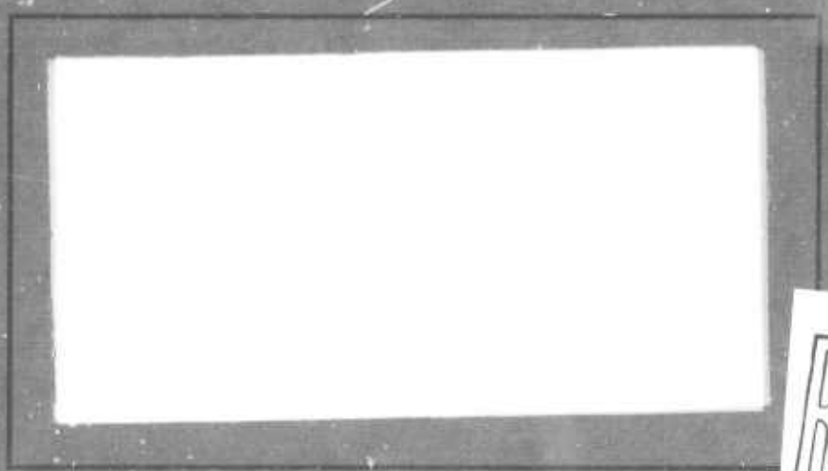
0

AIR FORCE INSTITUTE OF TECHNOLOGY

AIR UNIVERSITY
UNITED STATES AIR FORCE



SCHOOL OF ENGINEERING

WRIGHT-PATTERSON AIR FORCE BASE, OHIO

GE/EE/77S-6

(9) Master's thesis

(11) Sep 77

(12) 109 p.

(6) STRUCTURED ANALYSIS AND DESIGN OF A SATELLITE SIMULATOR

THESIS

(14) AFIT/GE/EE/77S-6

(10) Kenneth L. Marvin
Captain USAF

Approved for public release; distribution unlimited.

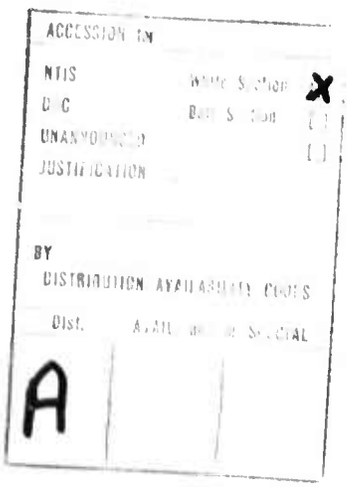
012 225

GE/EE/77S-6

STRUCTURED ANALYSIS AND DESIGN OF A SATELLITE SIMULATOR

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science



by

Kenneth L. Marvin, B.S.E.E.
Captain USAF
Graduate Electrical Engineering

September 1977

Preface

This thesis presents an effective application of a structured analysis and structured design methodology to a complex satellite command system. The methodology used combines features of three different design techniques. The key to the successful development is the preliminary design which gives a complete requirements definition in an understandable diagram form. A series of these diagrams are combined to form an activity model and a data model of the simulator using a top-down design strategy. The activity model is the starting point of the design refinement. A first cut structure chart is drawn directly from the activity model. A data flow graph is then created which in turn is converted into a refined structure chart. The refined structure chart is the primary vehicle which facilitates the programming phase. Even though the programming has not been done in this thesis, a structured analysis and design has been performed that presents a satellite simulator that is modifiable, understandable, and free of implementation constraints.

I thank my thesis advisor, Captain Peter E. Miller, for his professional influence and guidance throughout this project. Thanks must also be extended to Captain J. B. Peterson for sharing his expertise with me in this endeavor. A very special thanks goes to my wife, Maryanne, and to our children, Kevin and Cindy. Without their patience and love this effort would have been wasted. I cannot close without thanking God whose eternal wisdom has prevailed throughout.

Kenneth L. Marvin

Contents

List of Figures

List of Figures

List of Tables

Abstract

The problem addressed in this thesis is the analysis of a complex satellite command system and the design of a software simulation of that system. The problem is solved in three steps. First, a written requirements definition establishes a sound viewpoint and purpose on which the analyst can base his design. This requirements definition explains why the simulator is to be created, how it is to be constructed, and what it is to do. Second, a top-down strategy called "structured analysis" is applied to create the preliminary design. The structured analysis is presented in a blueprint-type language with activity and data models. The models represent graphically the functions performed by the simulator and the information upon which those functions act. A final design refinement is performed with a structured design methodology called "transform analysis." The structure charts drawn during the transform analysis reveal system characteristics which illustrate design quality. The activity model acts as a catalyst to a successful transition from a top-down analysis to a structured design which can be evaluated. The resulting simulator design, with minor revisions, satisfies the design goals established for the project. The methodology used is highly recommended for the analysis and design of any software system.

STRUCTURED ANALYSIS AND DESIGN OF A SATELLITE SIMULATOR

I. Introduction

Objectives

This thesis presents a "structured" development of the requirements and the design needed to create a software simulation of an operational satellite. A good development technique is required to make the final programming easier and more understandable. The primary objective of this thesis is to avoid all of the negative effects of a premature system design (Ref 4:2). To meet that objective, it is necessary to do a complete and understandable requirements analysis of the system to be designed. A requirements analysis is the first step towards the design of a high quality system that is efficient and reliable.

Any quality development project, be it hardware or software, must be based on an orderly, controlled, and disciplined methodology, (Ref 6:5). While searching for such a methodology, the top-down design phases of the software life-cycle are considered. The first two steps in the life-cycle are classified as "system requirements" and "software requirements" (Ref 2:3-4). A thorough explanation of the system and the functions it performs must be presented. This explanation is given by a Requirements Definition as defined by SofTech, Inc. (Ref 4:4). The Requirements Definition is the first portion of SofTech's Structured Analysis and Design Technique (SADT). It is this portion of the SADT that is used in the

requirements analysis and preliminary design of the satellite simulator. Requirements Definition deals with three subjects: context analysis, design constraints, and functional specifications. These subjects give a definition of the requirements for an efficient and long-range, cost-effective system. The context analysis explains why the system should be created and why certain technical and operational capabilities set the boundary conditions for the system. Design constraints explain how the system is to be constructed. The objective here is not to specify which things will be in the system, but only to set limits for selection of those things at a later time. The functional specifications are of primary interest. These specifications give only the boundary conditions for requirements taken up in the design phase (Ref 6:5-6).

The next phase in the software life-cycle is the preliminary design. The objective is to define system requirements more explicitly and to present a functional analysis that is conceptually complete. During this phase in the development, a design is produced that makes coding, debugging, and modification easier, faster, and less expensive by reducing complexity (Ref 9:115). After the preliminary design, a departure is made from the SADT. This departure is required to provide a design refinement that can be evaluated. A design technique called "transform analysis" is performed which has characteristics that reveal design quality (Ref 10:254-300). The combination of the preliminary design based on structured analysis and the design refinement based on transform analysis creates a satellite simulator design that is modifiable, understandable, and free of implementation constraints.

The sponsor of this thesis is the 4000 Aerospace Applications Group (SAC), located at Offutt Air Force Base, Nebraska. They are responsible for command, control, and analysis of the weather satellite (Block 5D), which is to be simulated. Operational requirements place a limit on the amount of software development that can be done "in house." A need exists for a simulator that can be used as a training device for satellite controllers and as an analysis aid for satellite data analysts. The purpose of this thesis is to provide a design of such a simulator.

Scope

The initial investigation of the satellite system shows that a simulation of all the satellite functions is beyond the scope of this thesis (Ref 1). However, a simulation of the spacecraft command "uplink" and command verification (CV) "downlink" is of primary importance. A simulator is required that will accept spacecraft commands in a realistic format, simulate performance of the proper functions in response to those commands, and maintain records of spacecraft status (i.e. telemetry point values, transmitters on or off, etc.), which can be modified at user request. In addition, the simulator should be designed in a modular fashion to ensure the modifiability, maintainability, and understandability. The structured analysis and design techniques used in this development effort present a modular system that is functionally independent. This independence makes it easy to add or delete modules to further enhance the realistic nature of the simulation at a later time. No attempt is made to do the actual programming required

to implement the simulator. However, the structured techniques used to perform the analysis and design will facilitate the final programming phase.

Plan of Development

Chapter II is a presentation of the Requirements Definition with emphasis placed on the functional specifications. Chapter III presents a Preliminary Design of the simulator in the form of activity and data models. The approach used in both of these chapters is based on the technique prepared by SofTech, Inc. called "structured analysis." Chapter IV presents a refinement of the preliminary design, which is illustrated with structure charts and data flow graphs (bubble charts). These charts and graphs are used to do the transform analysis. Chapter V contains a summary of results obtained in this development as well as conclusions and recommendations.

II. Requirements Definition

Introduction

The phase of the software life-cycle most often abused, or neglected, is the Requirements Definition. The Requirements Definition presented in this chapter is in a written form to show a distinction between it and the illustrative functional design presented in Chapter III. This order of development allows the analyst to establish a sound viewpoint and purpose on which to base his first phase of design. The lack of a Requirements Definition usually results in rising costs, missed schedules, waste and duplication (Ref 4:2). In addition, the elimination of this important step results in the absence of a useful documentation package. This creates the most common problem with large systems in the Air Force: a lack of understanding by the engineer who takes over the project. He typically must spend an inordinate amount of time nearly "redeveloping" the system to bring his level of understanding to a point where he can be productive.

Included in this chapter is (1) a context analysis, which tells why the simulator is to be created, (2) a set of design constraints, to tell how the simulator is to be constructed, and (3) a set of functional specifications that describe what the system is to do.

Context Analysis

To better understand why a satellite simulator is to be created, an explanation of the environment (context) in which it is

to be used is needed. This explanation is given in the context analysis which follows.

The 4000 Aerospace Applications Group (SAC) is responsible for the command and control of weather satellites in support of Global Weather Central at Offutt Air Force Base, Nebraska. It is also responsible for monitoring spacecraft telemetry and data to aid in detection of any anomalies that may have occurred during its orbit. Over the years these satellites have increased in complexity and produce more data for analysis purposes. This has increased the need for a good, easy to use simulator to aid in anomaly analysis and to function as a training device for new system controllers.

To perform as an analysis tool, the simulator must display current telemetry point values upon request. It must also provide the analyst with current status information about transmitters, central processing units, and sensors on the spacecraft. This information must be provided after each spacecraft command is given to the simulator and upon request from the user.

The simulator must accept its input in the same format that is transmitted to the satellite. This formatting is currently done by an existing hardware device known as the 5D Interface (5DI). Commands will be input through the 5DI to the simulator. In addition to the outputs mentioned above, the simulator should provide the proper command verification words and an explanatory message.

Design Constraints

This section is a summary of the conditions specifying how

the simulator is to be constructed. No attempt is made to specify input or output devices. Those details should be considered at a later stage in the design (Ref 4:4).

There are four fundamental goals which should be considered by the software engineer when he begins his development process. They are (1) modifiability, (2) efficiency, (3) reliability, and (4) understandability (Ref 5:89). These are the constraints considered in the process of selecting the analysis and design techniques used. The techniques selected are SofTech's structured analysis for the preliminary design and a structured design method for the design refinement.

Functional Specifications

Functional specifications are imposed on the functional architecture of the system. They differ from system architecture specifications because they outline the purposes of the system instead of giving the languages, transmission links, and record formats that are in the system (Ref 4:8). The following functional specifications are a first step towards the creation of the functional architecture. The next chapter presents the functional architecture as the preliminary design of the simulator.

There are four main functions that should be performed by the simulator. It should (1) process the input word, (2) update the vehicle (simulator) status in response to the input word, (3) create the appropriate command verification words, and (4) create an output message informing the user what action has been taken. A brief description of each of those functions follows.

Process Input Word. A satellite simulator input should be classified as a spacecraft uplink command or a user command. A user command modifies the current vehicle status data, i.e. change a telemetry point value, add or delete a status table, etc. A spacecraft uplink command must be properly formatted before it can be executed. The formatted command simulates satellite functions by updating status data and creating the appropriate command verification message. The formatted command must first be checked for parity and wordlength. If an error is detected, it is saved and used later by the simulator to generate the proper command verification words and output messages.

After the initial format checks are made, the spacecraft command type must be determined. This determination will influence the kind of sequence checks that need to be performed. Some spacecraft commands require a series of sub-commands in a specified order, while others require a specific time interval between them before they can be executed.

The processing just described will produce a user command in the form of some type of status modification word, a spacecraft command that will contain some errors (invalid command), or a spacecraft command that is error-free (valid command). One of these inputs will be passed to the remaining functional activities for appropriate processing.

Update Vehicle Status. Only a user command or a valid spacecraft command effects the current status information. An invalid command is only acknowledged by the command verification function. When the required modification is determined, an update request

is issued and the necessary update function is performed. The new status information is then given to the user.

Create Command Verification Words. Previously detected errors should be listed and used to determine the applicable command verification (CV) codes. These CV codes are transmitted via the downlink from the spacecraft after a command has been received. To make the simulator a more valuable analysis/training device, an explanatory message should be output with the CV code. There are also verification codes associated with valid spacecraft commands which should be handled in the same manner as the erroneous command codes.

Create Output Messages. The CV words are now translated and a meaningful output message is produced. Any status modifications are noted with a specific update message. The messages are assembled and a complete and easily understood output text is produced.

Summary

This chapter presents a context analysis of the problem, design constraints for its solution, and functional specifications. The context analysis forms a perspective from which to view the overall problem. An effort is made not to let the context analysis be functional specifications, however, the context analysis and functional specifications are closely related and in some cases it is difficult to separate the two. The design constraints are imposed by the desired design goals. Analysis and design techniques are selected that should make the design meet those goals. The functional specifications are given in modular sections which are

the four basic units needed for construction of the next level of design. This construction consists of the SADT activity and data models presented in the next chapter.

III. Preliminary Design

Introduction

The activity and data models presented in this chapter represent the results of many attempts to illustrate the conceptual ideas conveyed by the functional specifications. These models, organized as a sequence of diagrams with supporting text, form the preliminary design.

The activity model is presented first, followed by the data model. These models graphically represent the functions performed by the system and the data upon which the functions act. Each model contains both data and activities with complimentary emphases. The combination leads to "a much richer understanding of the subject than is afforded by a single model" (Ref 7: Chap. 2, p. 5). Each level of modules is a more detailed decomposition of the level above. This structured decomposition is the substance of top-down design. The following is a brief explanation of the diagram syntax to aid the reader in understanding the models. A more detailed discussion can be found in reference 7.

Diagram Syntax

Structured Analysis diagrams are composed of boxes and arrows which are a vehicle for clearly expressing activity and data modules (Ref 7: Chap. 3, p. 1). Figure 1 illustrates examples of the activity and data modules. The "mechanism" arrows are shown for completeness but are not used at this stage of the design since they are intended to represent the functions or hardware needed

to realize the module. The "multiple branch" (exclusive OR) is used to indicate multiple, but not simultaneous outputs. The "multiple join" indicates multiple, but not simultaneous inputs. Both conventions are shown in Figure 2.

In general, data and activity modules are decomposed into more detailed data and activity modules. Each module that is decomposed is referred to as a "parent" module and modules that result from the decomposition are the "children." To relate the arrows of the children to those of the parent, an "ICOM" code is

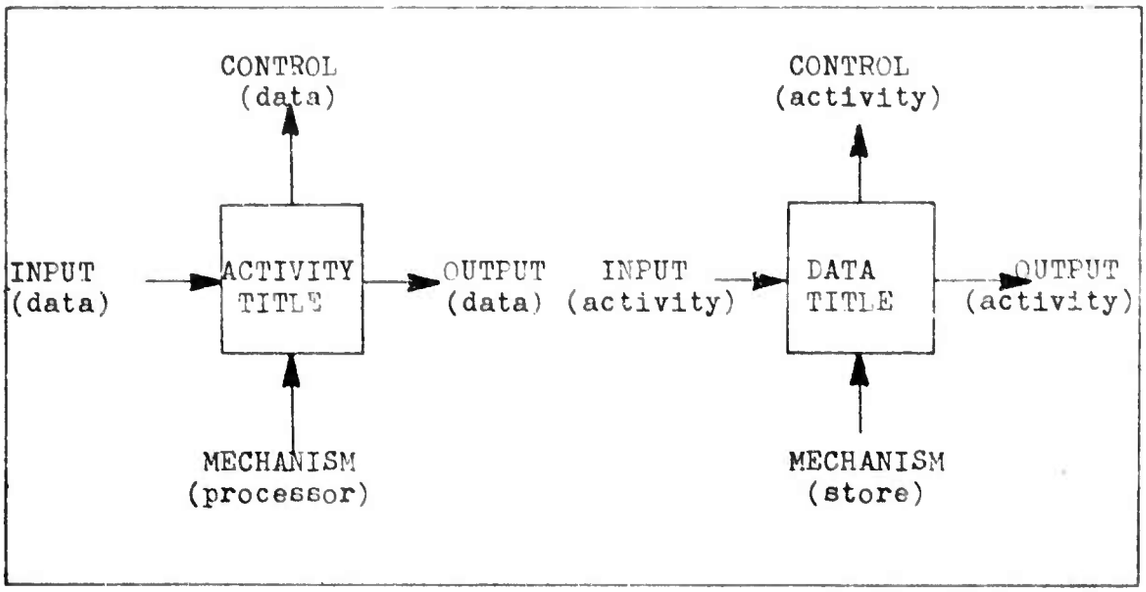


Figure 1. Module/Interfaces Arrow Conventions

is used. The acronym is derived from the arrow names: input, control, output, and mechanism. Each arrow at the parent/child boundary is uniquely labeled with the letter I, C, O, or M with prefix and suffix numbers. The prefix number refers to the module within the child and the suffix number refers to the top-down or left-right order of the arrow on a module. The boundary is the outer border of the activity or data diagram. Each diagram is called

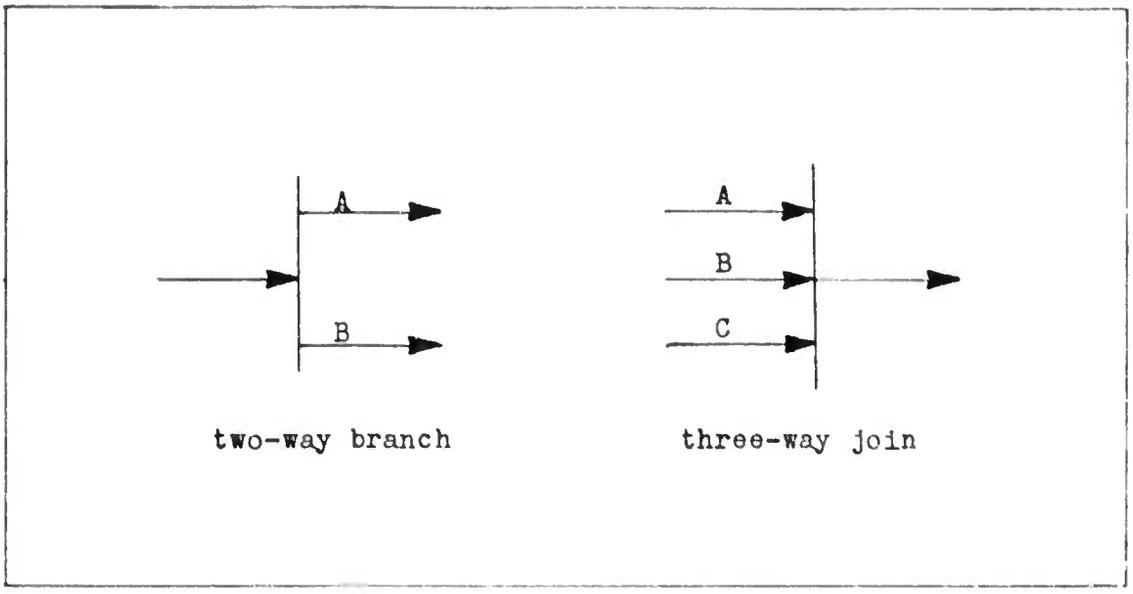


Figure 2. OR Branch and Join Structures

a "node."

Reading Sequence

The following is a suggested reading sequence, looking at each module in top-down order:

1. Scan the boxes to get a first impression.

2. Rethink the message of the parent module. Observe the boundary arrows.

3. Refer back to the current module, checking arrow attachments between it and the parent.

4. Consider internal arrows. Consider boxes from top to bottom and left to right.

5. Read the text.

Activity Model

Before reading the activity diagrams, it is recommended that the "node index" given below be scanned. This index serves as a table of contents, and gives an overview of the decomposition structure.

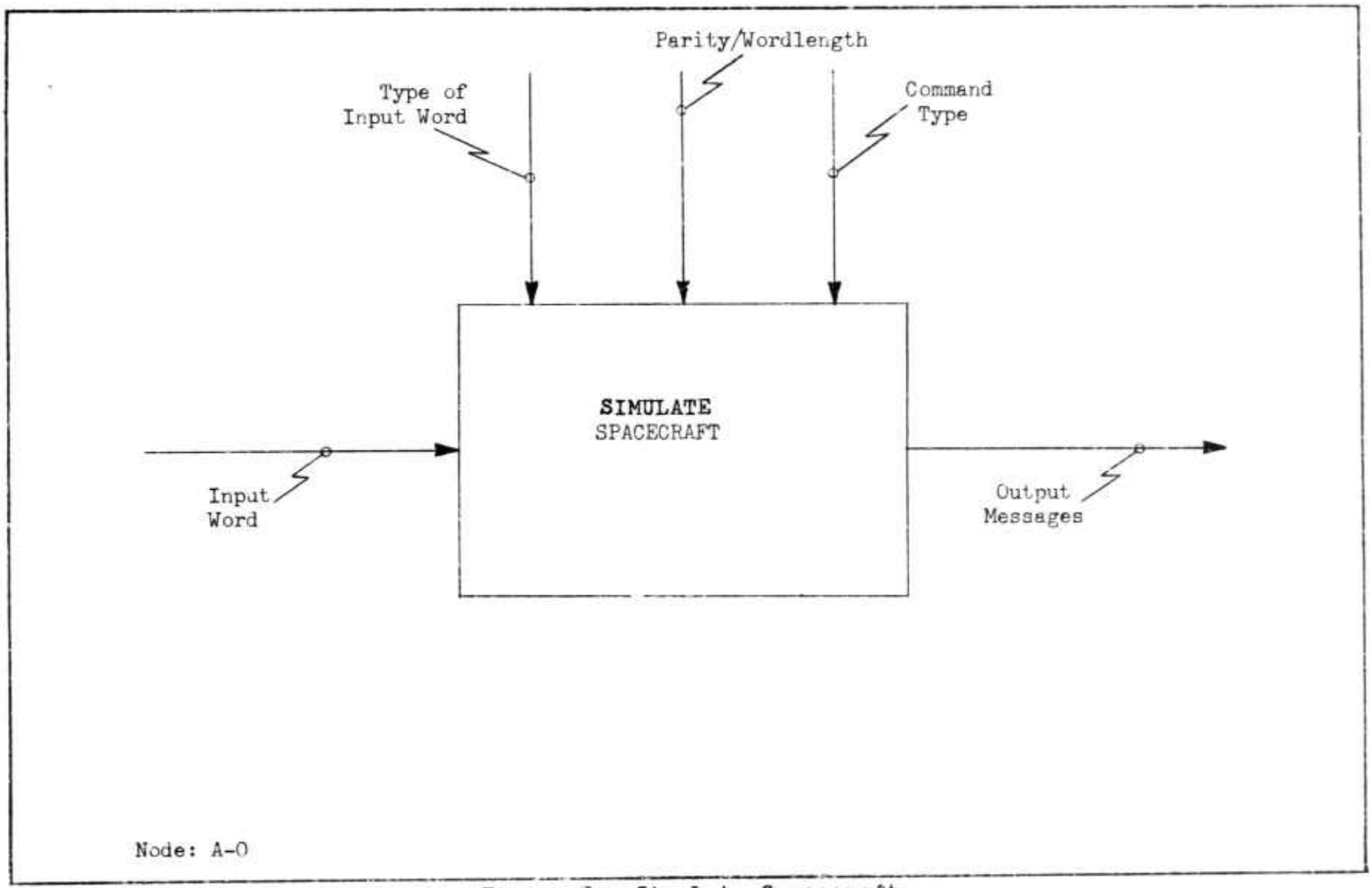


Figure 3. Simulate Spacecraft

A-0 Text: An input word (I1) is received by the simulator. That word is either a user command or an uplink spacecraft command, but not both at once. The activity performed ("SIMULATE SPACECRAFT") is constrained by the type of input word (C1), or by the parity, wordlength (C2) and type (C3), if the input is a spacecraft command. The final product of the activity is a complete and understandable message to the user (O1) for problem analysis or spacecraft interrogation.

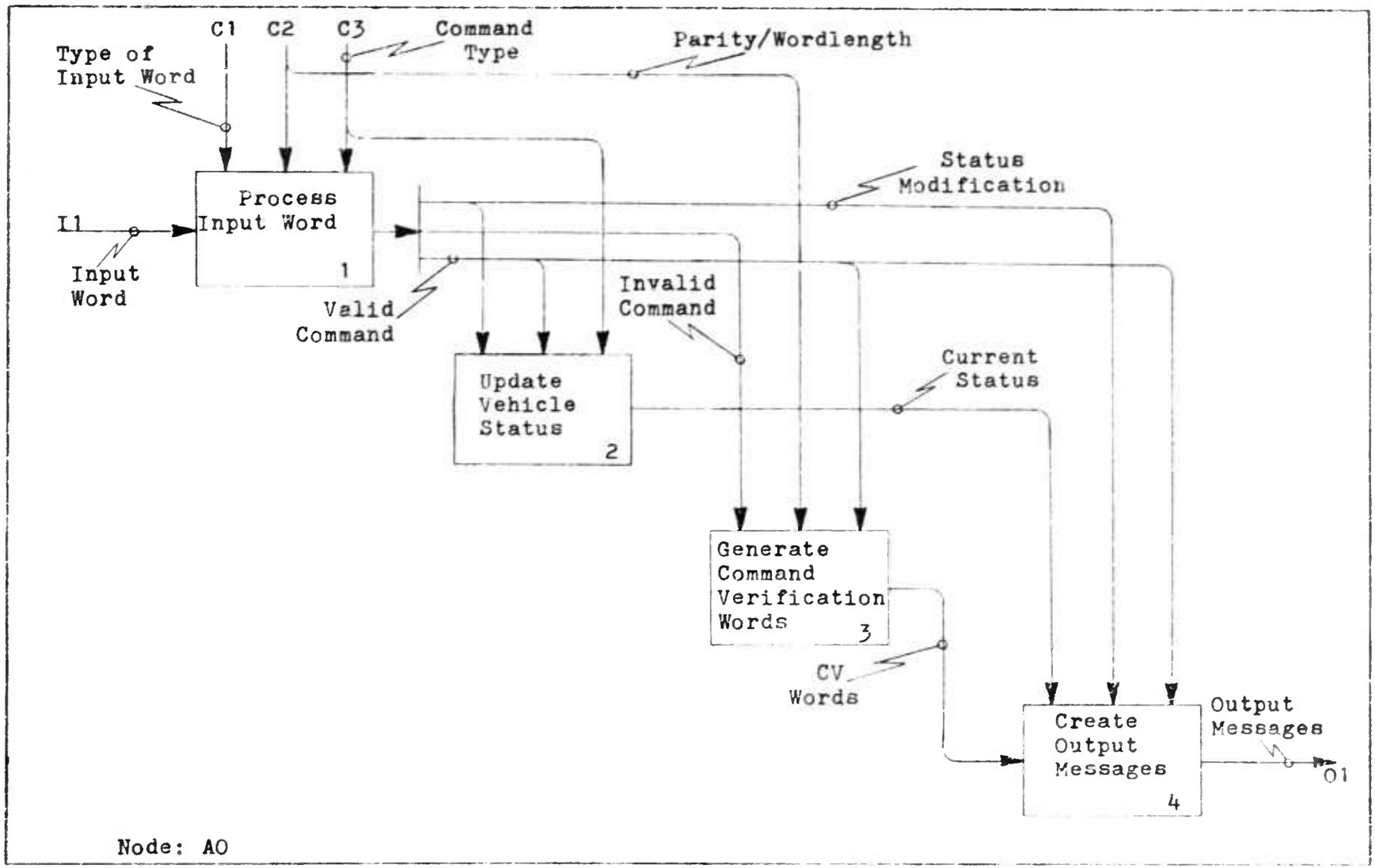


Figure 4. Simulate Spacecraft

AO Text: An input word (I1) is received by Process Input Word (1). That word is either a user command to modify vehicle (simulator) status or an uplink spacecraft command. If it is a user command, it is passed as a status modification (101) to control the Update Vehicle Status (2) and Create Output Messages (4) activities. If the input word is a spacecraft command, two constraints determine the type of output. These constraints are parity/wordlength (C2) errors and command type (C3). If there are no format (parity/wordlength) errors, a valid command (103) is used to control the other activities. Another possibility is an invalid command (102) which results from a parity/wordlength error. Update Vehicle Status (2) uses the status modification, the valid command, or the invalid command to determine the required vehicle status updates. The current status (201) is used as part of an output message (401). A valid command is acknowledged by a Command Verification (CV) Word (301). Generate Command Verification Words (3) produces these CV words and they are used by Create Output Messages (4), along with the constraints shown, to produce an output message responsive to the particular input word that is being processed.

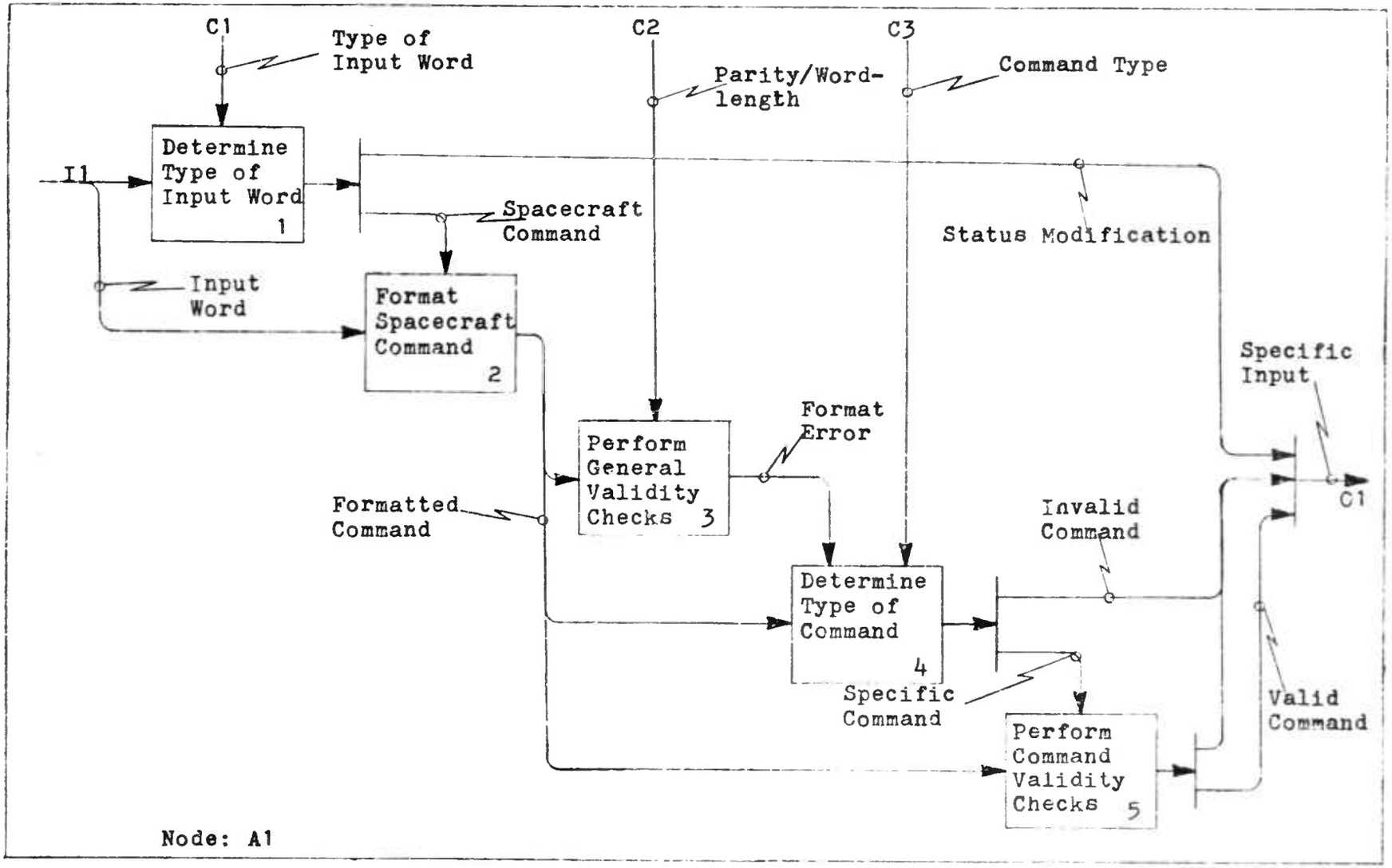


Figure 5. Process Input Word

A1 Text: Determine Type of Input Word (1) is primarily concerned with whether an input is in the format necessary to simulate an uplink spacecraft command (102), or in a user command format to be processed as a status modification (101). The spacecraft command is first formatted properly for use by the simulator. The formatted command (201) is passed to the remaining activities for processing. After the command is checked for parity/wordlength (C2) it is decoded to determine the command type (C3). Finally, it is checked for proper sequencing. Determine Type of Command (4) performs its function on a command even if it contains a format error. Perform Command Validity Checks (5) outputs an invalid command (501) or a valid command (502) to be used by the next activity.

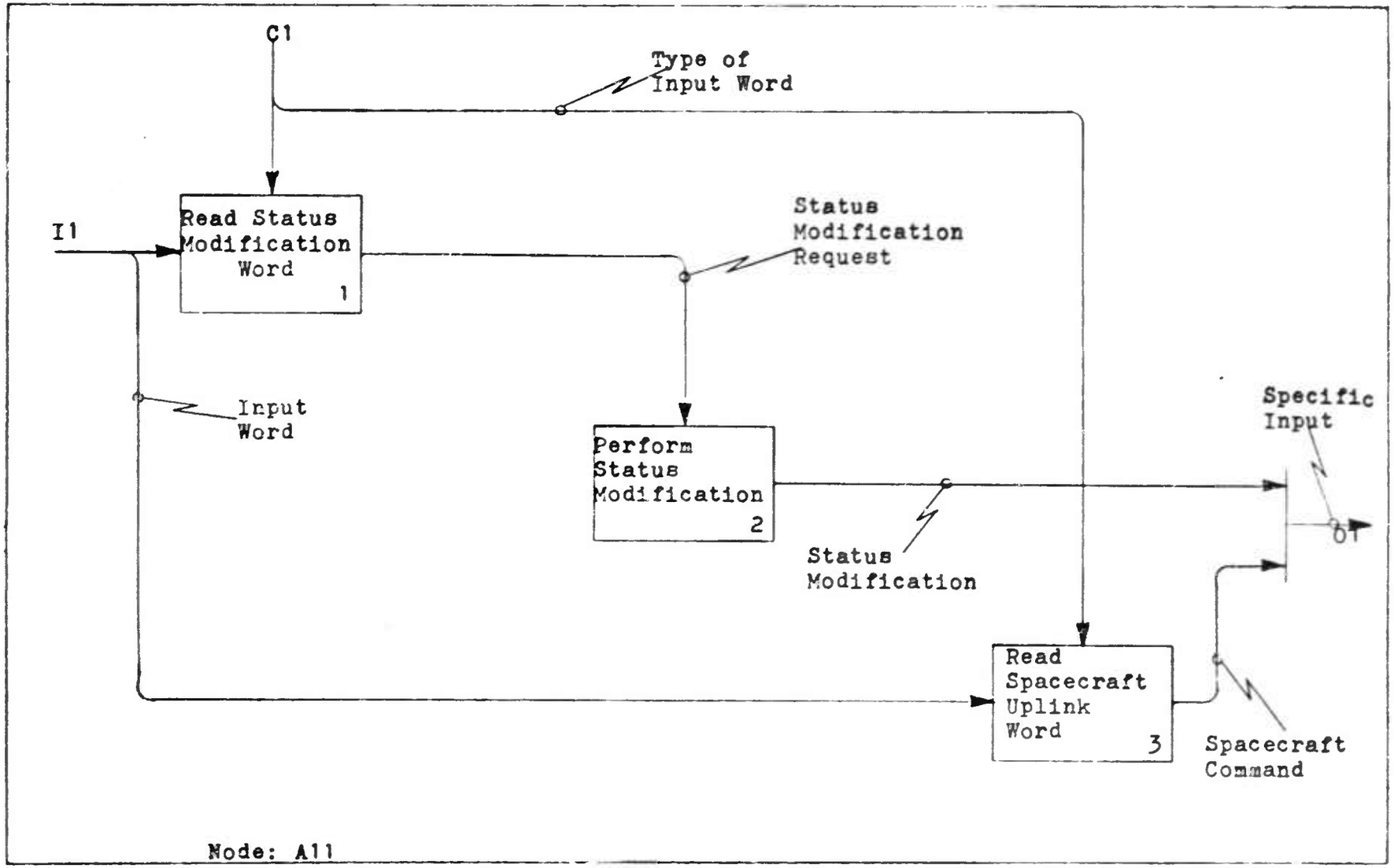


Figure 6. Determine Type of Input Word

All Text: If the Input Word (I1) is a user modification, Read Status Modification Word (1) outputs a status modification request (101) which controls Perform Status Modification (2). Upon receipt of the particular request, a status modification (201) is output.

Upon receipt of a word in the uplink format, activity (3) reads the spacecraft uplink word and outputs the spacecraft command (302) for use by the next activity.

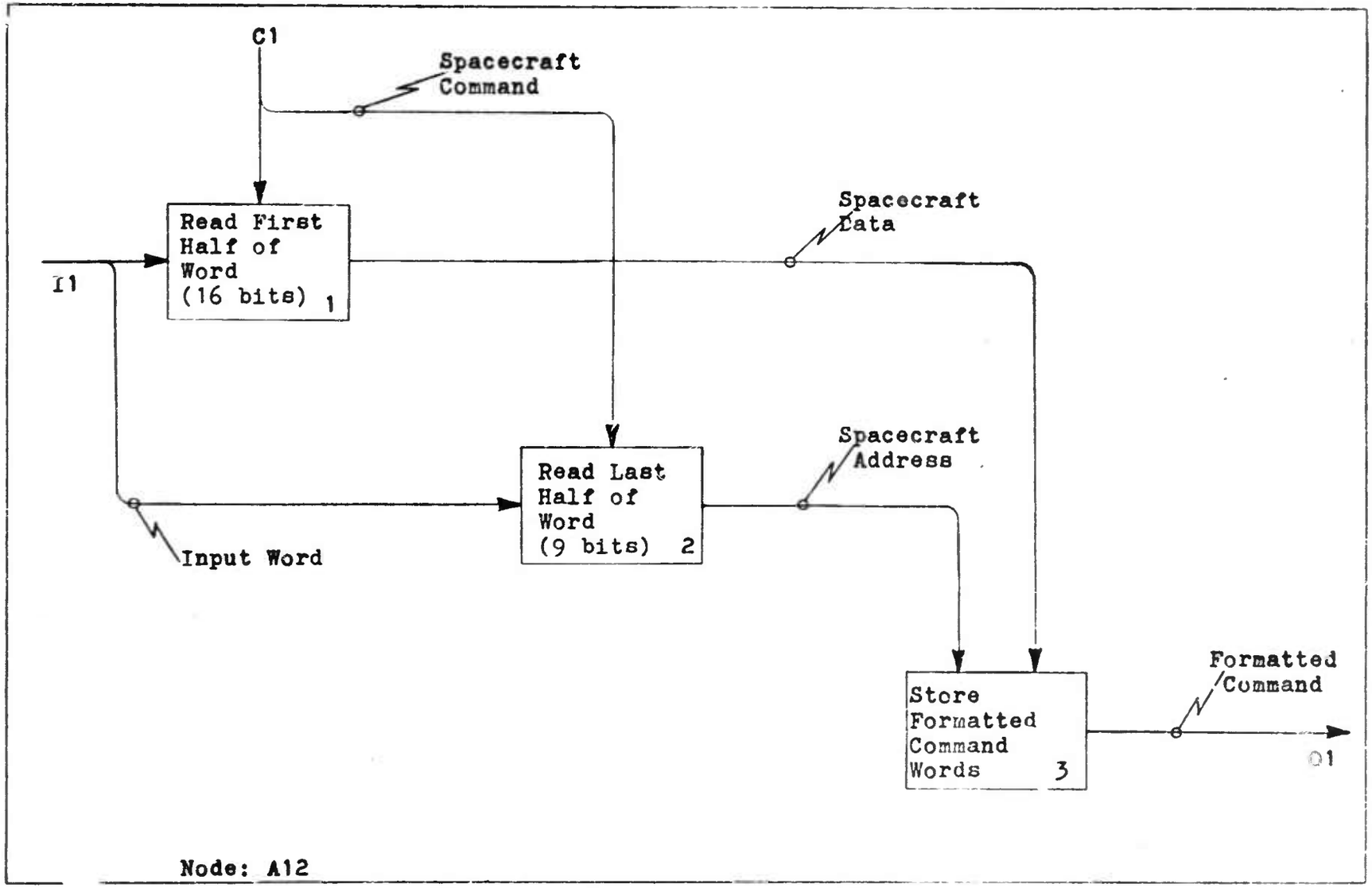


Figure 7. Format Spacecraft Command

A12 Text: After it is determined that the input word (I1) is a spacecraft command (C1), the first 16 bits of the word are read in activity (1) and the spacecraft data (101) is output. The remaining nine bits of the word are read in activity (2) and the spacecraft address (201) information is passed. Both of these outputs are stored by (3) and they are the formatted command (301).

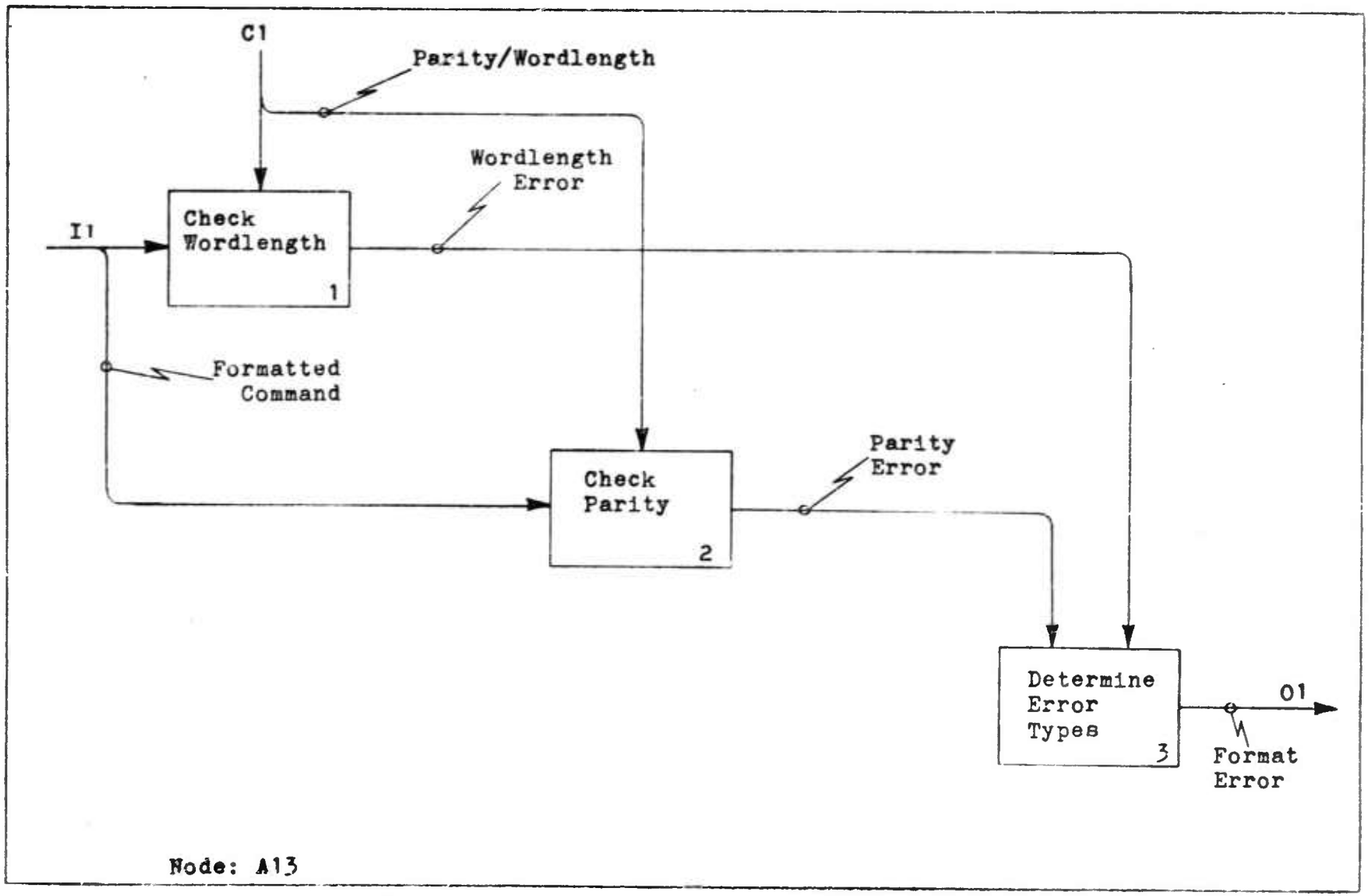


Figure 8. Perform General Validity Checks

A13 Text: The formatted command (I1) is input to both activity boxes (1) and (2) for a wordlength check and a parity check. There are various error codes that are associated with individual commands. After the error checks have been made, any errors that have been found are used to determine the codes that apply to the particular command and error (3).

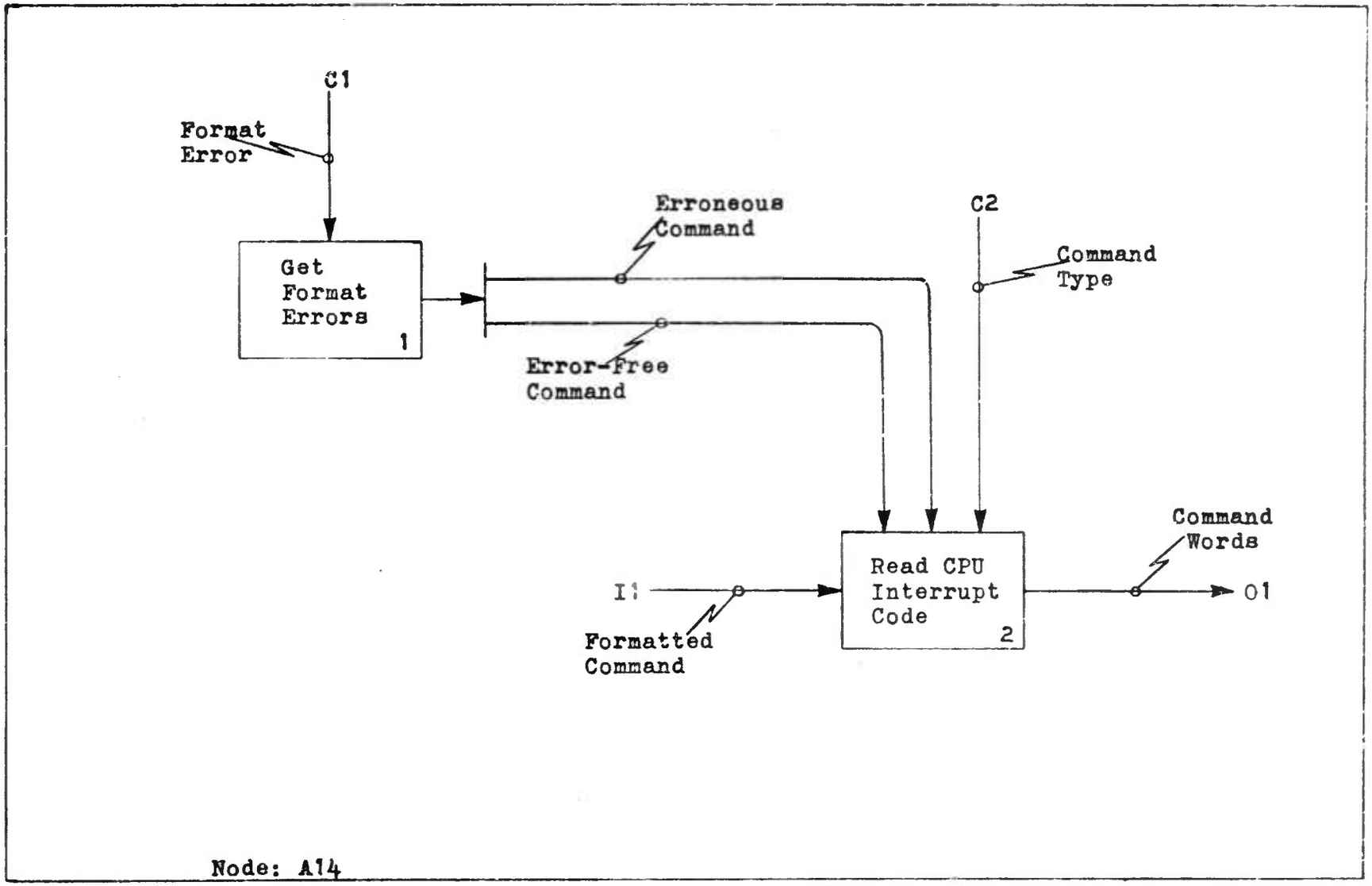


Figure 9. Determine Type of Command

A14 Text: The command type (C2) must be determined for all commands. If format errors (C1) exist, they are passed as indications of an erroneous command (101), or an error-free command (102).

The CPU Interrupt Code (CIC) is read (2) from the formatted command (I1). This code will relate to a certain type of command. The command words (O1) that are output are either invalid or valid, depending on ahether a format error has or has not been detected. These determinations must be made before the particular command can be executed.

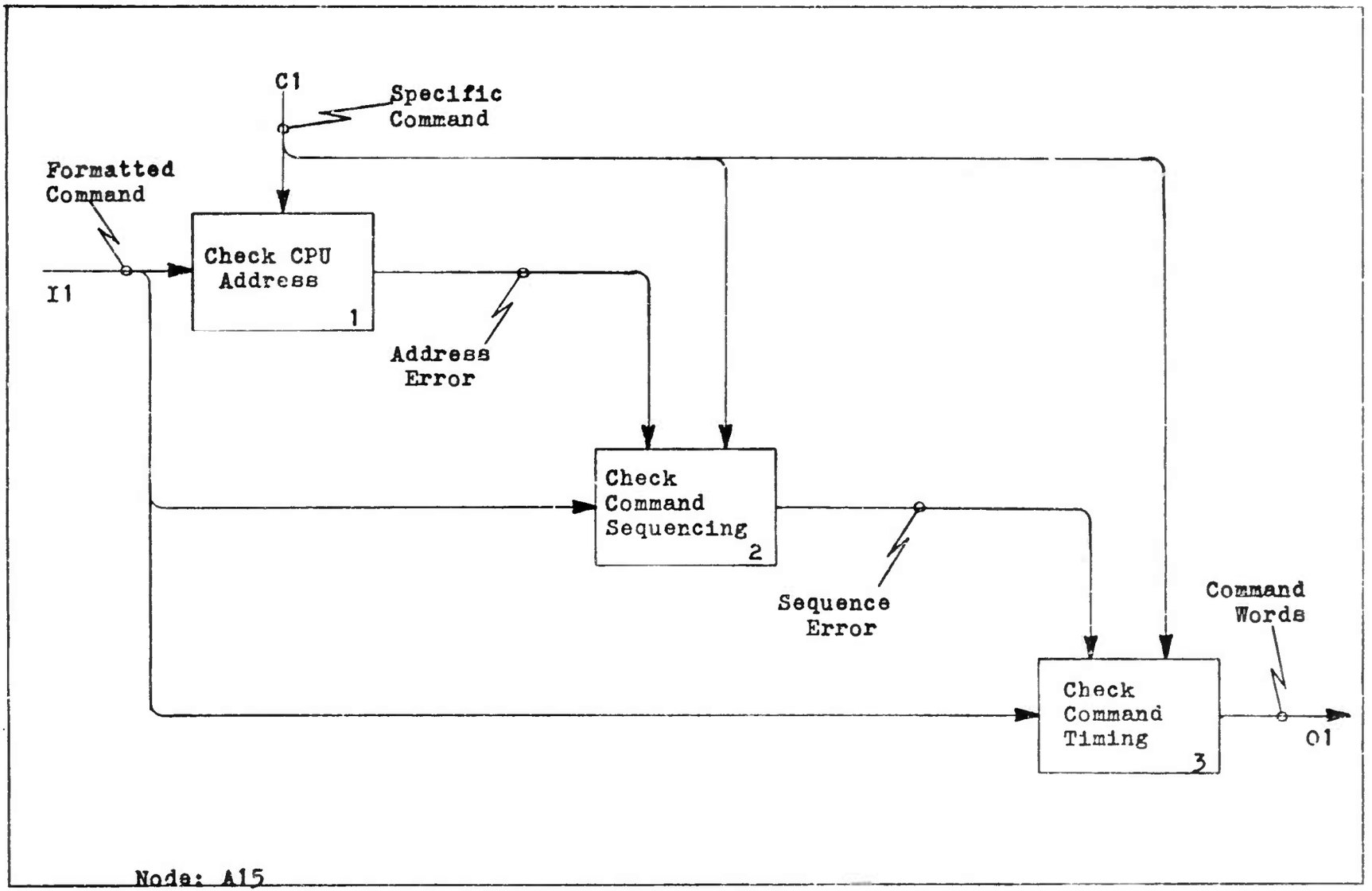


Figure 10. Perform Command Validity Checks

A15 Text: Each formatted command (I1) that has survived the General Validity Checks without error must go through a series of Command Validity Checks as controlled by the specific command (C1) that is input to the simulator. First it is determined if a CPU address error exists (1). If such an error exists, that error is saved for future use. If there is no CPU address error (101), the command sequence is checked in activity (2), and any sequencing errors (201) that may arise are saved in a like manner as the address error. Check Command Timing (3) works in a simular manner as (2) and is added for completeness. A timing error will result in the same error code as a sequence error. If this module is implemented, the sequence error and the timing error should be assigned a different error message so the user can distinguish between the two when they occur.

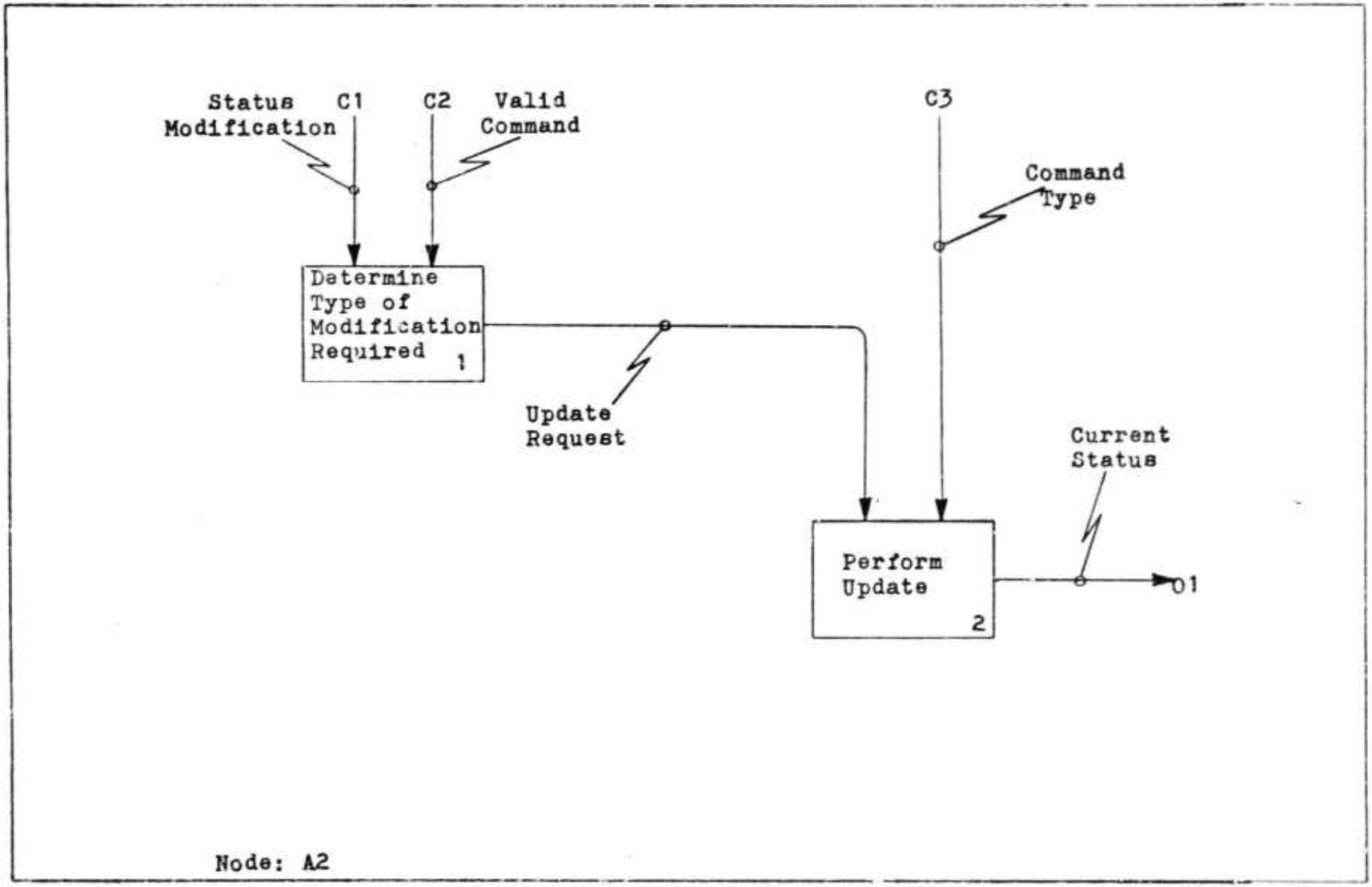


Figure 11. Update Vehicle Status

A2 Text: A status modification (C1) results from a user command input. Determine Type of Modification Required (1) analyzes it, or a valid command (C2) that may have been transmitted, to see what kind of update request (101) is necessary. After that determination has been made, Perform Update (2) is activated and the status information is passed to the output modules.

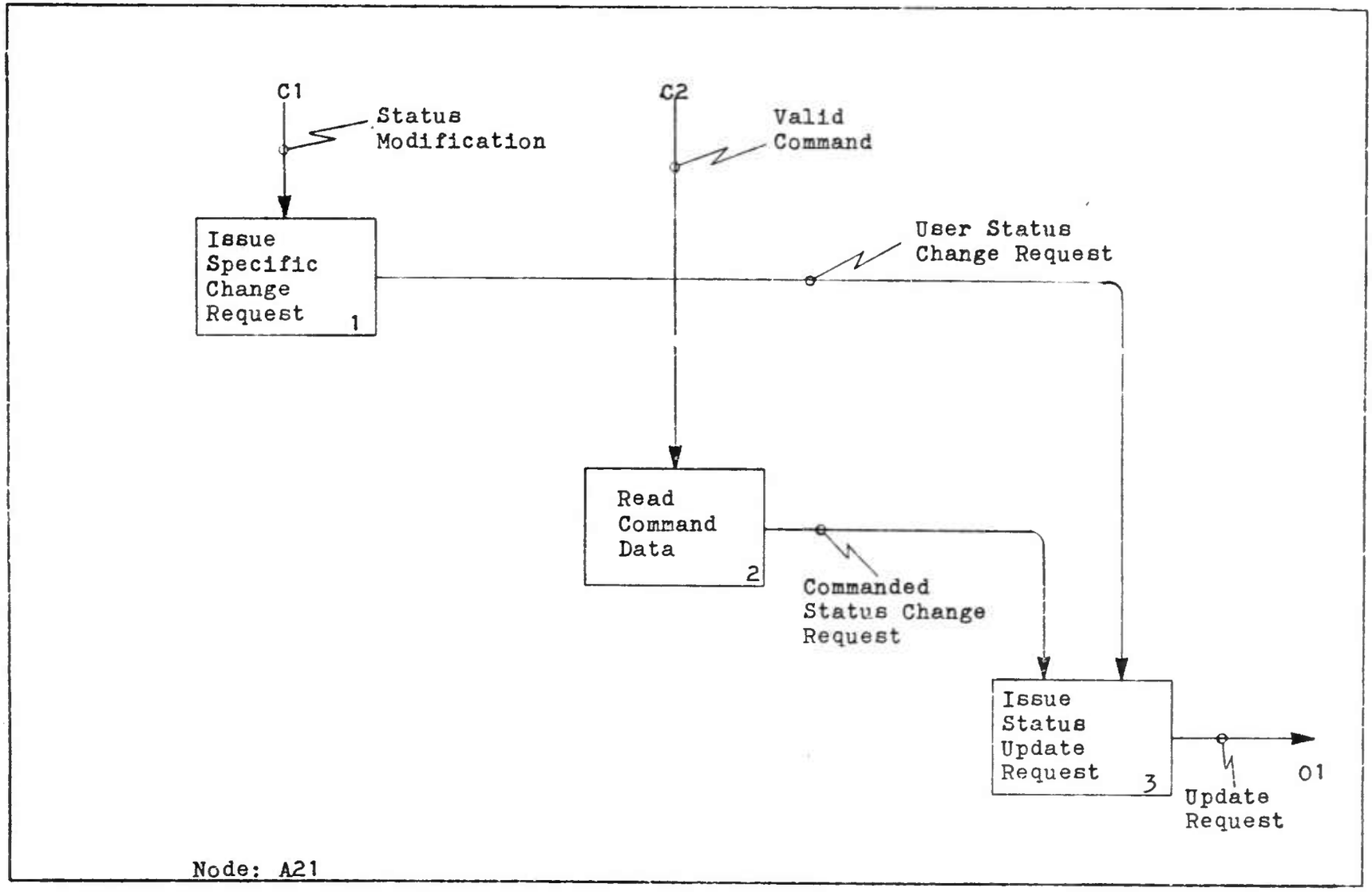


Figure 12. Determine Type of Modification Required

A21 Text: Issue Specific Change Request (1) utilizes the status modification (C1) that is being performed as a constraint which determines whether there should be a request made or not. When a valid command has been transmitted, the command data is read from it by (2) and the appropriate change request is passed to (3). With either request, (3C1) or (3C2), as the determining factor, a particular update request (01) is then issued.

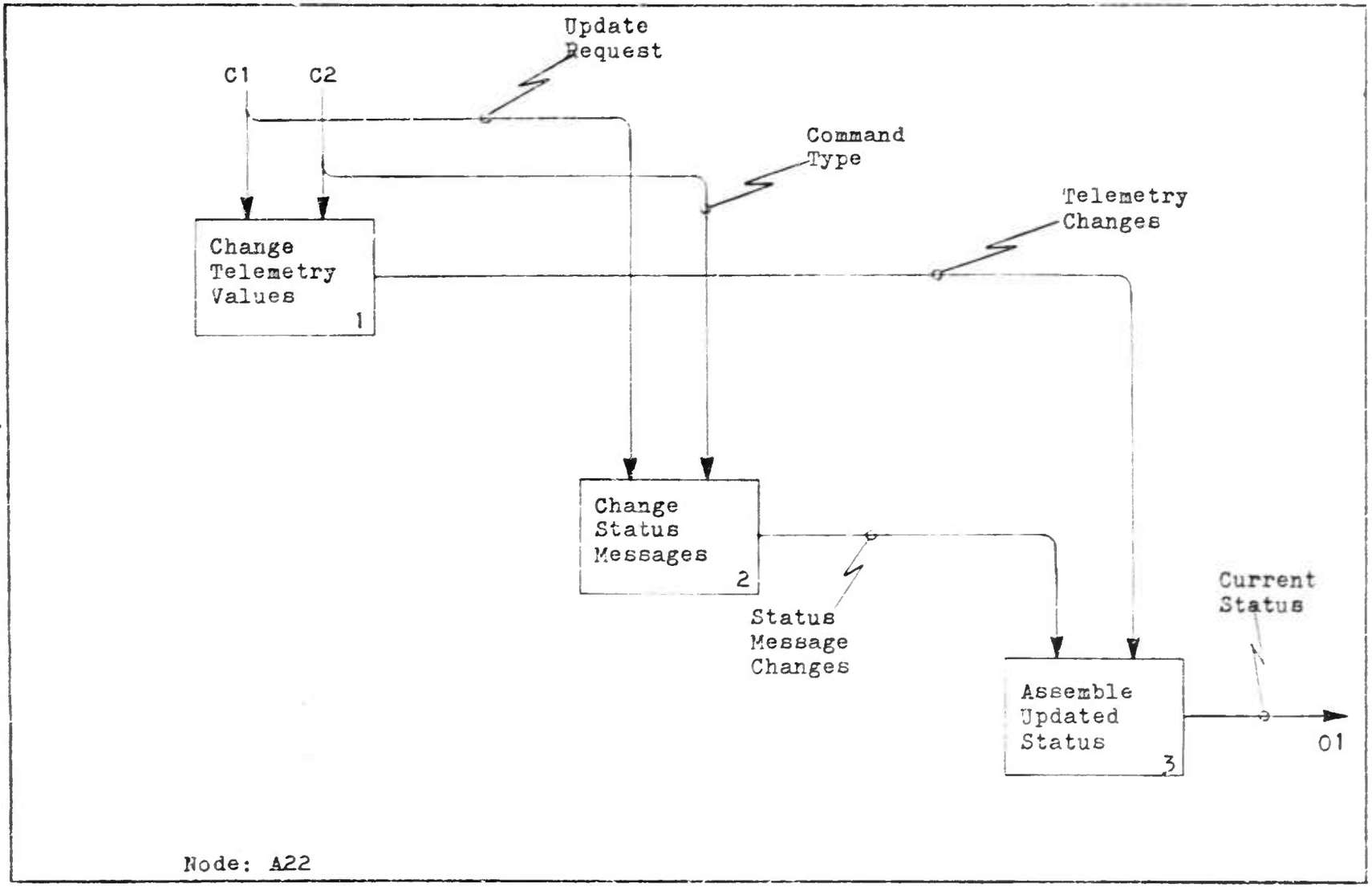


Figure 13. Perform Update

A22 Text: An update request that is issued as a result of either a user command or a valid spacecraft command, can require telemetry value changes to be made (1), status message changes to be made (2), or both. The changes that are made are assembled to form a list of current status data (01) to be used to create an informative output as a final product of the simulator.

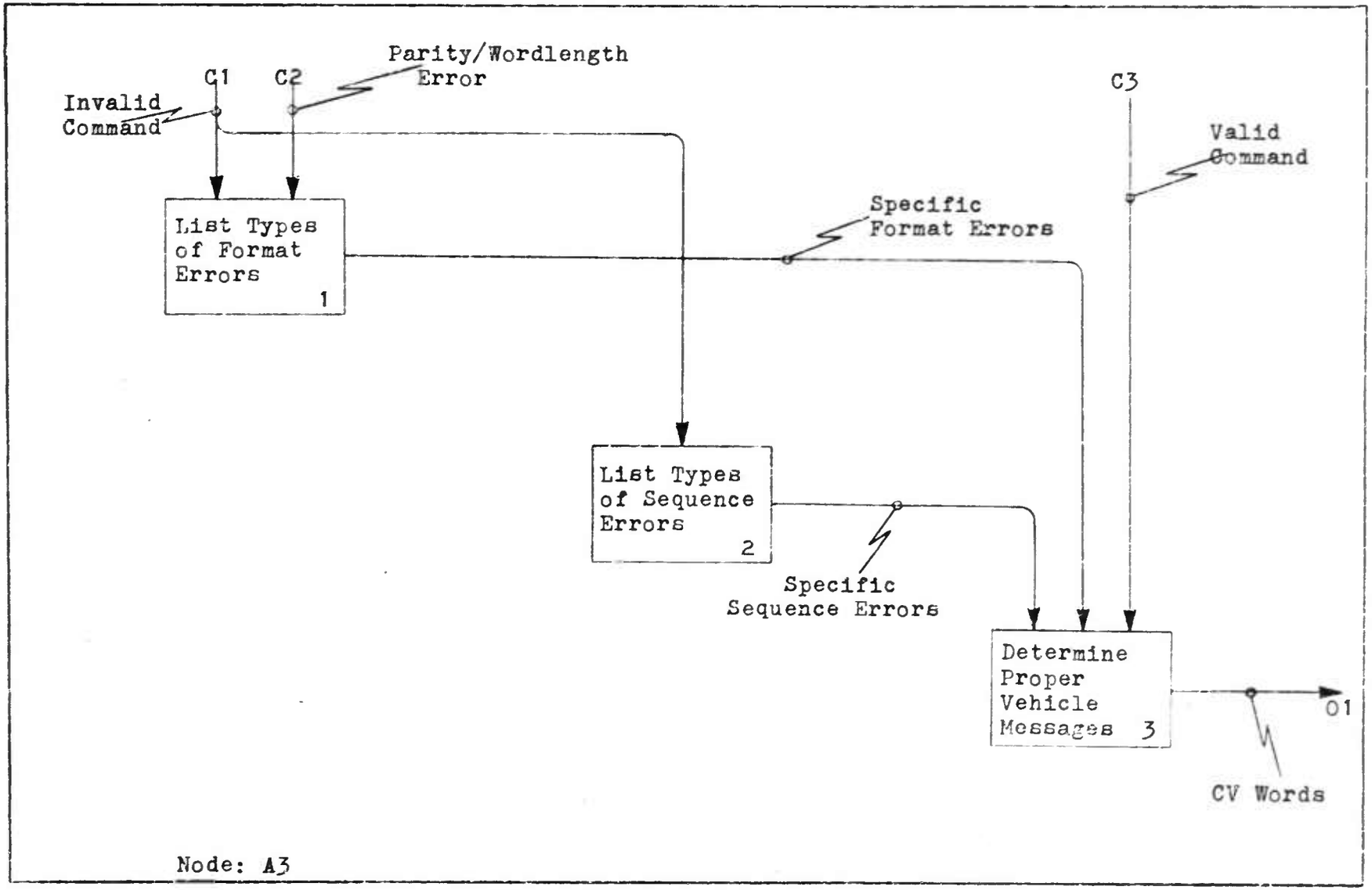


Figure 14. Generate CV Words

A3 Text: An invalid command (C1) can be the result of a parity error, a wordlength error, or some type of command sequencing error. List Types of Format Errors (1) extracts errors listed as a result of the general validity checks performed earlier.

The specific format errors (101) which are output are used to aid in determination of the proper CV words (01) to be passed. The other constraining factors are the specific sequence errors (3C1), and the valid command (C3) when there are no errors. Almost all of the spacecraft commands require a command verification (CV) message, whether they be valid or invalid.

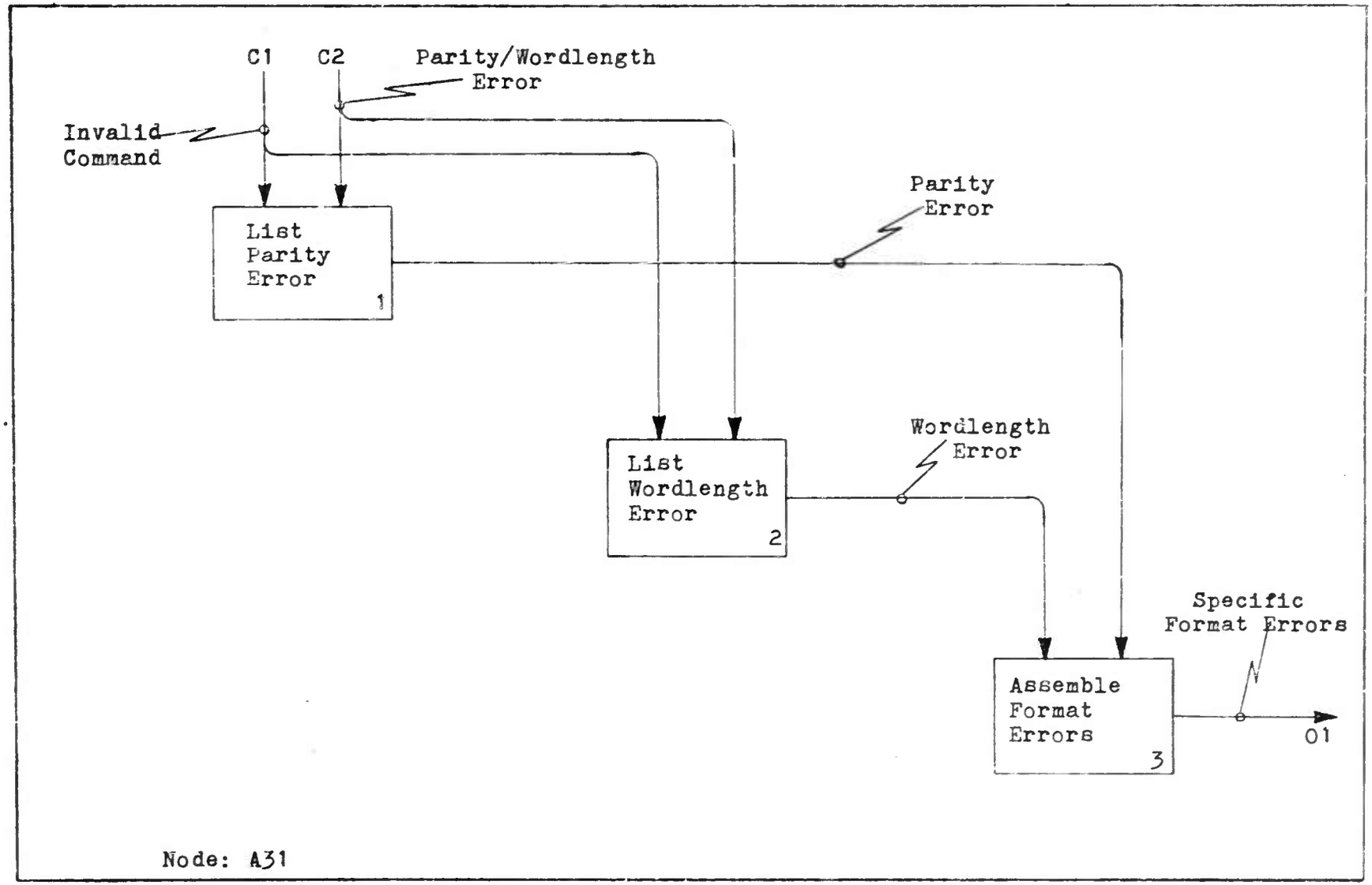


Figure 15. List Types of Format Errors

A31 Text: A parity error (101) and/or a wordlength error (201) is listed by Assemble Format Errors (3). The specific format errors (01) that are ouput are used to create the proper command verification words that will be placed in the simulated downlink.

This node presents a decomposition that is almost down to the coding level. Even though a small amount of new information is introduced, it is included to enhance the clarity of the parent node (A3).

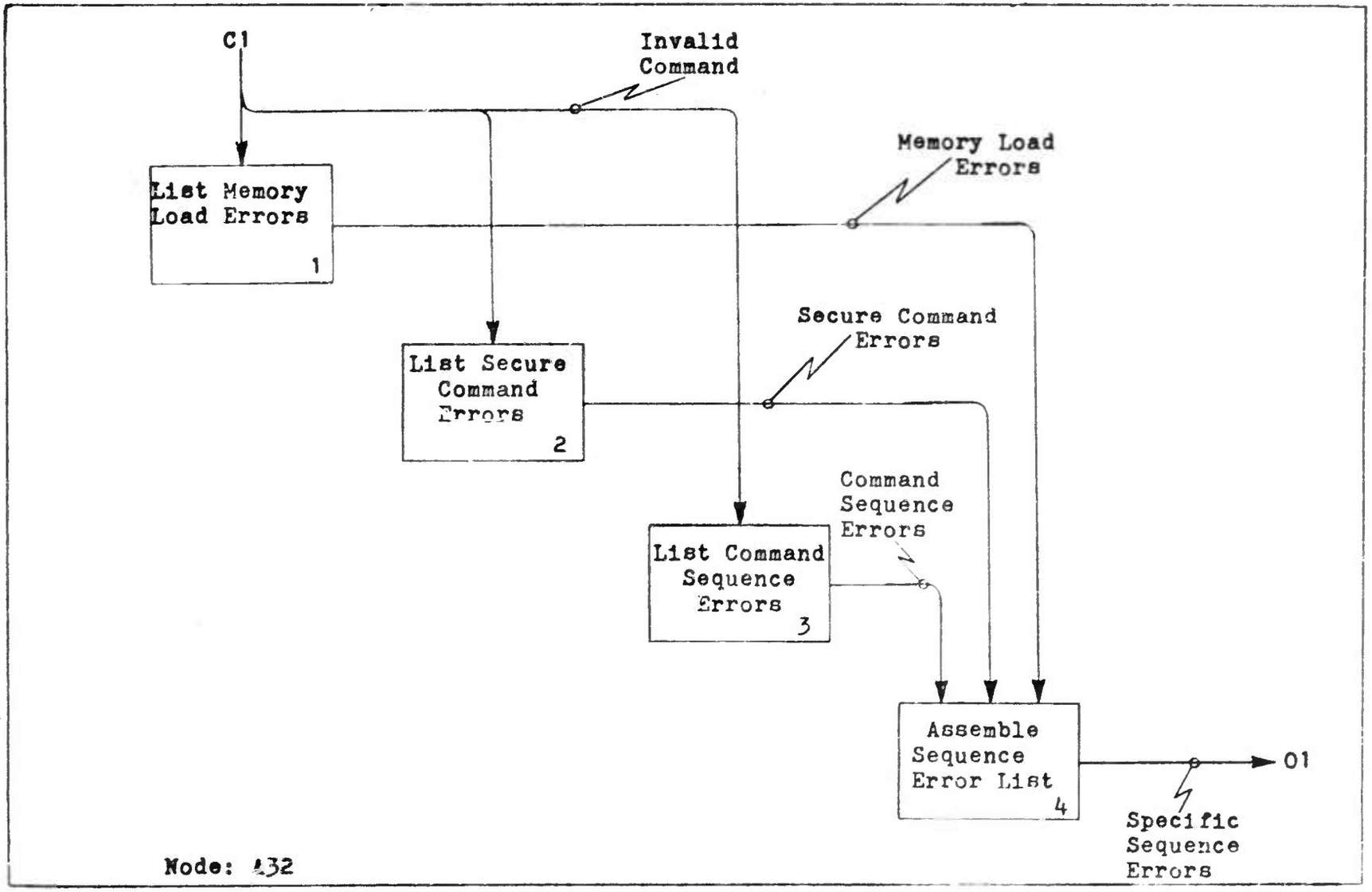


Figure 16. List Types of Sequence Errors

A32 Text: An invalid commadn (C1) is generated due to a command sequencing problem. List Memory Load Errors (1) is activated if the invalid command is an erroneous memory load command. A distinction is made between secure commands and non-secure commands by activity blocks (2) and (3). These two types of commands generate their own set of command verification (CV) codes as a result of errors. The various sequencing errors that occur during a command process are assembled in (4) and the specific sequence errors (401) are used to determine the proper vehicle message codes required to generate the CV words.

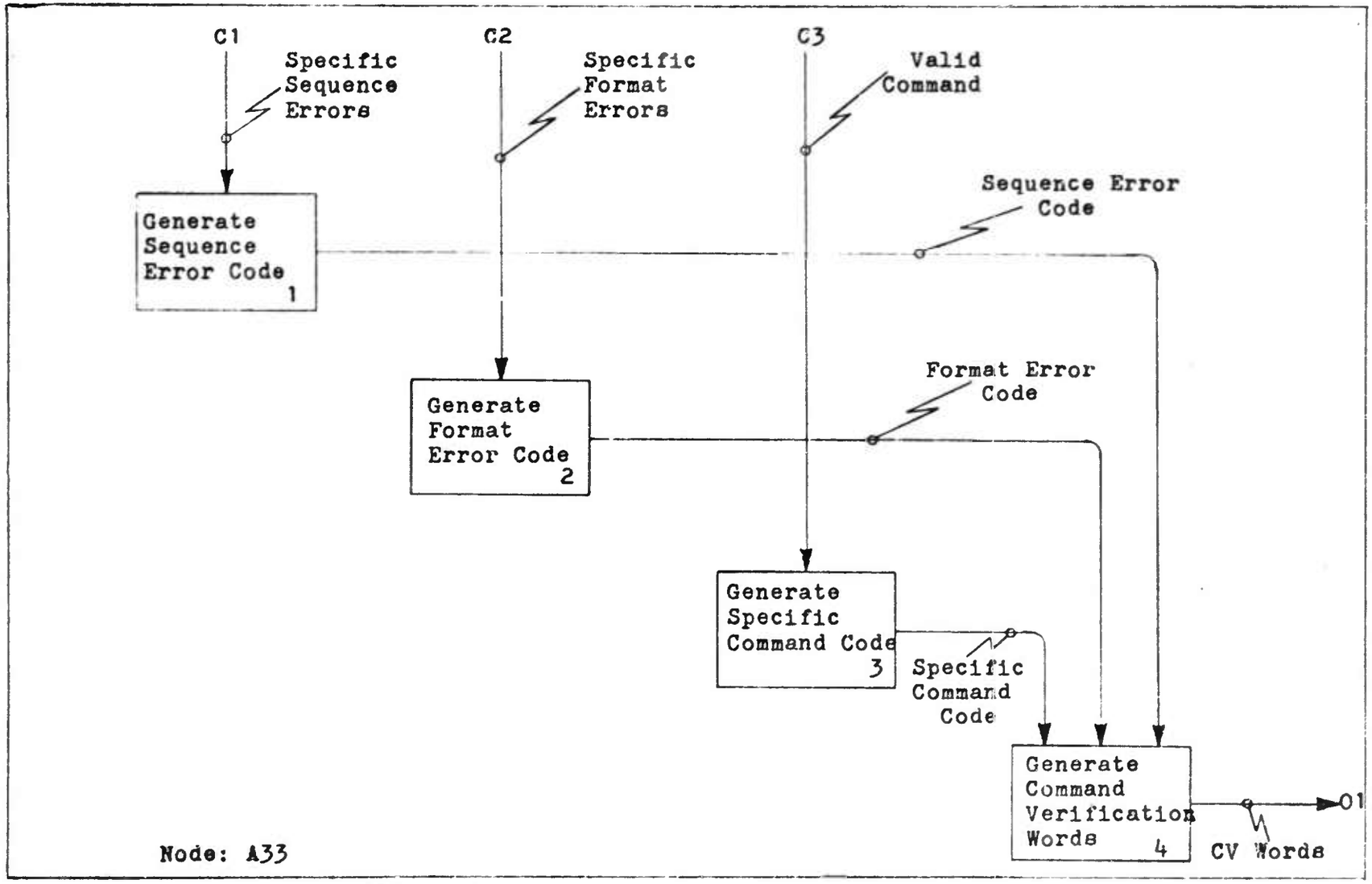


Figure 17. Determine Proper Vehicle Messages

A33 Text: The specific errors, (C1) and (C2), that are generated by an invalid command, are used to control the generation of the applicable error codes in activities (1) and (2). If a valid command (C3) is processed, Generate Specific Command Code (3) is activated to provide the required code for Generate Command Verification Words (4). All of the codes generated contribute to the creation of the appropriate CV words (401).

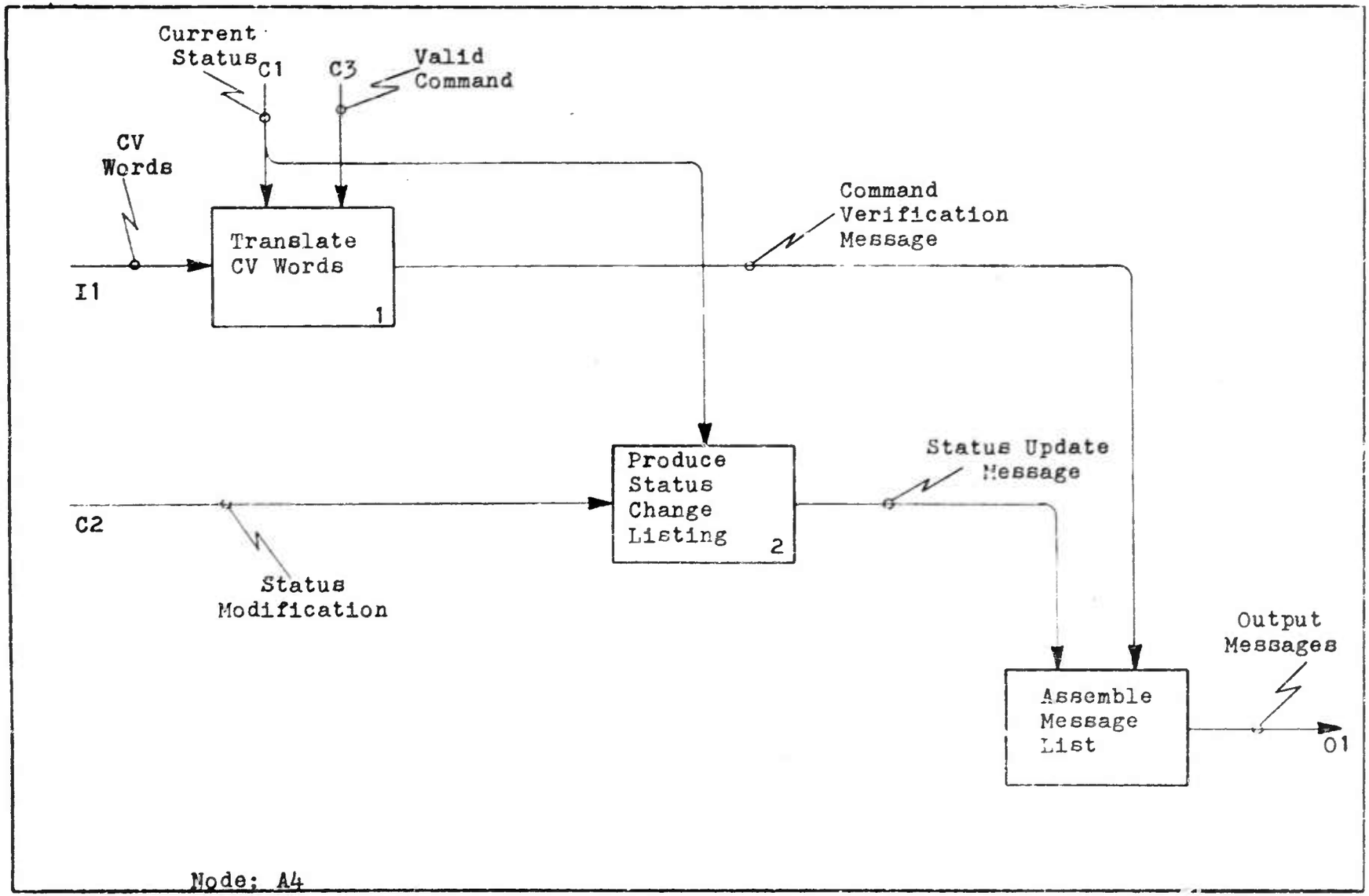


Figure 18. Create Output Messages

A4 Text: CV words (I1) are input to Translate CV Words (1) in a 16 bit hexidecimal code. The words are translated, and the CV message (101) that results is determined by the current status (C1), and the valid command that may be present (C3). The status modification (C2) inputs to Produce Status Change Listing (2) and, under control of the current status, a list of status changes (201) is produced. These status changes, combined with the CV message, aid in producing a meaningful output message (O1).

Data Model

Again, as with the activity model, it is recommended that the following "node index" be scanned for an overview of the decomposition.

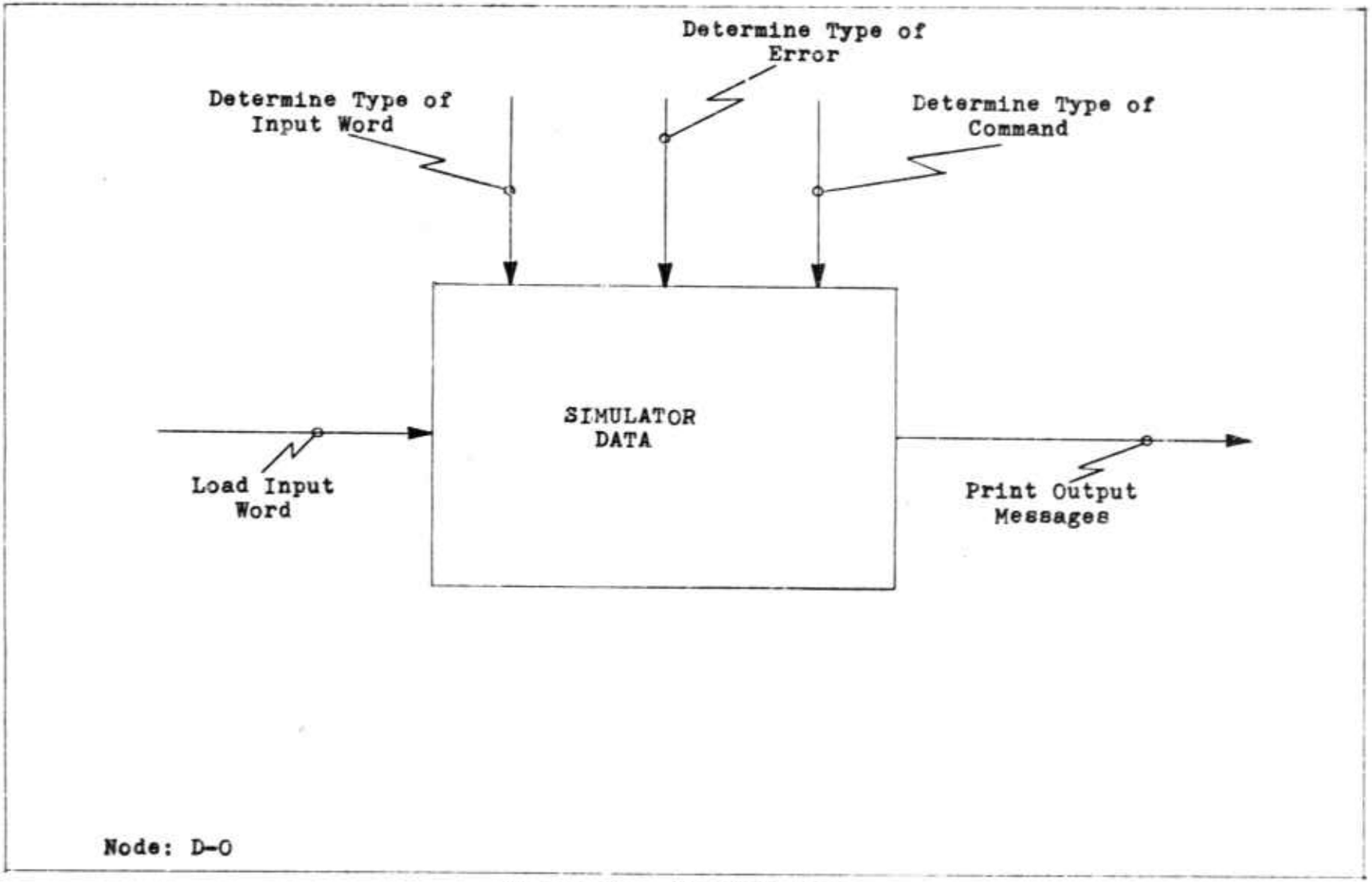


Figure 19. Simulate Data

D-O Text: The input activity that creates or modifies the Simulator Data is the loading of the input word (I1). What is done as a result of the input word is constrained by the determination of the type of input word loaded (C1), the type of error (parity, wordlength, or sequence) that exists (C2), and the type of spacecraft command that is transmitted (C3). Any input that is loaded results in the printing of an output message (O1) for user information.

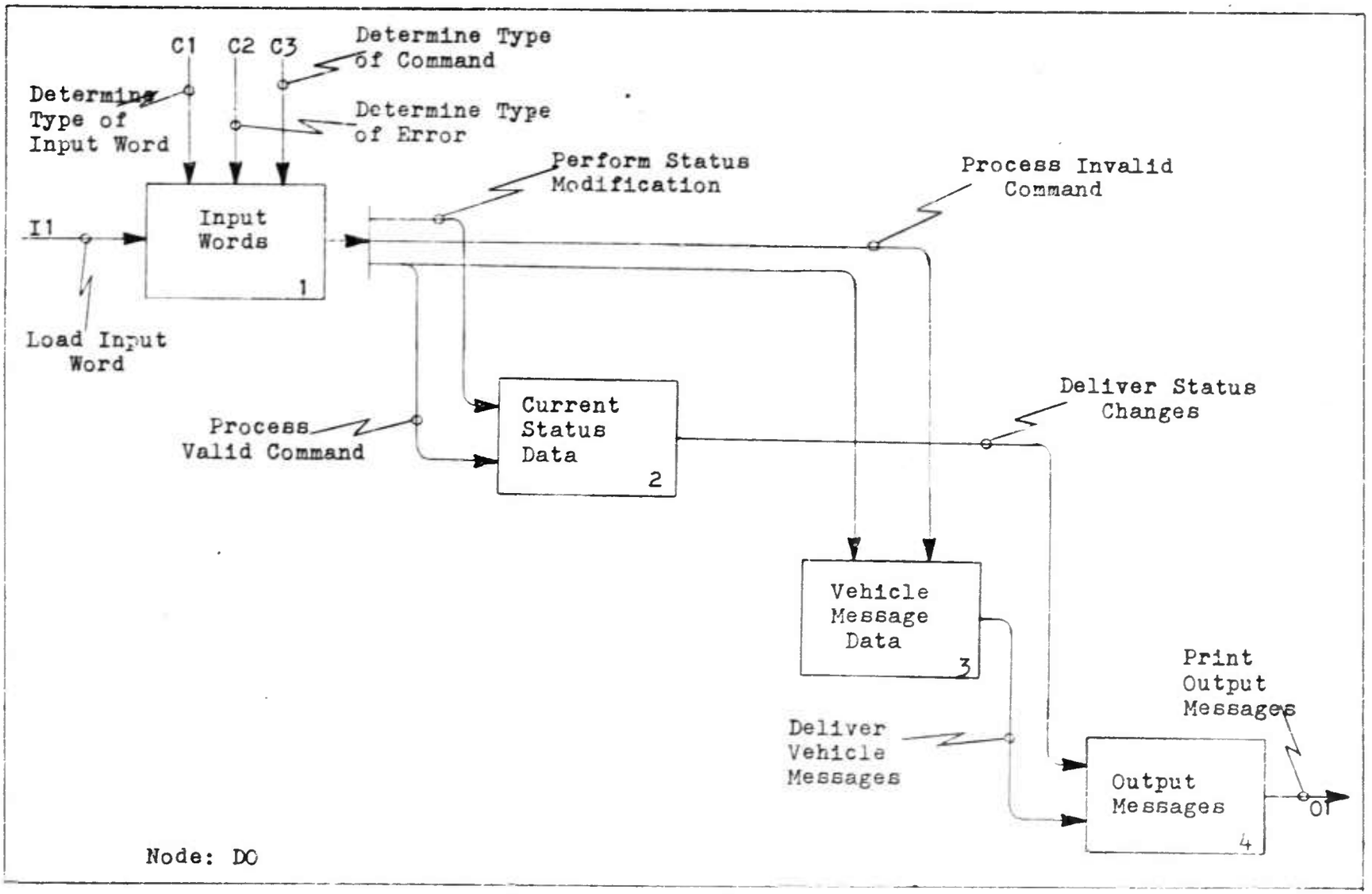


Figure 20. Simulate Data

DO Text: An input word is loaded (I1) which is either in the form of a user command or a spacecraft command. The Input Words (1) that are created are then used by one of three possible activities. The activity that uses the Input Word is constrained by the type of input word (C1), type of error (C2) that may exist, and type of command (C3) that may be present.

If the input word is a user command, some type of status modification or vehicle information update will be performed (101). This results in either the creation of some Current Status Data (2), or a change to some data that already exists. If a spacecraft command is input, either an invalid or a valid command will affect the Current Status Data.

The valid or invalid command process determines what type of vehicle message will be delivered (301). The Vehicle Message Data (3) is shown without an input process to create it. This data will have been created as an external function and will reside in the simulator for access. The data consists of command verification and error codes that will never be altered as a function of the simulator.

The status changes and vehicle messages created during the command process are assembled as Output Messages (4) and used to print output messages (01).

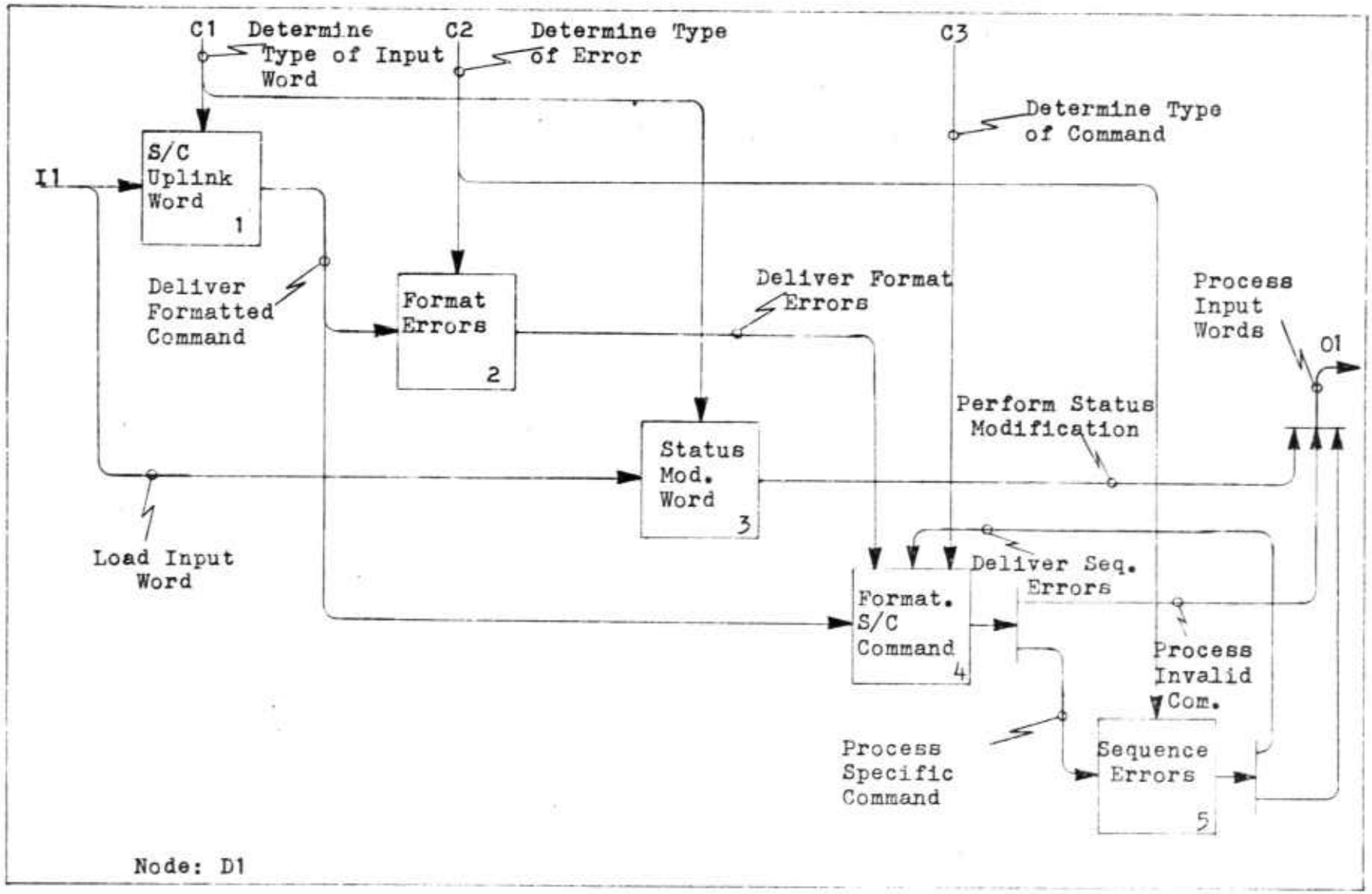


Figure 21. Input Words

D1 Text: If the input word (I1) is a spacecraft command the Spacecraft Uplink Word (1) is created. If it is a user command a Status Modification Word (3) is created. The Status Modification Word is used to perform the particular status modification (301). The Spacecraft Uplink Word (1) is delivered in the required format to create a Formatted Spacecraft Command (4) and Format Errors (2) if they exist. The Formatted Spacecraft Command is used to process the invalid command (402) or it is delivered as a specific command which generates Sequence Errors (5). Sequence Errors are fed back to the Formatted Spacecraft Command (4) and used to process the invalid command.

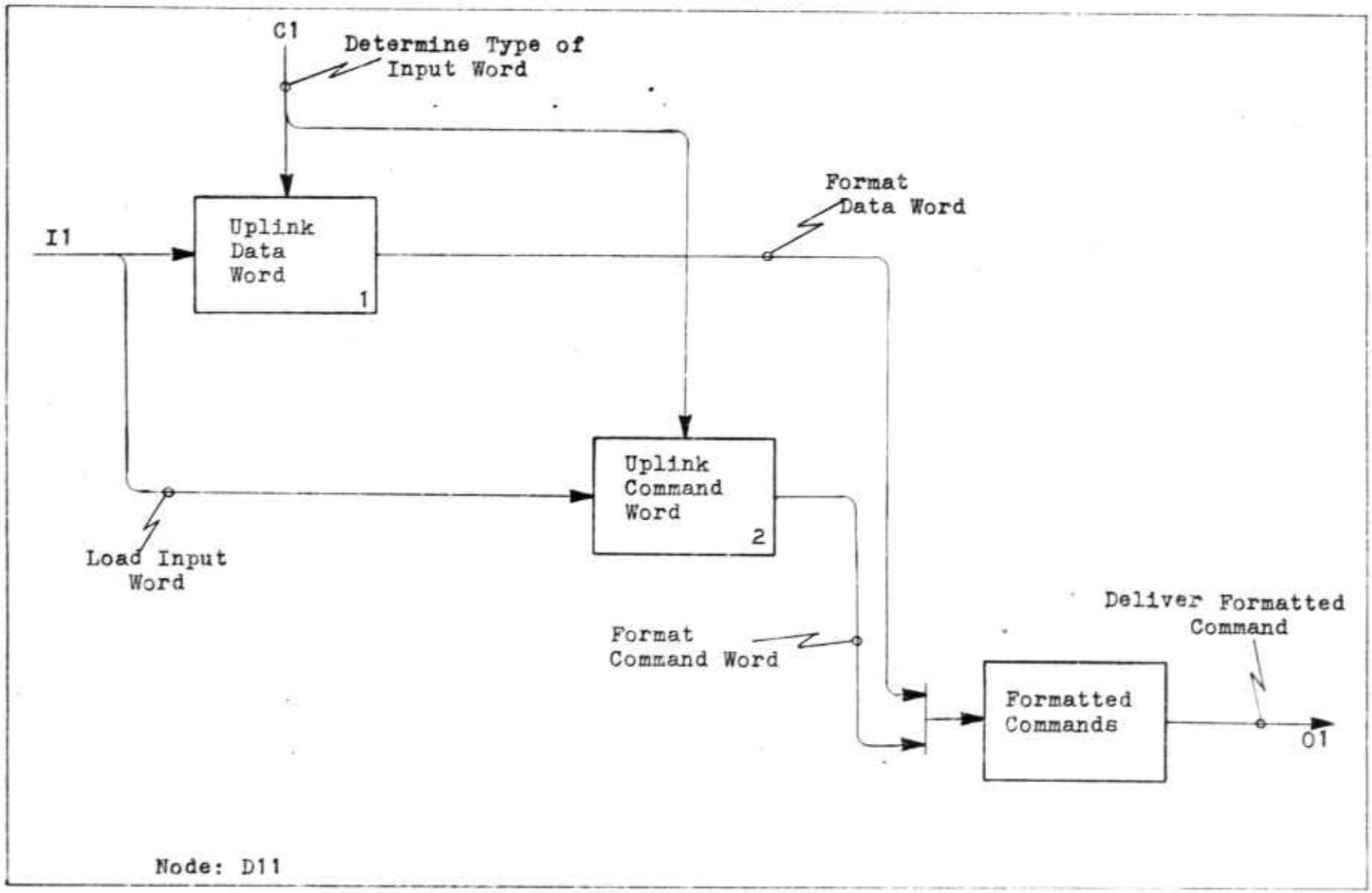


Figure 22. Spacecraft Uplink Words

D11 Text: An input word will either create an Uplink Data Word (1) or an Uplink Command Word (2). Whichever of the two is created, it must be placed in the proper format and wordlength to be used by the simulator. After the Formatted Commands (3) are created they are delivered for further processing (01).

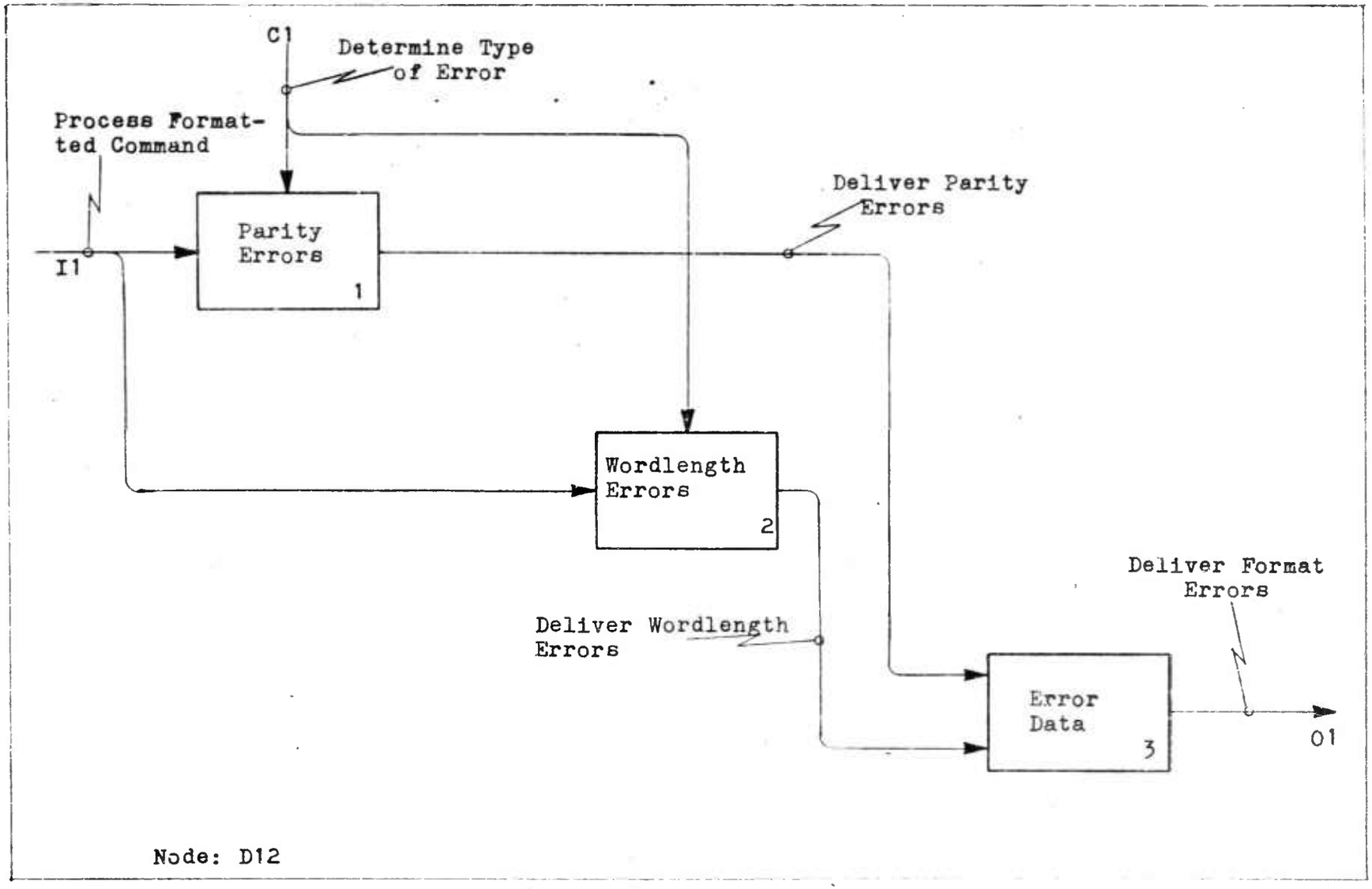


Figure 23. Format Errors

D12 Text: When processing the formatted command (I1), parity and wordlength are checked. The occurrence of either or both types of errors results in the creation of Parity (1) and/or Wordlength Errors(2). These errors are then used to create the Error Data (3).

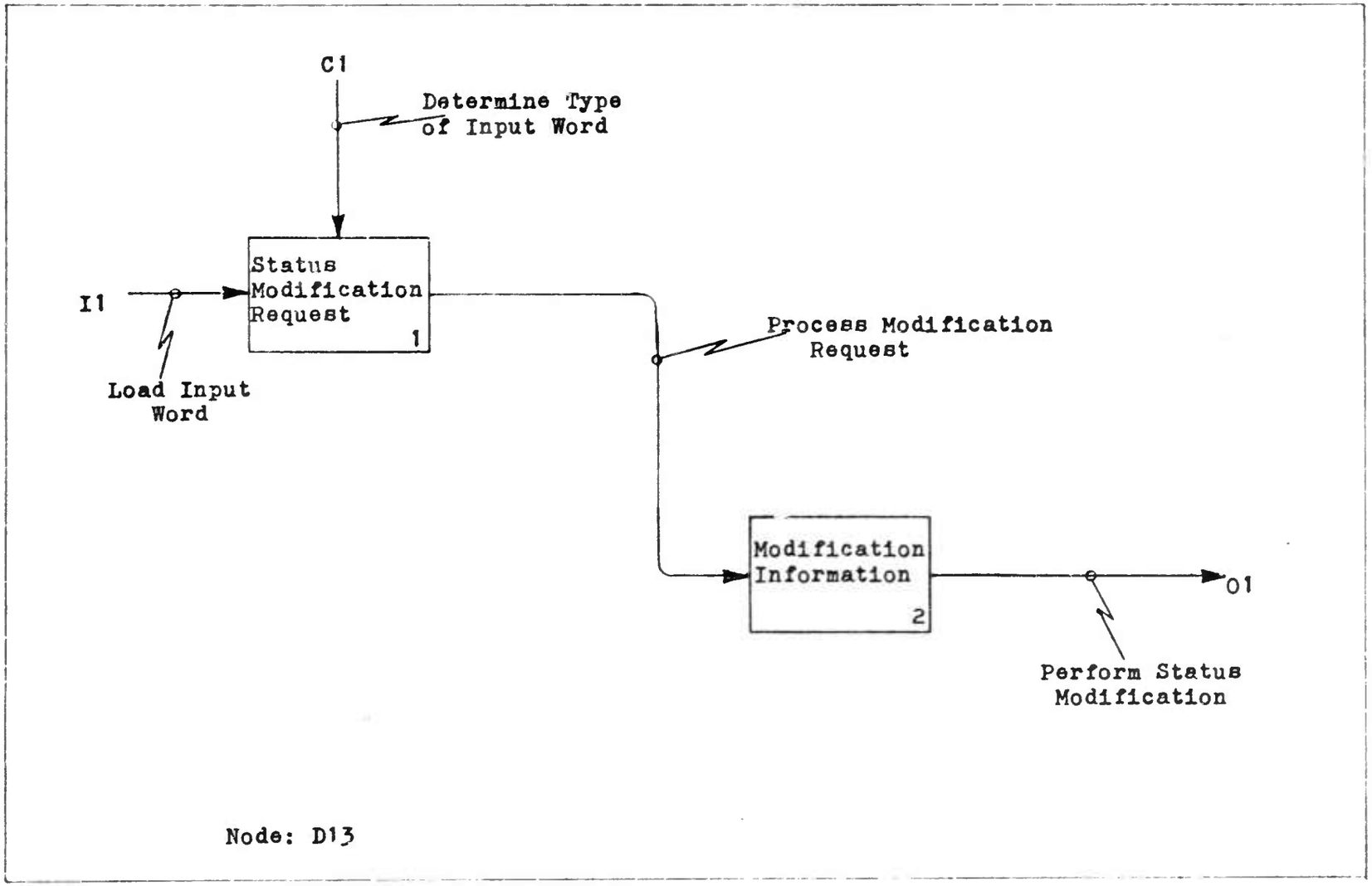


Figure 24. Status Modification Word

D13 Text: After it has been determined by (C1) that the input word (I1) is a user command, a Status Modification Request (1) is created. This request is then processed to determine the particular modification required, and the Modification Information (2) is created to be used to perform the modification (O1).

It should be noted at this time that much of the software required to perform the modifications or updates to the simulator may already exist. The node is included as a reminder that the function must be incorporated in the simulation.

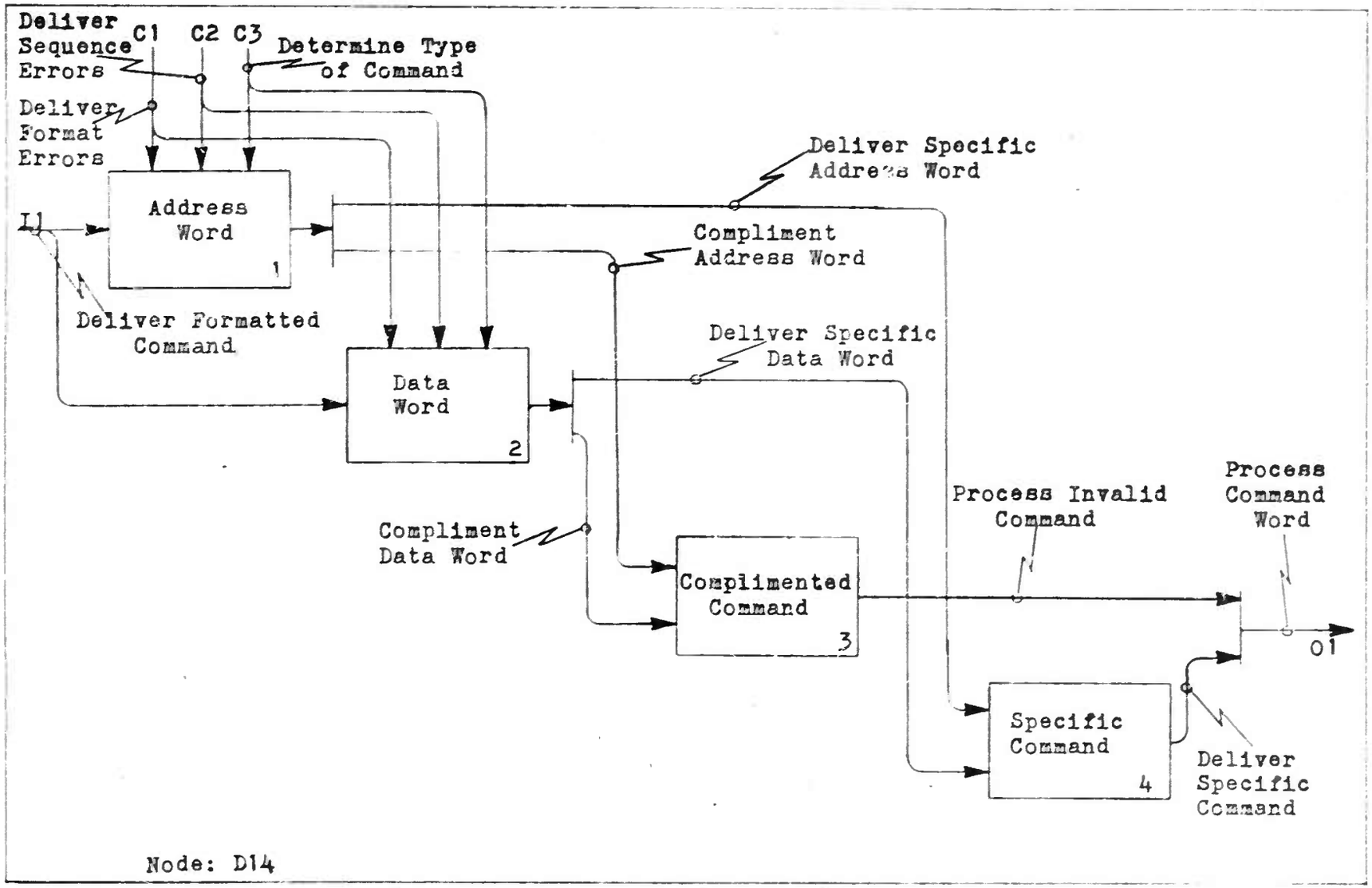


Figure 25. Formatted Spacecraft Command

D14 Text: The formatted command (I1) is made up of an Address Word (1) and a Data Word (2). The specific content of these words is determined by C1, C2, and C3. Many different things could be the cause of a particular command to be complimented. The complimenting is dependent upon the type of command word present, and on the existance of some kind of error in that command. It is also dependent upon any combination of the two circumstances. The same constraints apply to the complimenting of the Data Word (2). The Complimented Command (3) created by both the address word and the data word is used to further process the invalid command (301). By knowing a priori which commands are to be complimented and which commands are not to be complimented, a means of detecting commanding errors is provided. The Specific Command (4) is made up of a good address word and a good data word. It must be processed (401) to determine if any sequence errors exist.

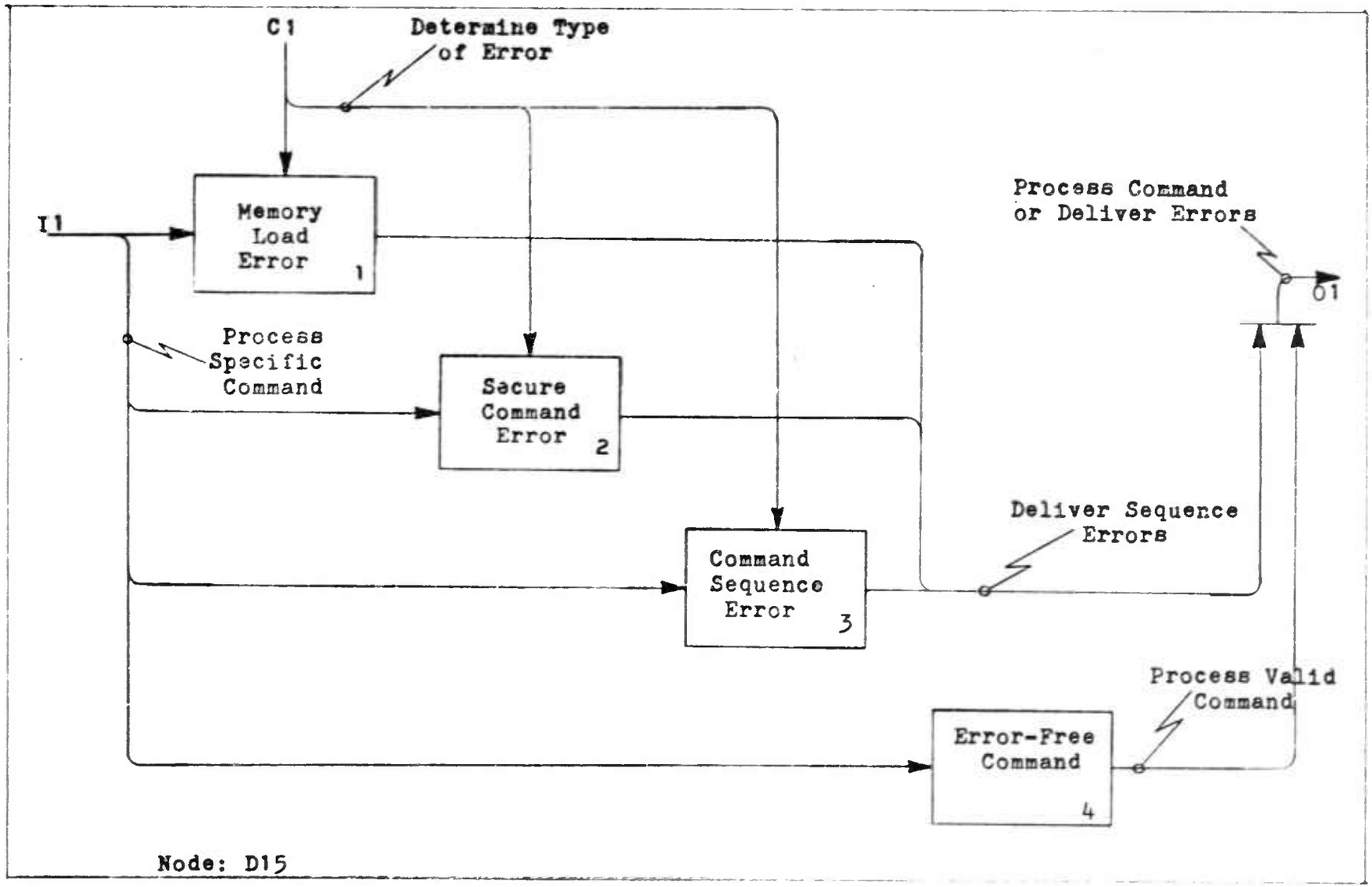


Figure 26. Sequence Errors

D15 Text: After a command is found to have no parity or wordlength errors, it must be checked for proper sequencing. If a sequence problem is detected, either a Memory Load Error (1), a Secure Command Error (2), or a Command Sequence Error (3) is produced. When these errors are created, they are delivered (301) back to the previous node (D14) for further processing. When a command is received that has no sequence problem, an Error-Free Command (4) is created and used to process the valid command (402).

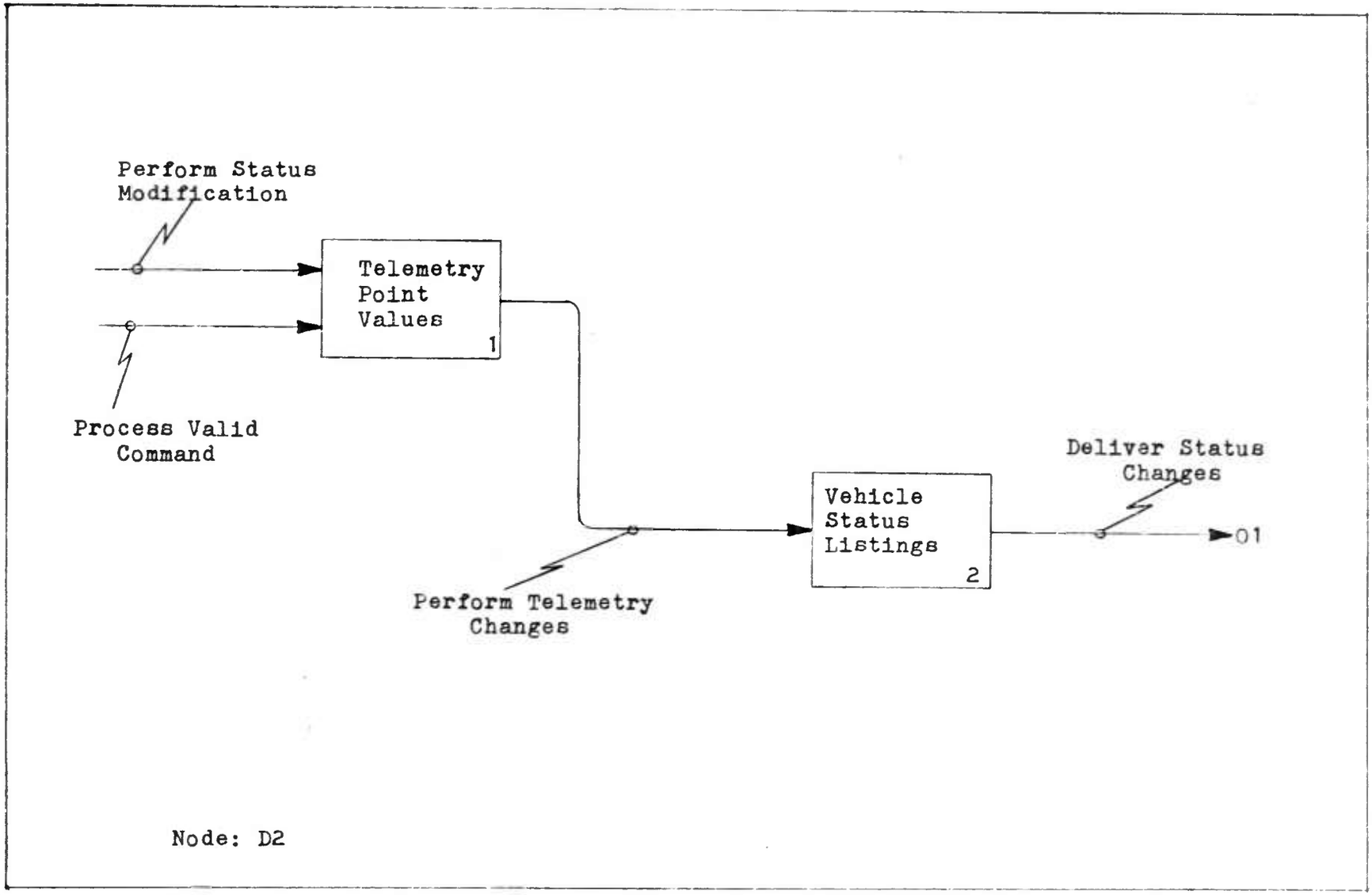


Figure 27. Current Status Data

D2 Text: Telemetry Point Values (1) are the main source of status information. A valid command (I2) must be processed to determine which telemetry indications are affected by its execution. Once the telemetry changes are performed (101), they are used in the update of the Vehicle Status Listings (2). The status changes (201) are then used to aid in compilation of the output messages. Performance of status modifications (I1) requires the same data processes as above.

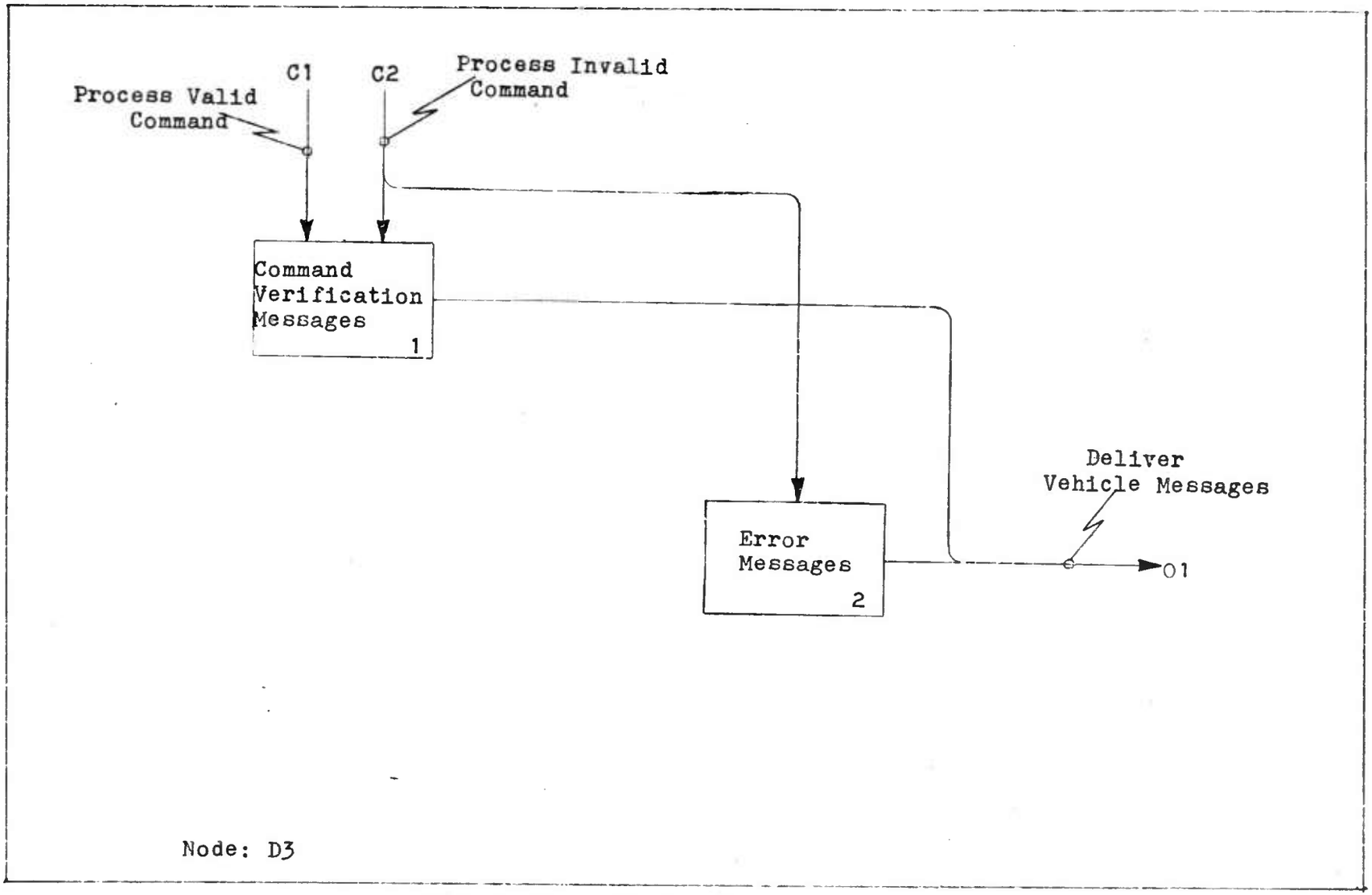


Figure 28. Vehicle Message Data

D3 Text: The processing of a valid or an invalid command causes the delivery of a command verification (CV) message (101). These messages are derived from a file of CV codes that are available for access during simulator use. The invalid command also accesses error codes that are on file for creation of proper Error Messages (2). These vehicle messages are passed on be assembled into a meaningful output message.

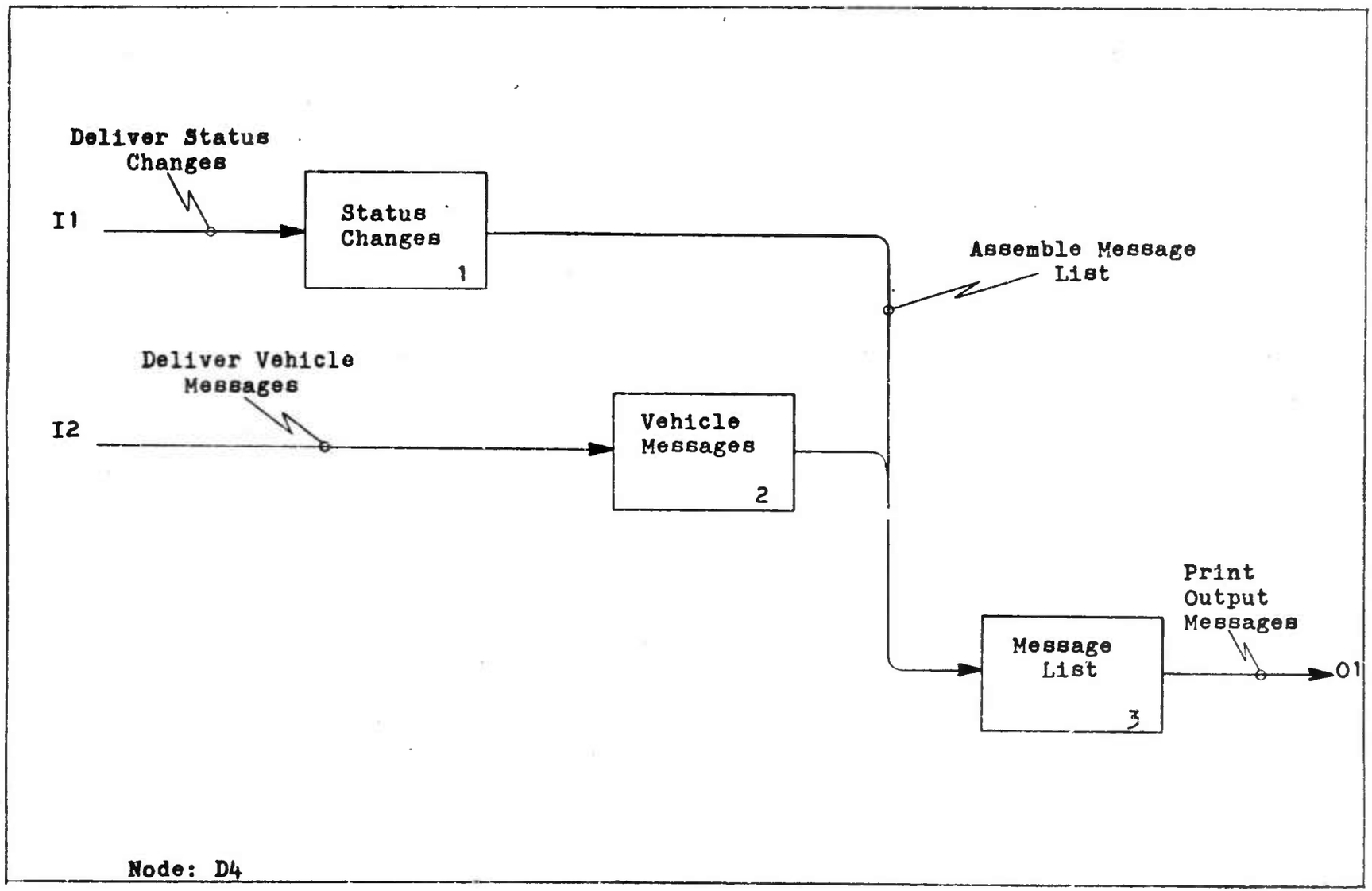


Figure 29. Output Messages

D4 Text: Status Changes (1) and Vehicle Messages (2) are created as a result of spacecraft command or user command processing. A Message List (3) is created which is comprised of all the user or vehicle generated changes that have occurred as a result of a command. This Message List is printed (01) for use by the system operator.

Summary

After a brief introduction and explanation of some diagram syntax, this chapter presents the activity and data models of the simulator. They are given as a complete functional specification of the software. None of the box titles or arrow labels are binding, however, and in the design refinement of the next chapter some names and labels are changed to better meet the software engineering goal of understandability.

IV. Design Refinement

Introduction

In the previous chapter, a top-down design strategy is used to create activity and data models which identify a basic design structure. To better prepare the design for easy implementation, a structure is needed which reveals the relative "goodness" of the design. This goodness is measured by observing the following attributes: (1) coupling, (2) cohesion, (3) span of control, and (4) scope-of-effect/scope-of-control (Ref 10:340). If two modules are totally independent of each other, and one can function completely without the other, then they are loosely coupled, or uncoupled. Loosely coupled modules are more maintainable than tightly coupled modules. Low cohesion occurs when an isolated module has internal elements that are loosely related. Cohesiveness and coupling are interrelated. The greater the cohesiveness of individual modules, the lower the coupling between any pair of modules will be (Ref 10:144). Excessive span of control is bad because it indicates too much decomposition of a module into subordinates. This is caused by a failure to identify intermediate levels of abstraction. Finally, scope-of-effect/scope-of-control should be examined to get a measure of how well the system has adhered to the subordinate structure required of any decision process: all modules that are affected by a decision (scope of effect) should be subordinate to (scope of control) the module which makes the decision (Ref 10:240). To recognize and isolate the existence of these effects, and then to eliminate them with a design refinement, a technique is employed

which is based on a top-down, structured design strategy called "transform analysis." This strategy should lead to system structures which are fully factored and ready to code (Ref 10:254).

Transform analysis produces a "structure chart" which reveals design goodness measures. The structure chart contains modules arranged top-down and left-right which represent the processing functions of the system. The first step is the restatement of the problem as a data flow graph or "bubble chart." The basic elements in a bubble chart are called "transforms" which are represented by circles labeled with short descriptions of the transformation. Interconnections between transforms are labeled arrows which represent "data elements" to be transformed. Two or more data elements required simultaneously by a transform are indicated with an asterisk ("*") between the data elements. The "ring-sum" operator ("⊕") is used to denote exclusive-OR relationships between data elements. Bubbles and arrows are drawn which represent input branches and output branches. These branches are connected by bubbles that are the "central" transforms of the system. The second step is to identify "afferent" and "efferent" data elements. An afferent data element is an input to a central transform and an efferent data element is an output from a central transform. The third step is top-level factoring. A "main" module is specified and it is decomposed into afferent, efferent and central modules. The fourth step is the decomposition of the afferent, efferent and central modules. The top-level modules and their subordinates form the "first cut structure chart." Finally, the first cut structure chart is examined for goodness and revisions are made to produce the final struc-

ture chart (Ref 10: Chap. 10).

After the preliminary design is complete the first cut structure chart is drawn from the activity model, bypassing the creation of a bubble chart. This decision is based on the conceptual resemblance of the activity model to the bubble chart in that both tie activities together with data elements. Reference 3 explains a method of design which creates an "intermediate" bubble chart from the first cut structure chart. Iterations of bubble-to-structure-chart and structure-to-bubble-chart transformations are made to further refine the design. This iterative process advocates starting with a structu . chart and then creating the bubble chart for the transform analysis.

In the remaining sections of this chapter, the first cut structure chart is constructed as a direct translation (input for input, output for output) from the activity model in Chapter III. The afferent and efferent data branches are identified to aid in construction of the intermediate bubble chart. Finally, a "refined" structure chart is constructed and examined to determine the relative goodness of the design.

First Cut Structure Chart

Figures 30a, 30b, 30c and 30d illustrate a direct translation of the activity model of Chapter III to a structure chart. Decisions and loops are not identified at this time; they are included in the refined structure cha derived later. The number to the left of each arrow identifies the input and output parameters listed in Table I. Controls are not indicated in this list because they

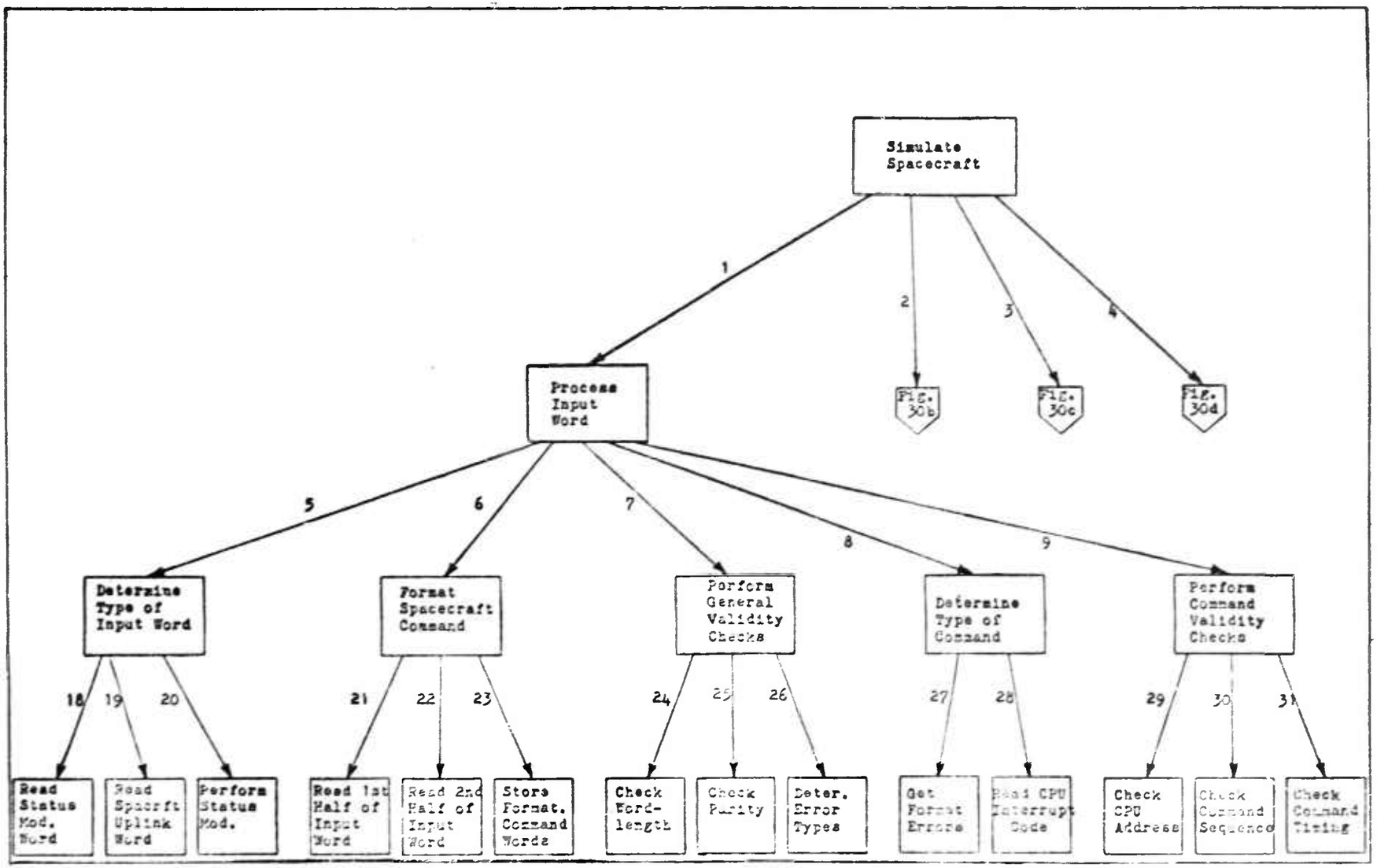


Figure 30a. First Cut Structure Chart

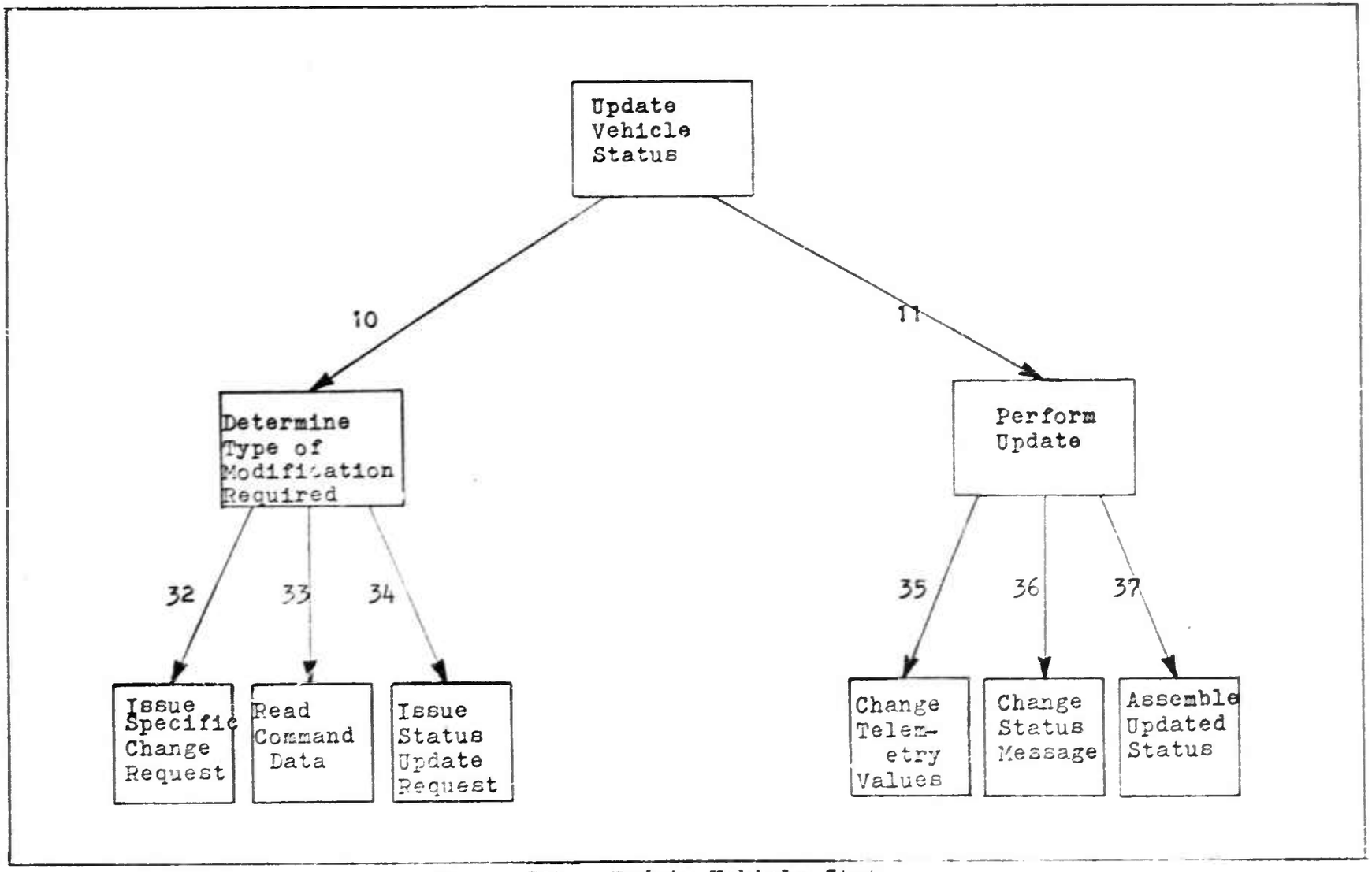


Figure 30b. Update Vehicle Status

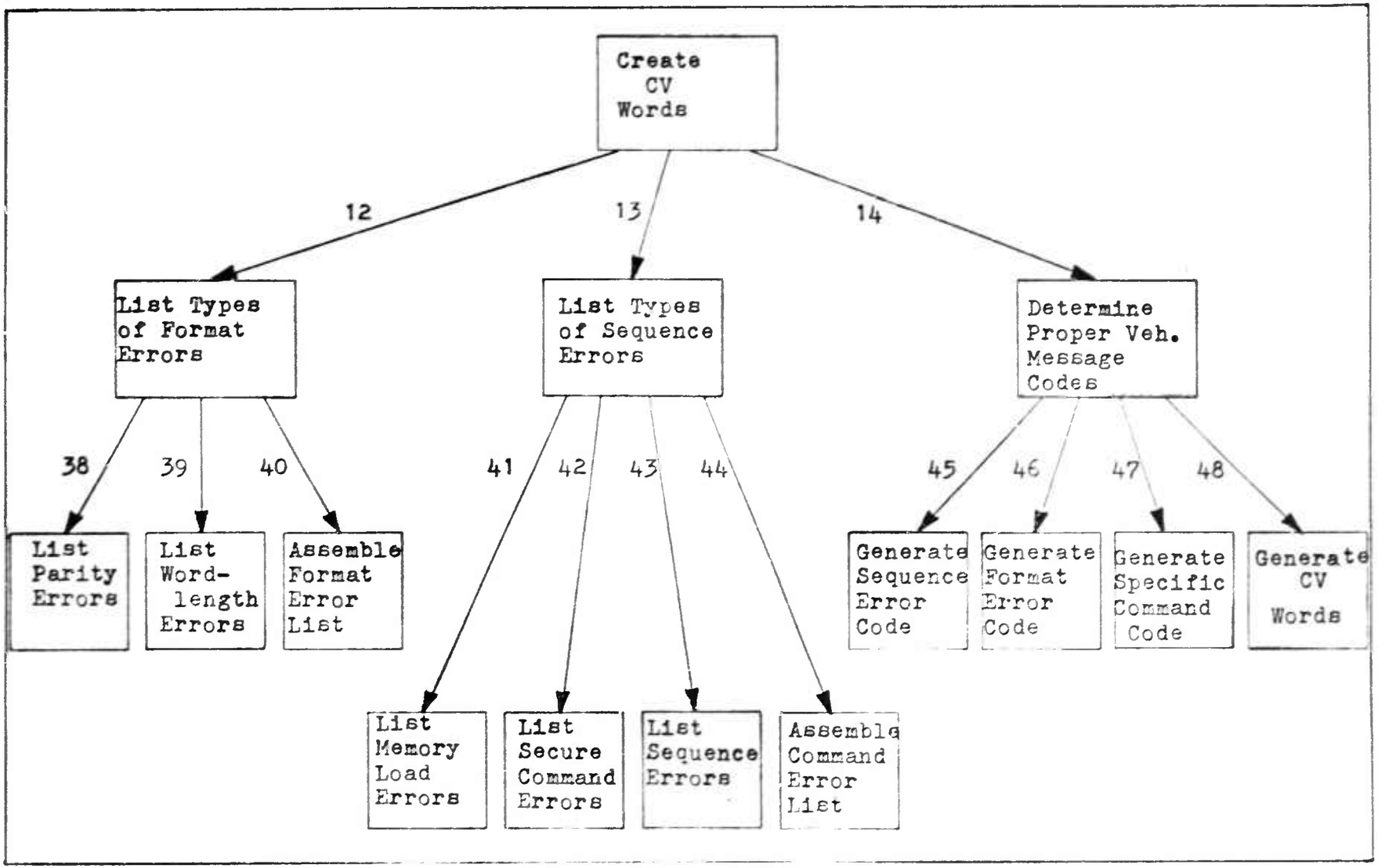


Figure 30c. Create CV Words

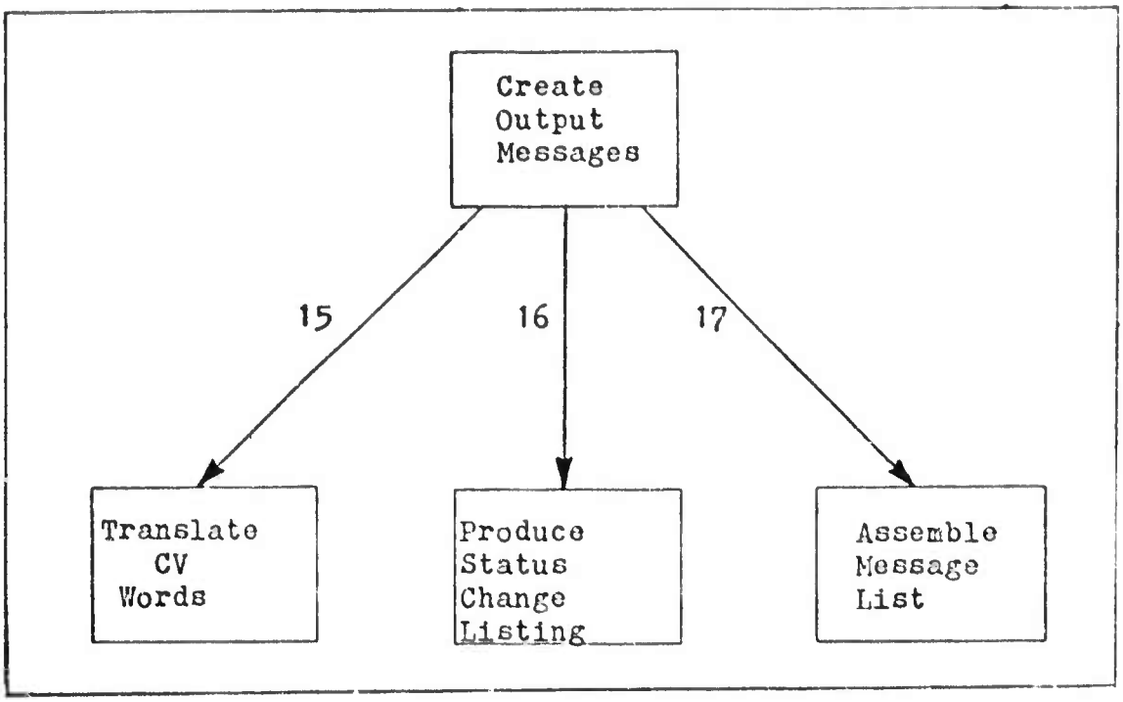


Figure 30d. Create Output Messages

are not considered when creating a bubble chart for transform analysis. However, a control element in an activity module may be used as an input element in a bubble chart.

Intermediate Bubble Chart (Data Flow Graph)

The bubble chart transforms "conceptual inputs" into "conceptual outputs," which implies that the detail in the structure chart may be higher than in the bubble chart (Ref 3: Chap. 2, p. 25). A scan of the parameters in Table I shows some repeated input and output names that may have to be altered to clarify the data flow. If there are any errors in the structure chart, they should be corrected while drawing the bubble chart. Looking at the top level in Figure 30a, the afferent data branch is recognized as that branch which consists of the subordinates of the Process Input Word module.

Table I

First Cut Structure Chart Parameters

	Input	Output
1	Input Word	Status Mod, Invalid Cmd, Valid Cmd
2	----------	Current Status
3	----------	CV Words
4	CV Words	Output Messages
5	Input Word	Status Mod, S/C Cmd
6	Input Word	Formatted Cmd
7	Formatted Cmd	Format Error
8	Formatted Cmd	Invalid Cmd, Specific Cmd
9	Formatted Cmd	Invalid Cmd, Valid Cmd
10	----------	Update Request
11	----------	Current Status
12	----------	Specific Format Errors
13	----------	Specific Sequence Errors
14	----------	CV Words
15	CV Words	Command Verification Message
16	Status Mod	Status Update Message
17	----------	Output Messages
18	User Cmd	Status Mod Request
19	S/C Uplink Cmd	S/C Cmd
20	----------	Status Mod
21	Input Word	S/C Data
22	Input Word	S/C Address
23	----------	Formatted Cmd
24	Formatted Cmd	Wordlength Error
25	Formatted Cmd	Parity Error
26	----------	Format Error
27	----------	Erroneous Cmd, Error-Free Cmd
28	Formatted Cmd	Command Words
29	Formatted Cmd	Address Error
30	Formatted Cmd	Sequence Error
31	Formatted Cmd	Command Words
32	----------	User Status Change Request
33	----------	Commanded Status Change Request
34	----------	Update Request
35	----------	Telemetry Changes
36	----------	Status Message Changes
37	----------	Current Status
38	----------	Parity Error
39	----------	Wordlength Error
40	----------	Specific Format Errors
41	----------	Memory Load Errors
42	----------	Secure Command Error
43	----------	Command Sequence Errors
44	----------	Specific Sequence Errors
45	----------	Sequence Error Code
46	----------	Format Error Code
47	----------	Specific Command Code
48	----------	CV Words

In Figure 30d, the efferent data branch consists of the subordinates of Create Output Messages. This leaves Update Vehicle Status and Create CV Words as the central transforms. The afferent data branch, which identifies the major input data flow to the central transforms, is the first to be drawn in the bubble chart. Following that branch down to its lowest level, the first input parameter seen is a "User Cmd." It is the input to Read Status Mod Word and the output is a "Status Mod Request." This identifies the input, first bubble, and output in this portion of the bubble chart. To locate the next bubble, the module which uses "Status Mod Request" as an input must be found. It cannot be located as an input in Table I, but it may be a control element in the activity model. Looking back to Figure 6, which is the activity node that contains the modules of interest, "Status Mod Request" is a control on Perform Status Mod. "Status Mod Request" is used as an input to the next bubble, which is Perform Status Mod. The output of Perform Status Mod is shown in the activity model as "Status Mod." Looking in Table I "Status Mod" is found as an input to Produce Status Change Listing which is in the efferent branch. The activity model is examined to see if "Status Mod" is used for control in any intermediate activities and it is found as a control input to Update Vehicle Status which is a central transform. Continuing this process, the bubble chart in Figure 31 is constructed. The completed bubble chart is now reviewed to ensure that it accurately represents the processing illustrated in the structure chart (Ref 3: Chap. 2, p. 28), then the refined structure chart is drawn and analyzed.

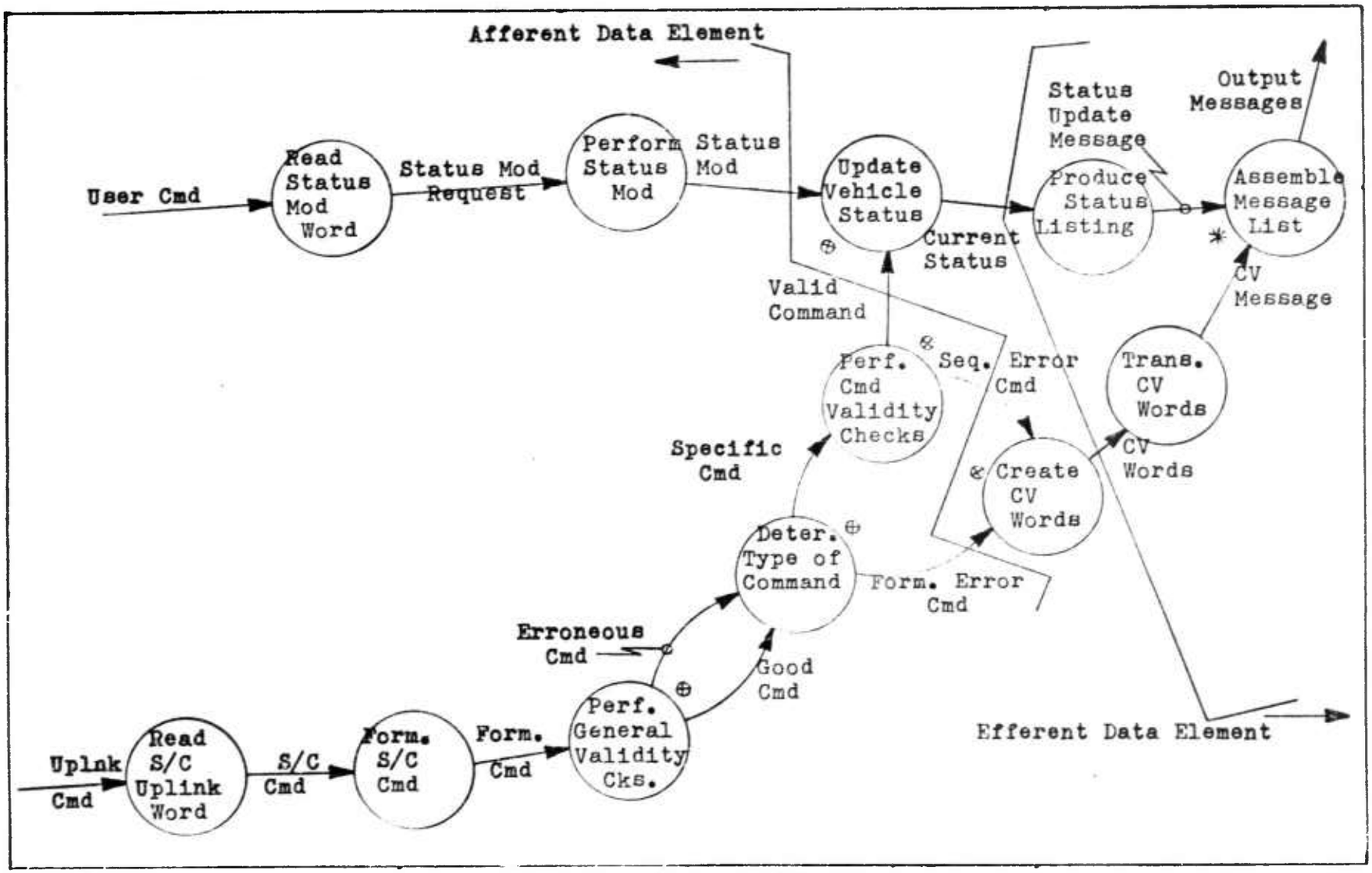


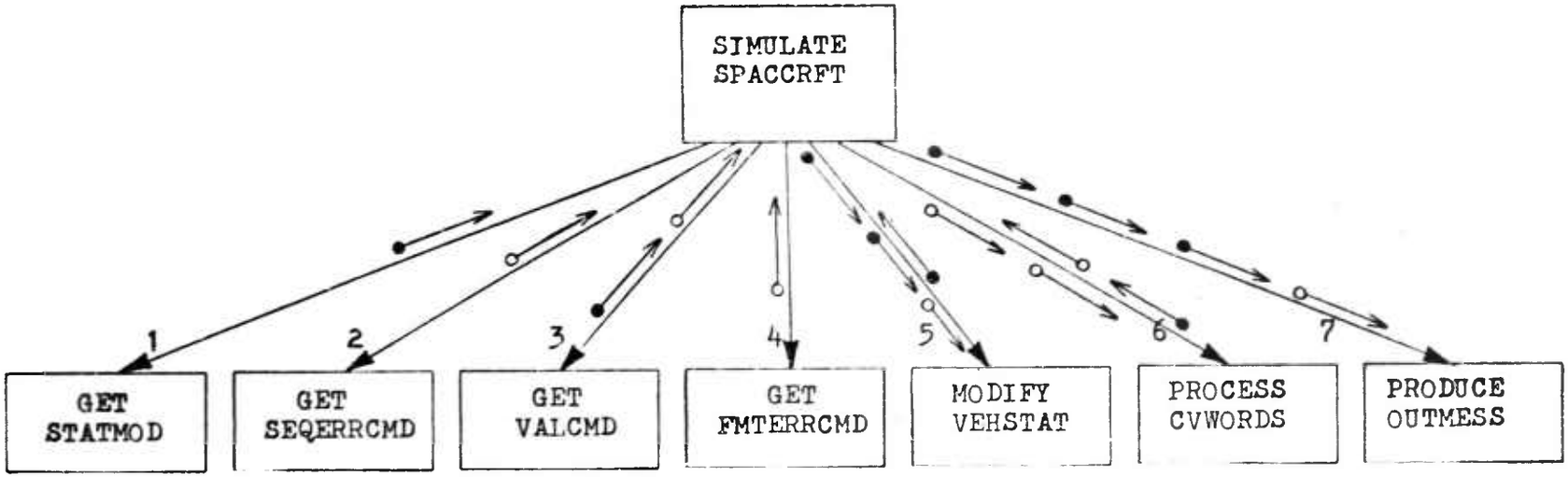
Figure 31. Intermediate Bubble Chart

Refined Structure Chart

Input and output arrows have been included on the structure charts in Figures 32, 33, 34, 35, 36 and 37 to enhance the clarity of each diagram. A dot at the end of the arrow indicates a control element, while a circle indicates a piece of data being passed between modules. The charts are presented in a top-down, left-right structure starting with the "main" or top-level module and each of its subordinates. Then each of those subordinates is decomposed and analyzed as independent sections of each branch. Accompanying each chart is a parameter list which adds to the understandability. The main module in Figure 32 identifies the afferent data branches by showing only inputs to the main module. The efferent branch has only outputs from the main module and the central transforms have data going in both directions. The following is a brief description of each module and the interconnections with its subordinate modules. At each level any decisions or loops that may occur will be explained and justified if necessary.

SIMULATE SPACCRFT. This is the main, or executive module. Its function is to control and coordinate the afferent, transform, and efferent modules dealing with the highest level data of the system. All of these subordinate modules are in the executive's scope of control. When the executive is invoked it will load the top-level afferent modules and output a "ready" message response when that loading is complete. This is a signal to the user that the simulator is ready to accept user command or spacecraft command inputs.

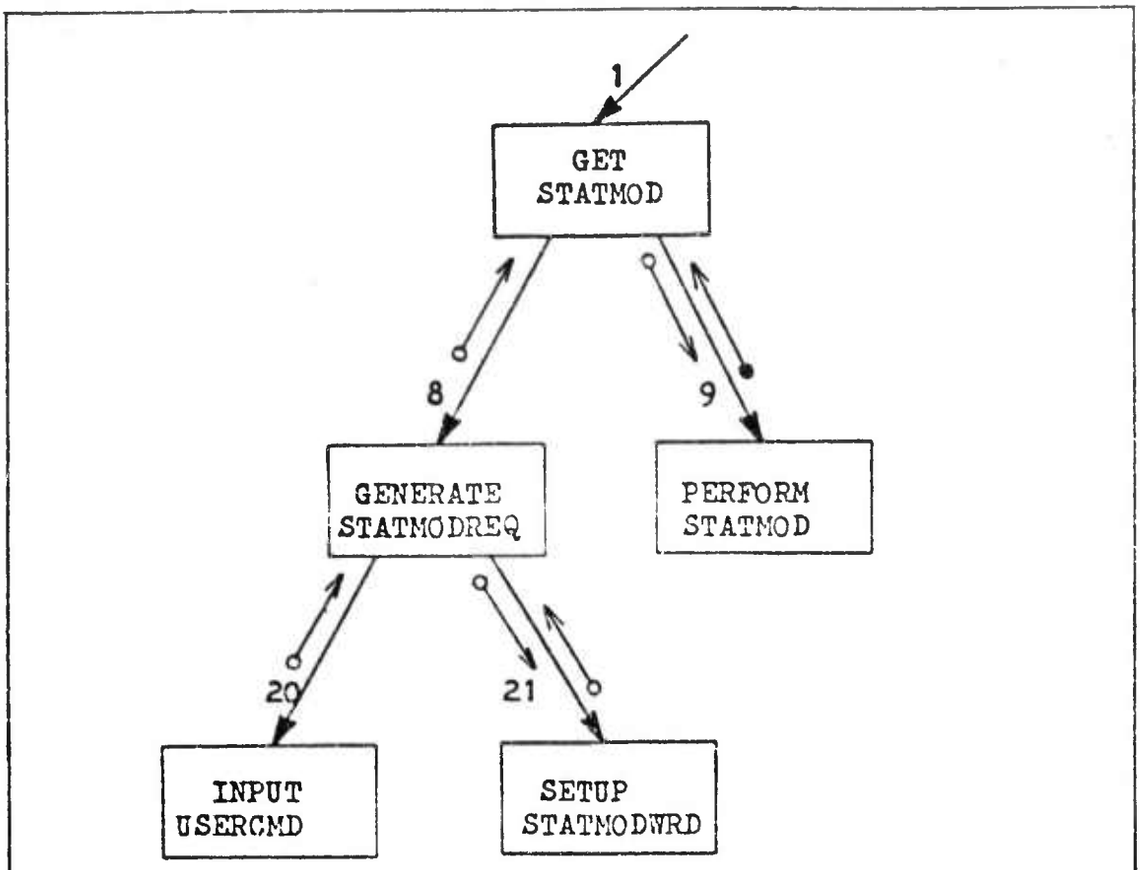
GET STATMOD. Loose coupling exists between this module (Fig-



Parameters

	Input	Output
1	-------	Status Modification
2	-------	Sequence Error Command
3	-------	Valid Command, Command Type
4	-------	Format Error Command
5	Status Modification, Command Type, Valid Command	Current Status
6	Valid Command, Invalid Command	CV Words, Errors
7	CV Words, Valid Command, Current Status, Status Modification	-------

Figure 32. Top Level Structure



Parameters

Input		Output
8	----------	Status Modification Request
9	Status Modification Request	Status Modification
20	----------	User Command
21	User Command	Status Modification Word

Figure 33. STATMOD Module and Subordinates

ure 33) and the executive. This allows the exclusion of the module from the simulator without affecting the performance of the rest of the system. The module is designed to perform functions such as adding entries to existing status and telemetry tables, deleting those entries, or changing them. These are functions that could

be performed externally if the operating system had an editing routine. If the option to use this module as part of the simulator is taken, a means of distinguishing between user and spacecraft commands should be devised. One way to do this is to prefix each user command with some identifying character.

Upon receipt of a "User Command" from INPUT USERCMD, GENERATE STATMODREQ controls its acceptance and the subsequent "set-up" necessary to pass the request to the PERFORM STATMOD module where the request is translated and the proper modification is performed. The "Status Modification" is then used as constraint data for other processes. INPUT USERCMD and SETUP STATMODWRD may be "dummy" modules whose implementation is so trivial that they may be compressed into their superordinate (Ref 10:344). Dummy modules are a common product of a top-down design. Although dummy modules rarely degrade system performance, they should be reconsidered before implementation.

Spacecraft Command Afferent Branch. This afferent data branch performs the major task of the simulator: spacecraft command processing. To enhance its description, the entire branch is drawn in Figure 34. The left-right construct convention has been violated in some places to avoid crossing arrows. The parameter list in Table II should be referenced while reading the following description.

The diamonds in the diagram represent decision processes. The presence of a decision requires examination of the scope-of-effect/scope-of-control attributes discussed earlier. GET VALCMD has PERFORM CMDVALCKS and ACCEPT SPECCMD in its scope of control.

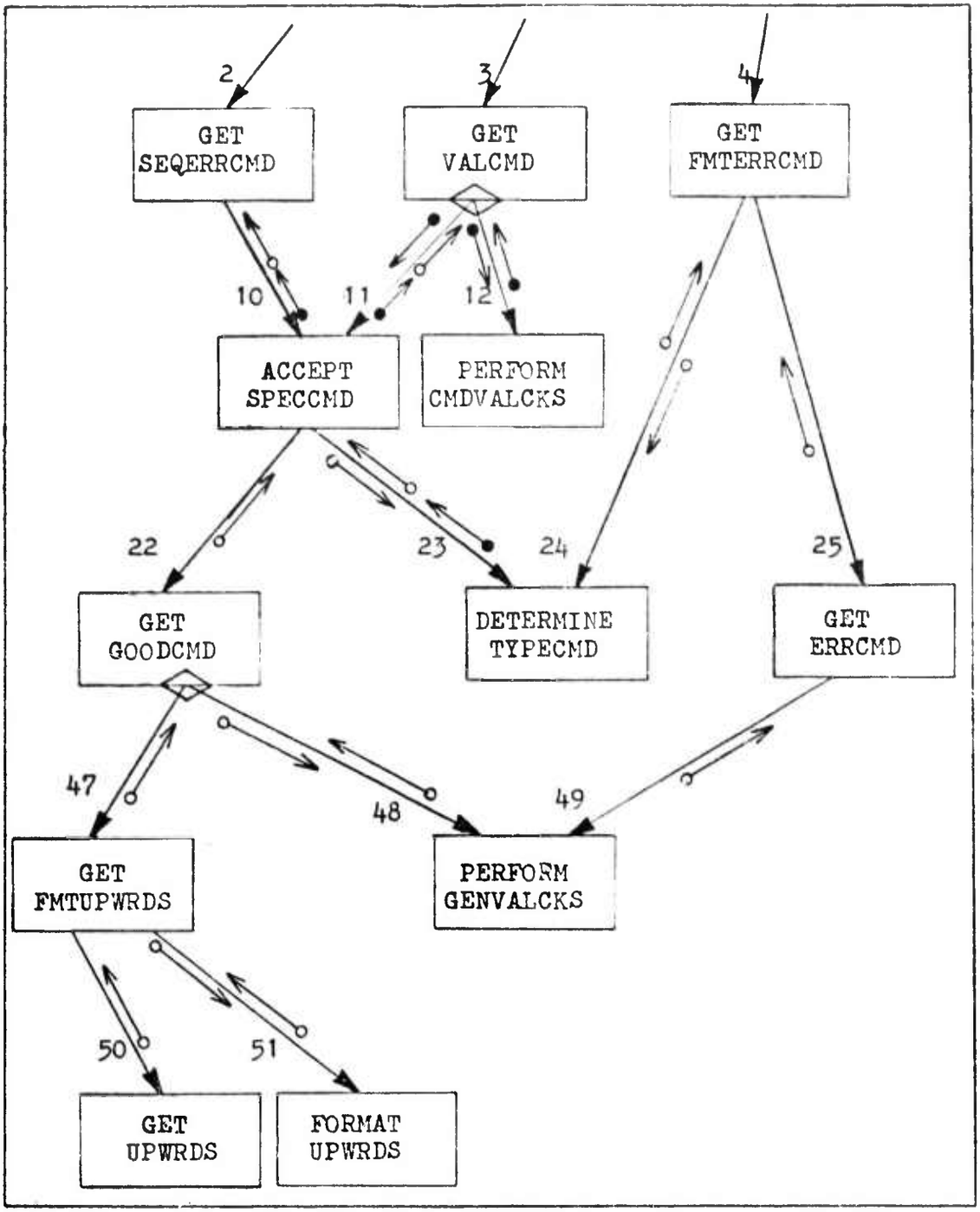


Figure 34. Spacecraft Command Afferent Branch

A "Specific Command" must be accepted by GET VALCMD and the command validity checks (sequence checks, timing checks) must be performed by PERFORM CMDVALCKS before a "Sequence Error" can be detected. Since PERFORM CMDVALCKS determines the existence of a "Sequence

Table II

Spacecraft Command Afferent Branch Parameters

	Input	Output
10	----------	Sequence Error, Specific Command
11	Sequence Error	Specific Command, Command Type
12	Specific Command	Sequence Error
22	----------	Good Command
23	Good Command	Specific Command, Command Type
24	Erroneous Command	Format Error Command
25	----------	Erroneous Command
47	----------	Formatted Uplink Words
48	Formatted Uplink Words	Good Command
49	----------	Erroneous Command
50	----------	Uplink Command
51	Uplink Command	Formatted Uplink Words

Error," it is the only module within the scope of effect of GET VALCMD. The next decision process takes place in the GET GOODCMD module. Its scope of control includes GET FMTUPWRDS and PERFORM GENVALCKS. PERFORM GENVALCKS is in the scope of effect of GET GOODCMD. No other module is effected by the process. Control coupling exists between GET VALCMD and GET SEQERRCMD because a sequence error, if one exists, is passed through GET VALCMD to GET SEQERRCMD before the "Sequence Error Command" can be transmitted up to the executive. This coupling could reduce maintainability and should be examined before implementation to see if it can be eliminated. This entire afferent branch is the most complex of the whole simulator. However, the loose coupling between modules resulting from the design methodology yields a structure that is relatively easy to implement. The relationship between coupling and cohesion implies that the loose coupling produces high cohesion in each module which is a desired attribute. A brief explanation of the spacecraft command processing follows.

An "Uplink Command" is received by GET UPWRDS and the FORMAT UPWRDS module does necessary formatting of the uplink command words. GET GOODCMD accepts these "Formatted Uplink Words" and passes them to the PERFORM GENVALCKS module where they are checked for parity and wordlength. If either one of those errors exists the "Erroneous Command" is received by the GET FMTERRCMD module and passed to DETERMINE TYPECMD where the type of command is determined. The command with the format error is then passed to the executive for further processing. If, after the general validity checks are performed, no errors are found, the "Good Command" is passed to the control of the GET VALCMD module and held for sequence checking. The sequence checks are performed after the specific command type is determined. If a sequencing error occurs, control is passed to the GET SEQERRCMD module where the "Erroneous Command" is passed upward to the executive. Erroneous commands are used to generate the appropriate command verification message and output message which gives the user a description of the errors, but they do not affect the performance of any other modules.

MODIFY VEHSTAT. The scope-of-effect/scope-of-control is more obvious in this module than it was in the afferent branch. Figure 35 and Table III should be referred to while reading the explanation of the functions performed.

If READ CMDDATA receives a "Valid Command," the "Command Data" portion of the command is read and a "Status Modification Request" is generated. The "Status Modification Request" is then passed as control data to PERFORM MODIFICATION. The loop indicates the possibility of multiple telemetry changes and status message changes

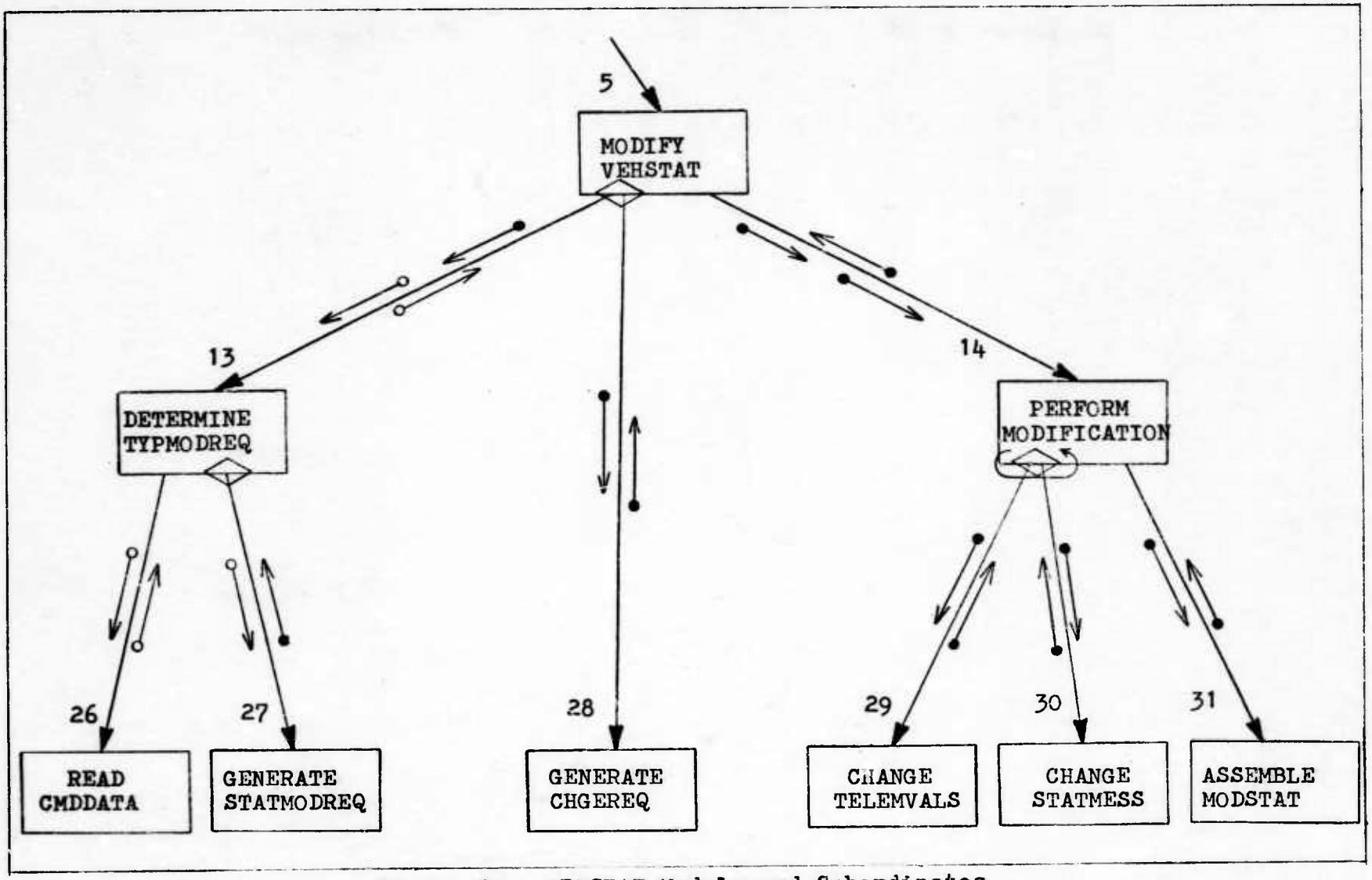


Figure 35. VEHSTAT Module and Subordinates

Table III

VEHSTAT Parameters

	Input	Output
13	Valid Command, Command Type	Status Modification Request
14	Status Modification Request, Change Request	Current Status
26	Valid Command	Command Data
27	Command Data	Status Modification Request
28	Status Modification	Change Request
29	Change Request	Modified Status
30	Status Modification Request	Modified Status
31	Modified Status	Current Status

required by one command. The status modifications generated by a valid spacecraft command are assembled and the "Current Status" is relayed to the PRODUCE OUTMESS module and a explanatory message is produced. A modification request could also result from a "User Command," in which case the "Status Modification" is used to generate a "Change Request." The portion of the module which consists of GENERATE CHGREQ can be omitted if an edit routine is used for "User Command" processing.

PROCESS CVWORDS. Command verification (CV) codes exist for most commands. The subordinates of PROCESS CVWORDS creates appropriate "CV Words" that are used to produce output messages (see Figure 36 and Table IV). The transform analysis has revealed a possible weakness in the design of this portion of the simulator. This weakness is indicated by the "pancake" structure that has occurred. "Pancaking" is a common phenomenon that results from a top-down design. It usually indicates a lack of sufficient decomposition. The FIND TYPSEQERR module should be decomposed further because it requires intermediate processing that is not shown.

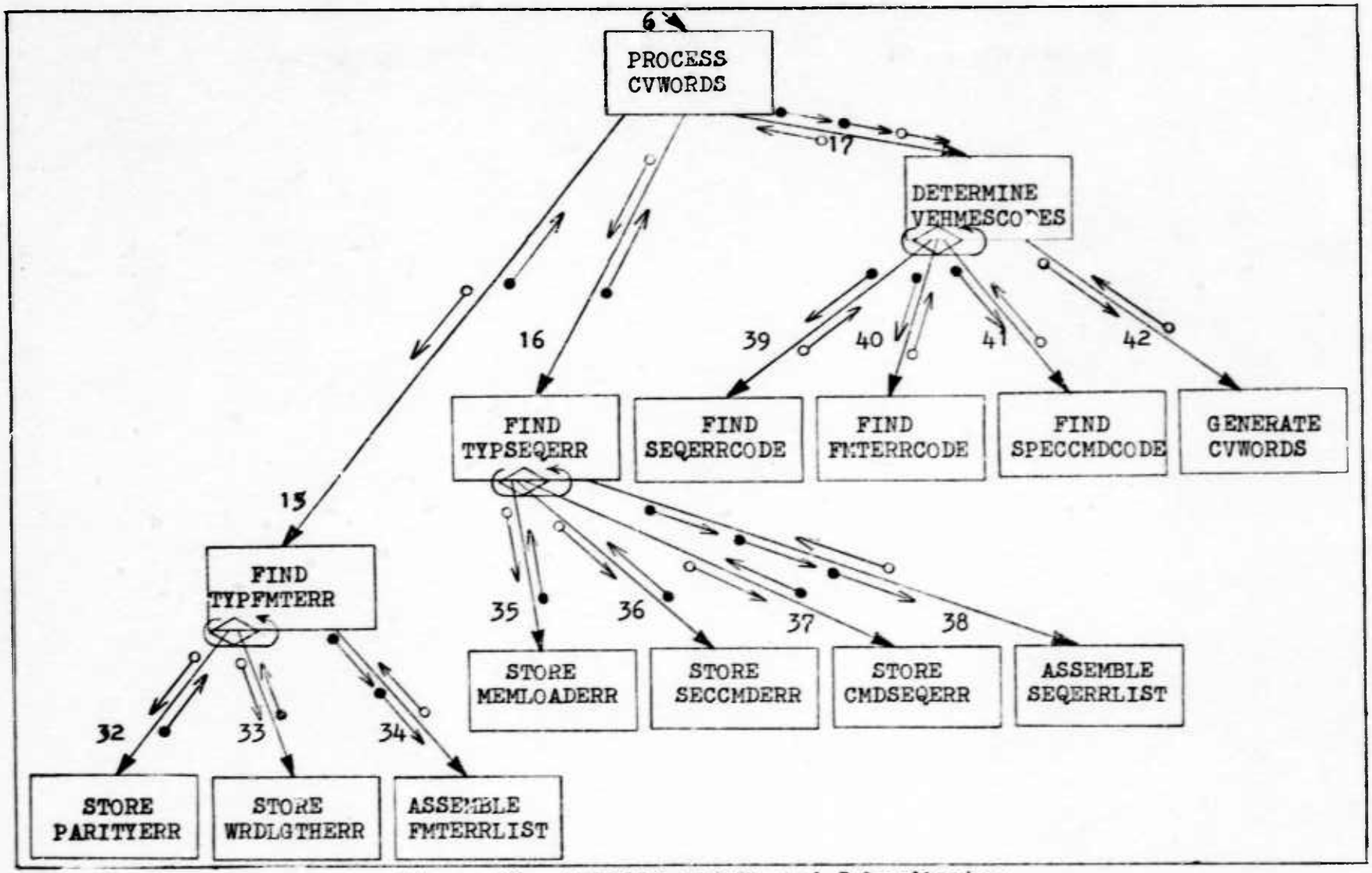


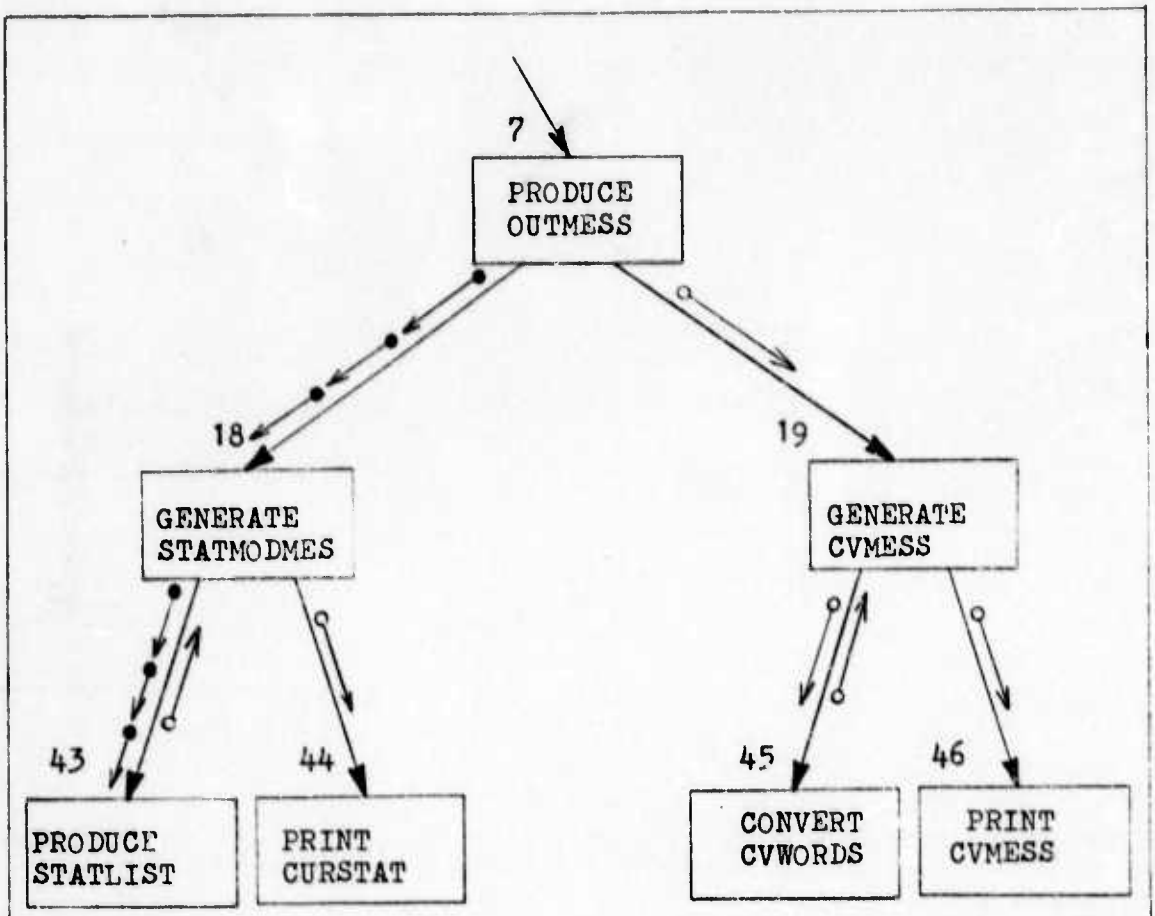
Figure 36. CVWORDS Module and Subordinates

Table IV

CVWORDS Parameters

	Input	Output
15	Format Error Command	Format Errors
16	Sequence Error Command	Sequence Errors
17	Valid Command, Format Errors, Sequence Errors	CV Words
32	Parity Error Command	Parity Error
33	Wordlength Error Command	Wordlength Error
34	Parity Error, Wordlength Error	Format Error List
35	Memory Load Words	Memory Load Error
36	Secure Command	Secure Command Error
37	Timed Command	Sequence Error
38	Memory Load Error, Secure Command Error, Sequence Error	Sequence Error List
39	Sequence Errors	Sequence Error Code
40	Format Errors	Format Error Code
41	Valid Command	Specific Command Code
42	CV Codes	CV Words

A command with format errors (parity, wordlength) is processed by FIND TYPFMTERR to determine the types of errors it contains. The iterative decision suggests a sequential check of both parity and wordlength. The "Format Error List" created is accessed by DETERMINE VEHMESCODES and the specific "Format Errors" are used to determine the proper CV codes needed to produce a CV message. The FIND TYPESQERR module is required to process a variety of errors that are classified as "Sequence Errors." These include (1) "Memory Load Errors" resulting from improper memory word lengths, (2) "Secure Command Errors" generated when a "Secure Command" is not preceded by the proper "prologue" command (Ref 1), and a (3) "Sequence Error" that results from improper timing between the prologue command and the executed spacecraft "Secure Command." These errors must all be recognized and assembled in the same manner as



Parameters

	Input	Output
18	Valid Command, Current Status, Status Modification	----------
19	CV Words	----------
43	Valid Command, Current Status, Status Modification	Status Listing
44	Modification Message	----------
45	CV Words	CV Message
46	CV Message	----------

Figure 37. OUTMESS Module and Subordinates

parity and wordlength errors. DETERMINE VEHMESCODES accepts the errors that have been found during the command processing and locates the proper command verification codes that adequately describe

the error.

PRODUCE OUTMESS. This is the efferent, or output branch of the simulator. It accumulates all errors and status modifications generated during command processing and provides a meaningful output message. The module and its associated parameter list are shown in Figure 37.

The "Valid Command" and "Current Status" information are used to produce a current "Status Listing" which is an explanatory statement of changes made and commands executed. "CV Words" are also listed with descriptions of their meaning.

Summary

Three previously tried design techniques are combined into one design methodology in this chapter. The activity model created during the preliminary design is converted directly to a first cut structure chart with no design modifications. An intermediate bubble chart is constructed from the first cut structure chart. During construction of the bubble chart a reevaluation of each module is made with emphasis on the conceptual representation intended by the bubble chart. After ensuring the bubble chart properly illustrates the desired system data flow, a new structure chart is created which represents a design refinement. This structure chart is then analyzed for "goodness" of design by looking for deficiencies in coupling, cohesion, scope-of-effect/scope-of-control.

V. Design Analysis

Introduction

This chapter is a summary of the information gained throughout this thesis project. The value of the design is discussed and suggestions to improv the design are given. Deficiencies noted during phases of the design are pointed out, and recommended remedies for those deficiencies are presented. Since the software design methodology employed in this project is relatively new and untried, this chapter also contains observations and recommendations for its continued usage.

Summary

The first problem addressed in this design project is the need for a sound analysis procedure that aids in defining requirements for a design. A study of SofTech's Structured Analysis and Design Technique shows that their methodology yields a top-down, modular, hierarchic, and structured design. These desirable design characteristics facilitate programming for implementation. The activity and data models are constructed with satisfactory results. Throughout the construction of the activity model an inability to define the input, output, or control is an indication of a lack of understanding of the particular activity. Consequently the system is studied further to a level of understanding that allows progress to continue. The same is true when constructing the data model. The data model is useful as a continuity check between blocks in the activity model. The two models are excellent tools that make

a complete, understandable requirements definition possible and provide a well defined preliminary design.

Structure charts are used as efficient means of refining the simulator design to prepare it for the programming phase of development. The structured analysis (activity and data models) leaves the question of "goodness" of design unanswered. The question is answered by the construction of structure charts. The problem is how to make a smooth transition from the SADT methodology to the Transform Analysis used to derive the structure charts. The activity model is redrawn into a structure chart. This transition yields a structure chart that better identifies the essential elements in the system. Next, an "intermediate" bubble chart is drawn based on a structured design method by Hughes Aircraft (Ref 3). The re-evaluation of intermodular connections at each stage of bubble construction results in the location and correction of poorly defined connections. Now Transform Analysis (Ref 10: Chap. 10) can be used to derive a more refined structure chart. Upon completion of the refined structure chart, a thorough examination of each module results in identification of some areas that need further analysis.

The combination of the three design methodolgies, structured analysis, structured design, and Hughes' iteration of structured design results in a simulator design that meets or exceeds the original design goals. The system is modular and understandable. The loose coupling between modules enhances the modifiability and maintainability of the simulator. The well defined scope-of-effect/scope-of-control within a branch of the system maintains the functional independence desired, even in the most complex branches.

After incorporating the recommendations given below, the structured design can be given to a programming section with full confidence that the system requirements will be met.

Conclusions and Recommendations

In Chapter IV, the first cut structure chart would be easier to understand if the control elements were included in the parameter list. It is recommended that all inputs, outputs, and controls be included in the first cut structure chart as a central reference for bubble chart construction.

The TYPSEQERR module shown in Figure 36 on page 90 needs further decomposition to better define its function. This module can be examined independently and modified as required without affecting the other subordinates of PROCESS CVWORDS. That examination should be made before coding is attempted.

To be consistent with a software life-cycle, each module in the given structure charts should be flow-charted for coding. Before this is attempted, however, the structure charts should be studied by the coders to clarify any vague portions of the structure. This clarification should be made by reading the Design Refinement in Chapter IV. If further clarification is needed, another iteration from a new bubble chart to another structure chart should be performed.

This thesis addresses the simulation of the command section of the Block 5D satellite with successful results. It is recommended that other spacecraft functions be analyzed using the methodology presented in this design. Each main function can be designed independently and added to this design at a later time.

Bibliography

1. Annex L. Systems Concepts and Procedures. Offutt Air Force Base, Nebraska: Headquarters 4000 Aerospace Applications Group (SAC), March 1976.

2. Boehm, B. W. Software Engineering. Redondo Beach, California: TRW, 1976.

3. Jensen, E. "Structured Design." 1975 IR&D Structured Design Methodology, 2: Hughes Aircraft Company, April 1976.

4. Ross, D. T., and K. E. Schoman, Jr. Structured Analysis for Requirements Definition. Waltham, Massachusetts: SofTech, Inc., April 1976.

5. Ross, D. T., et al. "Software Engineering: Process, Principles, and Goals." Computer: 89-99 (1975)

6. Ross, D. T. Quality Starts With Requirements Definition. Waltham, Massachusetts: SofTech, Inc., February 1977.

7. SofTech, Inc. Structured Analysis Reader Guide. Waltham, Massachusetts: May 1975.

8. SofTech, Inc. An Introduction to Structured Analysis and Design Technique. Waltham, Massachusetts: November 1976.

9. Stevens, W. P., et al. "Structured Design." IBM Systems Journal, 2: 115-139 (1974).

10. Yourdon, E., and L. L. Constantine. Structured Design. New York: Yourdon, Inc., February 1976.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: GE/EE/77S-6	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): STRUCTURED ANALYSIS AND DESIGN OF A SATELLITE SIMULATOR	5. TYPE OF REPORT & PERIOD COVERED: MS Thesis 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Kenneth L. Marvin, Captain USAF	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Air Force Institute of Technology (AFIT-EN), Wright-Patterson AFB, Ohio 45433	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: 4000 Aerospace Applications Group (SAC), Offutt AFB, Nebraska 68113	12. REPORT DATE: September, 1977 13. NUMBER OF PAGES: 108
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Approved for public release; IAW AFR 190-17

JERRAL F. GUESS, Captain, USAF
Director of Information

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Structured Analysis
Structured Design
Bubble Chart
Structure Chart

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The problem addressed in this thesis is the analysis of a complex satellite command system and the design of a software simulation of that system. The problem is solved in three steps. First, a written requirements definition establishes a sound viewpoint and purpose on which the analyst can base his design. This requirements definition explains why the simulator is to be created, how it is to be constructed, and what it is to do. Second, a top-down strategy called "structured analysis" is applied to create the preliminary design. The structured analysis is

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

presented in a blueprint-type language with activity and data models. The models represent graphically the functions performed by the simulator and the information upon which those functions act. A final design refinement is performed with a structured design methodology called "transform analysis." The structure charts drawn during the transform analysis reveal system characteristics which illustrate design quality. The activity model acts as a catalyst to a successful transition from a top-down analysis to a structured design which can be evaluated. The resulting simulator design, with minor revisions, satisfies the design goals established for the project. The methodology used is highly recommended for the analysis and design of any software system.