Re-engineering software: A case study

by R. N. Britcher

In 1986, the Federal Aviation Administration formed a contract with three companies to re-engineer a major portion of the New York terminal approach control (TRACON) application software—the software that supports air traffic control in the New York City and Newark, New Jersey, area. This paper discusses the techniques used to successfully re-engineer the software to run on an IBM System/370™, illustrating that real-time software can be logically converted from one computer to another, reliably and cost-effectively.

In 1981, the Federal Aviation Administration (FAA) issued a plan to upgrade the national airspace system. An overview of the system is depicted in Figure 1. The new plan included the host computer system that would replace the existing IBM 9020 computers (derivatives of the IBM System/360™) in the en route air traffic control centers, and the advanced automation system that would upgrade the host computer system and the terminal approach control (TRACON) and tower facilities.

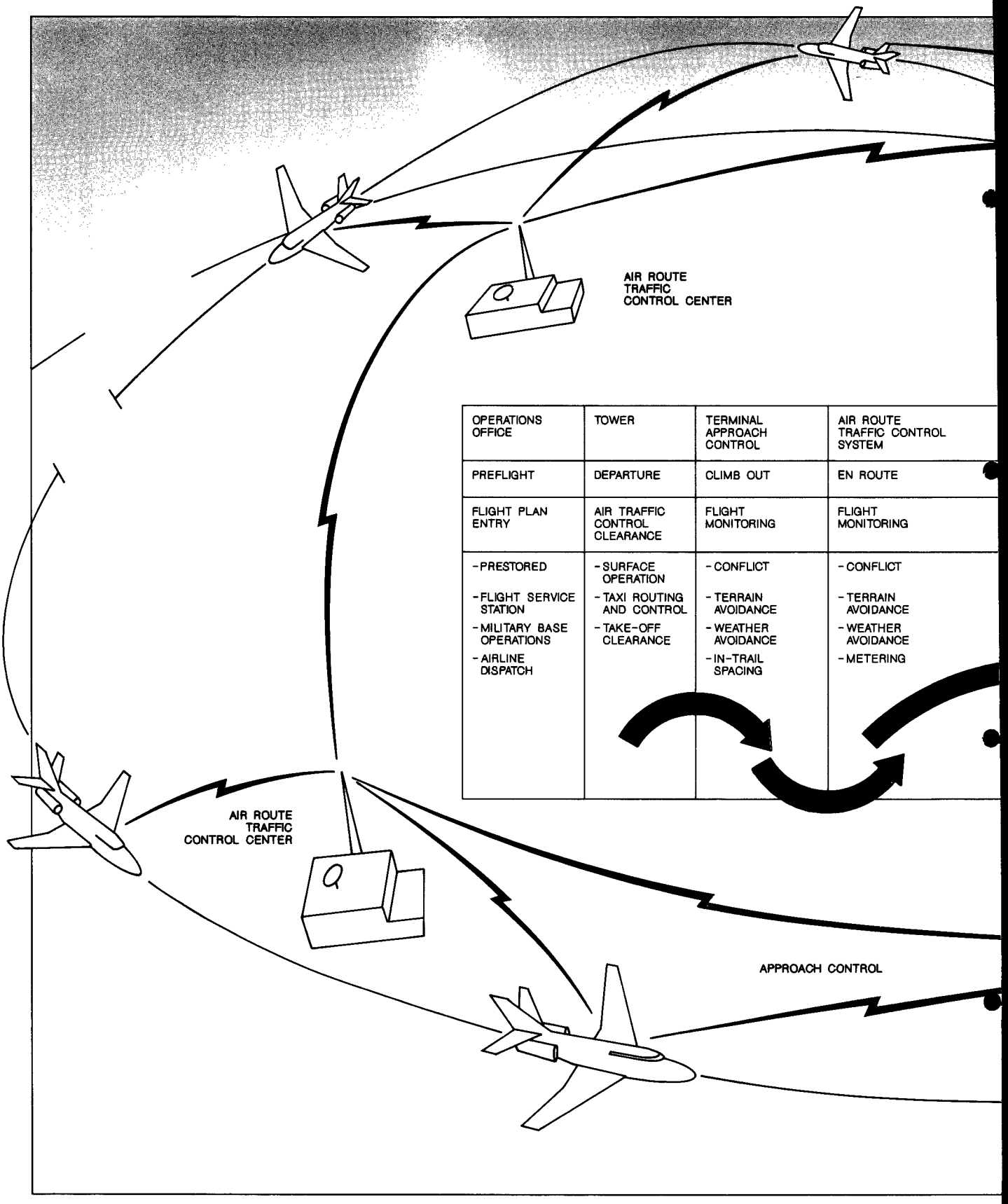
In 1986, during the host computer system acquisition and two years prior to the start of the advanced automation system acquisition, the FAA awarded a small contract, of approximately $2.5 million, to Data Transformation Corporation, International Business Machines Corporation, and Pailen-Johnson Associates to re-engineer[1] part of the New York TRACON software in a high-order language, to run on an IBM System/370™ Model 3083.

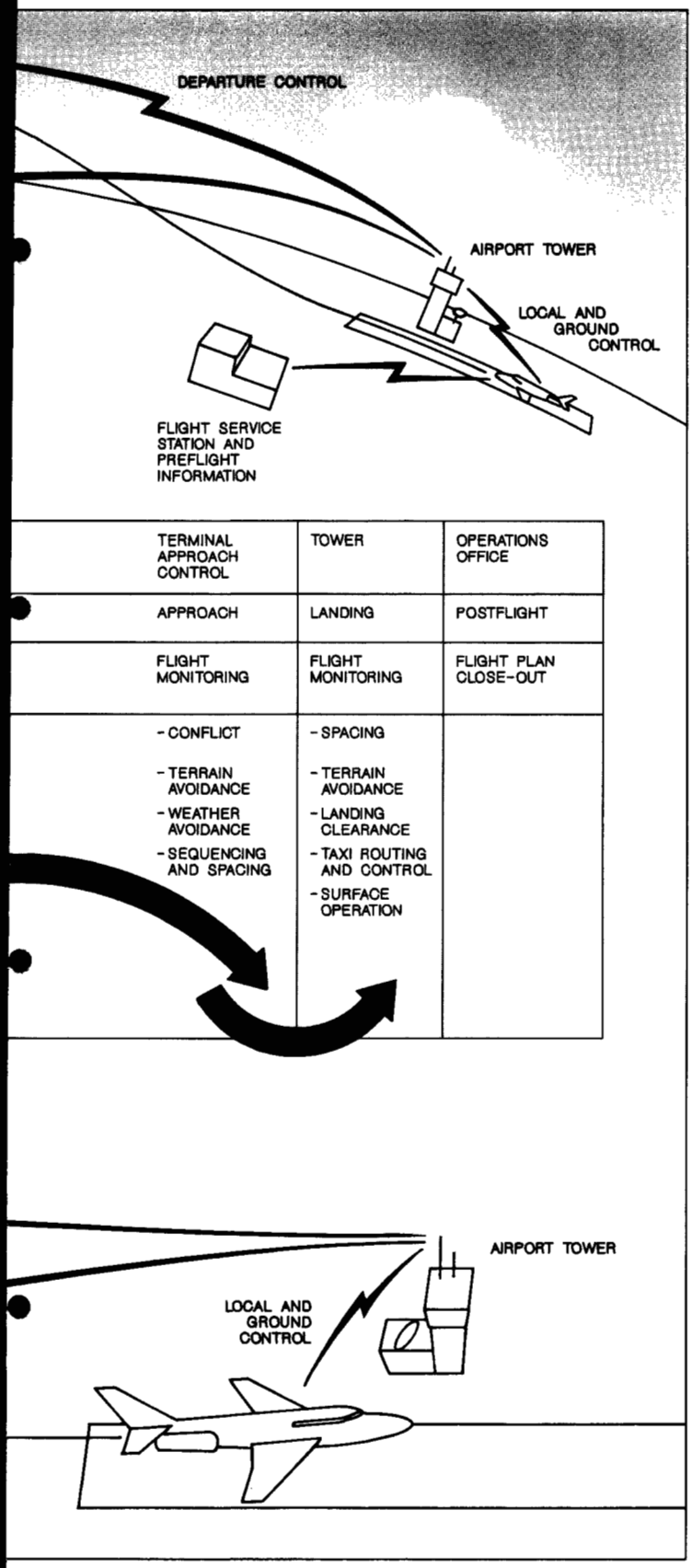
This paper describes the scope of that project and the tools and methods the contractors used to re-engineer the software. The results and conclusions indicate that re-engineering the software of an existing system may be preferable to reinventing it. Table 1 is provided to assist the reader in identifying the terms and abbreviations used frequently in this paper.

The scope of the project

The project was officially entitled "New York TRACON Demonstration of Program Recoding" and was aimed at recording the methods used to re-engineer the (then) current New York TRACON application software and verify the equivalence of the re-engineered version to its source.[2] The FAA's statement of work required the contract team to demonstrate and document a logical conversion of the existing New York TRACON operational software assembler source code (ULTRA™ from UNIVAC™) into a compiler language. The newly generated software had to be functionally identical and directly traceable to the existing code. The work was to be completed in nine months and culminate in two demonstrations: the first demonstration in early

Figure 1 Flying through the national airspace system





April of 1987, the second in mid-May, the ninth month. If it were successful, the project would bolster the idea of improving systems through planned evolution.

The objective in re-engineering. The FAA wanted to determine if a 20-year-old, real-time system could be cost-effectively re-engineered onto a commercial platform—including commercial hardware, commercial tools and languages, and commercial operating system software—while retaining the behavior of the applications. As a by-product of the re-engineering, the architecture of the applications and their database would be simplified to eliminate unnecessary dependencies, while creating a visible architecture. This is in keeping with the long-range objective of the FAA (and other United States government agencies) to eliminate customized system components and to develop and use application programs that are reliable and easy to modify and extend. Customized components typically require specialized training and maintenance and are vulnerable to changes in a product line or technology. The National Bureau of Standards publication in Reference 3 lists characteristics of systems that are candidates for redesign. Among them are code over seven years old, overly complex structure, code written for a previous generation of hardware, hard-coded parameters, and very large modules. All of these characteristics applied to the New York TRACON system.

It is essential that all parties involved in a particular application of re-engineering clarify its meaning and scope, and define their rationale for doing it. Does it include designing, coding, specifying, testing? Houtz,[4] in writing about how much software improvement differs from conventional system development, argues convincingly for a precisely documented software improvement program that would record which software would be affected; a strategy for each case such as purging, conversion, or replacing; a cost benefit assessment; and the detailed plan for implementing each strategy.

It was only after beginning the New York TRACON demonstration that the FAA and the contract team clearly understood the extent and type of re-engineering that would be applied to the TRACON software. For example, a recently completed project[5] moved the applications of the national airspace en route system from an IBM 9020 to an IBM System/370

Table 1 Terms used frequently in this paper

Term	Definition
FAA	
ARTS	automated radar terminal system
CDR	continuous data recording
FAA	Federal Aviation Administration
PSRAP	preliminary sensor receiver and processor
RETRACK	replay radar and controller data
TRACON	terminal approach control
UNIVAC	
ARTS IIIA	ARTS system coded in ULTRA
UNIVAC IOP	(input/output) multiprocessor (operational system)
MPE	multiprocessor executive for ARTS IIIA
Sperry 1100	development and support system
ULTRA	UNIVAC assembler source
IBM	
IBM 3083	demonstration (and target), development system
VM/SP-HPO	virtual machine control program for 3083

host computer system. Isolated portions of the software were respecified and redesigned, then coded in the original source language. The re-engineered system was tested by verifying its functional equivalence to the original IBM 9020 version. The New York TRACON demonstration, on the other hand, moved the applications from a UNIVAC host computer to an IBM System/370 host, replaced the platform software (the operating system), redesigned the software architecture and top-level design of the applications, specified the design (but not the system functions), re-engineered the software in a new target language, and tested it by verifying functional equivalence.

In both projects the results justified the cost. But without a well-defined rationale and plan, re-engineering could be at best too expensive, and at worst the wrong thing to do. For example, Sneed,[6] describing work performed for the Bertelsmann Publishing Corporation in West Germany, reported that the work of specifying and retesting, without re-engineering, 232 PL/I programs was grossly underestimated, and that only the judicious use of tools—such as a static analyzer—made the job feasible: the cost being two-thirds of the cost of developing the original software. Sneed also reported that the effort to find out whether software systems could be economically renovated resulted in no new software, only better documentation and a testbed for future testing.

The applications to be re-engineered. The New York TRACON is the largest terminal and approach control area in the United States, supporting five major airports in and around New York City. The automated radar terminal system that supports TRACONs around the country (ARTS IIIA), was developed in the early 1970s. The heart of the application software is the ARTS IIIA tracking algorithm. Because it is central to all air traffic automation, the FAA required that the contract team re-engineer the tracker as faithfully as possible. Tracking enables an air traffic controller to observe an aircraft's position and speed (and other important information, such as its transponder code) at the controller's workstation, by continually correlating the computer-modeled vector with the digitized radar return, every sweep of the radar. Tracking is supported by 1) an application program that buffers the digital radar returns, 2) an application that enables the TRACON to communicate with other facilities, and 3) an application that supports the controller workstation.

Of the ARTS IIIA applications, the FAA and the contract team agreed to re-engineer the tracker, the radar input program, or PSRAP (for preliminary sensor receiver and processor), a small portion of the interfacility support, and the controller workstation support. The PSRAP and tracking algorithms were coded virtually line for line, so that the FAA and the contract team could evaluate in some detail the results of re-engineering a real-time system. Although there were accommodations to the change in structure of the applications (the design process is described below) and to the lexical differences between source and target languages, the re-engineered software was traceable to its source without much difficulty. Because the target hardware and the operating environment differed from the source system, the other applications were re-engineered—with FAA approval—to achieve functional equivalence, but not necessarily algorithmic equivalence. In particular, because the prototype situation display used in the demonstration system was not equivalent to the TRACON workstation, the applications that support the data entry and display system were not re-engineered line for line.

Two support programs developed for the ARTS were key to the demonstration, because of their role in testing and verifying functional equivalence. The continuous data recording (CDR) editor reduces the data recorded by the operational software so that programmers and analysts can evaluate the functional performance of the system. A simulator,

RETRACK, uses the CDR data recorded from a previous run to feed the operational software, so that testing need not depend on the availability of live radar and controller inputs.

The contract team developed one new program: a CDR conversion program to translate the FAA-provided CDR file from ULTRA-compatible format to Pascal/VS records that could be processed by an IBM System/370.

The support programs and the conversion program provided a coherent thread of automation that connected airplanes to the controller monitoring them. It was enough to give the demonstration the characteristics of a complete system.

The software to build and execute the applications was provided by the contract team. The estimated number of lines of source code (in ULTRA) to be re-engineered was 8000 for the tracker and PSRAP, 7000 lines to support the workstation, 3000 lines for the operational database, and about 13 000 lines of support software, to drive the system and to record and reduce its outputs. (The actual number of lines converted exceeded the estimate by 66 percent: the contract team converted 53 000 lines of source code; the bulk of the 22 000 lines excluded from the estimate was taken up by the database specifications and the site (such as the location of the radar sensors) database.

The FAA provided the New York TRACON source code and listings, specifications and design data, database definitions, and files containing recorded radar and air traffic controller inputs that would be used to drive the system. In addition to providing contract team members, IBM also provided the laboratory and equipment to develop and demonstrate the converted software.

The current system architecture

The New York TRACON development and support system runs on a commercial Sperry™ 1100 processor under a commercial operating system, but uses specially-built software to build and reduce data generated by the operational system. The operational software can be assembled and linked on either the Sperry 1100 or on the operational system, UNIVAC's Input Output (IOP) multiprocessor. Data are reduced only on the IOP.

The operational system runs on the IOP, a multiprocessor developed for the automated radar terminal system (ARTS), and a derivative of the UNIVAC 8300. It drives the data entry and display subsystem, a configuration of Texas Instrument's workstations, containing neither software nor microcode.

The ARTS IIIA is coded in ULTRA, a 16-bit assembler developed in the 1960s for the UNIVAC processors. Like the assemblers of its generation, it was developed to get the most out of the target processor. The version of ULTRA in use now is not much different from the original, in that it offers no macro facility and no structured programming control structures.

The operational software is supervised by a customized multiprocessor executive (MPE) written for the ARTS. The rules for concurrent and sequential execution of the applications are encoded in a lattice and carried out and enforced by the MPE. Because the IOP is a multiprocessor, the MPE assigns work—radar to be buffered and sorted, tracks to be correlated—to the applications in a strict time-sequenced order. The MPE allocates each work-application pair to an available processor, the object being to keep the processors busy.

The application software is organized as tasks. The application tasks, in general, correspond to the applications: PSRAP, which processes incoming radar data; the tracker, consisting of the applications to support the controller workstation data entry and display; and the support for interfacility communications.

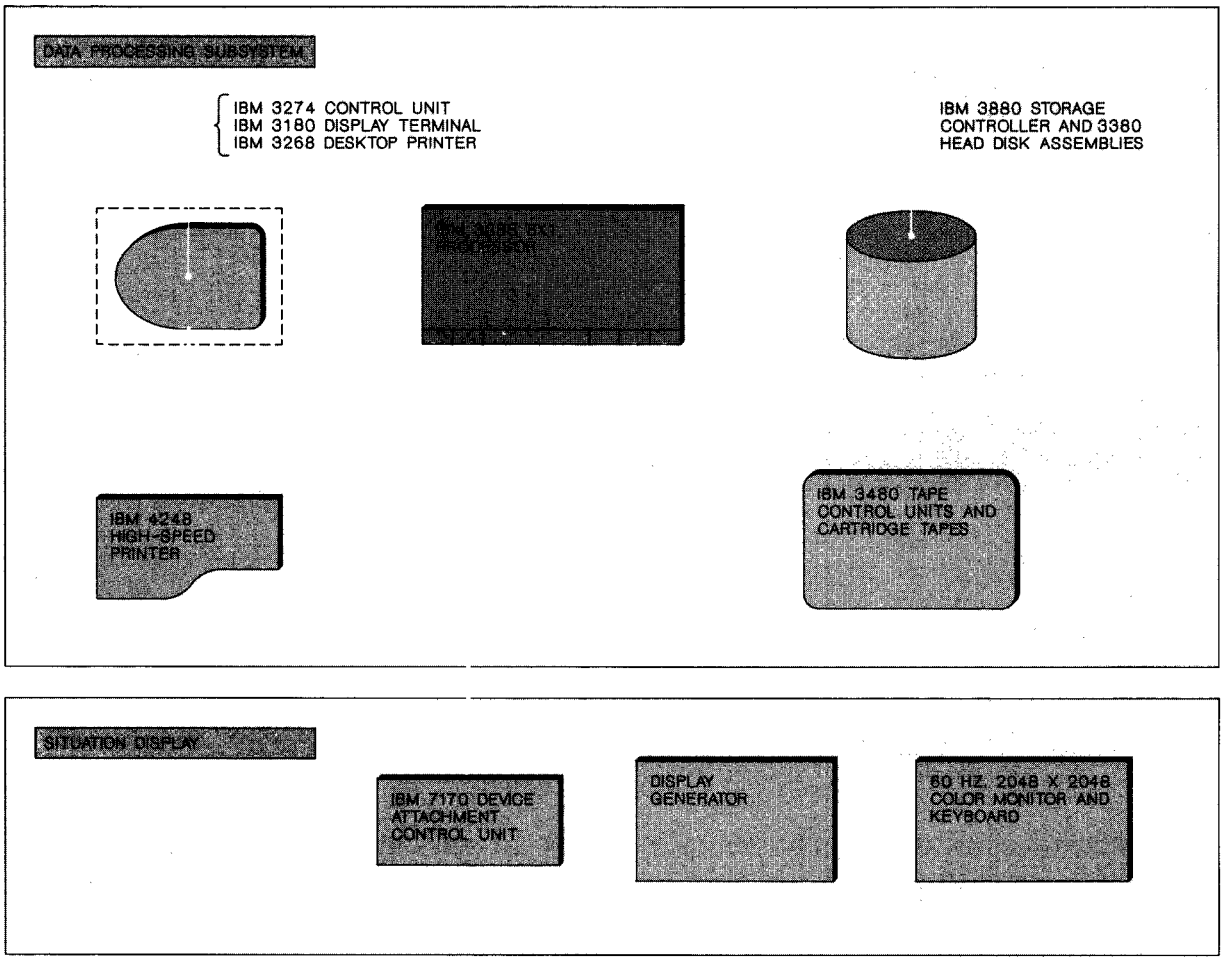
The small global database, comprising predominantly radar returns and tracks and the information describing the current state of each workstation, is organized in tables accessible by the applications without executive supervision.

The target system architecture

The contract team developed and demonstrated the re-engineered New York TRACON software on an IBM System/370 Model 3083. Unlike the UNIVAC IOP, the IBM 3083 is a uniprocessor. It executes at roughly four times the cycle speed of the IOP and contains 16 megabytes of primary storage. Its channels operate at about three megabytes per second. Each of its attached storage devices, both direct and sequential access, holds gigabytes of data.

The IBM 3083 drove a prototype situation display enabling the FAA to observe the display outputs. The situation display supports a 2048 by 2048 pixel color

Figure 2 Demonstration system hardware configuration

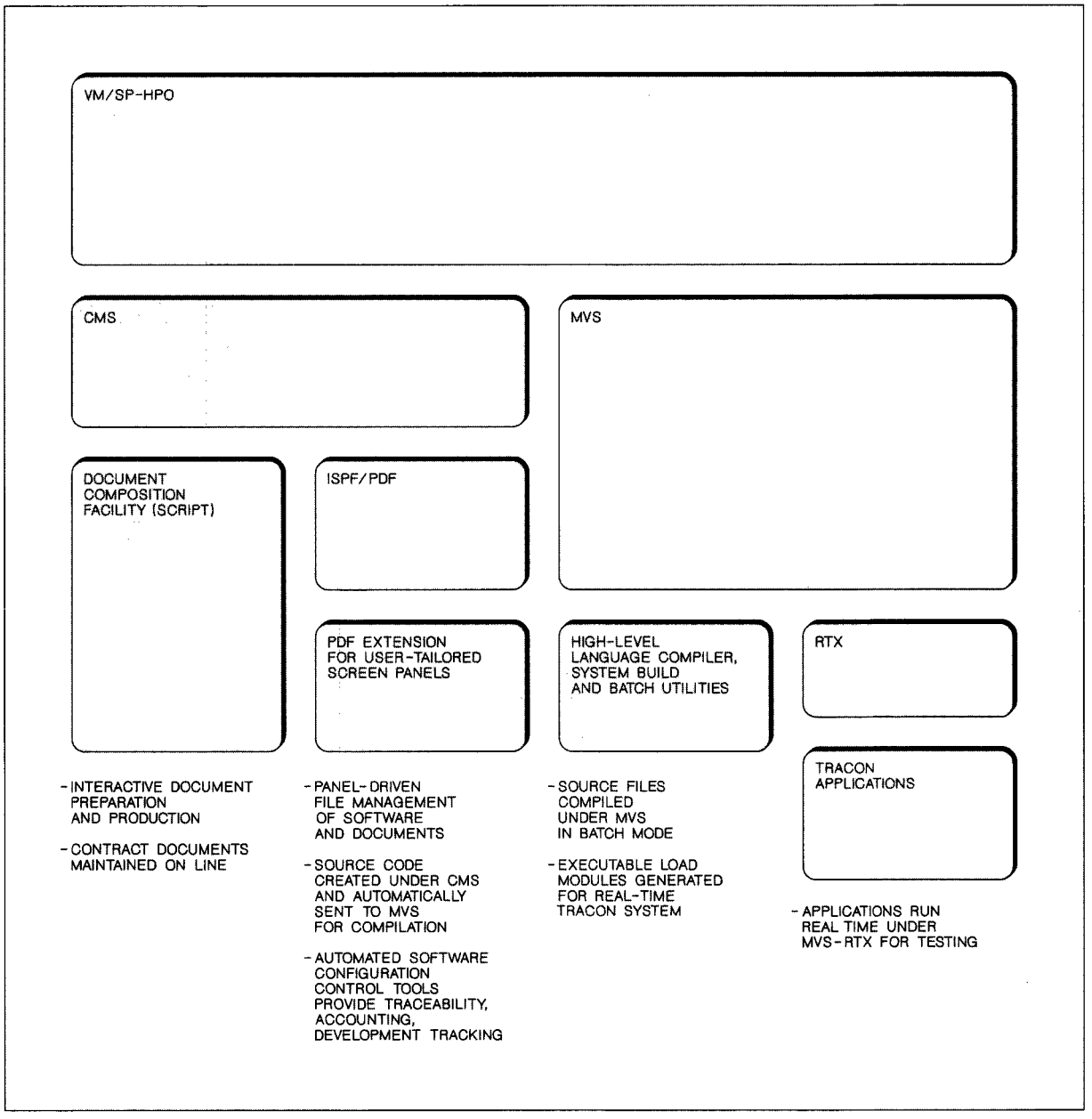


raster presentation driven by a state-of-the-art display generator, and it embeds a workstation processor. For the demonstration, it was used in a mode that emulated the current ARTS IIIA situation displays as closely as possible. Figure 2 describes the configuration.

The development processor and the demonstration processor were one unit. IBM's virtual machine operating system, VM/SP-HPO, made this possible. It allows users to run different systems and different kinds of systems, such as batch systems, interactive systems, and real-time systems, on a single processor.

The contract team chose Pascal/VS as the target compiler. The compiler and the operational system were controlled by Multiple Virtual Storage (MVS), whereas the development system was run under the Conversational Monitoring System (CMS). Both MVS and CMS can run concurrently under VM/SP-HPO. (Although the same processor supported both the development and the demonstration system, the customer demonstrations were given with the demonstration system running in native mode, that is, MVS interacting directly with the IBM 3083, without VM/SP-HPO and the development software, CMS, configured. The intent was to present the re-engineered system in an environment as close to operational as possible.) Figure 3 shows the commercial platform under which the re-engineered applications were developed and executed.

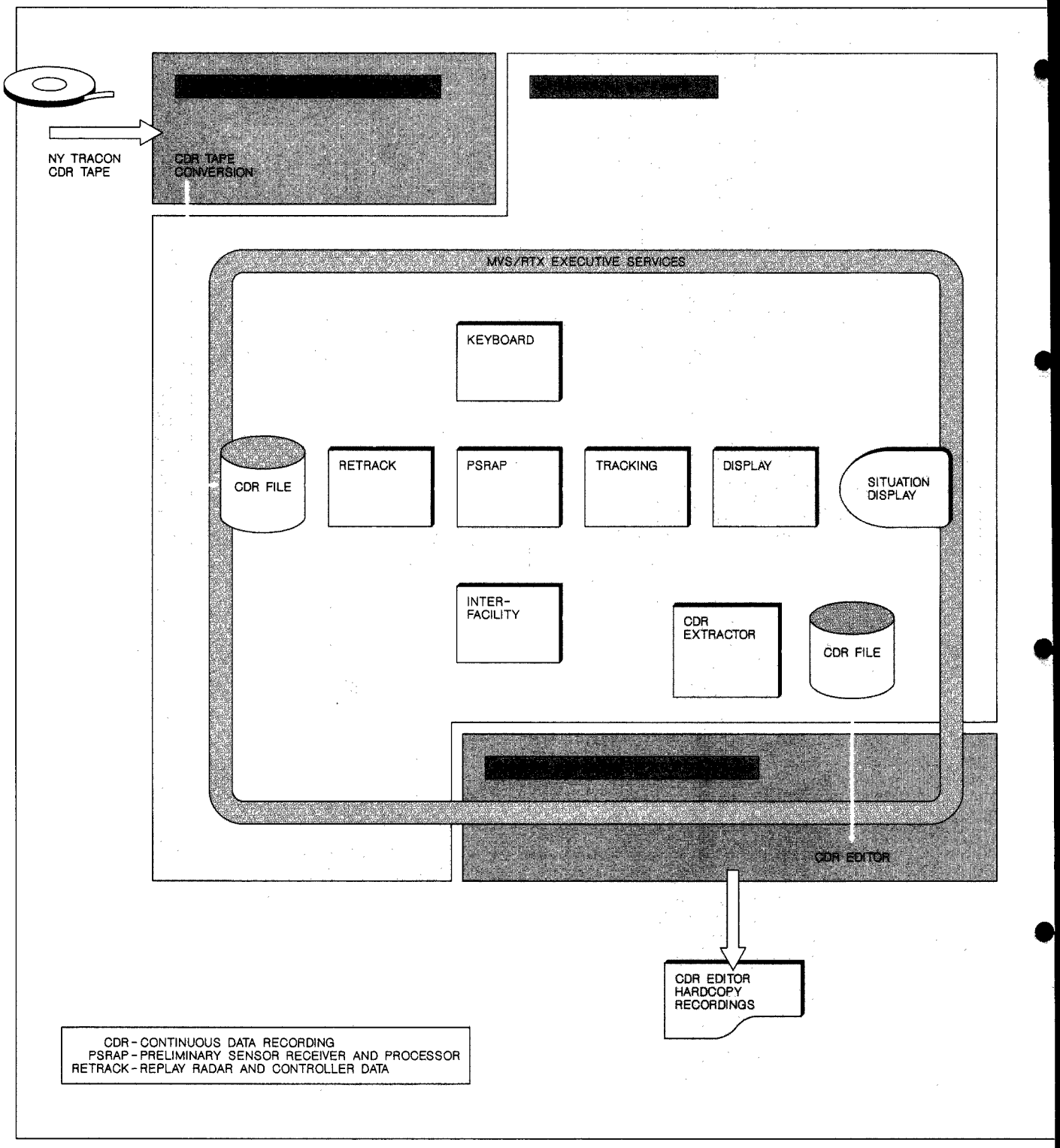
Figure 3 Development and demonstration system software architecture



Demonstration environment and data flow

The demonstration was run in test mode; no live inputs were permitted. It lasted about 20 minutes. The FAA-supplied CDR file, containing recorded radar, interfacility, and workstation inputs from a live New York TRACON operation, was read by RETRACK. RETRACK primed the input buffers of the radar input program PSRAP, the workstation keyboard and interfacility input programs; tracking was executed; and the resulting data were written to the display. The applications ran in real time—as if the inputs were

Figure 4 Demonstration flow



live—and recorded their CDR data. After the run was complete, the newly-generated CDR file was reduced by the CDR editor and the listing compared with a New York TRACON test run made using the identical CDR inputs. (See Figure 4.)

Tools

The tools were critical to the success of the demonstration. The development tools were similar to those IBM used in re-engineering the IBM 9020 en route system into the IBM System/370 host computer system.[5] They consisted of interactive programming and management tools, built on IBM commercial products, executing under CMS. The products included an interactive editor and library manager (ISPF/PDF), an interpretive language that enables programmers to take advantage of the ISPF/PDF dialog manager by developing their own panels and executive software (REXX), a relational database management system (SQL/DS), and an automated document preparation package (Document Composition Facility). For the host computer system IBM developed an automated software development plan to track the completion of all software and testing milestones from the level of the module to that of the system; an automated design issues and trouble report system; and an automated software configuration control and accounting system. These were changed slightly for the New York TRACON project.

The unifying architecture of the target system increased productivity significantly over that of the source system. (Growth in productivity might be one of the factors that would encourage a customer to re-engineer an old system, if the life span of the system were long enough.) Because the development system ran on the same processor as the demonstration system, a programmer could develop a schedule for a module, design, code, compile, unit test, repair, promote, and transmit it—through VM/SP-HPO—to the demonstration system for integration testing in a single session from the programmer's interactive terminal.

A tool that was central to the success of the demonstration was a PC-based relational database, hosted under dBASE III®, because it contained a data dictionary identifying every variable and constant in the source system, with the following attributes:

- the company responsible for coding the identifier
- the identifier
- the name of the table, if any, in which the identifier resides (in the source system)
- the page number in the New York TRACON coding specifications where the identifier is described
- the names of the procedures that reference the identifier
- with each procedure, a value—given here in parentheses—indicates whether the procedure sets the variable (1), uses it (2), both (3), or that data are not yet known (0)
- the type of identifier (e.g., character, Boolean, array, real)
- the new name of the identifier in Ada and Pascal/VS
- the name of the new design package that would own the identifier (the re-engineered system was designed to eliminate global data)
- the page number in the new database design document where the identifier is described

Figure 5 illustrates a page of the data dictionary.

The target language, Pascal/VS, and the target operating system, Multiple Virtual Storage/Extended Architecture (MVS/XA™), although they were part of the demonstration system architecture, were also tools, in that their characteristics facilitate and encourage certain methods. Pascal/VS was chosen because all of the programmers had experience with some form of Pascal, which is lexically similar to Ada used to develop the design, and because Pascal/VS runs under MVS. The FAA had requested that the demonstration system be coded in Ada. Because the contract team was not experienced in writing Ada and the schedule was quite short, all parties agreed on an Ada design (described in the section on methods) and a Pascal/VS implementation.

Lexically, the source system and the target system differed markedly. ULTRA is a 16-bit assembler that provides very little built-in support for development (e.g., no macro facility), for execution (no distinction is made between instruction space and the data space), or for testing. Because it is almost entirely a specification for the computer, its nature is lexical—and mechanical. Pascal/VS, because it is a specification for the programmer, provides not only a lexicon, but an emphasis; one that is mathematical rather than procedural. For example, variables and data types are specifications, and are distinguished from the instructions, which are fabricated of single-entry, single-exit control structures. Of equal importance, Pascal/VS implies rules for the development of programs, and it promotes the decomposition of

Figure 5 Data dictionary sample records

Record#	COMPANY	DATANAME	DATABASE	PAGENUMBR	P1	S1	P2	S2	P3	S3	P4	S4	P5	S5	P6
14		BOTIT9	BOT	4	TPUR	3		0		0		0		0	
15		ABEAT1	CTS	7		0	CRIT	0	DBATM	0	IFI	2	KOF	3	
16		ABEAT1	CTS	7	TEDC	3		0	TINIT	3		0	TPUR	3	TPRED
17		ABEAT1	CTS	7	PSBLD	0	RETRACK	1	SMOTH	0	TEDCRS	0	TI	0	TPSEC
18		ABEAT2	CTS	7	TPSEC	3	TCRSS	2	TINIT	3	IFI	3	TPUR	3	TPRED
19		ABEAT3	CTS	7	TEDC	3	TCRSS	2	TINIT	3	IFI	3	TPUR	3	TPRED
20		ABEAT4	CTS	7	TPSEC	3	COMA	3	TINIT	1		0	TPUR	3	
21		ABEAT5	CTS	7	TEDC	3	TCRSS	2	TINIT	3	DOP	3	TPUR	3	SLINK
22		ACTYPT	CTS	18	TPRED	0	TPSEC	3	TPUR	3	KOF	3	IFI	1	
23		ACTYPT2	CTS	18	TFI	3		0	KOF	3		0		0	
24		ALT1	CTS	17	TINIT	3	AUT	2	TPUR	1	ALTRKR	0	CDR	2	COMA

abstract programs into smaller, less abstract programs. It also imparts structure to a program and, to the advantage of large software products, among programs.

The choice of MVS to be the host for the ARTS applications was in keeping with the overall approach: providing a commercial platform for the re-engineered applications. MVS, unlike the specially-built supervisor developed for the ARTS IIIA (MPE), is a general-purpose operating system. It supports a wide range of applications, including both batch and interactive development, database management systems, network systems, production systems, and real-time systems. For example, the contract team used MVS to support batch compilations, to perform interactive reduction of CDR data using the CDR editor, and to run the ARTS applications. The last, because the ARTS executes in real time, required the presence of a real-time control program. The real-time control program used was RTX, developed by IBM to support the real-time applications for the National Aeronautics and Space Administration (NASA) and United States Air Force ground control space programs. Although the demonstration was carried out in a test environment using simulated inputs, the re-engineered system was designed and implemented as if it would be run in real time.

Methods

Within the contract team there were three teams, totaling 25 people, roughly aligned with the three companies. The first team was responsible for implementing the tools and the new software architecture. The second team concentrated on re-engineering the on-line applications. The third team coded RETRACK, the CDR editor, and the database constants, and conducted system testing. The teams were directed by a software architect, who developed the program plan, including the methods and the standards by which the software would be developed, and the software architecture. The teams met two or three days a week to discuss both methods and architecture. Every developer knew the approach, and frequent and formal communication marked the project. All project data, plans, standards, design, and code were on line and easily accessible through a hierarchy of directories.

A software development plan defined the software build plan and the standards for designing and developing the software, for software configuration control, and for software quality control. The standards contained templates, such as the gateway program that enabled each application to interface with the application control programs using one standard protocol. Every rule the developers might need was defined and in most cases examples were provided.

A formal life cycle, based on the traditional waterfall life cycle, was strictly adhered to. It was:

- Requirements analysis, methods, and architecture
- Level-1 software design (in Ada)
- Level-2 software design (in Ada)
- Coding, in three increments (in Pascal/VS)
- Software integration and testing
- Demonstrations and delivery of documentation

Figure 6 describes the contract team's approach to re-engineering. Although no software improvement program[4] was formally documented, the methods were defined in detail in the team's proposal to the FAA.

S6	P7	S7	P8	S8	P9	S9	P10	S10	TYPE	VARNAME	DBNAME	NEWPGNUM
0		0		0		0		0	E	bot_time_of_last_correl	TRACK	0
0		0	MSAW	0	MTGCT	0		0	L	last_correl_on_radar_only_trk	TRACK	0
0	ALTRKR	0	CDR	2	COMA	3	COMB	2	L	last_correl_on_radar_only_trk	TRACK	0
3	TSUB0	0	TSUB1	0		0		0	L	last_correl_on_radar_only_trk	TRACK	0
2	COMA	3		0		0		0	L	initial_correl_proc_enabled_trk	TRACK	0
2	SLINK	2	TPSEC	3	RETRACK	2	KOF	3	E	assigned_beacon_code_trk	TRACK	0
0		0		0		0		0	I	constant_zero_trk	TRACK	0
3	IFI	3	TPSEC	1	KOF	3	TPRED	2	I	beac_code_status_code_trk	TRACK	0
0	SCTME	2	TINIT	3	SLINK	0	TEDC	2	I	aircraft_type_code_trk	TRACK	0
0		0		0		0		0	C	sp_amen_site_ad_alphy	TRACK	0
3	COMB	2	KOFB	0	KOFC	0	MSAW	0	I	rep_altitude_trk	TRACK	0

The project began in September 1986. Within a month, two critical milestones were met: the development tools were adapted (from the host computer system) to suit the demonstration. For example, interactive panels that invoked the Jovial compiler on the host computer system were modified to invoke Pascal/VS, and a set of macros were developed that enabled RTX and the Pascal/VS run-time software to work together in a real-time environment (Pascal/VS was developed for a batch and interactive environment and its built-in storage management and error processing had to be modified).

The contract team did not reinvent the system. The source system requirements, an English description of its expected behavior, on the whole were not changed. To complete the requirements analysis, the team recorded their intent in the *New York TRACON Demonstration of Program Recoding Requirements Analysis Document.*

Re-engineering the software architecture. The official name given to the project, "New York TRACON Demonstration of Program Recoding," is somewhat misleading. A new architectural platform had to be designed before the applications could be re-engineered. The architecture of the source system is profoundly different from that of the target. (This would be the case in re-engineering many old systems.) Although both are von Neumann machines, the UNIVAC IOP is a multiprocessor, its storage shared among processors; the IBM 3083 is a uniprocessor. In the IOP, work is distributed among the processors to meet the system's demand for service in real time. The system was designed to get the most out of all the data processing elements, processors, channels, storage, and devices. In the IOP system, *the work is moved to the machine.* The supervisory software (MPE), schedules tasks from a prefabricated two-dimensional lattice, a directed graph that specifies the interdependency, priority, and, in some cases, the expendability of each unit of work. PSRAP must run before tracking, tracking must run before display, and so on. In the IOP system, the application is aware of its processing constraints. For example, tracking must execute within a fixed period of time, or work is flushed from the input queues.

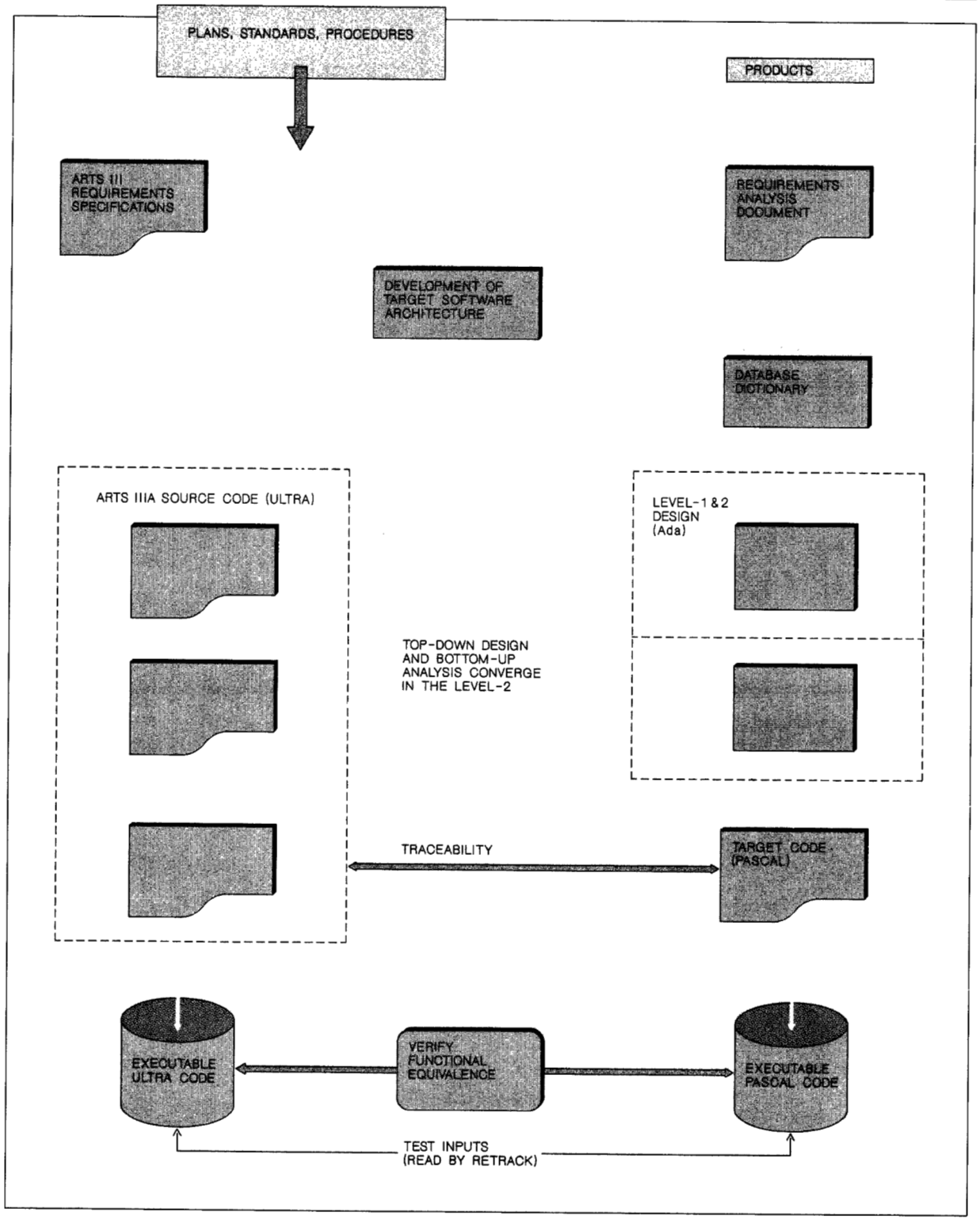
The IBM System/370 architecture is the inverse where *the machine is moved to the work,* at least in the abstract. The System/370 computer family, of which the IBM System/370 Model 3083 is representative, is event-driven. Work is defined and allocated dynamically. MVS dispatches applications when they are available, not according to a predefined sequence. The rules of concurrency, of data coherency, and of application communications are left to the designer.

The new software architecture centralized the control program services within its run-time system (that included MVS, RTX, and Pascal/VS) and a layer of applications control programs, thereby decoupling the applications from their processing environment, and insulating them so that they could be re-engineered free of operating system dependencies. A single gateway package enabled all the applications to interface with the control program services.

The architecture was developed as a set of rules and formally documented as part of a requirements analysis document. It contained the following:

- Definitions to include names of methods, attributes of behavior, objects (e.g., Ada package, application work, conversation, pipeline, process)

Figure 6 Overall approach to re-engineering



- Rules for packages and their attributes to define the five classes of applications and identification of the applications that fell within each: off line (O), control (C), interactive (I), pipeline (P), and data (D), and definition of the rules of behavior (e.g., pipeline packages shall not converse with each other)
- Rules for system work to define the units of work (e.g., radar targets, tracks) and their relationship to each other
- Rules for system parts and their work to describe the behavior of the packages with respect to their attributes (e.g., pipeline)
- Rules for the decomposition of parts to define the design levels (there were two) and their relationship to one another (e.g., the relationship between a Level-1 package and its Level-2 packages can be one-to-one or one-to-many)
- Rules for tasks and concurrency to define the behavior of the applications as real-time programs
- Rules for the communications between subtasks to define a message control system that would enable applications to operate independently, and send and receive messages to and from one another
- Rules for database and data coherency to define the rules for data aggregation and acquisition (data were defined as records in the demonstration system, rather than as tables in the source system, and, because the applications were redesigned as state machines, there were no global data)

The abstract rules provided a natural specification for the applications control packages, shown in the upper left of Figure 7.

Figure 7 describes one dimension of the software architecture, the Level-1 Ada packages, the attributes of each, and their interfaces.

Re-engineering the database. The new software architecture gave the applications independence, and as a result, it gave the teams coding them independence. The data dictionary progressed in parallel. As the architecture unfolded, developers studied the source system code and coding specifications of the applications, without having to worry about the environment in which they would run. The developers identified the variables and constants that are set and used by each application (e.g., PSRAP). By identifying the data space of the source system and mapping it to the target system, and recording both the domain (source) and range (target), the developers reduced the re-engineering of the applications to a simple mathematical function. The function is the set of ordered pairs (s,t), where s is the set of identifiers that define the potential states of the source system and t is the set of identifiers that define the potential states of the target system. In many cases the domain mapped to an empty set: variables were not needed because the target system excluded capabilities, for example, the live interface to the PSRAP.

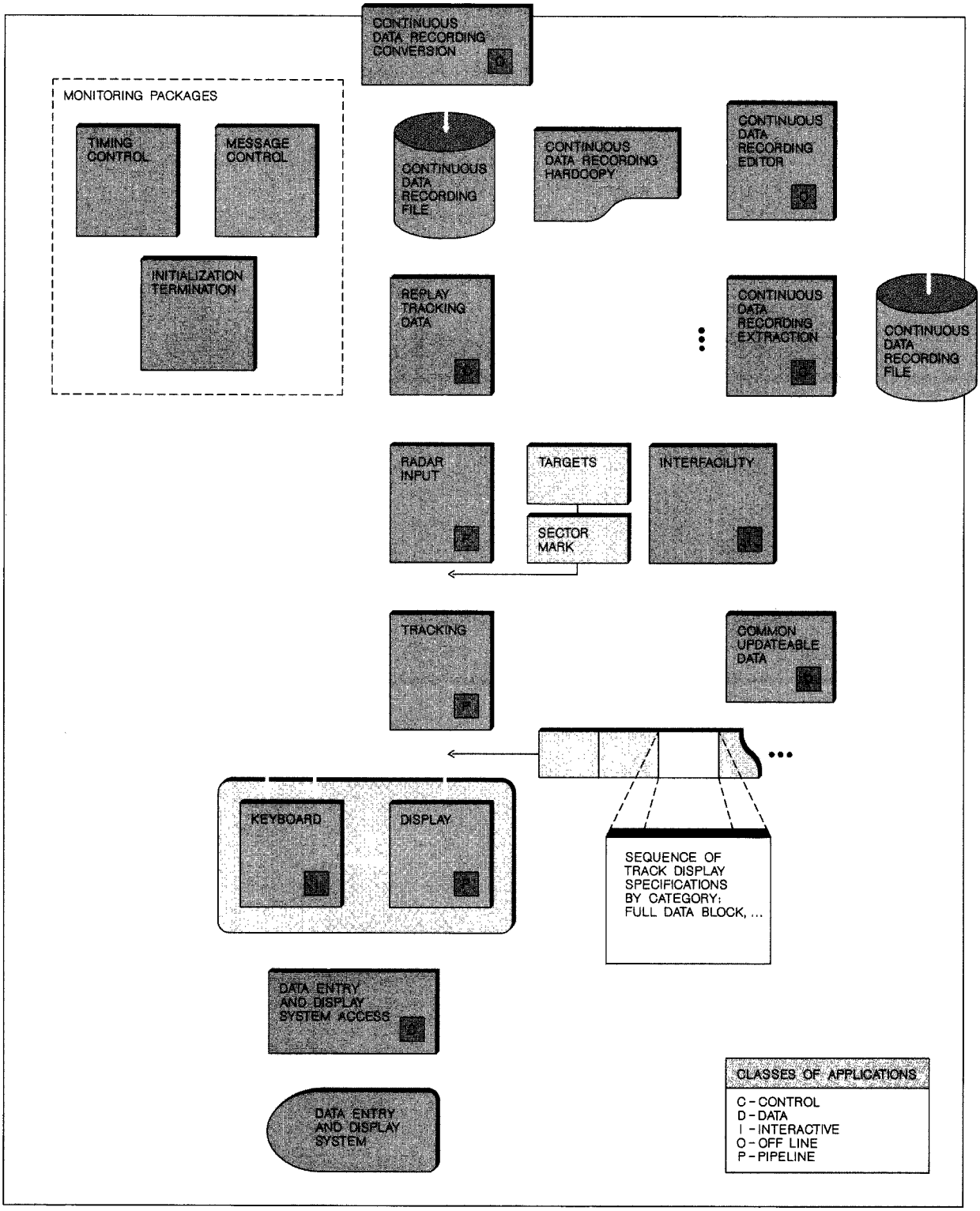
Formally defining the on-line software architecture and the data space, before elaborating the design of programs, was central to producing a target system that met the design and performance constraints within the short schedule.

Re-engineering the design of the applications. The design of the applications was re-engineered and recorded in two levels. The design was predicated on the methods used by IBM on all its major software development projects. The methods treat (computer) programs as mathematical objects, either functions or state machines.[7] A function maps inputs to outputs; a state machine is a function with a memory. Most modules or collections of modules behave as state machines. Designing a program as a state machine promotes the use of abstraction, where interfaces are specified before modules are fully elaborated, and data hiding. For example, a state machine implemented as an Ada package would specify the inputs, outputs, and a private set of state data, accessible to users of the package only by invoking one of its collection of procedures. In summary then, the contract team re-engineered the design of the applications as a set of communicating state machines.

The team used an Ada process design language to record the design. Using Ada as a design language turned out to have several advantages. Ada encourages both the use of abstraction and data hiding, so it is a good fit for using the concept of state machines; both the FAA and the contract team learned Ada and its characteristics, at least as a tool for design; the FAA is committed to using Ada on its large projects, and this project increased their confidence; Ada provided a common vernacular for talking about the design and the design process.

Three months after the project began, the Level-1 Ada packages were completely specified. All state machine interfaces, state data (the data owned by the Level-1 package), and CDR records were classified by type. The gateway package was specified. The successor packages, the Level-2 Ada packages that

Figure 7 Level-1 Ada packages and their relationships



would decompose from Level-1, were identified. All recording was inspected.

Incremental implementation. About a month later, the first of three software builds was completed and running. It included the support software, the CDR conversion program and the CDR editor, the on-line platform (including MVS, RTX, the Pascal/VS runtime support), the application control services (initialization/termination, timing control, message control), and the application gateways. The complete system structure was in place. The first build verified that records could be read from the converted CDR file and that messages (internal work) could be transmitted between tasks.

The Level-2 design (the full elaboration of the applications in the Ada design language) and the conversion of Ada to Pascal/VS proceeded in step with the build plan. The second build contained all the applications except tracking and parts of keyboard processing, both of which were completed in the third build.

Each Level-2 Ada package was inspected after it was completed and then converted to Pascal/VS. In writing the Pascal/VS code, the application programmer would use the Ada, the ULTRA, the ULTRA coding specifications, and the database dictionary to ensure that the re-engineering resulted in a functionally equivalent system.

The second and third builds were elaborations of the first. No structural changes were made to the system after the first build. The applications were laid into the system as they became available, and then verified by the software integration and test team.

Testing. During software integration and testing, formal error reporting and correction procedures and configuration control procedures were practiced, just as they would be on a large-scale development project.

Each build was tested at the unit and system level. Pascal/VS provided excellent debugging facilities; most modules ran in the system environment without error. System testing was performed under VM/SP-HPO and then natively, with MVS running directly on the processor. Not allowing enough time to test natively was a difficulty uncovered in the first demonstration. Errors were masked in the virtual machine environment. One of the advantages of transporting software, over inventing it, is the cost-effectiveness of testing by comparison. Because the source system and re-engineered system were functionally equivalent (with predetermined allowances), testing proved to be inexpensive and incisive. The outputs of the re-engineered system were compared with the outputs of the source system. Since both were driven by the same inputs, errors showed up as markers (discrepancies) rather than as the object of human evaluation and judgment, the reliability of which depends on many complex factors such as interpreting requirements, training, and experience.

The results

The contract team completed the re-engineering of the New York TRACON applications software and successfully demonstrated its functional equivalence in nine months. The budget was met. The team gave a final presentation and demonstration in May 1987. The demonstration was given using an FAA-provided input file, containing the recorded output of a then current run of the New York TRACON system. The FAA and the contractors compared the output recorded from the demonstration run with the New York TRACON generated output and found no unanticipated discrepancies. The analysis showed that the re-engineered tracker was functionally equivalent to the source original. The digitized radar and track data blocks stepping across the projected airspace on the modern situation display gave impressive visual evidence that the system worked.

The team converted over 53 000 lines of ULTRA to 83 000 lines of Pascal/VS (62 percent comments). The team developed 340 Pascal procedures. One-hundred-and-fifty-six (156) procedures, mostly in tracking, were implemented with no deviations from the original ULTRA. The Pascal/VS was converted from 47 000 lines of Ada process design language, including Level-1, Level-2, and commentary. The source code ratio of ULTRA to Pascal/VS, without commentary, was 1.6:1. The team identified and documented 58 design issues; all were resolved within a week. There were 78 errors found and corrected during software integration and testing.

Conclusions

The authorized logical conversion and demonstration of more than a third of the New York TRACON software, from an UNIVAC IOP ULTRA architecture to an IBM System/370 architecture, coded in Pascal/VS, is sound evidence that real-time software is transportable, even if the environments of the source and target system are markedly different.

In the final report delivered to the FAA the following factors were identified as key contributors to the success of the project:

- The software development environment, including the laboratory and software tools, was available shortly after the project started and was easy to learn and use.
- The team used a consistent and proven set of development methods and followed the standard software development life cycle, including requirements analysis and architecture, two levels of design, incremental software builds, design and code inspections, and an independent software integration and test team.
- The software architecture was formally recorded.
- A complete data dictionary mapped ULTRA variables to the re-engineered system.
- The on-line applications were designed as independent items, with no global data, with precisely-defined interfaces, and without embedding information about their operating environment.
- Early in the project, the system inputs were converted to System/370 format; the input records were defined as Ada data types to ensure consistency throughout the entire software system (the CDR editor, for example, used the same data types as RETRACK).
- The software was re-engineered in Pascal/VS, which made the transition from the Ada process design language easy, and, because of its rules for data typing and program construction, Pascal/VS code proved to be reliable.

The most significant problem, because of the tight schedule, was the short time allotted to test the system, in its final demonstration environment. This was overcome by working considerable overtime. At that, there were latent faults and errors, discovered after the final demonstration.

The advantages of re-engineering a software product or system, over reinventing it, are significant. Re-engineering avoids 1) the effort of, and the errors that accrue from, developing new engineering requirements and functional specifications, and 2) the effort of full-scale verification and validation, the very activities Sneed[6] found so expensive. In short, re-engineering takes advantage of the profound effects of evolution. It preserves the functional behavior of a system that had been specified, designed, implemented, repaired, enhanced, verified, validated, and most importantly, used over years, while improving its quality. The cost, as demonstrated by this project, need not be prohibitive.

Acknowledgments

The performance of the contract team was outstanding, as was the commitment and cooperation of the FAA and their system integration contractor Martin-Marietta. The FAA program manager, Dick Bock, deserves special recognition for his support and encouragement throughout the project. Art Smock from Martin-Marietta was particularly helpful in providing information about the New York TRACON system. The work of each member of the team was critical; in that sense every participant made an outstanding contribution to the success of the project: from Data Transformation Corporation—Bob Bosworth, Marie Brown, Annette Cabbell, Wai Cheung, Ben Dennis, Bill Earley, Hussein-Khomassi, Eugene Liberman, Stan Mead, Andre Small; from IBM—Dick Allardyce, Larry Boulia, Stuart Cramer, Jeff Dollyhite, Austin Gallow, Connie Hayes, Stephan Landau, Michielle Looser, Michele Ichniowski, Lana Massung, Margaret McCandless, Barb Morten, Bob Scheuble, Richard Wei; and from Pailen-Johnson Associates—Tom Baxter, Lalitha Bhat, Clarke Thomason, Stephen Hall, Michael Gandee, James Atkinson, Marvin Sendrow.

Cited references and note

1. In this paper, *re-engineering* means the authorized logical conversion of a customized architecture (implemented in assembler source code) to a commercial architecture (implemented in a compiler source language). Re-engineering does not include unauthorized reverse compilation of object code to form source code as the basis for the derivation of a substitute product, generally referred to as "reverse engineering."
2. *New York TRACON Demonstration of Program Recoding Software Translation and Verification Methodology Document*, DOT/FAA/CT-87/33, Federal Aviation Administration, U.S. Department of Transportation (August 1987); available through the National Technical Information Service, Springfield, VA 22161.
3. *Guideline on Software Maintenance*, Section 5, Federal Information Processing Standards Publication 106, National Bureau of Standards (June 15, 1984), pp. 14–17.
4. C. A. Houtz, "Software Improvement Program: A Treatment for Software Senility," *Proceedings of the 19th Computer Performance Evaluation Users Group*, National Bureau of Standards Special Publication 500-104 (October 1983), pp. 92–107.
5. R. N. Britcher and J. J. Craig, "Using Modern Design Practices to Upgrade Aging Software Systems," *IEEE Software* **3**, No. 3, 16–24 (May 1986).
6. H. M. Sneed, "Software Renewal: A Case Study," *IEEE Software* **1**, No. 3, 53–56 (July 1984).
7. A. B. Ferrentino and H. D. Mills, "State Machines and Their Semantics in Software Engineering," *Proceedings, IEEE Com-*

puter Society First International Computer Software and Applications Conference COMPSAC (1977), pp. 242–251.

Robert N. Britcher *IBM Federal Sector Division, 9231 Corporate Boulevard, Rockville, Maryland 20850.* Mr. Britcher received a B.A. in chemistry from Gettysburg College in 1968, and joined IBM in 1969. He has worked primarily on the automation of air traffic control systems and is currently assigned to the FAA advanced automation system (AAS). He has written about several aspects of software, including design, standards, correctness, size estimation and metrics, maintenance, and software systems engineering. His articles have appeared in a number of journals and magazines, including *IEEE Software* and (IEEE) *Computer*. Mr. Britcher's research interests include program correctness and verification, and the development and evolution of software systems. He is a member of the IEEE Computer Society.

Reprint Order No. G321-5418.