Automated Distributed Computer Management

Presented at the 1986 Fall DECUS Symposium
San Francisco, CA

October, 1986

Larry L. Johnson

CONTENTS

ABSTRACT

A data flow model of a distributed, loosely-coupled, time-sharing computer network is discussed. This model has been used to initiate an evolving implementation of an automated software distribution/installation and configuration management system capable of handling software distribution and management for a system involving many machines. The model defines a generic software product, independent of its function, in the context of a distributed computer services shop.

1 Introduction

The following is a discussion of the top-levels of architecture used in our effort to automate distributed computer services management. The discussion is from the point of view of a systems analyst and is consequently centered around the "whats" rather than the "hows ". Also, architectural aspects are discussed with no regard to their state of implementation. The discussion wanders in and out of those things which have been accomplished and those which may be accomplished someday.

When using the expression "system configuration", we speak in the context of nodes in a distributed system and their attributes of software, hardware, and personnel. We are not generally speaking of classic project system configuration management or control.

2 Statement of Problem

A computer services shop which maintains a large number of time-sharing computer nodes on behalf of its customers is presented with problems in efficiently managing the configuration and day-to-day operation of those nodes. As the number of nodes and their geographical dispersion increases, the head count and physical dispersion of the maintaining staff must also increase. Without automated assistance the maintaining staff becomes overwhelmed with problems in keeping large amounts of data organized, in keeping up intra-staff and user communication, in keeping the system up-to-date, and in keeping management apprised of the system's state.

The installation of software on a node amounts to high-tech clerical work, The process is simple in concept, complex in detail, and time consuming. The resulting boredom makes the installation operator error-prone, resulting in error situations which are often time-consuming in recovery. The installation chore is characterized by long periods of boredom, punctuated by moments of sheer panic.

Without automated support, system configuration management is costly and approaches impracticality as the number of nodes increases. This presents legal hazards by making it more likely that software might propagate to a node which does not have a license. It creates a nightmare for those responsible for software updates. Additionally, management is constrained to manage the unknown. It is difficult to assess and control the efficacy of computer services without knowing the state of the collective system.

Another problem which arises in a large distributed computing environment is the creation of unnecessarily diverse environments, Each node creates its own system structure which is often poorly documented. System engineers maintaining the system must re-orient themselves for every system on which they work. This diversity also impairs the migration capabilities of user groups. Users require retraining simply because they have moved their accounts to other machines.

3 System Goals

In order to address the problem an automated distributed computer management (ADCM) system was proposed with the following major goals:

- Accommodate system management.

  System configuration management was to be part of the architecture of any software distribution system conceived.

- Automated distribution.

  Menu selectable distribution. When the distribution is entered, it proceeds transparently, shipping the software over network facilities where available, and coordinating the generation and traffic of physical distribution media.

- Automated installation.

Pre-programmed installation will be supplied whenever feasible, allowing a programmer to "install" it once for all systems.

- System manager control.

  Despite the level of automation achieved for a given product, system manager approval will be required before it is a candidate for installation by an operator or by automated facilities.

- Uniform node architecture.

  There will be a core of software common to all machines to provide a common base of familiarity for operation and use of any machine in the system.

- Tailorable node architecture.

  The core software will offer hooks and extensions by which node tailoring is made simple. Functionality provided by the core software will be selectable for special purpose machines. Software will be organized in functional packages which closely cooperate with other packages, or alternatively, operate without them using documented hooks to provide surrogate processing.

- Logical isolation of physical node structure.

  The physical system structure will be transparent to normal operational and applicational use, allowing rearrangement of physical structure for tuning or architectural enhancement.

- Incremental Release

  Due to the size of the undertaking, it was imperative that the system be incrementally releaseable so that portions of the system could be used while the rest was developed. The architecture would be such that previously released portions would require minimal rework to accommodate new functionality.

- Broad Applicability

  Support was needed for machines applied to extremely diverse applications, e.g., software development, modeling, CAD/CAM, signal processing, etc. The system must be layered to allow matching the supplied software and features to almost any node's mission. There will be a base kernel which is highly general and makes no assumption of the use of the machine. Additional products will be supplied within the system to assist particular common applications, e.g. time-share computing services as opposed to dedicated machine control.

4 Development Strategy

It was important that the analysis of the problem proceed by a broad-scoped top-down analysis of the function of managing a distributed computer services shop. Yourdon style data flow diagramming was selected as the primary vehicle of analysis.

The broad-scoped analysis provided a framework in which problem areas could be identified and prioritized. Detailed analysis proceeded in critical areas providing a detailed list of problems. The problems were prioritized in light of their scopes of effect and estimated cost of solution. Solutions were then developed in order of the priority of the problems.

It is important to note that an attempt was made to avoid any pre-disposition as to automating or not automating functions. The implementation of a solution could involve any combination of automated and manual procedures. The human element was treated as part of the system in both contexts of system support and system use. Manual procedures would be designed to dove-tail with automated procedures and vice-versa.

Early development concentrated on quick solutions to severe problem areas. This enabled the installation of a broad base of "adequate" prototype software to do the *job* while the "final" solution was under development. Re-prioritization was made for the upgrade of these early subsystems. Incremental refinement of subsystems then proceeded under the same prioritized task management under which the system had been conceived. The incremental refinements were possible and manageable due to the top-down analysis which had been made. A system architecture was defined by the specification, allowing subsystems to be easily up-graded with minimal perturbation to other subsystems.

The system was designed to accommodate evolution. No particular subsystem was to be considered final. It was perceived that our customers' requirements and upper-management's requirements changed naturally in the course of normal business. It was therefore important that the system be designed to evolve to meet the changing needs. It was imperative that evolution (change) of the system be considered as normal and not handled as an alternate development mode.

5 Data Flow Model of Distributed Computing Services

The development framework was a data-flow model of a distributed computing services shop. The top-levels of the model are presented in an appendix and are the focal point of the remaining discussions. It is important to note that the diagrams have been edited to simplify them for the purposes of illustration. This has resulted in flow imbalances in several places. A data dictionary of flows and process descriptions (mini-specs) has not

been offered. The names of the flows at this top-level provide sufficient definition of the items for the purposes of this discussion. Most of the text of this article is a filtered, narrative form of the mini-specs.

The entire model involves well over one-hundred mini-specs and over eight hundred data dictionary entries. The diagram depth averages around six to seven levels, going as low as twelve.

A brief tutorial on reading data flow diagrams is presented in an appendix.

The first level of the diagram (under the context diagram) defines three major subsystems.

1.0 -- Produce Integrated Software

This subsystem is responsible for providing software to the distribution subsystem in a form it can use. This involves re-packaging externally acquired software as well as developing new software.

2.0 -- Manage Distributed System

This subsystem is responsible for the distribution and coordination of resources among all nodes and the coordination of those resources. "Resources" includes software, hardware and human resources. It is this subsystem which maintains the central database which details the state of the entire distributed system, managing distributed audits to keep itself up to date.

3.0 -- Provide Node Services

This subsystem is responsible for providing node services which may be those of a hardware inventory cage, or those of a full-service time-share computer node. Although many other classes of services could be defined, these were the two which were considered. The hardware inventory cage was treated exactly as a computing node, except it had no users and could not compute.

Note that software is the flow which hops across the subsystems. In order to define the system it was essential to construct a model of a software product (which has come to be called an "Integrated Product") which lends itself to application in all three major subsystems.

6 Generic Software Model -- "Integrated Products"

In order to accommodate the requirements of software production, distribution, and use, a model of generic software was constructed.

The model defines a product to the system, regardless of its function. The generic product was partitioned into operational contexts which were defined against system events. Such partitioning amounts to a pre-installation or "pre-integration" of the software. Consequently, software packaged in compliance with the model came to be known as "Integrated Products", or "IP's".

The following is a breakdown of the operation contexts:

- System Functions -- those functions which have system-wide effects or availability.
  - Startup -- those functions provided at each system boot.
  - Shutdown -- functions provided at each system shutdown; typically the converse operation of the startup function.
  - Monitor -- those functions which are provided by the product to be executed by the system on the products behalf, and which may have system-wide effects.
  - Alarm -- definitions of alarms, alarm recipients, and alarm conditions. The primary vehicle of the machines request for human action. Alarm conditions are generally tripped by sister functionality supplied as a monitor component.
  - Eventlog -- definitions of entries for error logging and audit trails.

- Process Functions
  - Login -- non-optional functions provided at process creation.
  - Logout -- non-optional functions provided at process deletion; typically the converse of operations provided at login.
  - Setup -- user initiated functions which establishes the users access to a product resolving any logical/physical associations.

- Rundown -- the converse of setup functions; generally run at process deletion before logout components.

0 Product Functions

- Install -- functions beyond the system default required for the successful installation of a product.

- Remove -- functions *which* undo the operations done by an install component prior to the default removal operations.

- Document -- descriptions of product documentation, on-line and off-line. Used primarily with a product documentation catalog utility.

- Test -- procedures to provide minimal exercise of a product sufficient to be confident of proper installation of the software. These tests are not the exhaustive test performed by developers to assure correct operation.

0 User Definition Functions -- Functions which will be called under a pre-defined protocol on the addition, modification or removal of a user from the system.

- Add User

- Modify User

- Delete User

Local Data -- repositories of data generated or accumulated by a product which must survive product update.

Software to operate in these contexts is provided as components of the product. Typically a component is an image or a command file, e.g. startup or login functions. Additional conventions exist for some components such as:

+ Menu selection titles (User Definition)

+ Standard product description information (Document)

+ Execution specifications (Monitor)

Each integrated product (IP) is supplied with source and a prepare stream. The distribution manager can build a new product release for upwardly compatible operating system revisions without returning to the developer. This is true for all software dependent on some other piece of software which has been marked as an upwardly compatible dependency. Of course this upward compatibility is an assumption. The prepare stream often will contain a coarse thread test for the product. Passing this test, the new release is a candidate for beta distribution.

The option exists to ship images and to use the prepare stream simply for packaging, a major function of the prepare stream.

The prepare stream packages the product according to context in a pre-defined way which permits node resident software to locate the software components of each context. Currently these components are located through a pre-defined directory structure. A second packaging technique is under development which will describe a product using an Integrated Product Description Language (IPDL), making no assumptions of directory structure beyond the location of the root where the description is contained. This second version will facilitate the conversion of externally acquired software, requiring only that a description be written, minimizing physical repackaging.

Each IP has associated with it a distribution class, which operates as a "red-tape filter". The distribution class is assigned by management, specifying the amount of justification required for a piece of software from "for the asking" to full cost-benefit analysis. Obviously the price of the software has a great deal to do with the class assignment.

Completing the Loop of Human Interface

In the previous discussion on operational context partitioning, most of the contexts are intuitively recognizable as being characteristic of a generic software product. However, the alarm component, and to a lesser extent, the monitor component deserve additional discussion.

When a person wants a computer to do something, he simply logs in and makes his request of the computer. However, when software detects that something requires human intervention there is no robust way to make a request of a person, and if necessary, nag him. Alarm components provide the ability to do this.

Alarm components are supplied as descriptions of alarms and their meaning. Any alarm so defined can be "tripped" by software which calls routines supplied by the alarm subsystem. When an alarm is tripped, it becomes an alarm condition which remains in effect until explicitly dismissed. Part of the alarm description specifies escalation levels. Each level describes the age or

multiple trip count for a condition required to escalate it to the next level.

The operator uses a utility to enter subscriptions of users to alarms at particular escalation levels. The alarm system issues reports of outstanding alarm conditions based on the terms of the subscription.

Therefore, one can arrange to have an operator receive "backup overdue" messages. If ignored for a pre-set period of time the alarm may escalate to a secondary operator. A further escalation may cause the alarm to escalate to the operator manager... and so forth.

Monitor components are operated by a driver which is much like a batch job controller. The controller is supplied with more flexible timing and periodicity options, but it accepts jobs only in the form of monitor components. Frequently the use of a monitor component is to periodically run schedule and sanity checks, issuing alarms for anything that requires human action or inspection.

Using the combination of monitor and alarm components, all products have available to them a means by which they can request action from people, and if necessary escalate the request to backup people. This capability makes the automation of coordination of efforts requiring manual operations straight-forward.

8 Software Production Subsystem

The software production subsystem (1.0, PRODUCE INTEGRATED_ PRODUCTS) defines the approach toward the development of the system itself, as well as the development of integrated products (IP's) and the conversion of external products to IP format.

The development team is itself operated much like a time-share system in which:

- the CPU is the development team (PERFORM EFFORT CYCLES).
- the job-controller is the team's management (MANAGE IP PRODUCTION).
- the ACTIVE EFFORT CYCLES file is the list of inswapped tasks.

- the CANDIDATE CYCLEDEFNS file is the list of outswapped tasks.

The management of development (1.1, MANAGE IP PRODUCTION), is responsible for maintaining a list of CANDIDATE EFFORT_ CYCLES comprised of edited and reviewed DEVELOPMENT REQUEST's originating from inside or outside the development team. DEVELOPMENT_PRIORITIES are gathered from upper-management. The priorities are used to order CANDIDATE_EFFORT_CYCLES, selecting some for migration to ACTIVE_EFFORT_CYCLES when AVAILABLE_ RESOURCE_ DATA indicates appropriate resources are available.

An EFFORT_CYCLE can consist of work defined among the following categories in any proportion:

- Ramp-up -- Project staff is brought up to date on work previously done and the requirements which need to be satisfied on this cycle of effort.

- Analysis -- Consists of deriving data flow diagrams as a functional specification. Often database requirements, time limitations, and external interfaces are defined. Focus is kept on what is to be done as opposed to "how". Rough resource and delivery estimates are made,

- Design -- Consists of refinement of analysis to the definition of modules and the actual engineering of the system and its interfaces. Early design cycles are made to aid the estimations being made by the analyst.

- Implementation -- A broad category of those things involved in taking a designed system and creating an incarnation of it. Includes:
  - Coding
  - Testing
  - Integration
  - Packaging

- Wrap-up -- Whether the cycle reaches the end of its definition, or is terminated prematurely by management, a minimum period of time is allocated for wrapping-up the effort. Completed work is inventoried as well as work in progress. Work which is too young is discarded and remarked as unstarted. The procedure for reactivating the cycle is documented. The project is shelved in the CANDIDATE CYCLE DEFNS to be activated at a future time as changes in priorities and available resources dictate. When a cycle is completed naturally, part of the wrap-up effort includes analysis for the definition of the next suggested cycle of effort, or modification of that cycle's definition if it already exists. At the end of each cycle, estimates

are refined. As each cycle completes, a more accurate picture emerges of the amount of work required to reach a development milestone.

Generally subsystems or products begin in the definition of a cycle involving primarily analysis and a little design work for feasibility testing. This effort generally has a pre-defined man-power expenditure of two to four man-weeks. Whatever the result at that time, the cycle is completed and recommendations made to management. If management is still interested in the project based on the initial estimates, another cycle is initiated involving detailed design of critical subsystems. The goal of this effort is primarily the refinement of initial estimates. Analysis continues concurrently to attempt to define partial delivery sub-packages so that portions of the functionality can precede the entire deliverable, allowing experience in using a product to help direct the development effort.

Subsequent cycles are defined and committed to execution until the priorities change and the project is outswapped, or the effort reaches fruition.

Management can therefore change the direction of the development team in a smooth pre-defined fashion. However, care must be exercised to avoid "thrashing" the development team into rapid wrap-up/ramp-up sequences.

This technique is referred to as an "Evolutionary Life Cycle" approach (ELC). It is derived from a wide number of references on project management (too numerous to cite), and from the development staff's collective experience. It produces a little anxiety in an implementation team the first few times a cycle is wrapped-up before completion. However, confidence is restored the first time one of those cycles is re-activated and completed.

The ELC approach also affords the opportunity to weed out pet projects which "seemed like a good idea at the time". By committing resources incrementally and seeking refined estimates as the primary product of early cycles, one detects projects which are bigger than they had seemed very early. One can then determine that the results are still worth the cost and continue, or that they are not and shelve (or kill) the project. In either case, management is apprised of any changes in the perceived size of an effort before it is "too late".

9 System Management Subsystem

9.1 Configuration Database

At the heart of the configuration management subsystem (2.0 MANAGE DISTRIBUTED SYSTEM) is the SYSTEM CONFIG database.

The subsystem (2.1, MANAGE _ RESOURCE_ POOL) maintains entries for software, hardware, node, and personnel resources. New resources are acquired and old resources disposed in keeping with SYSTEM REQUIREMENTS.

Software resources are presumed to be in integrated product (IP) form. Hardware resources are tracked from submittal to the corporate purchasing department, through shipping, at receiving docks, destination node and through check-out. In this context, vendors and docks are treated as temporary "hardware bins", and therefore as nodes in their own right.

Software licenses are carried as resources in the pool, and are mapped to nodes in the same fashion as software.

The resources of personnel are mapped to nodes and tasks based on a percentage man-power basis. A minimum time for any task is provided to avoid fragmentation of assignments.

NODE_CONFIG ENTRY's are managed in (2.2, MANAGE_SYSTEM SYSTEM_CONFIG). This system records and tracks movement of resources among nodes. RESOURCE_ UTILIZATION_ REPORTS are used to move under-used resources to systems which require them. This subsystem maps software, hardware, personnel, licenses, etc. to specific nodes.

For maximum flexibility and speed of implementation, a relational database was selected for the implementation of the central system configuration database. DEC's RDB product was used. The database was organized under principles of entity formalism suggested in the outline of Codd's RM/T model as presented in C. J. Date's book, "An Introduction to Database Systems, Volume II", Addison-Wesley, 1983.

9.2 Software Distribution

The subsystem (2.2 MANAGE SYSTEM_CONFIG) receives integrated product releases from (2.1 MANAGE_RESOURCE_POOL). There is some controversy over having an apparent software production function under a resource management subsystem whose primary responsibility is to record the entrance or exit of a resource from the system. However, placement of the function here enables regeneration of software for new revisions of the operating system without the overhead of going back to the development organization. This provision is made for software which is expected to require only recompilation or relinking for upward compatibility.

A configuration operator enters the intended distribution via a menu interface. The distribution is entered in the configuration database as the first act of distribution, meeting the design goal of architecturally embedded configuration management. The intended distribution is checked for compatibility against products existing at the node and other products targeted for the node. If other products are required for proper operation the configuration operator is presented with the option to include them in the distribution. If the distribution would cause an incompatible profile of products the distribution is flagged incompatible until a subsequent edit of the distribution brings it into compatibility compliance.

In addition to inter-software dependencies, hardware dependencies are similarly checked.

Software licensing for the node is checked. If no licensing is required for the software being distributed, it is marked as a candidate for distribution immediately. Otherwise it placed in a licensing wait mode, awaiting confirmation of license receipt. At confirmation the software is then marked as distributable. (Actually, distributability is indicated by a number of state flags in the database involving compatibility checks, licensing checks, managerial approval, configuration control board approval, etc.)

A licensing mode has been provided for those vendors with whom a great deal of business is done. The software is marked as distributable, and a confirmation to the licensing vendor is automatically entered and sent periodically. On receipt the vendor adds the license to its database, effective the date of distribution, and bills the company. This minimizes administrative delays in the distribution of commonly required software.

Assuming that all the state flags of the software-to-node configuration entry indicate deliverability, the Automated Distribution System (ADS) takes over.

Distributions which are approved and have network lines available will transmit the distribution electronically. The transmission can take place at a pre-determined time or at a time when the ADS detects low processor/network demand (of course this later mode requires a deadline to avoid the transmission dying on a busy node). An automated acknowledgment system implemented in the receiving half of the ADS confirms receipt.

Those distributions for which there is no network accessibility are queued. When a distribution operator lets the system know that he is ready to make distribution media, the ADS notifies the operator to mount the appropriate media, generates labels and attendant manuals, and dequeues and transfers the software distribution to the media. Compatibility checks pre-determine the bundling of software going to the same node to ensure that

inter-dependent software products are shipped together.

Regardless of the mode of transmission (network or physical media), the system maintains confirmation flags for reception of distribution, acceptance/rejection of distribution, and installation of distribution. Those nodes which have network support report these confirmations automatically. The confirmation for the "footnet" physical media are entered manually by the configuration operator.

9.3 Hardware Distribution

At the time request for purchase is sent to the corporate purchasing department, the ordered hardware is entered as a resource mapped to a vendor node. Expectation dates are kept, as well as the node of first receipt inside the company. This enables aged lists of overdue hardware to be compiled for follow-up.

Status flags are maintained to track the shipment of the order. The hardware is then mapped to its intended node of use under a unique configuration number. Using a unique configuration number for each node/item/duration mapping enables an audit history to be maintained on the travel of a particular hardware item.

Most of the actual purchasing tracing is done by the corporate purchasing department operating its own inventory system. The SYSTEM CONFIG database tracks only milestones and overall status to detect bottlenecks and to provide status information in resource summary reports covering the entire system.

9.4 Personnel Distribution

Personnel records are kept to define the role of an individual in the distributed system. Personnel are divided into support and user categories. Within support, personnel are recorded with a capabilities matrix indicating their ability to serve as operators, product consultants, system managers, system engineers, etc.

Personnel are assigned to nodes and functions by percent of time. These assignments are under control of their supervisors and coordinated by the database. The database performs periodic integrity scans to detect lapses in coverage due to vacations, illness, or termination; over-booking of an individuals time; responsibilities which have no person currently assigned to them; etc.

Central records are also kept of the existence of a user account on any node. On termination or reassignment of the employee, his accounts can be marked for reassignment or deletion, and immediately disabled for access. This is an important security consideration in a large, dispersed system in which an employee can have a half dozen accounts on machines separated by hundreds of miles.

9.5 System Configuration Audits

The configuration audit subsystem (2.3 AUDIT_SYSTEM CONFIG) coordinates audit schedules, receives manual and automated node audit reports and compares the audit against the appropriate SYSTEM CONFIG ENTRY.

The audit system is used as a state detector. When there is disagreement between the audit and the configuration entry, the configuration entry is marked with the audit identification and a variance flag. The system compiles reports of the conflicts which are resolved by an audit operator. The operator must determine if the audit or the configuration entry is in error, and make a traceable correction to the configuration entry through audit support software.

The audit system also generates late notices for audit reports which have not been received on schedule.

10 Node Management Subsystem

10.1 Software Installation

The installation of software begins with the reception of the software by the node portion of the Automated Distribution System (ADS). The system operates as a network task for electronic distribution and interactively for reception of physical media.

The ADS places the distribution in a receiving area, and updates the node's database to reflect receipt of the distribution and a table of its contents. Each distribution entry in the node database has a system manager approval flag. The distribution is not installable until that has been set.

The system manager runs privileged software to review distributions and to accept or reject them. The primary purpose of this pause is to keep the system manager apprised of the configuration of his machine, and to provide a last chance to catch software sent to the wrong node. Normally the system manager approves the distribution for installation and sets a time for installation or a deadline by which the installation is to be done. The installation can be auto-scheduled and/or

auto-initiated if elected by the system manager. Should the system manager reject the distribution, he enters his reason during the transaction. The reason for rejection is sent back to central distribution electronically or by inter-office mail, depending on the node's available resources.

An installation subsystem is invoked interactively by the operator, or automatically under a schedule selected by the system manager or operator. Only software which has been marked in the database as approved for installation is candidate for the installation subsystem.

The installation subsystem unpacks the distribution kit, placing the product in a product/revision root under an integrated product root (directory structure). Software which has been originally packaged as an integrated product commonly requires no further action.

Those products which require installation activity beyond the default supply an Install Component (image or command procedure) which is activated after the default installation operations.

Interdependent integrated products are packaged in kits. All products in a kit are installed during the same install operation and in a prescribed order determined when the kit is generated. If the installation of any product in a kit fails, the entire kit is forced to fail so that a partial installation does not result.

After installation, the Test Component is operated to ensure that the product is properly installed. If the test succeeds then the product is marked in the database as available. Otherwise the installation operator receives an error message and the product remains marked as unavailable.

10.2 Integrated Product Operation (COREDRIVE)

Each node within the system has two base products:

1. ADSNODE -- The receiving half of the Automated Distribution System

2. COREDRIVE -- Coordinates integrated product operation.

The COREDRIVE product understands the IP organization. From the description of the products' components, scripts are compiled for each class of function, e.g., startup, login, etc. Each IP component is inserted into its appropriate script at installation.

Each system event (as described in the discussion on operational context partitioning of products) has associated with it a functional driver(s). Drivers are triggered by the operating system, the operator or the user as appropriate for the component. For example, the startup driver is triggered by the boot of the operating system, the login driver is triggered by the creation of a process, and the setup driver is invoked by the user.

COREDRIVE activates each component according to the script automatically.

Many of the drivers are utilities which provide a standard interface to the components.

- IP Receiver (OPER, SYSMGR) -- Used to pick up the IP after it has been placed in the receiving root by the ADS. A portion of functionality restricted to the system manager deals with installation approval.

- **INSTALL (OPER) --** Installs kits of products previously approved by the system manager. The operator selects the kits to be installed and optionally determines schedules for auto-installation. The operation of this utility has been discussed at length in a previous section.

- User Definition (OPER) -- Menu driven utility to define users. User definition components are mapped at run time and supply menu selection line titles for use in optional activation.

- IP Control (OPER) -- Allows the operator to turn on/off specific product availability, component operation, and other properties common to all products.

- External Product (OPER) -- Currently implemented as an external component utility. This enables a local node to describe locally generated or acquired code to COREDRIVE for execution.

- Document Catalog (USER) -- A utility to describe the documentation as presented in documentation components. Documentation on and off-line is described. Facilities are provided to copy on-line documentation to printer or user directory, and to order documentation which is off-line.

- Software Catalog (USER) -- Describes the software currently on the system. Documentation components supply product name, short description, abstract description, etc. The level of description desired is specified by the requester.

- Hardware Catalog (USER) -- Describes the hardware configuration of the node.

- Product Setup (USER) -- Establishes a logical link between the user and the physical location of the product. It is this utility which provides the isolation of physical product structure. This utility runs the setup component for the named products.

10.3 Time Share Node Management Products

The context partitioning of products, automated distribution, reception and installation are all more general than the context of a time share computer service node.

The data-flow model specifically models a time-share node operation. The products which emerge from the model have been called Ancillary Core Products (ACP's, not to be confused with ancillary control processes).

Most of the ACP's are strictly applicable to time-share nodes whereas others are much more general. These products are considered advised to our nodes, but are optional. When an ACP requires cooperation from another which is not available on the node, the node's technical staff must see to it that surrogate functionality is provided.

The following is a list of some of the ACP's:

- HRDWRMGR -- Loads device drivers, maintains hardware database, provides hardware status callbacks for use by other ACP's which need to check the availability of hardware.

- VOLMGR -- Coordinates disk mounts, disk archive, tape archive, backup, and volume integrity.

- USERCOMM -- Coordinates communications with users. Provides login banners, news utility, broadcast utilities, etc.

- USERSCRN -- Screen users against system mode. Uses authorize and extended UAF facilities.

- VASP -- Accounting utility, Resource usage by account.

- PRINTQMGR -- Menu driven print queue handling. Automated modes of operation for common functions. Call backs for products requiring queue management services.

- BATCHQMGR -- Similar to PRINTQMGR

- PMIS -- Performance monitoring, load monitoring.

Where applicable the above products maintain communication with counter-part functionality on the central node to maintain the central database and provide auto-auditing capabilities. Where networking is not available, the software contacts system operators via the alarm subsystem to see to it that required manual operations are performed.

APPENDIX A

INTRODUCTION TO READING DATA FLOW DIAGRAMS

Data flow diagrams are used to describe the functions of a system in terms of processes and data flow. Flow diagrams are essentially inventories of functionality without regard to sequencing and similar design related issues. All functions are asynchronous from the point of view of a diagram. A function is "triggered" by the presence of data on which to operate. This orientation is not necessarily carried forward into design, unless it is natural to do so.

Circles represent processing and are labeled with the name of the process. The name is selected to give an extremely brief description of the process.

Processes are connected by labeled arrows or vectors which represent data flow. Data flows are aggregates of data named to suggest their content. The contents are generally detailed in a data dictionary; however, in this article no data dictionary has been supplied since we are not addressing details.

Data stores are indicated by labeled straight lines which have ingressing/egressing data flows from one or more processes. The word "stores" is used in lieu of the frequently used term, "files". This is to emphasize that data stores represent storage which could be temporary or permanent, resident in memory, on disk, or on a piece of paper in a file cabinet. The only requirement on a data store is that it be in existence long enough for cooperating processes to use them meaningfully.

Data flow diagrams are "leveled". To describe the functionality of a process, one goes to a decomposition of that process which is another data-flow diagram. Data flows ingressing and egressing the "edge" of the paper represent flows in and out of the parent process.

When a processes function can be explained in a page of text, it is not further decomposed as a diagram, but textually described in what is called a "mini-spec", or "primitive process". Mini-specs have not been supplied here because only the top-levels are being examined and the mini-specs occur considerably lower in the leveled diagram.

Data flow diagrams are not generally read from start to finish. One generally examines the top-levels to get a feel for the overall architecture of the system. When an area of particular interest is encountered, one often pursues the reading down through mini-spec descriptions. Alternately, some peruse the mini-specs and then supply themselves with the context from the diagrams afterward. Data flow diagrams are more comfortably read in "random access" than in "serial access".

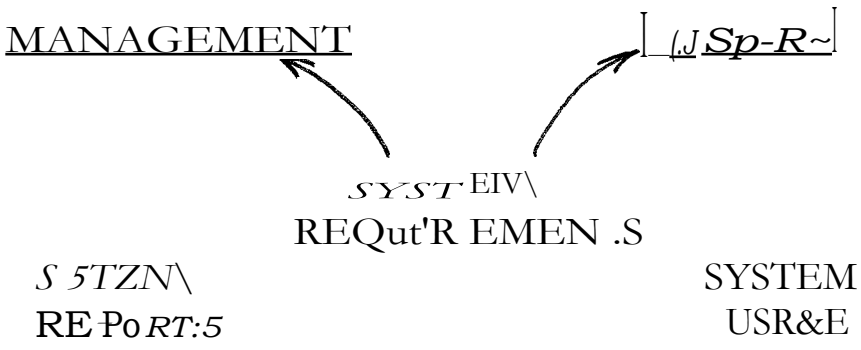
Clearly the diagrams presented in this paper are not intended to convey the full specification, but to give some flavor of our approach and its results. Edited diagrams have been included for illustrative purposes only.

APPENDIX B

DATA FLOW DIAGRAMS

✓.UrO/V1 A-rE D ttSTRt BUT-ED
COMPUTER„ SER.')C-E5

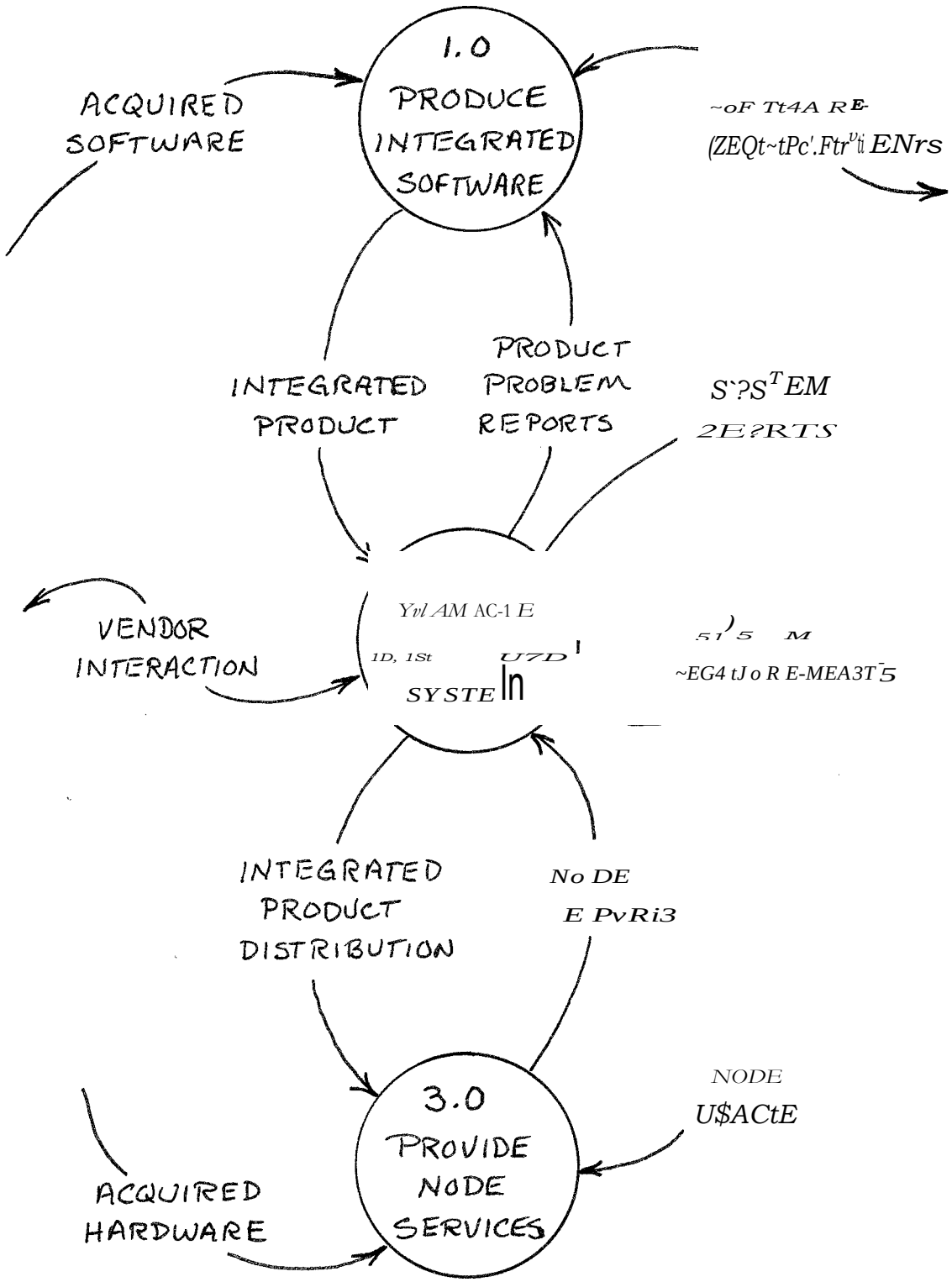
COAlr''EX i 1t IA6RI1F

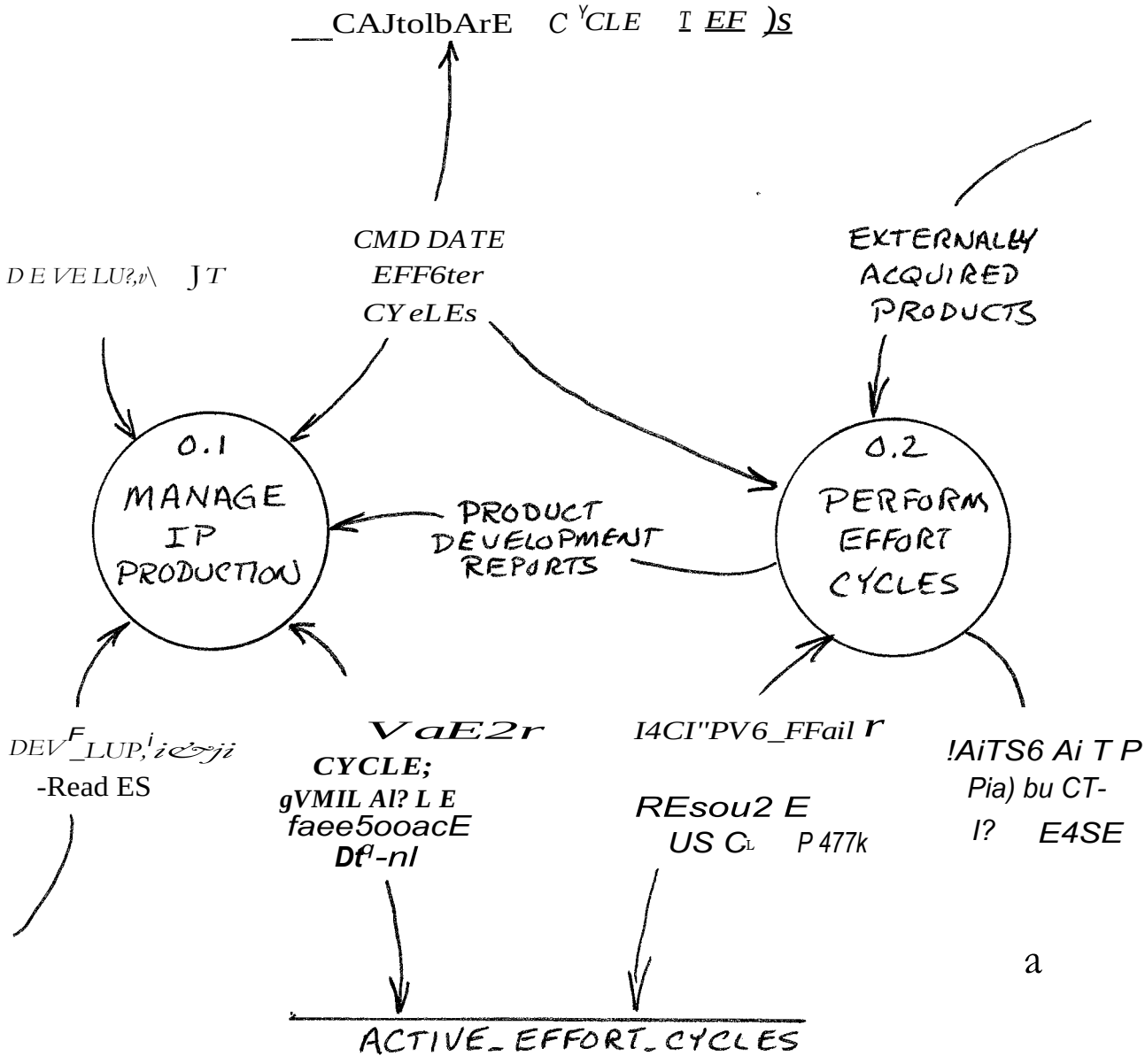


O.O
PROV(DE
L7' s- rc'a C~3 U QED
cowl PUTS .
'5.EkV ICE

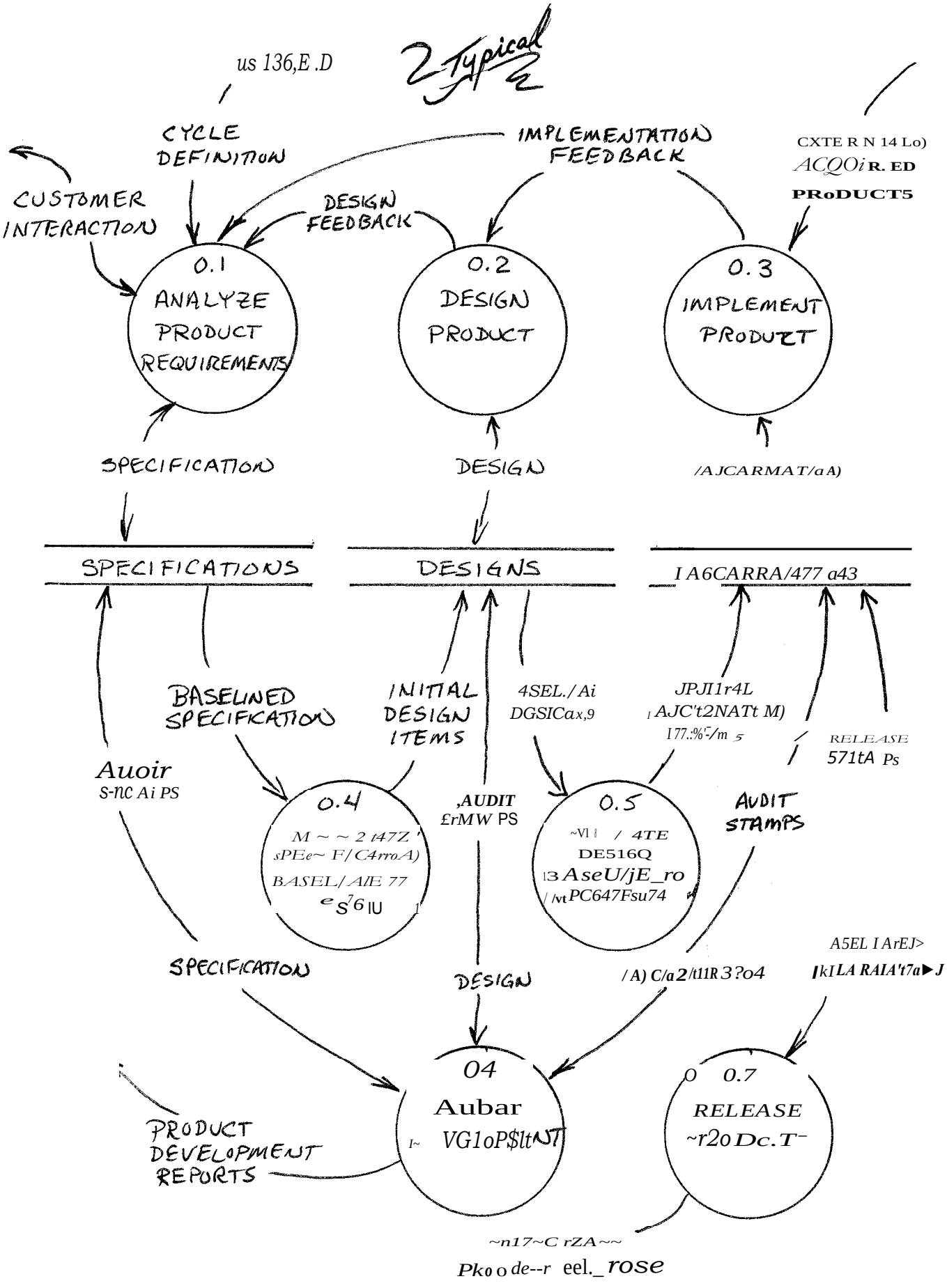
ACOOLIe D
RE-Sou ece
VENDOQ
1N1- 4erfo.)

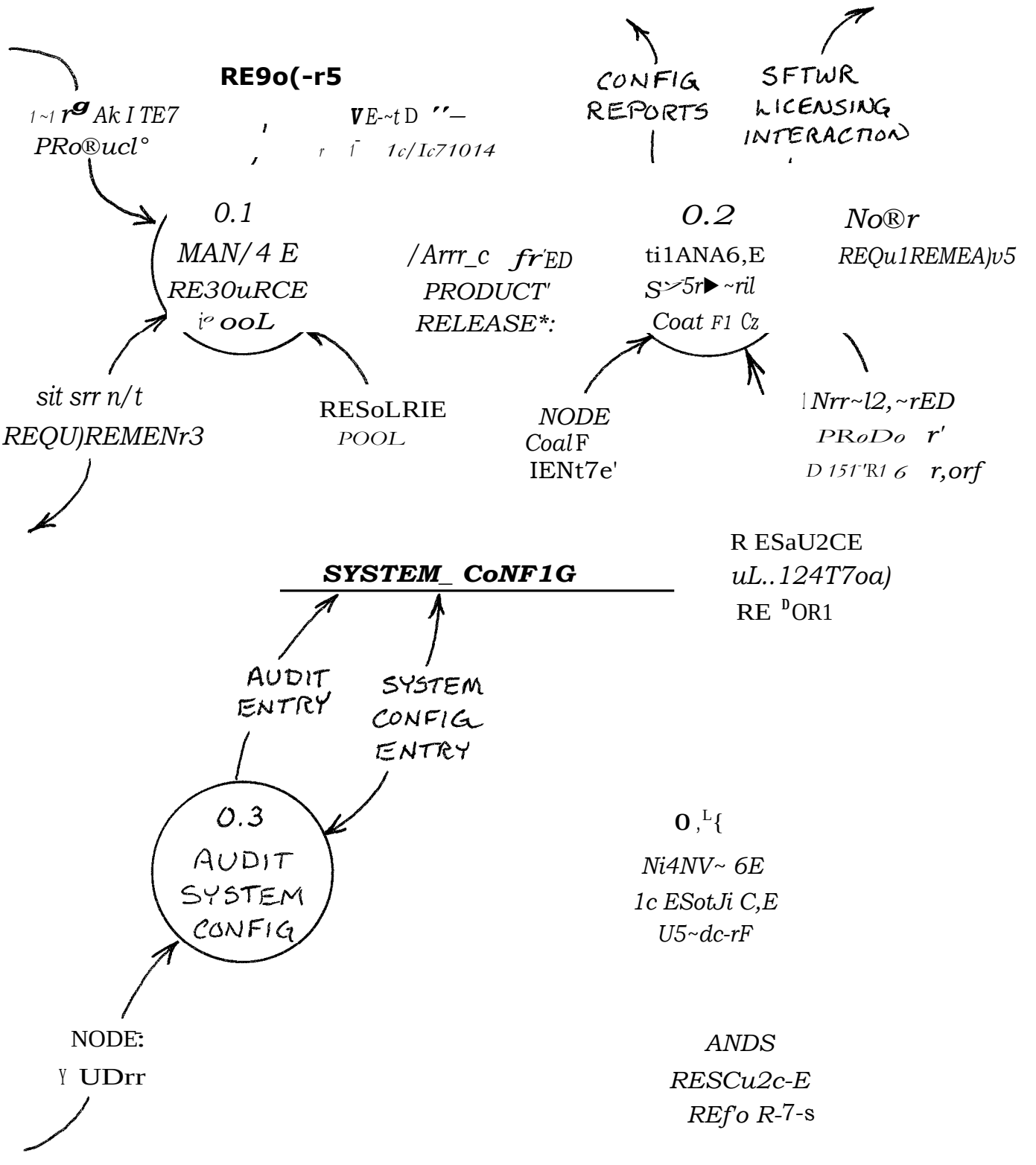
IVE!)D0i5

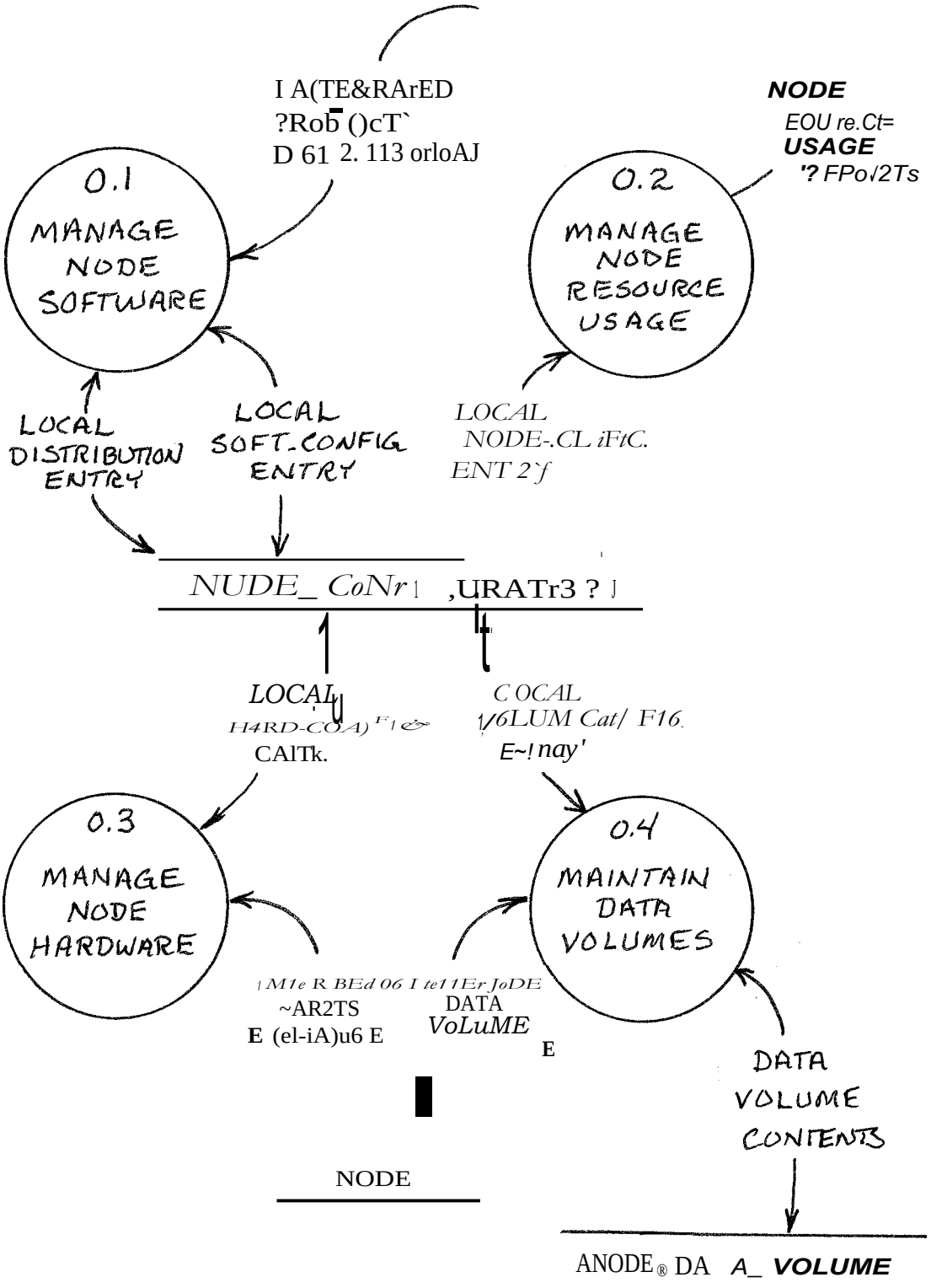

1.0
PRODUCE
INTEGRATED
SOFTWARE
ACQUIRED
SOFTWARE
~oF Tt4A RE-
(ZEQt~tPc'.Ftr'ti ENrs
INTEGRATED
PRODUCT
PRODUCT
PROBLEM
REPORTS
S`?S^TEM
2E?RTS
VENDOR
INTERACTION
YnlAM AC-1 E
1D, 1St
U7D'
SYSTE In
51)5 M
~EG4 tJ o R E-MEA3T5
INTEGRATED
PRODUCT
DISTRIBUTION
No DE
E PvRi3
3.0
PROVIDE
NODE
SERVICES
NODE
U$ACtE
ACQUIRED
HARDWARE

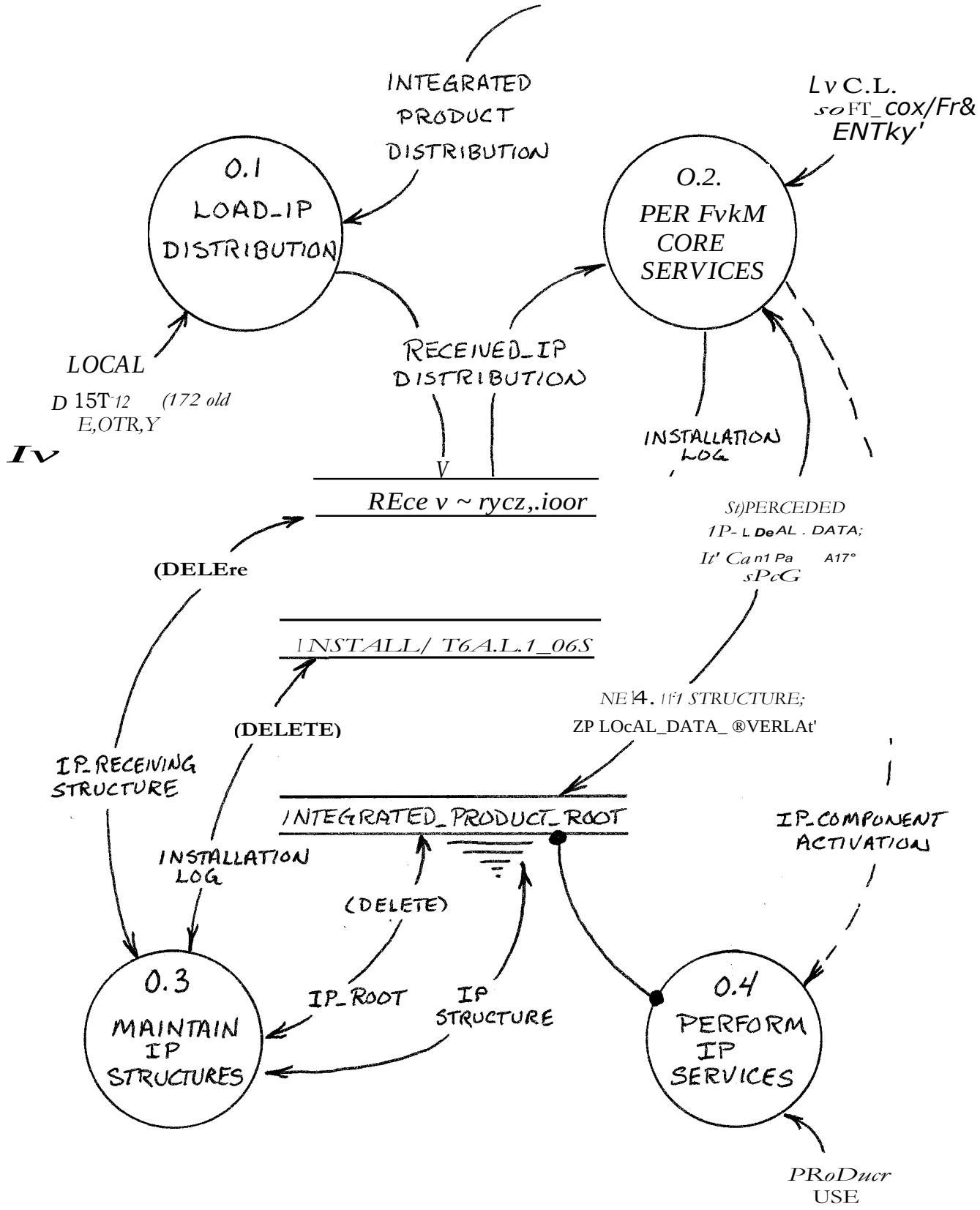

__CAJtolbArE C YCLE I EE)S
CMD DATE
EFF6ter
CYeLEs
EXTERNALLY
ACQUIRED
PRODUCTS
0.1
MANAGE
IP
PRODUCTION
PRODUCT
DEVELOPMENT
REPORTS
0.2
PERFORM
EFFORT
CYCLES
VaE2r
CYCLE;
gVMIL Al? L E
faee5ooacE
Dt9-nl
I4CI"PV6_FFail r
REsou2 E
US CL P 477k
!AiTS6 Ai T P
Pia) bu CT-
I? E4SE
-Read ES
ACTIVE_EFFORT_CYCLES

a


us 136,E .D
2 Typical
CYCLE DEFINITION
IMPLEMENTATION FEEDBACK
CXTE R N 14 Lo)
ACQOi R. ED
PRoDUCT5
CUSTOMER INTERACTION
DESIGN FEEDBACK
0.1 ANALYZE PRODUCT REQUIREMENTS
0.2 DESIGN PRODUCT
0.3 IMPLEMENT PRODUCT
SPECIFICATION
DESIGN
/AJCARMAT/aA)
SPECIFICATIONS
DESIGNS
I A6CARRA/477 a43
BASELINED SPECIFICATION
INITIAL DESIGN ITEMS
4SEL./Ai DGSICax,9
JPJI1r4L
AUDIT STAMPS
RELEASE
571tA Ps
Auoir
S-nC Ai PS
0.4
0.5
,AUDIT
£rMW PS
SPECIFICATION
DESIGN
A5EL I ArEJ>
PRODUCT DEVELOPMENT REPORTS
04
Aubar
0.7
RELEASE
~n17~C rZA~~
Pko o de--r eel._ rose


RE9o(-r5
CONFIG REPORTS
SFTWR LICENSING INTERACTION
0.1
MAN/4 E RE30uRCE
0.2
No®r
REQu1REMEA)v5
/Arrr_c fr'ED PRODUCT' RELEASE*:
sit srr n/t REQU)REMENr3
RESoLRIE POOL
NODE Coal F IENt7e'
PRoDo r'
R ESaU2CE
uL..124T7oa)
SYSTEM_ CoNF1G
AUDIT ENTRY
SYSTEM CONFIG ENTRY
0.3 AUDIT SYSTEM CONFIG
NODE:
UDrr
ANDS
RESCu2c-E
REf'o R-7-s


I A(TE&RArED
?Rob ()cT`
D 61 2. 113 orloAJ
NODE
EOU re.Ct=
USAGE
'? FPo/2Ts
0.1
MANAGE NODE SOFTWARE
0.2
MANAGE NODE RESOURCE USAGE
LOCAL DISTRIBUTION ENTRY
LOCAL SOFT-CONFIG ENTRY
LOCAL NODE-.CL iFtC. ENT 2 ƒ
NUDE_ CoNr | ,URATr3 ? J
LOCAL
H4RD-COA) F1 &
CAlTk.
C OCAL
1/6LUM Cat/ F16.
E~! nay'
0.3
MANAGE NODE HARDWARE
0.4
MAINTAIN DATA VOLUMES
| M1e R BEd 06 I te11Er JoDE
~AR2TS
E (el-iA)u6 E
DATA
VoLuME
E
DATA VOLUME CONTENTS
NODE
ANODE® DA A_ VOLUME


INTEGRATED
PRODUCT
DISTRIBUTION
Lv C.L.
soFT_cox/Fr&
ENTky'
0.1
LOAD_IP
DISTRIBUTION
O.2.
PER FvkM
CORE
SERVICES
LOCAL
D 15T-12 (172 old
E,OTR,Y
RECEIVED_IP
DISTRIBUTION
INSTALLATION
LOG
REce v ~ rycz,.ioor
St)PERCEDED
1P- L DeAL . DATA;
It' Ca n1 Pa A17°
sPcG
(DELERe
INSTALL/ T6A.L.1_06S
NE 4. 1f1 STRUCTURE;
ZP LOcAL_DATA_ ®VERLAt'
(DELETE)
IP_RECEIVING
STRUCTURE
INTEGRATED_PRODUCT_ROOT
IP_COMPONENT
ACTIVATION
INSTALLATION
LOG
(DELETE)
0.3
MAINTAIN
IP
STRUCTURES
IP_ROOT
IP
STRUCTURE
0.4
PERFORM
IP
SERVICES
PRoDucr
USE

usE