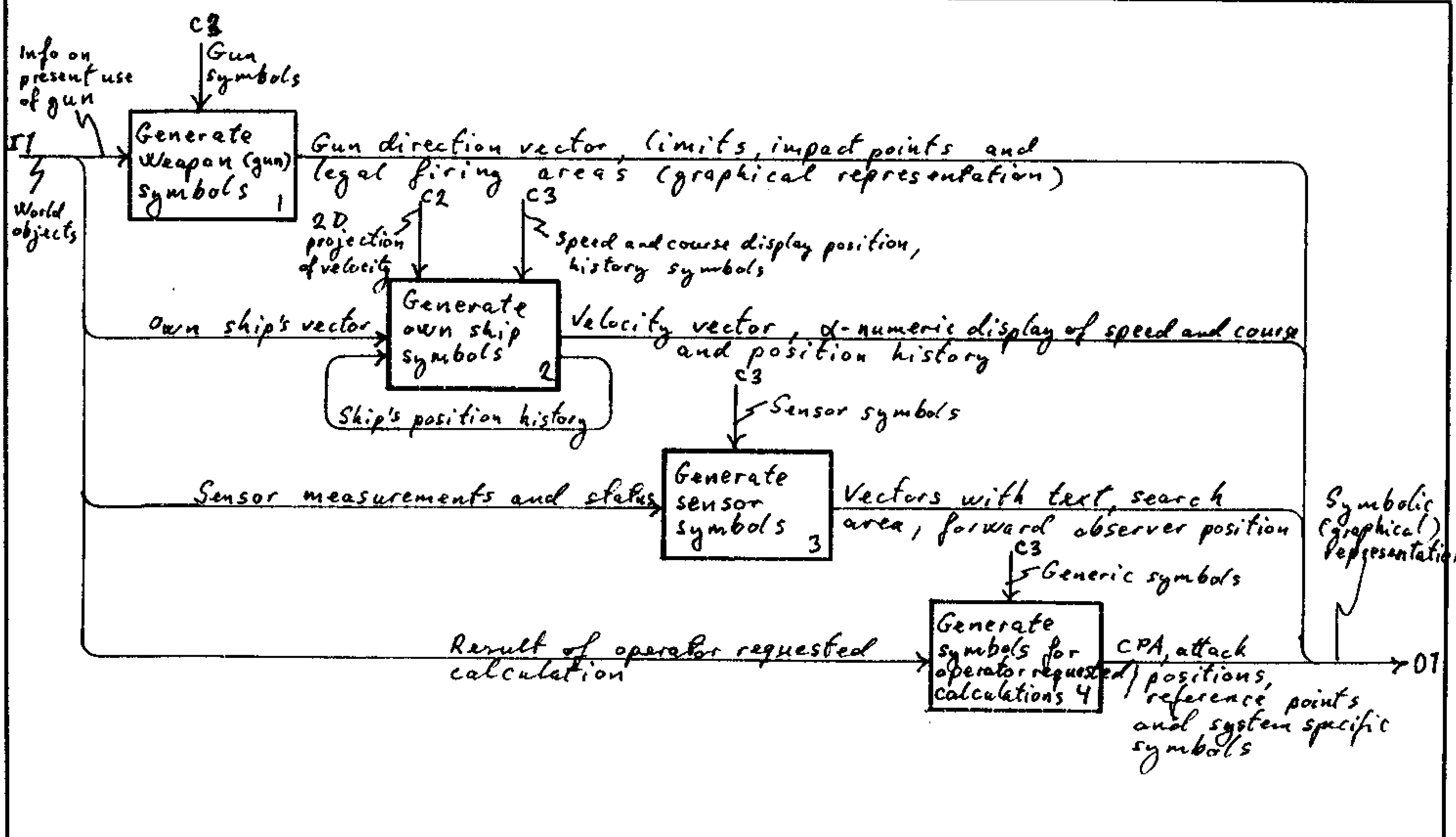
SADT(TM) DIAGRAM FORM ST098 9/75
Form © 1975 SofTech, Inc., 460 Totten Pond Road, Waltham, Mass. 02154, USA

AUTHOR: Sigmund Sundfør
PROJECT: Draco Domain Analysis
DATE: March 4-83
REV:
NOTES: 1 2 3 4 5 6 7 8 9 10

X WORKING
X DRAFT
X RECOMMENDED
X PUBLICATION

READER DATE

CONTEXT: A3 SiSu 253



NODE: A35 Tactical Display
TITLE: Generate graphical representation for other "world objects"
NUMBER: SiSu 257

6.2 The Domain Language

The final workproduct of the Draco domain analysis is the domain language. This is presented in the following. The description contains a user's manual, syntax definition, prettyprinter and a few sample system definitions using the domain language.

The user's manual is written as though the domain was fully implemented with transformations and refinements. As previously mentioned, the transformations and refinements are not defined as yet.

The syntax definition is presented in the form of the parser definition for Draco. This follows BNF, but has the addition of node constructors for the internal form tree and some other parts specific to the parser.

The language is basically a non-procedural, specification language. It covers both the specification for how it will look to the end user, and the interface to the rest of the gun control system. The language has 6 major sections:

1. Configuration
2. World model
3. Access to world model
4. Commands
5. Graphic presentation
6. Tasks

Sections 2, 3 and 4 deal primarily with the interfacing towards the rest of the gun control system. Sections 5 and 6 deal with the specification more as it is seen by the end user. Section 1 defines what objects are to be shown on the tactical plot and also the data used by the graphical plot subsystem. Section 1 is therefore a mixture of both specifications for the user and a specification of the interface.

6.2.1 User manual for Tactical Plot Domain Language

The user of this language is the builder of a specific tactical display subsystem. It is aimed primarily at ship borne gun control systems, but may be applicable to other related systems.

This language is especially developed for the definition of tactical display subsystems. It allows the system builder to specify the subsystem using terms that are familiar from the application. He/she [specifies in effect what the system is to do using terms from the application, not how the system shall do it.

The user will have to define what objects are to be displayed on the particular system and give some parameters defining the exact graphic representation, e.g. the length/speed conversion for a target's velocity vectors. Since this will be one of several subsystems, the interface to the other parts will have to be defined. This will include the definition of how the objects that are to be displayed can be retrieved, and also what commands and data the tactical plot subsystem will receive from other parts of the system. In addition to these, the editing of the display can be described, e.g. specifications like: "update all hostile targets every second and when push button #1 is pushed, colour them all red, otherwise blue" .

The language knows about the common type of objects and their representations on the tactical display. In addition, it provides the user with the facility to define new objects and symbols for them. The user is also provided with an interface facility for tieing in special procedures written in the implementation language.

The tactical plot specification is divided into the following parts:

- Head: naming the particular subsystem, dates etc. .
- Configuration: defining the objects to be displayed and configuration of the display itself like size and available colours.
- World model: definition of how the objects are stored in the systems database.

- Access to world model: definition of how the system database is accessed to get the required data on the objects.
- Commands and parameters: definition of commands and parameters like push buttons and scaling.
- Graphic representation: giving parameters for the graphic representation and also definition of special symbols.
- Tasks: definition of the rate and priority of updates as well as the editing of the screen.

These different parts are described in the following subsections.

6.2.1.1 Head

All that has to be done is defining a name for subsystem. However, it will generally be useful to attach comments to the head that specifies date, who the system builder is, what the system name or project name is and possibly a short description of the particular implementation.

6.2.1.2 Configuration

The user has to define what objects are to be displayed. The definition must distinguish between objects that are already defined for the domain and new objects. (The objects that are already defined in the domain has a representation defined and the user only has to provide some parameters. For new objects, the user will also have to define the symbols for them. This is done in the graphic representation section.)

The objects are defined in the form of relations and their attributes. The relations has also to be stated for the objects that have predefined relations because this is the definition of what objects are to be displayed.

The configuration definition shall also include the definition of display size, number of picture planes, cursors and rulers as well as the colouring available. (Colouring is used as a collective term for colour, line types, intensity and special modes like blinking.)

6.2.1.3 World Model

The world model is the definition of the part of the system database that has to be accessed from this subsystem.

In the present form of the language, the existence of a database is assumed. It is realized that this is not always the case for this type of system. The data may for example be stored in fixed, labeled positions in the memory. It is conceivable to extend the language to handle this also using the same scheme, namely: 1) define internal representation in the subsystem; 2)define the external (system) representation of the data; 3)define the mapping between the two.

The world model must be defined in terms of the syntax defined for the interface domain. That is one of the underlying domains and contains the definition of operations on the system database.

6.2.1.4 Access to World Model

This is the mapping between the internal representation of the objects and the world model (the system database).

The mapping is defined by giving the access commands to the database and defining how the resulting data bound to the internal objects (relations) and their attributes. The access commands should be defined using the syntax defined for the system database queries. The tactical plot domain will simply keep them in the way they are written here without any further processing. The refinements will be done by the interface domain.

It would have been possible for the Draco system to generate the correct access calls from a definition of the basic access mechanisms. However, this would probably limit the range of databases one could interface to, e.g. it might only be able to handle nonprocedural access commands to relational databases.

6.2.1.5 Commands and Parameters

This section defines commands and data that influences the way objects are displayed. It also allows the user to define translations of these commands and parameters into some other, internal representation. An example of the latter would be to define the range values corresponding to the set of positions on the range switch.

This is only the definition of what commands and parameters there are. The use of these are defined in the later sections.

6.2.1.6 Graphic Representation

This is the definition of what the objects should look like on the display. All objects defined in the configuration section has to be given a graphic representation here.

All objects that are built-in has a graphic representation already defined. The user only has to give a few parameters for these. Objects that are new to the domain ("new" objects in the configuration definition) has to have a complete definition of the graphic representation.

There are two types of representations provided for. One is symbols. These are defined in screen increments and characters. Symbols have a fixed size and orientation on the display regardless of scaling and rotation. The other type is a mapping from world coordinates to the display coordinates. The latter will follow the scaling and the rotation of the display.

An example of symbol is the symbol for an impact points. An example of the other type is the position of the impact point. The impact point is the (predicted) point where the projectile will hit the target.

In the case of a new object that is to be represented by a single symbol, the user will first have to state the relation and the attributes that defines the position in world coordinates. Thereafter the symbol to be centered at this location must be defined in screen coordinates. The screen coordinates are defined relative to the position of the symbol (i.e. relative to the last position given in world coordinates).

If the new object is to be a mapping from world coordinates to the display, the user will have to define the relation and attributes defining the points. In addition, the connection between the points has to be defined. This can be straight lines or circular arcs.

The representations can be made to depend upon the commands defined in the commands and parameter section. The predefined objects like cursors that have several modes of operation, can be set either to depend upon the commands and parameters or given a fixed value.

The objects predefined in the domain with a defined representation are all named in the syntax. The names used are the same as those used in the definition of the objects (the name of the relations). The same convention must be used for user defined objects.

6.2.1.7 Tasks

This section specifies 3 types of information:

- The frequency and priority with which the data are to be updated on the display.
- The condition for displaying the objects.
- The colouring of the objects. Colouring denotes use of colour, line types, intensity and any other feature that may be available for discriminating between different graphical objects.

The specification must include all the objects or collection of objects that are to appear on the screen. This is because this section defines the conditions under which the objects will be displayed.

All parts of an object need not necessarily be treated in the same manner, e.g. it is possible to specify conditions for displaying target vectors but not the target symbols. If no such condition is specified, the default is to display all.

The condition for the display statements may be stated in the form of conditions on external commands, on conditions on the attributes, or a combination of both. An example of the latter would be specifications like: "When pushbutton #3 is pressed, display all targets where the category is hostile every 500msec and with priority 4" (not strictly according to syntax).

6.2.1.8 General Syntax Rules

Names can be any string of letters and digits and the underline '_' character, but must start with a letter. Semicolon ';' is used as terminator for most of the statements. Comments can be inserted after most statements.

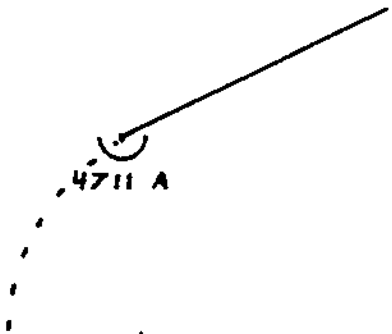
6.2.1.X.9 Syntax

For the complete syntax, refer to the Draco syntax definition. Note that the specification is nonprocedural. The order of the definition does not influence the way the processing in the final program will be performed. The syntax does actually enforce a fairly strict ordering on the specifications, which is done to improve readability and ensure completeness. It does not reflect any procedural characteristic of the final program.

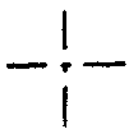
The parser for the language does not check for all consistencies one may want it to. If for example a user defined object is referenced in the task portion, there is no check that it is defined in the configuration part or the other relevant parts. Another example is that if one uses a user defined set value, there is no check that this is in fact defined. There is no check on datatypes either, but the names used in the syntax definition should make it clear which are legal in the different constructs. The user has therefore to check for these consistencies manually. Undetected errors will probably manifest themselves during some stage of the refinements, but some may end up as errors in the final program.

6.2.1.10 Predefined Objects and Graphic Symbols

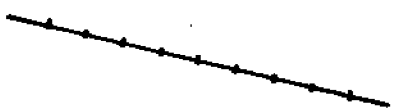
Below follows sketches of the predefined objects in the form of their graphical representation on the tactical display. The names of the different parts are also included. In addition, some general symbols are shown.



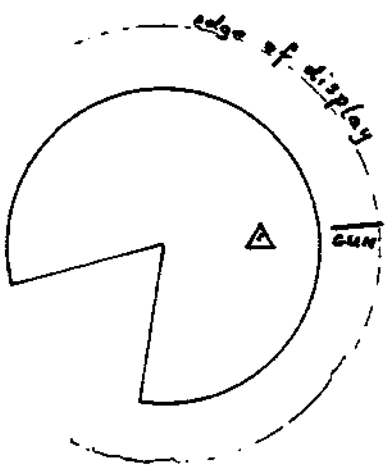
target
- vector -direction = course
  - length = how far the target will move in the specified time
- history points -past position markers
- alpha-numeric -text placed close to target



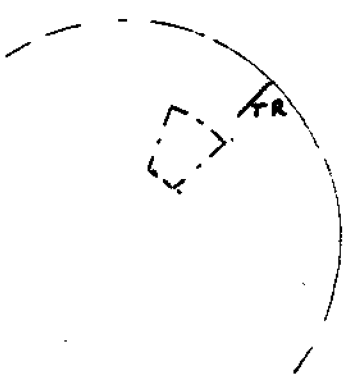
cursor
- symbol -freely specified symbol, different for each cursor



ruler
- ruler -line between end points, normally set using cursor
- scale -scale marks on ruler



gun
- vector -present direction of gun
- coverage -gun coverage limited by max-min distance and two vectors from own ship
- alpha-numeric -text placed next to vector at edge of display
- impact_symbol -symbol at predicted impact point (shell and target) if gun is fired at that time



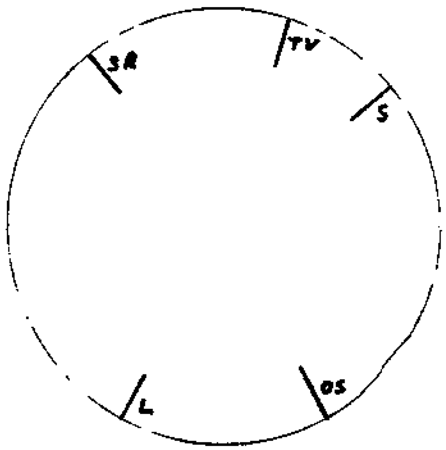
tracker_radar
- vector -present direction of antenna
- search_area -outline of search area, displayed

when in search mode
symbol -symbol at position of
the track
alpha_numeric -text placed
next to vector at edge of
display

The following objects:

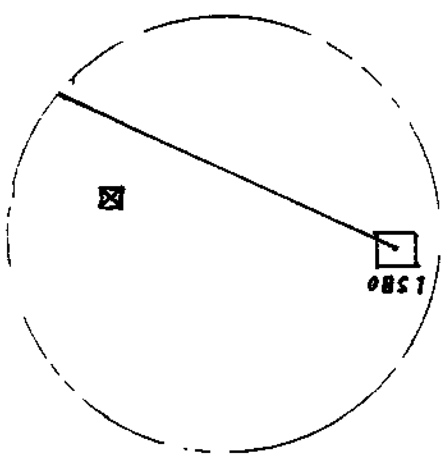
surveillance_radar
TV
optical_sight
laser_direction
sensor_platform

are all represented by:
vector -showing present
direction
alpha_numeric -text next to
vector at edge of screen,
in addition the text is
preceeded by "*" when
tracking



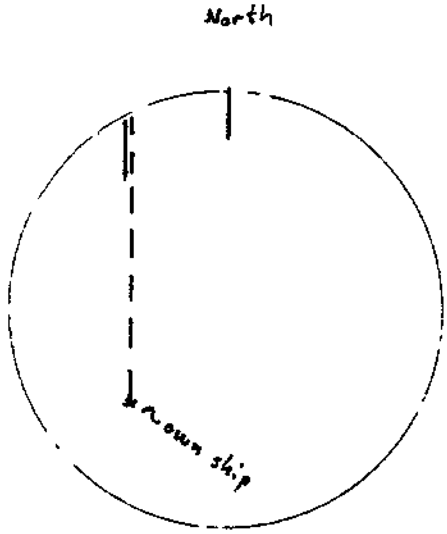
laser_measurement
symbol -at measured position
alpha_numeric -text placed
next to symbol

forward_observer
symbol -at observer position
alpha_numeric -text placed
next to symbol
vector -direction of
observation from observer
when in "direction" mode
observed_position -symbol
used for observation when
in "position" mode

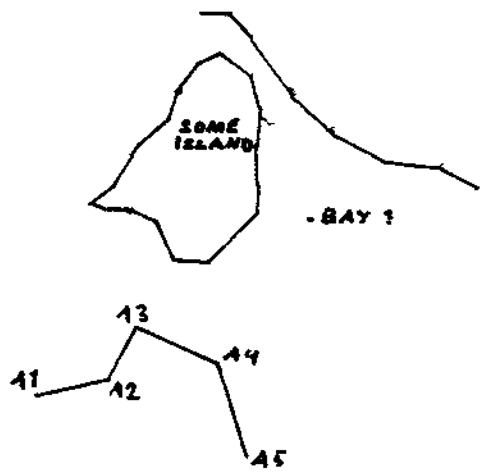


friendly_unit
graphics as for targets

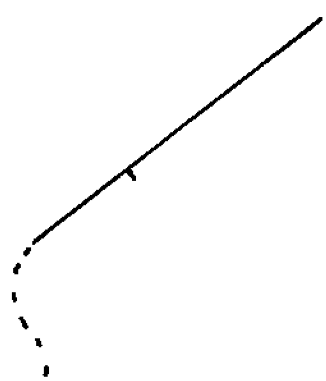
CPA
ref_point
attack_position
-are all represented by a
symbol and alpha_numeric



The direction vectors can be of 3 different types as illustrated here for a straight North direction from own ship, and with the ship off center



The map and area graphic are represented by lines between geographic positions and text (for maps) placed at specific positions. For area graphic, the points with same area "id" are joined by straight lines through the points in number order. The "id" and number are also displayed.



own_ship
   vector -similar to target vectors, but may run all the way to edge of screen with just a marker indicating the speed.
   history_points -as for target
   speed_course_display
      -alpha-numeric displaying of speed and course in a fixed position on the display

6.2.2 Syntax for the Tactical Plot Domain Language

The syntax definition below is in the form of the parser definition for Draco.

```
.DEFINE tactical_display_rule
[ TCTPR1  - Parser for Tactical Display Domain                              ]
[ by: SiSu using Draco, June 23, 1983                                       ]

[ This parser will parse a specification for a tactical display subsystem  ]
[ for ship borne gun control systems.                                       ]
[ The specification is basicly non procedural statements about what the    ]
[ system should do,  not how it is to be done.  The refinement of the      ]
[ specification will provide the 'how'.                                     ]

[ This parser definition uses the following naming conventions:            ]
[ suffix '_rule' : non terminal symbols (rules)                            ]
[ suffix '_tree' : root of right leaning tree                              ]
[ suffix '_seq'  : sequence nodes in right leaning tree                    ]
[ suffix '_def'  : denotes a definition of relation, attribute, record etc.]
[ suffix '_gr'   : the graphic representation of an object                 ]


tactical_display_rule =    comment_rule
                           "Tactical display" tctname_rule ":"
                           comment_rule
                           "Configuration:"
                              configuration_rule
                           comment_rule
                           "World model:"
                              world_model_rule
                           comment_rule
                           "Access to world model:"
                              access_rule
                           comment_rule
                           "Commands and parameters:"
                              command_rule
                           comment_rule
                           "Graphic representation:"
                              graphic_rule
                           comment_rule
                           "The tasks are:"
                              task_rule
                           comment_rule
                           .NODE(tactical_display_spec #15#14#13#12#11#10#9
                           #8#7#6#5#4#3#2#1)
                           ;

tctname_rule =             name_rule .NODE(tct_name#1);

comment_rule =             .TREE(comment_tree comment_seq
```

```
$("/*" any_string_rule "*/"
.NODE(comment#1)))
;
```

```
]
[------------------------------------------------------------------------]
[ Configuration definition syntax:                                      ]

configuration_rule =      attribute_def_rule
                          relation_def_rule
                          physical_rule
                          .NODE(config#3#2#1)
                          ;

[ Note: the attributes are global,  i.e. once an attribute is defined with  ]
[ the domain it takes its values from, the attribute can be used in several ]
[ relations.                                                                ]

attribute_def_rule =      .TREE(attribute_def_tree attribute_def_seq
                          $("/*" any_string_rule "*/" .NODE(comment#1)
                          |name_rule domain_rule .NODE(attribute_def#2#1)";"))
                          ;

[ World coordinates are real values.                                       ]
domain_rule =             "is a real number" .NODE(real)/
                          "is an integer" .NODE(integer)/
                          "is a boolean" .NODE(bool)/
                          "is a string of" integer_constant_rule "characters"
                          .NODE(string#1)/
                          "is an element from the set" set_rule
                          ;

[ The user can define new sets and the values of the sets.  These can then  ]
[ be used in the definition of new objects (relations).  In addition, the   ]
[ user has to define the set values for the following sets (if the objects  ]
[ they are used in are included):                                           ]
[   target_category, target_class       - used in target                    ]
[   tracker_radar_mode                  - used in tracker_radar             ]
[   surveillance_radar_mode             - used in surveillance_radar        ]
[   tv_mode                             - used in TV                        ]
[   unit_type                           - used in friendly_unit             ]
[ These set values are used for determening some of the graphic represen-   ]
[ tations,  and can also be used in condition statements in the task        ]
[ definitions.                                                              ]
[                                                                           ]
[ The following sets have these values:                                     ]
[   map_mode is an element from the set {off , on}; (curly brackets are     ]
[   used instead of square brackets because the parser builder uses the     ]
[   square brackets for comments.)                                          ]
[   observation_mode is an element from the set {none, direction, position};]
[   optical_sight_mode is an element from the set {no_track, tracking};     ]

set_rule =                .TREE(set_tree set_seq "["name_rule
                          $("," name_rule) "]")
                          ;
```

```
[ The definition differentiates between predefined relations (objects defined ]
[ in the domain) and new,  user defined relations.                            ]

relation_def_rule =         .TREE(relation_tree relation_seq
                            $("/*" any_string_rule "*/" .NODE(comment#1) /
                            ( "relation"
                            (  "target" target_def_rule
                            / "cursor" cursor_def_rule
                            / "ruler" ruler_def_rule
                            / "own_ship" own_ship_def_rule
                            / "gun" gun_def_rule
                            / "tracker radar" tracker_radar_def_rule
                            / "surveillance radar& surveillance_radar_def_rule
                            / "TV" uv_def_rule
                            / "optical sight" optical_sight_def_rule
                            / "laser measurement" laser_measurement_def_rule
                            / "laser direction" laser_direction_def_rule
                            / "sensor platform" sensor_platform_def_rule
                            / "map" map_def_rule
                            / "area" area_def_rule
                            / "forward observer" forward_observer_def_rule
                            / "friendly unit" friendly_unit_def_rule
                            / "CPA" cpa_def_rule
                            / "reference point" ref_point_def_rule
                            / "attack position" attack_position_def_rule
                            )
                            / "new relation" name_rule "[key" key_rule "] of"
                            attribute_list_rule ";"
                            "end" .NODE(new_relation_def#3#2#1)";"))
                            ;

target_def_rule =           "[key target_number] of"
                               "target_number,"
                               "X_position,Y_position,Z_position,"
                               "X_velocity,Y_velocity,Z_velocity,"
                               "target_category,target_class,"
                               "time_of_update;"
                            "end;"
                            .NODE(target_def)
                            ;

cursor_def_rule =           "[key number] of"
                               "number,"
                               "X_position,Y_position,Z_position,"
                               "time_of_update;"
                            "end;"
                            .NODE(cursor_def)
                            ;

ruler_def_rule =            "[key number] of"
                               "number,"
                               "X_start_position,Y_start_position,"
                               "Z_start_position,"
```

```
                              "X_end_postion,Y_end_position,Z_end_position;"
                            "end;"
                            .NODE(ruler_def)
                            ;

own_ship_def_rule =         "[key id] of"
                              "id,"
                              "north,east,"
                              "X_position,Y_position,Z_position,"
                              "X_velocity,Y_velocity,Z_velocity1"
                              "roll,pitch,heading,"
                              "time_of_update;"
                            "end;"
                            .NODE(own_ship_def)
                            ;

gun_def_rule =              "[key number] of"
                              "number,"
                              "target_number,"
                              "azimuth,elevation,"
                              "X_impact_position,Y_impact_position,"
                              "Z_impact_position;"
                            "end;"
                            .NODE(gun_def)
                            ;

tracker_radar_def_rule =    "[key number] of"
                              "number,"
                              "target_number,"
                              "tracker_radar_mode,"
                              "X_track_position,Y_track_position,"
                              "Z_track_position,"
                              "search_gate_length,"
                              "search_gate_angular_width;"
                            "end;"
                            .NODE(tracker_radar_def)
                            ;

surveillance_radar_def_rule =
                            "[key number] of"
                              "number,"
                              "surveillance_radar_mode,"
                              "azimuth;"
                            "end;"
                            .NODE(surveillance_radar_def)
                            ;


tv_def_rule =               "[key number] of"
                              "number,"
                              "tv_mode,"
                              "azimuth,elevation;"
                            "end;"
                            .NODE(tv_def)
```

```
                            ;

optical_sight_def_rule =   "[key number] of"
                              "number,"
                              "optical_sight_mode,"
                              "azimuth,elevation;"
                           "end;"
                           .NODE(optical_sight_def)
                           ;

laser_measurement_def_rule =
                           "[key number] of"
                              "number,"
                              "distance,"
                              "range_number,"
                              "X_position,Y_position,Z_position;"
                           "end;"
                           .NODE(laser_measurement_def)
                           ;

laser_direction_def_rule = "[key number] of"
                              "number,"
                              "azimuth,elevation;"
                           "end;"
                           .NODE(laser_direction_def)
                           ;

sensor_platform_def_rule = "[key number] of"
                              "number,"
                              "azimuth,"
                              "time_of_update;"
                           "end;"
                           .NODE(sensor_platform_def)
                           ;

[ Map data are assumed to be in separate files (not in database). The only     ]
[ info. from the database should be the identifier of the map to be displayed.]
[ All information on projection of map should be in the file.                  ]
map_def_rule =             "[key map_id] of"
                              "map_id,"
                              "map_mode;"
                           "end;"
                           .NODE(map_def)
                           ;

[ Areas are like maps,  except that they are typically operator defined by    ]
[ for example using the cursor.                                                ]
area_def_rule =            "[key area_id] of"
                              "area_id,"
                              "point_number,"
                              "north,east;"
                           "end;"
                           .NODE(area_def)
                           ;
```

```
forward_observer_def_rule ="[key number] of"
                              "number,"
                              bX_position,Y_position,Z_position,"
                              "observation_mode,"
                              "azimuth,elevation,"
                              "X_relative_position,Y_relative_position,"
                              "Z_relative_position,"
                              "time_of_update;"
                           "end;"
                           .NODE(forward_observer_def)
                           ;

friendly_unit_def_rule =   "[key id] of"
                              "id,"
                              "X_position,Y_position,Z_position,"
                              "X_velocity,Y_velocity,Z_velocity,"
                              "unit_type,"
                              "time_of_update;"
                           "end;"
                           .NODE(friendly_unit_def)
                           ;

cpa_def_rule =             "[key target_number] of"
                              "target_number,"
                              "X_position,Y_position,Z_position,"
                              "X_relative_ship,Y_relative_ship,"
                              "Z_relative_ship,"
                              "time_of_update;"
                           "end;"
                           .NODE(cpa_def)
                           ;

ref_point_def_rule =       "[key number] of"
                              "number,"
                              "X_position,Y_position,Z_position;"
                           "end;"
                           .NODE(ref_point_def)
                           ;

attack_position_def_rule = "[key target_number] of"
                              "target_number,"
                              "X_position,Y_position,Z_position;"
                           "end;"
                           .NODE(attack_position_def)
                           ;


key_rule =                 .TREE(key_tree key_seq name_rule $("," name_rule))
                           ;

attribute_list_rule =      .TREE(attribute_list_tree attribute_list_seq
```

```
                        name_rule $("," name_rule) )
                        ;

physical_rule =         "number of cursors is" integer_constant_rule ";"
                        .NODE(cursor_no#1)
                        "number of picture planes is"
                        integer_constant_rule ";"
                        .NODE(picture_plane_no #1)
                        "available" ("colouring"/"coloring") "is"
                        .TREE(colour_available_tree colour_available_seq
                        ("colour("/"color(") integer_constant_rule ")"
                        $("," ("colour("/"color(") integer_constant_rule")")
                        ) ";"
                        "the size of the display is" integer_constant_rule
                        "display increments"";" .NODE(display_size #1)
                        "number of rulers is" integer_constant_rule ";"
                        .NODE(ruler_no#1)
                        .NODE(physical_def#5#4#3#2#1)
                        ;
```

```
]
[ --------------------------------------------------------------------]
[ Syntax for defining world model:                                    ]
[ The user has to define the objects as they exist in the world model, i.e.]
[ the system's database.  The syntax to be used is that of the interface  ]
[ provided to the database.  In this implementation,  the definition will ]
[ basicly only be treated as a string of characters.  At a later stage it ]
[ is feasible to use this definition for checking both the definition     ]
[ and the following mapping to the objects in the domain.  However,  it   ]
[ does also as it is serve the purpose of requesting the user to define the]
[ objects to be accessed in the world model.                          ]

world_model_rule =          .TREE(world_model_tree world_model_seq
                            $("{"any_string_rule"}") )
                            ;
```

```
]
[----------------------------------------------------------------------]
[ Syntax for defining access to world model:                           ]
[ This defines the mapping between the objects as they are known in the ]
[ domain and the way they are represented in the world model.          ]

access_rule =                .TREE(access_tree access_seq
                             $("/*" any_string_rule "*/" .NODE(comment#1)
                             / relation_name_rule "access is"
                             "{" database_operation_rule "}"
                             "tctdisplay" relation_rule
                             ":=" "{" database_operation_rule "}" ";"
                             .NODE(dbaccess#4#3#2#1)
                             | relation_name_rule "update is"
                             "{"database_operation_rule"}" ":="
                             "tctdisplay" relation_rule ";"
                             "{"database_operation_rule"}"
                             .NODE(dbupdate#4#3#2#1)";"))
                             ;

database_operation_rule =    any_string_rule .NODE(database_operation #1)
                             ;

relation_rule =              relation_name_rule "(" attribute_list_rule ")"
                             .NODE(relation#2#1)
                             ;
```

```
]
[------------------------------------------------------------------------]
[ Syntax for definition of commands and parameters controlling the display:  ]

command_rule =              .TREE(command_tree command_seq
                            $("/*" any_string_rule "*/" .NODE(comment#1)
                            / (name_translation_rule | value_translation_rule |
                            on_off_command_rule | do_once_command_rule)";"))
                            ;

name_translation_rule =     "external" name_rule "is" name_rule
                            .NODE(name_translation#2#1)
                            ;

[ Value translation can only translate from a continous set of integer        ]
[ values from 0 and up and translate them to a corresponding set of parameters]
[ for the display processing.  (Set values that are represented as integers   ]
[ in the external variable, are treated as integers.) The function is intended]
[ for defining values from a range scale switch,  but may be used for other   ]
[ purposes if that is found useful.                                           ]

value_translation_rule =    name_rule "is an element from the set of ["
                            parameter_list_rule "]"
                            .NODE(value_translation#2#1)
                            ;

parameter_list_rule =       .TREE(parameter_tree parameter_seq
                            (real_parameter_rule
                            $("," real_parameter_rule)
                            | integer_parameter_rule
                            $("," integer_parameter_rule)))
                            ;

real_parameter_rule =       real_constant_rule unit_rule .NODE(parameter#2#1)
                            ;

integer_parameter_rule =    integer_constant_rule unit_rule
                            .NODE(parameter#2#1)
                            ;

unit_rule =                 ("meter/secs" / "meter/sec" / "m/s")
                            .NODE(meter_per_sec)
                            /("meter"/"m") .NODE(meter)
                            / "NM" .NODE(nautical_mile)
                            / ("secs"/"sec"/"s") .NODE(second)
                            / ("knots"/"knot") .NODE(knot)
                            ;

on_off_command_rule =       "on-off commands are" "["
                            .TREE(on_off_command_tree on_off_command_seq
                            user_defined_boolean_function_rule | name_rule
                            $("," (user_defined_boolean_function_rule
                            | name_rule))) "]"
                            ;
```

```
[ The user defined boolean function provides the user a means of tying in a  ]
[ function that returns a boolean value that can be used just like the other ]
[ commands.  No parameters can be passed to the function.                    ]

user_defined_boolean_function_rule =
                           "boolean_function" "(" name_rule ")"
                           .NODE(boolean_function#1)
                           ;

do_once_command_rule =     "do-once commands are"
                           "[" .TREE(do_once_command_tree do_once_command_seq
                           record_def_rule $("," record_def_rule)) "]"
                           ;

record_def_rule =          name_rule "(" name_list_rule ")"
                           .NODE(record_def#2#1)
                           ;
```

```
]
[ ---------------------------------------------------------------------]
[ Syntax for defining the graphic representation:                      ]

graphic_rule =             .TREE(graphic_tree graphic_seq
                           $( "target graphic:" target_gr_rule
                           / "cursor graphic:" cursor_gr_rule
                           / "ruler graphic:" ruler_gr_rule
                           / "own_ship graphic:" own_ship_gr_rule
                           / "gun graphic:" gun_gr_rule
                           / "tracker radar graphic:" tracker_radar_gr_rule
                           / "surveillance radar graphic:"
                             surveillance_radar_gr_rule
                           / "TV graphic:" tv_gr_rule
                           / "optical sight graphic:" optical_sight_gr_rule
                           / "laser measurement graphic:"
                             laser_measurement_gr_rule
                           / "laser direction graphic:"
                             laser_direction_gr_rule
                           / "sensor platform graphic:" sensor_platform_gr_rule
                           / "map graphic:" map_gr_rule
                           / "area graphic:" area_gr_rule
                           / "forward observer graphic:"
                             forward_observer_gr_rule
                           / "friendly unit graphic:" friendly_unit_gr_rule
                           / "CPA graphic:" cpa_gr_rule
                           / "reference point graphic:" ref_point_gr_rule
                           / "attack position graphic:" attack_position_gr_rule
                           / "/*" any_string_rule "*/" .NODE(comment#1)
                           | user_defined_object_gr_rule))
                           ;


target_gr_rule =           "vector length =" integer_constant_rule "seconds"";"
                           "length limit =" integer_constant_rule "knots"";"
                           "time between history points ="
                           integer_constant_rule "seconds"";"
                           "number of history points ="
                           integer_constant_rule";"
                           "alpha-numeric is [" alpha_tree_rule "]" ";"
                           "target symbols:" "[" target_symbol_rule"]" ";"
                           .NODE(target_gr#6#5#4#3#2#1)
                           ;

target_symbol_rule =       .TREE(target_symbol_tree target_symbol_seq
                           $(target_category_rule "," target_class_rule
                           "="  gr_symbol_rule
                           .NODE(target_symbol#3#2#1)))
                           ;

target_category_rule =     name_rule .NODE(target_category #1)
                           ;

target_class_rule =        name_rule .NODE(target_class #1)
```

```
                           ;
cursor_gr_rule =           "cursor""("integer_constant_rule")"":"
                              "cursor symbol =" gr_symbol_rule
                              "true motion =" (boolean_constant_rule |
                              command_name_rule) ";"
                              "center cursor =" (boolean_constant_rule |
                              command_name_rule)";"
                              "own ship to cursor =" (boolean_constant_rule |
                              command_name_rule)";"
                              "cursor to own ship =" (boolean_constant_rule |
                              command_name_rule)";"
                              "cursor movement X =" real_rule ";"
                              "cursor movement Y =" real_rule ";"
                           .NODE(cursor_gr#8#7#6#5#4#3#2#1)
                           ;
ruler_gr_rule =            "scale on ruler =" (boolean_constant_rule |
                           command_name_rule)";"
                           "distance between scale marks ="
                           integer_constant_rule "meter"";"
                           .NODE(ruler_gr#3#2#1)
                           ;
own_ship_gr_rule =         "velocity vector type =" vector_type_rule ";"
                           "speed marker / speed vector length ="
                           integer_constant_rule "seconds"";"
                           "time between history points ="
                           integer_constant_rule "seconds"";"
                           "number of history points ="
                           integer_constant_rule ";"
                           "alpha numeric speed and course display ="
                           boolean_constant_rule ";"
                           "screen position of speed and course display ="
                           "("integer_constant_rule","integer_constant_rule");"
                           .NODE(own_ship_gr#7#6#5#4#3#2#1)
                           ;
gun_gr_rule =              "direction vector type =" vector_type_rule ";"
                           "coverage =""-"integer_constant_rule "degrees"
                           "to""+"integer_constant_rule "degrees"",""range"
                           integer_constant_rule "meter"";"
                           "alpha-numeric is ["alpha_tree_rule"]"";"
                           "impact symbol =" gr_symbol_rule ";"
                           .NODE(gun_gr#6#5#4#3#2#1)
                           ;
tracker_radar_gr_rule =    "direction vector type =" vector_type_rule";"
                           "search mode =" (boolean_constant_rule |
                           set_constant_rule)";"
                           "tracker_symbol =" gr_symbol_rule ";"
                           "alpha-numeric is""[" alpha_tree_rule "]"";"
                           .NODE(tracker_radar_gr#4#3#2#1)
                           ;
```

```
surveillance_radar_gr_rule =
                            "direction vector type =" vector_type_rule ";"
                            "alpha-numeric is""[" alpha_tree_rule "]"";"
                            .NODE(surveillance_radar_gr#2#1)
                            ;

tv_gr_rule =                "direction vector type =" vector_type_rule ";"
                            "track mode =" (boolean_constant_rule |
                            set_constant_rule) ";"
                            "alpha-numeric when tracking is"
                            "[" alpha_tree_rule "]"";"
                            "otherwise alpha-numeric is"
                            "[" alpha_tree_rule "]"";"
                            .NODE(tv_gr#4#3#2#1)
                            ;

optical_sight_gr_rule =     "direction vector type =" vector_type_rule ";"
                            "alpha-numeric when tracking is"
                            "[" alpha_tree_rule "]"";"
                            "otherwise alpha-numeric is"
                            "[" alpha_tree_rule "]"";"
                            .NODE(optical_sight_gr#3#2#1)
                            ;

laser_measurement_gr_rule =
                           "alpha-numeric is" "[" alpha_tree_rule "]"";"
                           "laser measurement symbol =" gr_symbol_rule ";"
                           .NODE(laser_measurement_gr#2#1)
                           ;

laser_direction_gr_rule =  "direction vector type =" vector_type_rule ";"
                           "alpha-numeric is""[" alpha_tree_rule "]"";"
                           .NODE(laser_direction_gr#2#1)
                           ;

sensor_platform_gr_rule =  "direction vector type =" vector_type_rule ";"
                           "alpha-numeric is""[" alpha_tree_rule "]"";"
                           .NODE(sensor_platform_gr#2#1)
                           ;

[ Maps and area graphic have a standard graphic representation.        ]

map_gr_rule =              "standard"";" .NODE(map_gr)
                           ;

area_gr_rule =             "standard"";" .NODE(area_gr)
                           ;

forward_observer_gr_rule =
                           "alpha-numeric is""[" alpha_tree_rule "]"";"
                           "observer symbol =" gr_symbol_rule ";"
                           "observation vector type =" vector_type_rule ";"
                           "symbol for observed position =" gr_symbol_rule ";"
```

```
                              .NODE(forward_observer_gr#4#3#2#1)
                              ;

friendly_unit_gr_rule =        "vector length =" integer_constant_rule "seconds"";"
                               "length limit =" integer_constant_rule "knots"";"
                               "time between history points ="
                               integer_constant_rule "seconds"";"
                               "number of history points ="
                               integer_constant_rule";"
                               "alpha-numeric is" "[" alpha_tree_rule "]" ";"
                               "unit symbols:" "[" unit_symbol_rule"]" ";"
                               .NODE(friendly_unit_gr#6#5#4#3#2#1)
                               ;

unit_symbol_rule =            .TREE(unit_symbol_tree unit_symbol_seq
                               $(unit_type_rule "="
                               gr_symbol_rule  ";"
                               .NODE(unit_symbol#2#1)))
                               ;

unit_type_rule =               name_rule .NODE(unit_type#1)
                               ;

[ CPA (Closest Point of Approach) can be displayed in two ways.  It can be  ]
[ displayed as the actual geographic position the target is predicted to be ]
[ at when own ship and target will be closest.  The other possibility is to ]
[ show the position relative to own ship like it will be when they come to  ]
[ the CPA.  For navigation purposes,  the latter is normally preferred as it]
[ shows what side the target will pass on.                                  ]

cpa_gr_rule =                  "alpha-numeric is""[" alpha_tree_rule "]"";"
                               "symbol =" gr_symbol_rule ";"
                               "relative to own ship =" boolean_constant_rule ";"
                               .NODE(cpa_gr#3#2#1)
                               ;

ref_point_gr_rule =            "alpha-numeric is""[" alpha_tree_rule "]"";"
                               "symbol =" gr_symbol_rule ";"
                               .NODE(ref_point_gr#2#1)
                               ;

attack_position_gr_rule =      "alpha-numeric is""[" alpha_tree_rule "]"";"
                               "symbol =" gr_symbol_rule ";"
                               .NODE(attack_position_gr#2#1)
                               ;

[ There are four types of vectors (apart from those defined in symbol and     ]
[ world rules):                                                               ]
[ 1: speed_vector - is used to represent velocity.  The direction is equal    ]
[    to the velocity vector's,  while the length is a user defined function   ]
[    of the object's speed.                                                   ]
[ 2: point_to_edge_vector - indicates an angle by drawing a vector from the   ]
[    point to the edge of the screen.                                         ]
[ 3: edge_marker - is like the point_to_edge_vector except that it is only    ]
```

```
[    a short vector at the edge of the screen.                              ]
[ 4: compass_marker - indicates an azimuth angle by drawing a short vector  ]
[    at the edge of the screen as though the screen was a compass.          ]

[    The difference between the last two vectors is that the first one is   ]
[    always drawn relative to to the point given (usually own ship) while the]
[    second one is drawn as though the display was a compass.  When,  say   ]
[    own ship is in the display centre,  they will be equal,  but not when  ]
[    own ship is off centre.                                                ]

vector_type_rule =          "speed_vector" .NODE(speed_vector)
                            / "point_to_edge_vector" .NODE(point_to_edge_vector)
                            / "edge_marker" .NODE(edge_marker)
                            / "compass_marker" .NODE(compass_marker)
                            ;

alpha_tree_rule =           .TREE(alpha_tree alpha_seq
                            $("digit(" integer_constant_rule
                            ".." integer_conwtant_rule ")" "="
                            relation_variable_rule ";"
                            .NODE(alpha_num_gr#3#2#1)))
                            ;

gr_symbol_rule =            "symbol" "("
                            .TREE(symbol_tree symbol_seq
                            $("(arc" $<1:3>("," screen_coordinate_rule) ")"
                            .NODE(arc#3#2#1)
                            /"(circle""," integer_constant_rule
                            .NODE(circle#1) ")"
                            /"(point"","screen_coordinate_rule .NODE(point#1)")"
                            /"(string"","(relation_variable_rule |
                            record_variable_rule | string_rule)
                            .NODE(string#1) ")"
                            /"(vector" .TREE(vector_tree vector_seq
                            $("," screen_coordinate_rule)) ")" )) ")" ";"
                            ;


screen_coordinate_rule =    integer_constant_rule ","
                            integer_constant_rule .NODE(screen_xy#2#1)
                            ;

[ The user can define graphics for new objects as shown below.             ]
[ The way it is done is by giving the name of the object followed by the   ]
[ the text "graphic:".  The graphic is then defined as straight line,      ]
[ circular arcs and alphanumeric. The graphic can be in both screen        ]
[ coordinates and world coordinates.  The screen coordinates will allways be]
[ relative to the last given world coordinate pair.                        ]
[ It may be specified that different parts of the object's graphic repre-  ]
[ sentation should be displayed under various conditions.  The user have   ]
[ therefore the option to name different elements of the graphic so that   ]
[ they subsequently can be addressed specificly in the definition of tasks. ]
```

```
user_defined_object_gr_rule =
                            relation_name_rule "graphic:"
                            (.TREE(object_gr_tree object_gr_seq
                            ("element" element_name_rule ":"
                            element_gr_rule .NODE(element#2#1)
                            $("element" element_name_rule
                            ":" element_gr_rule .NODE(element#2#1))
                            / $(object_gr_rule)))).NODE(user_defined_object#2#1)
                            ;

element_gr_rule =           .TREE(element_tree element_seq
                            $(object_gr_rule))
                            ;

element_name_rule =         name_rule .NODE(element_name #1)
                            ;

object_gr_rule =            gr_world_rule  | gr_symbol_rule
                            .NODE(object_gr #1)
                            ;

gr_world_rule =             "world coordinates" "("
                            .TREE(world_tree world_seq
                            $("(arc" $<1:3>("," world_coordinate_rule) ")"
                            .NODE(w_arc#3#2#1)
                            /"(circle"","  real_rule .NODE(w_circle#1) ")"
                            /"(point"",''world_coordinate_rule
                            .NODE(w_point#1)")"
                            /"(string"","  (relation_variable_rule |
                            record_variable_rule | string_rule) ")"
                            .NODE(w_string#1)
                            /"(vector" .TREE(w_vector_tree w_vector_seq
                            $("," world_coordinate_rule)) ")" )) ")" ";"
                            ;

world_coordinate_rule =     real_rule "," real_rule .NODE(world_xy #2#1)
                            ;

[ The operators take precedence from left to right!                     ]
real_rule =                 real_primary_rule("+" real_rule .NODE(real_add#2#1)
                            / "-" real_rule .NODE(real_minus#2#1)
                            / "*" real_rule .NODE(real_mpy#2#1)
                            / "/" real_rule .NODE(real_div#2#1)
                            / .EMPTY .NODE(real_value#1))
                            ;

real_primary_rule =         ( real_constant_rule
                            | user_defined_real_function_rule
                            | relation_variable_rule
                            | record_variable_rule)
                            .NODE(real_value #1)
                            ;

[ The user defined real function does like the boolean function provide a   ]
```

```
[ means for tying in special functions that cannot be specified in the syntax]
[ of this domain. The function can return one single real value. No parameter]
[ passing to it is possible.                                                  ]

user_defined_real_function_rule =
                          "real_function" "(" name_rule ")"
                          .NODE(real_function#1)
                          ;

relation_variable_rule =  relation_name_rule "(" attribute_name_rule ")"
                          .NODE(relation_variable#2#1)
                          ;

relation_name_rule =      name_rule .NODE(relation_name#1)
                          ;

attribute_name_rule =     name_rule .NODE(attribute#1)
                          ;

record_variable_rule =    record_name_rule "(" field_name_rule ")"
                          .NODE(record_variable#2#1)
                          ;

record_name_rule =        name_rule .NODE(record#1)
                          ;

field_name_rule =         name_rule .NODE(field#1)
                          ;
```

```
]
[------------------------------------------------------------------------]
[ Syntax definition for tasks:                                           ]

task_rule =                  .TREE(task_tree task_seq
                             $("/*" any_string_rule "*/" .NODE(comment#1)
                             / (condition_rule
                             |  action_rule )";"))
                             ;

condition_rule =             "when command is" command_name_rule
                             "then"
                                 task_rule
                             ("otherwise"
                                 task_rule
                             .NODE(when_otherwise #3#2#1)
                             / .EMPTY .NODE(when #2#1))
                             ;

command_name_rule =          name_rule .NODE(command_name#1)
                             ;

[ The actions specified here are:                                        ]
[    display - must be specified for all objects to be displayed and     ]
[              defines the update frequency and priority to be used under ]
[              the given conditions.                                     ]
[    show    - is used to name graphic parts of objects such that one part]
[              may be treated different to the rest of an object, e.g. the]
[              target vector may have different colour than the rest of the]
[              target graphic.  If no show statement refers to an object, ]
[              the default is to treat all parts in the same way.        ]
[    colour  - defines colouring which is here taken to cover all things ]
[              like use of colour,  different line types and different   ]
[              intensities.  Colouring can be specified for complete     ]
[              objects or part of objects.  All objects must have a      ]
[              specified colour.                                         ]

action_rule =                "display" tuple_selection_rule
                             "every" frequency_rule ","
                             "priority" priority_rule .NODE(display#3#2#1)
                             / "show" gr_element_selection_rule .NODE(show#1)
                             / ("colour"/"color")
                             (gr_element_selection_rule | tuple_selection_rule)
                             colouring_rule
                             .NODE(colour#2#1)
                             ;

frequency_rule =             integer_constant_rule "sec" .NODE(freq_seconds#1)
                             | integer_constant_rule "msec"
                             .NODE(freq_milliseconds#1)
                             ;

priority_rule =              integer_constant_rule .NODE(priority#1)
                             ;
```

```
colouring_rule =            ("colour" / "color") "(" integer_constant_rule ")"
                            .NODE(colouring#1)
                            ;

tuple_selection_rule =      "all" relation_name_rule
                            ("where" relation_condition_rule
                            .NODE(select_tuples#2#1)
                            / .EMPTY .NODE(all_tuples#1))
                            ;

relation_condition_rule =   attribute_name_rule "is not" domain_value_rule
                            .NODE(is_not#2#1)
                            | attribute_name_rule "is" domain_value_rule
                            .NODE(is#2#1)
                            | attribute_name_rule "=" domain_value_rule
                            .NODE(equal#2#1)
                            | attribute_name_rule ">" domain_value_rule
                            .NODE(greater_than#2#1)
                            | attribute_name_rule "<" domain_value_rule
                            .NODE(less_than#2#1)
                            ;

domain_value_rule =         (real_constant_rule
                            | integer_constant_rule
                            | set_constant_rule)
                            .NODE(domain_value#1)
                            ;

gr_element_selection_rule =
                            "all" relation_name_rule "graphic"
                            ("except" gr_element_name_rule
                            .NODE(select_gr_element#2#1)
                            / .EMPTY .NODE(all_gr_element#1))
                            | gr_element_name_rule "of" relation_name_rule
                            .NODE(one_gr_element#2#1)
                            ;

[ The graphic representation of the objects consists of several parts. These ]
[ can be selected and treated separatly from the rest of the object.  The    ]
[ names of these parts are defined for all the predefined objects mentioned  ]
[ in the configuration and graphic representation sections.  In addition,    ]
[ the user may also name parts of user defined objects.                      ]
[ The predefined names refers to general parts used for many of the objects: ]
[ vector, alpha_numeric and symbol (symbol specified in screen coordinates,  ]
[ centered at position of object). In addition there are some special parts. ]
[ It should be clear from the names what parts they refer to.  Further       ]
[ explanation is therefore not given.  Map and  area graphic do not have     ]
[ named parts.                                                               ]

gr_element_name_rule =     "vector" .NODE(vector_element)/
                           "symbol" .NODE(symbol_element)/
                           "alpha-numeric" .NODE(alpha_numeric_element)/
                           "history points" .NODE(history_points_element)/
```

```
"gun coverage" .NODE(gun_coverage_element)/
"speed and course display"
.NODE(speed_course_display)/
name_rule .NODE(user_defined_element#1)
;
```

```
]
[-----------------------------------------------------------------------]
[ Syntax rules common to several of the sections:                       ]

name_list_rule =          .TREE(name_tree name_seq name_rule $(","name_rule))
                          ;

any_string_rule =         any_string .LITERAL
                          ;

any_string:               .TOKEN $(.ANYBUT('{ ! '} ! '*)) .DELTOK
                          ;

string_rule =             string .LITERAL
                          ;

string:                   .TOKEN $<1:?>(.ANYBUT('/ ! '* ! '( ! ') )) .DELTOK
                          ;

set_constant_rule =       name_rule .NODE(set_constant#1)
                          ;

real_constant_rule =      real_constant .NODE(real_constant *)
                          ;

real_constant:            PREFIX .TOKEN $<0:1>(sign) $(digit)
                          .ANY('.) $(digit) .DELTOK
                          ;

boolean_constant_rule =   ("true"/"TRUE") .NODE(true)
                          / ("false"/"FALSE").NODE(false)
                          ;

integer_constant_rule =   integer_constant .NODE(integer_constant *)
                          ;

integer_constant:         PREFIX .TOKEN $<0:1>(sign) $<1:?>(digit) .DELTOK
                          ;

name_rule =               name .LITERAL
                          ;

name:                     PREFIX .TOKEN letter $(char) .DELTOK
                          ;

char:                     .ANY('A:'Z ! 'a:'z ! '0:'9 ! '_)
                          ;

letter:                   .ANY('A:'Z ! 'a:'z)
                          ;

digit:                    .ANY('0:'9)
                          ;
```

```
sign:                  .ANY('+ ! '-)
                       ;

[ PREFIX scans off any blanks, carriage returns and line feeds.        ]

PREFIX:                $(.ANY(32 ! 13 ! 10))
                       ;

.END
```

6.2.3 Prettyprinter for the Tactical Plot Domain Language

Below follows the prettyprinter definition:

```
.PRETTYPRINTER TCTPR1

access_seq =                #1 .SLM ;
access_tree =               .TREEPRINT(access_seq,1,,);
all_gr_element =            "all " #1 " graphic ";
all_tuples =                "all " #1 ;
alpha_num_gr =              " digit(" #1 ".." #2 ") = " #3 ";";
alpha_seq =                 #1 .SLM ;
alpha_tree =                .TREEPRINT(alpha_seq,1,,);
arc =                       "(arc, " #1 "," #2 "," #3 ")";
attribute =                 #1 ;
attribute_def =             #1 #2 ";" .SLM;
attribute_def_seq =         #1 ;
attribute_def_tree =        .TREEPRINT(attribute_def_seq,1,,) ;
attribute_list_seq =        #1 ;
attribute_list_tree =       .TREEPRINT(attribute_list_seq,1, ",".SLM , ";");
bool =                      " is a boolean" ;
boolean_function =          "boolean function(" #1 ")" ;
circle =                    "(circle," #1 ")" ;
colour =                    " colour " #1 #2 ";" .SLM ;
colouring =                 " colour(" #1 ")" ;
colour_available_seq =      " colour(" #1 ")" ;
colour_available_tree =     .TREEPRINT(colour_available_seq,1, ",",";") ;
command_name =              #1 ;
command_seq =               #1 ;
command_tree =              .TREEPRINT(command_seq,1,,.SLM) ;
comment =                   .SLM "/* " #1 "*/"  ;
comment_seq =               #1 ;
comment_tree =              .TREEPRINT(comment_seq,1,,) ;
config =                    #1 #2 #3 ;
cursor_no =                 .SLM "number of cursors is " #1 ";" .SLM ;
database_operation =        #1 ;
dbaccess =                  .SLM #1 " access is " .LM(+3)
                               .SLM "{" #2 "}"
                               .SLM "tctdisplay " #3
                               .SLM " := {" #4 "};" .LM(0) ;
dbupdate =                  .SLM #1 " update is " .LM(+3)
                               .SLM "{" #2 "}"
                               .SLM " := tctdisplay " #3 ";"
                               .SLM "{" #4 "};" .LM(0) ;
display =                   "display " #1 .SLM "every " #2 ",  priority " #3 ";" ;
display_size =              .SLM "the size of the display is " #1
                            " display increments ;" ;
do_once_command_seq =       #1 ;
do_once_command_tree =      "do-once commands are" .LM(+3)
                               .SLM "[" .TREEPRINT(do_once_coomand_seq,1,",","];")
                               .LM(0) ;
domain_value =              #1 ;
element =                   "element " #1 " :" .LM(+3)
                               .SLM #2 .LM(0) ;
```

```
element_name =           #1 ;
element_seq =            #1 ;
element_tree =           .TREEPRINT(element_seq,1,.SLM,9 ;
equal =                  #1 " = " #2 ;
false =                  " false" ;
field =                  #1 ;
graphic_seq =            #1 ;
graphic_tree =           .TREEPRINT(graphic_seq,1,.SLM,) ;
greater_than =           #1 " > " #2 ;
integer =                " is an integer " ;
integer_constant =       #1 ;
is_not =                 #1 " is not " #2 ;
is =                     #1 " is " #2 ;
key_seq =                #1 ;
key_tree =               .TREEPRINT(key_seq,1,",",) ;
knot =                   " knot" ;
less_than =              #1 " < " #2 ;
meter =                  " meter" ;
meter_per_sec =          " meter/sec" ;
freq_seconds =           #1 " sec" ;
freq_milliseconds =      #1 " msec" ;
name_seq =               #1 ;
name_translation =       "external " #1 " is " #2 ";" ;
name_tree =              .TREEPRINT(name_seq,1,",",) ;
nautical_mile =          " NM" ;
new_relation_def =       "new relation " #1 " [key " #2 "] of" .LM(+3)
                            .SLM #3 .LM(0)
                         .SLM "end;" ;
object_gr =              #1 ;
object_gr_seq =          #1 ;
object_gr_tree =         .TREEPRINT(object_gr_seq,1,.SLM,) ;
on_off_command_seq =     #1 ;
on_off_command_tree =    "on-off commands are" .LM(+3)
                            .SLM "["
                            .TREEPRINT(on_off_command_seq,1,",".SLM,"];")
                            .LM(0) ;
one_gr_element =         #1 " of " #2 ;
parameter =              #1 " " #2 ;
parameter_seq =          #1 ;
parameter_tree =         .TREEPRINT(parameter_seq,1,",",) ;
physical_def =           #1 #2 #3 #4 #5 ;
picture_plane_no =       .SLM "number of picture planes is " #1 " ;" ;
point =                  "(point, " #1 ")" ;
priority =               #1 ;
real =                   " is a real number " ;
real_add =               #1 " + " #2 ;
real_constant =          #1 ;
real_div =               #1 " / " #2 ;
real_function =          "real function(" #1 ")" ;
real_minus =             #1 " - " #2 ;
real_mpy =               #1 " * " #2 ;
real_value =             #1 ;
record =                 #1 ;
record_def =             #1 "(" #2 ")" ;
```

```
record_variable =          #1 "(" #2 ")" ;
relation =                 #1 " (" #2 ")" ;
relation_name =            #1 ;
relation_seq =             #1 ;
relation_tree =            .TREEPRINT(relation_seq,1,.SLM,) ;
relation_variable =        #2 "(" #1 ")" ;
ruler_no =                 .SLM "number of rulers is " #1 ";" ;
screen_xy =                #1 "," #2 ;
second =                   "sec" ;
select_gr_element =        "all " #1  " graphic except " #2 ;
select_tuples =            "all " #1 " where " #2 ;
set_constant =             #1 ;
set_seq =                  #1 ;
set_tree =                 " is an element from the set of ["
                           .TREEPRINT(set_seq,1,",","]") ;
show =                     .SLM "show " #1 " ;" ;
string =                   "(string, " #1 ")" ;
symbol_seq =               #1 ;
symbol_tree =              "symbol(" .TREEPRINT(symbol_seq,1,.SLM,) ");" ;
tactical_display_spec =    #1
                           .SLM "Tactical display " #2 " :" .LM(+3)
                           .SLM #3
                           .SLM "Configuration:" .LM(+3) .SLM #4 .LM(0)
                           .SLM #5
                           .SLM "World model:" .LM(+3) .SLM #6 .LM(0)
                           .SLM #7
                           .SLM "Access to world model:" .LM(+3) .SLM #8 .LM(0)
                           .SLM #9
                           .SLM "Commands and parameters:" .LM(+3).SLM #10 .LM(0)
                           .SLM #11
                           .SLM "Graphic representation:" .LM(+3) .SLM #12 .LM(0)
                           .SLM #13
                           .SLM "The tasks are:" .LM(+3) .SLM #14 .LM(0)
                           .SLM #15 ;
task_seq =                 #1 ;
task_tree =                .TREEPRINT(task_seq,1,.SLM,) ;
tct_name =                 #1 ;
true =                     " true" ;
user_defined_object =      #1 " graphic:" .LM(+3)
                              .SLM #2 .LM(0) ;
value_translation =        #1 " is an element from the set of [" #2 "];" ;
vector_seq =               #1 ;
vector_tree =              "(vector, " .TREEPRINT(vector_seq,1,",",")") ;
w_arc =                    "(arc, " #1 " , " #2 " , " #3 ")" ;
w_circle =                 "(circle, " #1 ")" ;
w_point =                  "(point, " #1 ")" ;
w_string =                 "(string, " #1 ")" ;
w_vector_seq =             #1 ;
w_vector_tree =            "(vector, " .TREEPRINT(w_vector_seq,1,",",")") ;
when =                     "when command is " #1
                           .SLM "then" .LM(+3)
                              .SLM c2 .LM(0) .SLM ";" ;
when_otherwise =           "when command is " #1
                           .SLM "then" .LM(+3)
```

```
                           .SLM #2 .LM(0)
                        .SLM "otherwise" .LM(+3)
                           .SLM #3 .LM(0) .SLM ";" ;
world_model_seq =       #1  ;
world_model_tree =      "{" .TREEPRINT(world_model_seq,1,"}".SLM"{","}");
world_seq =             #1 ;
world_tree =            "world coordinates(".TREEPRINT(world_seq,1,.SLM,)");";
world_xy =              #1 "," #2 ;

target_def =            "relation target [key target_number] of" .LM(+3)
                           .SLM "target_number,"
                           .SLM  "X_position,Y_position,Z_position,"
                           .SLM  "X_velocity,Y_velocity,Z_velocity,"
                           .SLM  "target_category,target_class,"
                           .SLM  "time_of_update;" .LM(0)
                        .SLM  "end;"
                        ;

cursor_def =            "relation cursor [key number] of" .LM(+3)
                           .SLM  "number,"
                           .SLM  "X_position,Y_position,Z_position,"
                           .SLM  "time_of_update;" .LM(0)
                        .SLM  "end;"
                        ;

ruler_def =             "relation ruler [key number] of" .LM(+3)
                           .SLM  "number,"
                           .SLM  "X_start_position,Y_start_position,"
                           .SLM  "Z_start_position,"
                           .SLM "X_end_postion,Y_end_position,Z_end_position;"
                        .LM(0) .SLM  "end;"
                        ;

own_ship_def =          "relation own_ship [key id] of" .LM(+3)
                           .SLM  "id,"
                           .SLM  "north,east,"
                           .SLM  "X_position,Y_position,Z_position,"
                           .SLM  "X_velocity,Y_velocity,Z_velocity,"
                           .SLM  "roll,pitch,heading,"
                           .SLM  "time_of_update;" .LM(0)
                        .LM(0)  "end;"
                        ;

gun_def =               "relation gun [key number] of" .LM(+3)
                           .SLM  "number,"
                           .SLM  "target_number,"
                           .SLM  "azimuth,elevation,"
                           .SLM  "X_impact_position,Y_impact_position,"
                           .SLM  "Z_impact_position," .LM(0)
                        .LM(0)  "end;"
                        ;

tracker_radar_def =     "relation tracker radar [key number] of" .LM(+3)
                           .SLM  "number,"
```

```
                            .SLM  "target_number,"
                            .SLM  "tracker_radar_mode,"
                            .SLM  "X_track_position,Y_track_position,"
                            .SLM  "Z_track_position,"
                            .SLM  "search_gate_length,"
                            .SLM  "search_gate_angular_width," .LM(0)
                         .SLM  "end;"
                         ;

surveillance_radar_def = "relation surveillance radar [key number] of" .LM(+3)
                            .SLM  "number,"
                            .SLM  "surveillance_radar_mode,"
                            .SLM  "azimuth," .LM(0)
                         .SLM  "end;"
                         ;

tv_def =                 "relation TV [key number] of" .LM(+3)
                            .SLM  "number,"
                            .SLM  "tv_mode,"
                            .SLM  "azimuth,elevation" .LM(0)
                         .SLM  "end;"
                         ;

optical_sight_def =      "relation optical sight [key number] of" .LM(+3)
                            .SLM  "number,"
                            .SLM  "optical_sight_mode,"
                            .SLM  "azimuth,elevation," .LM(0)
                         .SLM  "end;"
                         ;

laser_measurement_def =  "relation laser measurement [key number] of" .LM(+3)
                            .SLM  "number,"
                            .SLM  "distance,"
                            .SLM  "range_number,"
                            .SLM  "X_position,Y_position,Z_position" .LM(0)
                         .SLM "end;"
                         ;

laser_direction_def =    "relation laser direction [key number] of" .LM(+3)
                            .SLM  "number,"
                            .SLM  "azimuth,elevation," .LM(0)
                         .SLM  "end;"
                         ;

sensor_platform_def =    "relation sensor platform [key number] of" .LM(+3)
                            .SLM  "number,"
                            .SLM  "azimuth,"
                            .SLM  "time_of_update;" .LM(0)
                         .SLM  "end;"
                         ;

map_def =                "relation map [key map_id] of" .LM(+3)
                            .SLM  "map_id,"
```

```
                               .SLM  "map_mode;" .LM(0)
                            .SLM  "end;"
                            ;

area_def =                  "relation area [key area_id] of" .LM(+3)
                               .SLM  "area_id,"
                               .SLM  "point_number,"
                               .SLM  "north,east;" .LM(0)
                            .SLM  "end;"
                            ;

forward_observer_def =      "relation forward observer [key number] of" .LM(+3)
                               .SLM  "number,"
                               .SLM  "X_position,Y_position,Z_position,"
                               .SLM  "observation_mode,"
                               .SLM  "azimuth,elevation,"
                               .SLM  "X_relative_position,Y_relative_position,"
                               .SLM  "Z_relative_position,"
                               .SLM  "time_of_update;" .LM(0)
                            .SLM  "end;"
                            ;

friendly_unit_def =         "relation friendly unit [key id] of" .LM(+3)
                               .SLM  "id,"
                               .SLM  "X_position,Y_position,Z_position,"
                               .SLM  "X_velocity,Y_velocity,Z_velocity,"
                               .SLM  "unit_type,"
                               .SLM  "time_of_update;" .LM(0)
                            .SLM  "end;"
                            ;

cpa_def =                   "relation CPA [key target_number] of" .LM(+3)
                               .SLM  "target_number,"
                               .SLM  "X_position,Y_position,Z_position,"
                               .SLM  "X_relative_ship,Y_relative_ship,"
                               .SLM  "Z_relative_ship,"
                               .SLM  "time_of_update;" .LM(0)
                            .SLM  "end;"
                            ;

ref_point_def =             "relation reference point [key number] of" .LM(+3)
                               .SLM  "number,"
                               .SLM  "X_position,Y_position,Z_position;" .LM(0)
                            .SLM  "end;" .LM(0)
                            ;

attack_position_def =       "relation attack position [key target_number] of"
                            .LM(+3)
                               .SLM  "target_number,"
                               .SLM  "X_position,Y_position,Z_position;" .LM(0)
                            .LM(0)  "end;"
                            ;

target_gr =                 .SLM "target graphic:" .LM(+3)
```

```
                        .SLM "vector length =" #1 "seconds"";"
                        .SLM "length limit =" #2 "knots"";"
                        .SLM "time between history points ="
                        #3 "seconds"";"
                        .SLM "number of history points =" #4 ";"
                        .SLM "alpha-numeric is [" #5 "]" ";"
                        .SLM "target symbols:" "[" #6 "]" ";" .LM(0)
                        ;

target_symbol =         #1 ", " #2 " = " #3 ;
target_symbol_seq =     #1 ;
target_symbol_tree =    .TREEPRINT(target_symbol_seq,1,.SLM,) ;
target_category =       #1 ;
target_class =          #1 ;

cursor_gr =             .SLM "cursor graphic:" .LM(+3)
                           .SLM "cursor""(" #1 ")"":" .LM(+3)
                           .SLM "cursor symbol =" #2
                           .SLM "true motion =" #3 ";"
                           .SLM "center cursor =" #4 ";"
                           .SLM "own ship to cursor =" #5 ";"
                           .SLM "cursor to own ship =" #6 ";"
                           .SLM "cursor movement X =" #7 ";"
                           .SLM "cursor movement Y =" #8 ";" .LM(0)
                        .LM(0);

ruler_gr =              .SLM "ruler graphic:" .LM(+3)
                        .SLM"scale on ruler =" #1 ";"
                        .SLM "distance between scale marks =" #2 "meter"";"
                        .LM(0);

own_ship_gr =           .SLM "own_ship graphic:" .LM(+3)
                        .SLM "velocity vector type =" #1 ";"
                        .SLM "speed marker / speed vector length ="
                        #2 "seconds"";"
                        .SLM "time between history points =" #3 "seconds"";"
                        .SLM "number of history points =" #4 ";"
                        .SLM "alpha numeric speed and course display =" #5 ";"
                        .SLM "screen position of speed and course display ="
                        "(" #6 ", " #7 ");" .LM(0)
                        ;

gun_gr =                .SLM "gun graphic:" .LM(+3)
                        .SLM "direction vector type =" #1 ";"
                        .SLM "coverage =""-" #2 "degrees"
                        .SLM "to""+" #3  "degrees"","""range" #4 "meter"";"
                        .SLM "alpha-numeric is [" #5 "]"";"
                        .SLM "impact symbol =" #6 ";"
                        .LM(0);

tracker_radar_gr =      .SLM "tracker radar graphic:" .LM(+3)
                        .SLM "direction vector type =" #1 ";"
                        .SLM "search mode =" #2 ";"
```

```
                              .SLM "tracker_symbol =" #3 ";"
                              .SLM "alpha-numeric is""[" #4 "]"";"
                              .LM(0);

surveillance_radar_gr =       .SLM "surveillance radar graphic:" .LM(+3)
                              .SLM "direction vector type =" #1 ";"
                              .SLM "alpha-numeric is""[" #2 "]"";"
                              .LM(0);

tv_gr =                       .SLM "TV graphic:" .LM(+3)
                              .SLM "direction vector type =" #1 ";"
                              .SLM "track mode =" #2 ";"
                              .SLM "alpha-numeric when tracking is"
                              .SLM "[" #3 "]"";"
                              .SLM "otherwise alpha-numeric is"
                              .SLM "[" #4 "]"";"
                              .LM(0);

optical_sight_gr =            .SLM "optical sight graphic:" .LM(+3)
                              .SLM "direction vector type =" #1 ";"
                              .SLM "alpha-numeric when tracking is"
                              .SLM "[" #2 "]"";"
                              .SLM "otherwise alpha-numeric is"
                              .SLM "[" #3 "]"";"
                              .LM(0);

laser_measurement_gr =        .SLM "laser measurement graphic:" .LM(+3)
                              .SLM "alpha-numeric is" "[" #1 "]"";"
                              .SLM "laser measurement symbol =" #2 ";"
                              .LM(0);

laser_direction_gr =          .SLM "laser direction graphic:" .LM(+3)
                              .SLM "direction vector type =" #1 ";"
                              .SLM "alpha-numeric is""[" #2 "]"";"
                              .LM(0);

sensor_platform_gr =          .SLM "sensor platform graphic:" .LM(+3)
                              .SLM "direction vector type =" #1 ";"
                              .SLM "alpha-numeric is""[" #2 "]"";"
                              .LM(0);

map_gr =                      .SLM "map graphic:" .LM(+3) .SLM"standard"";".LM(0)
                              ;

area_gr =                     .SLM "area graphic:" .LM(+3) .SLM"standard"";".LM(0)
                              ;

forward_observer_gr =         .SLM "forward observer graphic:" .LM(+3)
                              .SLM "alpha-numeric is""[" #1 "]"";"
                              .SLM "observer symbol =" #2 ";"
                              .SLM "observation vector type =" #3 ";"
                              .SLM "symbol for observed position =" #4 ";"
                              .LM(0);
```

```
friendly_unit_gr =        .SLM "friendly unit graphic:" .LM(+3)
                          .SLM "vector length =" #1 "seconds"";"
                          .SLM "length limit =" #2 "knots"";"
                          .SLM "time between history points =" #3 "seconds"";"
                          .SLM "number of history points =" #4 ";"
                          .SLM "alpha-numeric is" "[" #5 "]" ";"
                          .SLM "unit symbols:" "[" #6 "]" ";"
                          .LM(0);

unit_symbol =             #1 " = " #2 " ;" ;
unit_symbol_seq =         #1 ;
unit_symbol_tree =        .TREEPRINT(unit_symbol_seq,1,,) ;
unit_type =               #1 ;

cpa_gr =                  .SLM "CPA graphic:" .LM(+3)
                          .SLM "alpha-numeric is""[" #1 "]"";"
                          .SLM "symbol =" #2 ";"
                          .SLM "relative to own ship =" #3 ";"
                          .LM(0);

ref_point_gr =            .SLM "reference point graphic:" .LM(+3)
                          .SLM"alpha-numeric is""[" #1 "]"";"
                          .SLM "symbol =" #2 ";"
                          .LM(0);

attack_position_gr =      .SLM "attack position graphic:" .LM(+3)
                          .SLM "alpha-numeric is""[" #1 "]"";"
                          .SLM "symbol =" #2 ";"
                          .LM(0);
speed_vector =            "speed vector" ;
point_to_edge_vector =    "point_to_edge_vector" ;
edge_marker =             "edge_marker" ;
compass_marker =          "compass_marker" ;

vector_element =          " vector" ;
symbol_element =          " symbol" ;
alpha_numeric_element =   " alpha-numeric" ;
history_points_element =  " history points" ;
gun_coverage_element =    " gun coverage" ;
speed_course_display =    " speed and course and display" ;
user_defined_element =    #1 ;

.END
```

6.2.4 Example

The following is an example of definition of tactical plot system using the domain language. It is meant to simplify the understanding of the syntax and give the reader a basis for a qualitative assessment of the domain language. It may represent the requirements for a small system, but it is not meant to be representative of the typical systems. Its purpose is to serve as an example for explaining the domain.

```
/* Targetdisplay                                                    */
/* by: Sigmund Sundfor,  June 24  1983                              */
/*                                                                  */
/* Definition of a tactical display subsystem, called "Targetdisplay" */
/* using the domain language for tactical displays on ship borne gun  */
/* control system.                                                  */
/*                                                                  */
/* This is just a sample of how a very simple system conceivably could*/
/* be defined.  This subsystem only displays targets, own ship, some  */
/* special points and one cursor.                                   */

Tactical display Targetdisplay :

   /*--------------------------------------------------------------*/
   /* Configuration  definition:                                    */

   Configuration:

      target_category is an element from the set
         [friendly,hostile,unknown,neutral];
      target_class is an element from the set [air,sea,submarine];

      relation target [key target_number] of
         target_number,
         X_position,Y_position,Z_position,
         X_velocity,Y_velocity,Z_velocity,
         target_category,target_class,
         time_of_update;
      end;

      relation cursor [key number] of
         number,
         X_position,Y_position,Z_position,
         time_of_update;
      end;

      relation own_ship [key id] of
         id,
         north,east,
```

```
      X_position,Y_position,Z_position,
      X_velocity,Y_velocity,Z_velocity,
      roll,pitch,heading,
      time_of_update;
   end;

   new relation special_point [key name] of
      name,
      X_position,Y_position,Z_position;
   end;

   number of cursors is 1;

   number of picture planes is 6;

   available colouring is colour(1), colour(2), colour(3), colour(4);

   the size of the display is 1024 display increments;

   number of rulers is 0 ;

/*
```

```
*/
 /*------------------------------------------------------------------*/
 /* World model definition:                                          */

 World model:
  { relation target [key id] of
       id,
       X,Y,Z,
       XV,YV,ZV,
    end;

    relation category [key id] of
       id,
       cat;
    end;

    relation classification [key id] of
       id,
       class;
    end;

    relation targetobservation [key id] of
       id,
       observer,
       sensor,
       clock;
    end;

    relation own [key name] of
       name,
       X,Y,Z,
       XV,YV,ZV,
       R,P,C,
       clock;
    end;

    relation own_geographic_pos [key name]
       name,
       north,east;
    end;

    relation refpoints [key number] of
       number,
       X,Y,Z;
    end;

    relation cursorpos [key number] of
       number,
       X,Y,Z,
       clock;
    end; }
 /*
```

```
*/
 /*---------------------------------------------------------*/
 /* Definition of access to world model:                    */

 Access to world model:
    /* To access the target data as it is known in this domain,    */
    /* several relations in the world model will have to be combined*/
    /* and those attributes that are relevant tied to attributes    */
    /* of the internal representation.                              */
    /* Below,  a query interface of the type represented by TROLL   */
    /* is assumed.                                                  */

    target access is
       {templ := target.id join category.id;
        temp2 := templ.id join classification.id;
        temp3 := temp2.id join observation.id;}
        tctdisplay target(number,
           X_position,Y_position,Z_position,
           X_velocity,Y_velocity,Z_velocity,
           target_category,target_class,
           time_of_update)
        :=
       {temp3(id,
        X,Y,Z,
        XV,YV,ZV,
        class,cat,
        clock);}
        ;

    /* The cursor is generated in this domain, so instead of fetching  */
    /* data from the database,  the cursor data in the database is     */
    /* updated from here.                                              */

    cursor update is
       {cursorpos(number,X,Y,Z,clock)}
       := tctdisplay
       cursor(number,X_position,Y_position,Z_position,time_of_update);
       {} ;

    special_point access is
       {}
       tctdisplay special_point(name, X_position,Y_position,Z_position)
       := {refpoints(number, X,Y,Z)}
       ;

    own_ship access is
      {temp4 := own.name join own_geographic_pos.name;}
       tctdisplay own_ship(name,north,east,X_position,Y_position,Z_position,
       X_velocity,Y_velocity,Z_velocity,roll,pitch,heading,time_of_update)
       := {temp4(name,north,east, X,Y,Z, XV,YV,ZV, R,P,C, clock);}
       ;
 /*
```

```
*/
 /*-----------------------------------------------------------------*/
 /* Commands and parameters:                                       */

 Commands and parameters:

    /* The name ´scale´ corresponds to ´range´.                     */
    external scale is range;

    range is an element from the set of [1830meter, 3660meter, 7320meter];

    on-off commands are
       [pb_display_only_hostile,
       pb_delete_id,
       pb_colour_hostile_red,
       pb_true_motion];

    do-once commands are
       [center_cursor(), cursor_inc(X,Y)];

 /*
```

```
*/
 /*-----------------------------------------------------------------*/
 /* Graphic representation of the objects:                          */


 Graphic representation:

    target graphic:

      vector length = 180 seconds ;
      length limit =  50 knots ;
      time between history points = 180 seconds;
      number of history points = 6;
      alpha-numeric is [digit(1..4) = target(number);];
      target symbols:
        [friend,submarine  = symbol((arc, -5,0, 0,-5, 5,0));
         friend,surface    = symbol((circle, 5));
         friend,air        = symbol((arc, -5,0, 0,5, 5,0));
         hostile,submarine = symbol((vector, -5,0, 0,-5, 5,0));
         hostile,surface   = symbol((vector, -5,0, 0,-5, 5,0, -5,0));
         hostile,air       = symbol((vector, -5,0, 0,5, 5,0));
         ];
      /* The above definition of target symbols is incomplete in this */
      /* example.  It should give symbols for all combinations of the */
      /* target category and class.                                   */


    cursor graphic:

    cursor(1):
       cursor symbol = symbol((vector, 2,0, 10,0)(vector, 0,2, 0,10)
                       (vector, -2,0, -10,0)(vector, 0,-2, 0,-10));

       true motion = pb_true_motion;
       center cursor = center_cursor;
       own ship to cursor = false;
       cursor to own ship = false;
       cursor movement X = cursor_inc(X);
       cursor movement Y = cursor_inc(Y);
       /* The above defines what the different modes and actions     */
       /* in the cursor alorithm depends upon.  The l.h.s is the name*/
       /* of the mode and actions while the r.h.s. defines external  */
       /* commands or states they depend upon.  The r.h.s. can be a  */
       /* constant 'true' or 'false' also.                           */

    own_ship graphic:

       velocity vector type = point_to_edge_vector;
       speed marker / speed vector length = 180 seconds;
       time between history points = 30 seconds;
       number of history points = 6;
       alpha numeric speed and course display = true;
       screen position of speed and course display = (450,900);
```

```
/* The graphic representation for the new object defined, special_point,*/
/* has to be defined fully.  The below definition does this.  The       */
/* graphics will only have the name of the object as it is defined below*/
/* It would also be possible to name subparts of the objects.  This     */
/* would have been neccesary if there were conditions where only parts  */
/* of it should be displayed.  The parts would then have to be named.   */

 special_point graphic:

    world coordinates((point, special_point(X),special_point(Y)));
    symbol((vector, -5,0, 5,0) (vector, 0,-5, 0,5)
    (point, 5,5) (string, special_point(number)) );



/*
```

```
*/
 /*--------------------------------------------------------------------*/
 /* Task definition:                                                   */
 /* The tasks define the scheduling and priority of collecting the objects*/
 /* to be displayed and transforming them to display code. It defines  */
 /* "colouring" (adding colour, intensity and line type).  Conditions for */
 /* showing or not showing parts of the graphic representation of the  */
 /* objects may be defined (if not defined,  it is assumed that all shall */
 /* be shown.                                                          */

 The tasks are:

    when command is pb_display_only_hostile
    then
       display all target where target_category is hostile
       every 1sec, priority 2;
    otherwise
       display all target every 1sec, priority 2;
    ;


    when command is pb_delete_id
    then
       show all target graphic except alpha-numeric;
    otherwise
       show all target graphic;
    ;

    when command is pb_colour_hostile_red
    then
       colour all target where category is hostile colour(1);
       colour all target where category is not hostile colour(2);
    otherwise
       colour all target colour(2);
    ;

    display all cursor every 100msec, priority 1;
    colour all cursor colour(3);

    display all own_ship every 1sec, priority 2;
    colour all own_ship colour(2);

    display all special_point every 2sec, priority 3;
    colour all special_point colour(2);

/*-------- This is the end of the specification of 'Targetdisplay' -------*/
/*-----------------------------------------------------------------------*/
/*-----------------------------------------------------------------------*/
```

6.2.5 Transformations and Refinements

As mentioned previously, the transformations and refinements have not been implemented. One thing is that the underlying domains will have to be defined before the refinements can be implemented. However, the language has been designed with them in mind.

The source to source transformations in this domain are independent of any other domain. On the other hand, there will probably not be a great deal of transformations that can take place, because the syntax imposes a very stringent order on the definition with few alternative ways of writing the specifications. In general it is a question of whether a construct, e.g. an object to be displayed, should be included or not. Once it is decided to include it, there is in most cases only one way of specifying it. Transformations basicly deals with simplifications of the specification ("program" written in the domain language) by propagating informations so that it can detect superfluous specifications and eliminate these. Since this language gives few alternatives, there will probably be few transformations (simplifications).

The refinements can probably follow several different ways depending upon the implementation model one has in mind. The one that has been thought of here is one along the lines of the SADT model of the domain. It is envisaged that the domain must be implemented as several tasks to provide for the frequency and priority of update specified in the task section of the language. All the predefined objects would have to have components defined for them. It is likely that say most of the components for treating target data would be collected in one task. This means that data from several of the sections in the programming language will interact. It would therefore be necessary to have the refinement mechanism "walk" round the internal form tree and collect all information relevant to targets.

6.3 Rationalization for the Domain Language

One question that arose when the design of the syntax started was what form the language should have. One possibility was to design a set of powerful primitives relevant to tactical plots and conventional, procedural control structure. The user would then have to define what the objects should be and how they should be mapped from internal form to the tactical plot. The other possibility was to decide what objects there are, the basic form of their graphical representation and the algorithms for generating these. The user would then only be given the opportunity to decide what should be displayed, conditions and some parameters for the graphic representation.

The first choice would give the most powerful language with the ability to describe a multitude of objects and how they should be displayed. On the other hand, what objects are presented on the tactical display and the way they may be represented is indeed part of the domain knowledge. It was felt important to capture this domain knowledge in the language design. Therefore the second alternative was chosen. In addition, the language does include some limited possibilities for defining new objects to be displayed.

The language is meant to be a specification tool tailored for this domain. It is not meant to be a programming language (or what is commonly associated with traditional programming languages).

The alternative chosen has lead to a very big language. Big in the way that there are a lot of syntax rules. But that is to be expected when the language shall capture a lot of domain knowledge. Big in this context does not mean that it leaves a wide varieties of styles of using the language. It is in fact very restrictive. In most places it is only a matter of whether a particular construct should be included or not. Once it is included, there is a fixed set of parameters to define. It is not like ordinary programming languages, rich with recurrent structures leaving an almost infinite amount of freedom.

It appears feasible to construct a syntax directed editor based primarily on the information in the parser and prettyprinter definition. That would certainly ease the task of reading the syntax. The application specialists the language really is for, are in general not very good or keen at reading BNF type grammar rules. Another possibility for this language, would be to provide a template file that included all the different language structures so the user could fill it out more or less as a form.

The design goal for the language has been to form it such that the specifications written in the language follow close to how traditional english specifications for tactical plots are written. This includes the terms used in the domain, making it non procedural and also letting it be quite verbose to increase the readability of the final specifications written in this language.

The language tries to provide constructs that will be readable to the customer (purchaser of the ship borne gun control system). However, since the tactical plot is part of the total system, it does also have to provide for the interface to the rest. That is necessarily not meant to be understandable for the end user of the product. These are divided up in separate parts so that there can be a clear distinction between the specifications that addresses the end user requirements and those that addresses the interfaces to the rest of the system.

7. Summary

The reason for the work done was to test out the feasibility of the Draco approach to real-time, embedded systems and to gain more knowledge of the domain analysis process. It was decided to do a domain analysis on a specific kind of system, namely ship borne gun control system. The application was analysed and modelled using SADT. The conclusion after this initial analysis of the application, was that it covered too wide an area to be described as a single Draco domain. Such a system will probably have to be defined in terms of several domains, some of them possibly not using Draco. It is argued that the splitting of into domains may be done according to anticipated changes as suggested by Parnas.

The domain chosen to work further on was that of the tactical plot in the ship borne gun control system. The domain was analysed using SADT to model the domain, several trial specifications of actual systems and also writing a users manual. Finally this led to the construction of the domain language. It is expressed in the form of the parser and prettyprinter definition for Draco. The parser definition is basicly the BNF definition and does therefore serve as a syntax definition. The language is non procedural. It is verbose and contains a lot of domain specific constructs. This is to make it specific and to make the specifications written in the language readable for the domain specialists. The size of the syntax definition may give an impression of it being hard to use. This would be alleviated by a simple syntax directed editor. It could even be suitable for form/template type of input.

The language parser and prettyprinter has been implemented using Draco. Some sample specifications for tactical plots has been written using the language. The work done does not include transformations and refinements. At this point it is therefore not possible to generate working tactical plot subsystems. This work has concentrated on the domain analysis.

I. An Introduction to SADT

SADT (System Analysis and Design Technique) has been used successfully to model both software systems and social systems. Its ability to model both types of systems is important here since Draco advocates the use of a software system within a social system.

A complete SADT model consists of two kinds of diagrams: activity diagrams (called actigrams) and data diagrams (called datagrams). The view of an actigram is that data objects flow between activities while the view of a datagram is that activities during their operation access data objects. The only difference is the center of attention. Only actigram models will be discussed in this appendix.

I.1 The Elements of an Actigram

An actigram depicts three to six activities which are represented as boxes. The limit on the number of activities depicted helps to limit the amount of information a reader of an actigram must deal with. The boxes of an actigram are connected by arrows which represent data objects. Actigrams are data-flow diagrams. This means that the activity of a box takes place only when the data objects represented by incoming arrows to a box are present.

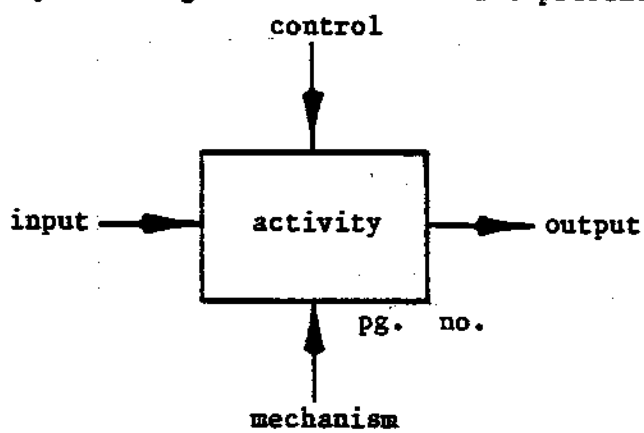


Figure 7-1: An SADT Actigram Box

The positions of the arrows on the box determines what type of data an arrow represents as shown in figure 7-1. When the input, control, and mechanism objects are present, the activity uses the mechanism as an agent to transform the input data objects into the output data objects under the guidance and constraints of the control data objects. Activity names should be verbs, while data object names should be nouns. Each activity must have at least one control and output.

A double headed dotted arrow may be used as a shorthand in SADT to denote data relations between activities as shown in figure 7-2.

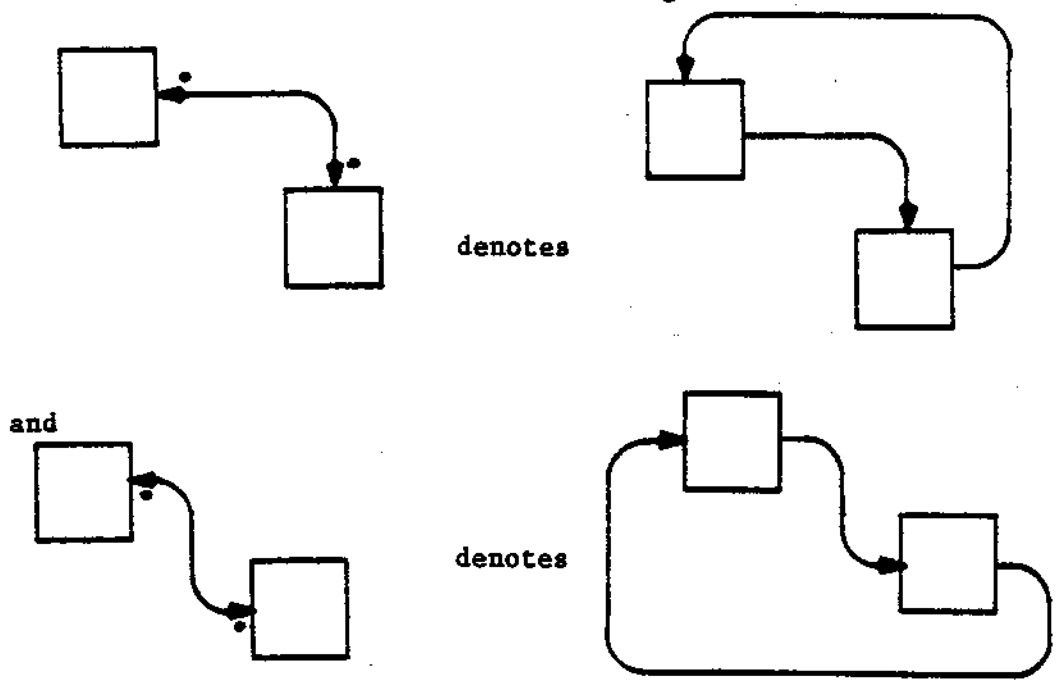


Figure 7-2: SADT Dotted Arrow Shorthand

I.2 The Structure of an SADT Model

Each actigram is an elaboration of an activity box in a higher-level diagram called the parent diagram. If a page number appears in parentheses just outside the lower right-hand corner of an activity box, then this number specifies the page of the actigram which elaborates the box. The inputs, outputs, controls, and mechanisms used in an actigram are the same as those on the corresponding activity box in the parent diagram. Each actigram should include from three to six activity boxes.

The highest-level actigram of a model is the only exception to the three to six activity rule and it presents only one activity, the one being modeled. The inputs, outputs, controls, and mechanisms which are used in the rest of the model are specified on this highest-level actigram called A-0. The A-0 actigram represents the context in which the system being modeled operates. As a part of the context the A-0 actigram explicitly states in prose the purpose of the model and from what viewpoint the model was made.

The external inputs, outputs, controls, and mechanisms used in an actigram are labeled with the position of the corresponding arrow on the corresponding box in the parent diagram. Inputs and outputs are numbered top to bottom while controls and mechanisms are numbered left to right. Thus, A2.3I2 (on actigram A2, box three, second arrow from top on left of box) would be shown as an external input labeled I2 on actigram A23. The numbering of the data objects with I ,C ,O , and M are called ICOM codes. If an external data object appears in an actigram and not on the corresponding box in the parent diagram then rather than being denoted by an ICOM code it is "tunneled." This means that the start or finish of the arrow is surrounded by parentheses to denote that the data object does not appear on the parent diagram.

The above discussion is a very brief introduction to SADT. More information about SADT can be found in [Ross 77].

I.3 Reading an SADT Model

There are three major stages in reading an SADT actigram model. At each stage the reader should ask the questions listed below.

1. Is the model syntactically correct?

    - All lines are commented with nouns. Each section of a split line is commented.

    - All boxes are labeled with verb phrases.

    - There are three to six boxes on each actigram (except the A-0 context diagram).

    - ICOM codes are accurate. All data produced is used. All data used is produced.

    - Each box has at least one control and one output.

2. Do I understand what the model says?

3. Do I agree with what the model says?

Usually comments written on the diagrams are returned to the author of the model. The author then responds to these comments and returns them to the reader. This cycle of written comments between a reader and an author is called the author-reader cycle.

II. A short introduction to Draco

It has been a common practice to name new computer languages after stars. Since the system described in this manual is a mechanism which manipulates special purpose languages it seems only fitting to name it after a structure of stars, a galaxy. Draco[2] is a dwarf elliptical galaxy in our local group of galaxies which is dominated by the large spiral galaxies Milky Way and Andromeda. Draco is a small nearby companion of the Milky Way ($1.2x10^5$ solar masses and 68 kiloparsecs from Earth). This small size and close distance to home is well suited to the current system which is a small prototype.

II.1 The Draco View of Software Production

The Draco system addresses itself to the routine production of many systems which are similar to each other. The theory behind its operation is described in detail in [Neighbors 80].

Three themes dominate the way Draco operates: the use of special-purpose high-level languages for the domains or problem areas in which many similar systems are needed; the use of software components to implement problems stated in these languages in a flexible and reliable way; and the use of source-to-source program transformations to tailor the components to their use in a specific context. The basic steps in the production of a specific system using a Draco supported domain-specific high-level language is as follows:

1. An analyst with experience in developing many systems in a certain problem domain decides that the domain is understood well enough to define a language suitable for comfortably and easily describing other systems in the problem domain. This person is called the Domain Analyst and the language described is called the Domain Language. The Domain Analyst describes the domain and its internal form with the parser generator part of the BUILD subsystem of Draco which is described in the Draco user's manual.

2. Once the Domain Analyst has described the external and internal form of the domain then how program fragments in the domain should be printed so that users find them easy to look at and accurate in their meaning must be described. This is called prettyprinter

---

[2] Draco is Latin for dragon

generation and it is done by the Draco BUILD subsystem.

3. The Domain Analyst must provide simplifying relations among the objects and operations of the domain. These are used for simplification and optimization of programs in the domain. These simplifications are accepted in terms of source-to-source program transformations by the BUILD subsystem which forms them into a library of transformations.

4. Finally, the Domain Analyst must prepare a prose description of the meaning of the operations and objects in his domain.

5. This prose description is turned over to a Domain Designer who specifies components for the objects and operations in the domain which refine the objects and operations of one domain into other domains known to the Draco system. These components are formed into libraries by the Draco subsystem. A component is a set of refinements each capable of implementing a domain object or operation under certain stated conditions while making certain implementation assertions.

6. A new system which can be described in a Domain Language known to Draco can inherit some analysis, design, and coding from the Draco library. The statement of the system to be constructed is cast in a Domain Language. The Domain Language program is then turned into an internal form by the PARSE subsystem. This internal form is then given to a System Specialist.

7. The System Specialist interacts with the transformation and refinement subsystem of Draco. The basic operation in this phase is the selection of an appropriate set of software components to implement the operations and objects in the domain which are used in the problem statement. Then these components are specialized by program transformation to the problem at hand and then separately refined into another (or the same) domain and the cycle begins again. The refinement subsystem allows the definition of refinement tactics capable of removing the burden of answering low-level questions from the System Specialist.

8. The process the System Specialist uses to refine the problem is, of course, not strictly top down but the refinement subsystem keeps a record of the process which makes it look top down. When the program is in an executable form it is printed out by the System Specialist and either acceptable or the specification cycle begins again with the existing Domain Language program.

9. The refinement history of a program may be examined by a user of the EXAMINE subsystem which states what refinements were used in the production of this program. A higher-level description of all parts of the program to whatever level (up to the original Domain Language) always exists in the refinement history. It is hoped that these higher levels of abstraction of an existing program will be useful in understanding the program during the maintenance phase of its lifecycle.

The process described briefly above is dealt with in more detail in [Neighbors 80] which presents an SADT[3] model of the process.

---

[3]SADT is a registered trademark of SofTech Inc.

REFERENCES

[Belady & Lehman 79]
Belady, L.A. and Lehman, M.M.
The Characteristics of Large Systems.
In Wegner, P., editor, Research Directions in Software Technology, chapter 1, pages 106-131. The Massachusetts Institute Technology Press, Cambridge, Mass., 1979.

[Freeman 82]
Freeman, Peter and Neighbors, James Milne.
Reusable Software Engineering, a Proposal Submitted for Consideration by the National Science Foundation.

[Gonzales 81]
Gonzales, Leonora San Luis.
A Domain Language for Processing Standardized Tests.
Technical Report RTP 007, Department Of Information And Computer Science University Of California, Irvine , 1981.

[Neighbors 80]
Neighbors, James Milne.
Software Construction Using Components.
PhD thesis, UCI, 1980.

[Parnas 79]
Parnas, David L.
Designing Software for Ease of Extension and Contraction.
IEEE Transactions On Software Engineering , SE-5(2):128,137, March , 1979.

[Ross 77]
Ross, Douglas T.
Structured Analysis(SA): A Language for Communicating Ideas.
IEEE Transactions On Software Engineering :?, January , 1977.

[Sundfor 83]
Sigmund Sundfor.
Draco Domain Analysis for a Real Time Application; Discussion of the Results.
Technical Report RTP 016, Department of Information and Computer Science, University Of California, Irvine , June, 1983.