Object Tracking and Designation (OTD) Phase 2 Final Report

November 1990

prepared for

Los Alamos National Laboratory

Los Alamos, NM 87545

Order No. 9-XG9-F5131

DISCLAIMER

MASTER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED 970

Object Tracking and Designation (OTD) Final Report

Hamilton Technologies, Inc. (HTI) demonstrated on the Object Tracking and Designation (OTD) project, which was commissioned by the Strategic Defense Initiative Organization (SDIO) under Los Alamos National Lab (LANL) contract 9-XG9-F5131, the effectiveness of the OO1 technology to the development of complex scientifically-oriented applications for SDI. This document summarizes the results of the project; it also describes the role played by the OO1 tool suite and the OO1 methodology.

There are five primary sections to this report. The first discusses the general approach used to develop the OTD prototype system from a requirements document that was provided by SDIO. The second section discusses the assumptions and changes made (including simplifications) to these requirements that were applied to the project. The third section provides a brief statistical summary of the resultant OTD prototype, from which productivity enhancements can be observed. The fourth section discusses aspects of the development process that were enhanced by the use of OO1. Section V presents a set of expected "next-step" actions that would occur if the prototype were to continue to evolve. Section V is followed by a reflective condensation of the experience of the OTD project, as a whole. Following the conclusion at the end of Section V are the Appendices which contain the OO1 AXES specification of the OTD prototype.

Section I: The OTD Rapid Prototyping Approach

In the previous phase to this OTD effort a prototype Object Tracking and Designation system was developed using the 001 technology. In this effort HTI demonstrated the effectiveness of 001 as a development environment for the OTD System.

In the OTD system, Object sightings from Measurement Processing are sorted by Object Sorting into azimuth/elevation bins, then passed to Object Screening. Object Screening separates sightings which are part of an established track from those for which no track has been established. In Process Sighting, sightings determined to be part of established tracks are sent to Update Tracks; uncorrelated sightings are sent to Generate Tracks. Generate Tracks first performs a rate smoothing of the data in Rate Smoothing. In Candidate Track Selection (CTS) uncorrelated sightings are compared with the previous five to seven frames of uncorrelated sightings. Those sightings which can be put together to form a candidate ballistic trajectory are sent to the Trajectory Fitting process. In the Trajectory Fitting process, candidate tracks are fitted to precision trajectories. Those which are found to be valid trajectories are sent to Radiometric Discriminant Initialization in the form of a new track message, and object sightings making up the trajectory are removed from the Object Sighting data structure. (The trajectory fitting process is thus also used for track filter initialization.) In Radiometric Discriminant Initialization, radiometric discriminants are produced from the new track message and the resulting discriminant values used to initialize a track record in the Object Track File data structure which is passed to the Prediction function. The Metric Discrimination process uses angle data to determine object lethality. The object's designation is then sent to the Prediction process. In the Prediction process the track's position and uncertainty on the next frame is predicted based upon a coefficient corresponding to the object's estimated class and on the expected interception of the track and the scanner on the next frame.

In the previous phase, the OTD system was defined in 001 AXES and automatically analyzed for interface errors with the 001 analyzer.¹ Once each module in the OTD

---

¹ Interface errors make up approximately 75% of the errors that are found after implementation with a conventional technique. With 001 they are found before implementation. This means that approximately 75% of the "testing" of the 001-defined and developed OTD system was completed before implementation if we compare the 001 process in development to that which is equivalent to a conventional development process. Interface errors include data flow, priority and timing errors throughout a system definition to a very fine grained level.

system was analyzed, the 001 RAT was used to automatically generate production-ready C source code. The OTD system was then performance tested for a first round with a set of our own defined tests and data.

The initial OTD effort was a ten man week effort starting with learning the requirements and ending with a round of preliminary performance testing. A major part of the effort was spent in becoming familiar with the OTD system requirements. Approximately one man month was spent learning the OTD requirements, one man month on developing the system with 001 not including performance testing and 2 man weeks on the first round of performance testing. The system developed with 001 is roughly estimated to be comparable to a 23,000-28,000² statement system in C source lines manually developed in a highly structured and modular conventional development environment or a 13,000-18,000² statement system in C source lines being manually produced with a generic reusable library with power equivalent to that of 001.

The earlier OTD phase focused on deriving from the requirements the TMap structures and top level FMap sketches needed to specify and implement the OTD system. A preliminary TMap was defined and analyzed using 001. Concurrently, FMaps for the top level of the system were being defined and analyzed. Both the TMap and the associated FMaps from that phase were developed using an incomplete requirements document. It was assumed that the rest of the requirements would be provided as needed. During the analysis phase, supported by the 001 Analyzer, it became evident that several areas of the requirements were either inconsistent or incomplete. As the missing requirements were provided, the TMap and FMaps were changed. Any changes made to the system being developed were then automatically propagated to the entire system via the 001 Tool Suite. If a change were made, for example, to the TMap, then 001 automatically forced the re-analysis of FMaps that could be invalidated by the change. Any invalidated FMaps were then properly adapted, taking into account the latest instance of the requirements. At any given point, a working executable prototype existed. This refinement cycle, of getting more accurate or new requirements, of analyzing the requirements specifications, of changing the TMap and FMaps to reflect specification

---

² This figure was obtained by counting the number of C lines of code automatically generated by 001 and reducing it by a factor of approximately forty to sixty percent to account for a typical programmer's stylistic preferences and removal of comments. (It should be noted that the executables of a 001 developed system are normally smaller than the executables for a conventionally developed system of the same functionality.)

changes due to the new requirements, of testing the new system, was repeated until the end of the project.

The original purpose of this phase of development was to performance test the OTD system. The project was a redevelopment effort in many respects due to aspects such as changing requirements and missing information. The functionality that comprises the final OTD prototype has been divided into 15 application specific libraries. The breakdown is structured in terms of major functional categories or algorithms. The following table is presented here, so as to simplify discussions and references that occur later.

Library Name	Typical Reference	Brief Functional Description
OTD	"OTD"	Main library from which system is linked and run.
OBJECT_SORTING	"Object Sorting"	Sorts sightings, builds database that is updated by rest of system.
OBJECT_SCREENING	"Object Screening"	Performs track correlations and some track maintenance.
CTS	"CTS"	Candidate Track Selection, frame to frame correlation on sighting data.
TRAGFIT	"Trajectory Fitting"	Controls the call to the Laplace fitting algorithm.
LAPLACE	"Laplace"	Performs a first order curve fit on data provided by CTS.
KALMAN	"Kalman"	Performs a precision fit on data provided by LAPLACE. *
METRICDISC	"Metric Discrimination"	Computes certain values needed by Track Update.
TRACKUPD	"Track Update"	Updates a track.
PREDICTION	"Prediction"	Computes predicted points in the next frame.
KNEWTKMSG	-	Initializes a new track.
ALGO_SUPPORT	-	Algorithmic support functions referenced by several libraries.
GEN_SUPPORT	-	General support functions.
TAT_DEBUG	-	Performance, debug, and statistics gathering functions.
STATREPORT	-	Run time statistics gathering functions.

Breakdown of OTD system, according to functional grouping in 001 Libraries

* this functional group was entirely simulated.

Table I.

Section II: Assumptions and Simplifications

A variety of simplifications were applied, in order to limit the scope of the project to that of a prototype. These are listed below.

- The state vector computed by the Laplace algorithm is used as the initial state vector of the object being tracked. The Kalman smoothing algorithm is not applied to the state vector computed by the Laplace algorithm. Instead, a simple Runge-Kutta integrator is applied to propagate the state vector to the time of the last frame.
- The Track Update algorithm does not apply the Kalman filter to the data correlated by the Object Screening function.
- Since the refraction table parameter and refraction indices were not provided, the error in elevation (EL) is assumed to be significant.
- The tracking of an object (correlation of tracks by Object Screening) is simulated by setting a simulation flag. It always correlates the predicted point with the correct track by performing a match on the track identification number. This flag can be turned ON or OFF for each simulation run.
- The Kalman smoothing algorithm is simulated and assumed to be perfect (i.e. it always filters out invalid tracks by looking at the track identification number).
- The OTD system was simplified to ignore the problem of multiple objects being associated with a single sighting.
- The OTD system was tested using a simplified data set. A five track data set was used to test the system.

Section III: The Resultant OTD Prototype

The final prototype OTD system is a collection of over 440 modules. All modules are written in the 001 specification language, 001 AXES. Approximately 6,500 lines of 001 AXES specification were written from which roughly 30,000 new lines of C was automatically generated by the 001 Resource Allocation Tool (RAT). Additionally, this body of C references a large portion of the functions supplied by the 001 Core Library; the rough equivalent, in lines of C-code that would have

otherwise been written, of these Core Library functions is at least 10,000 lines of C. Thus, the total effort is equivalent to a system, developed with conventional techniques, of approximately 40,000 lines of C code. Table II provides a detailed breakdown.

In the preliminary phase of the OTD project (before any changes were made to the original requirements), the system grew extremely quickly to almost the size that is represented in the table. That phase of the project has been documented elsewhere³, and is an example of the productivity that results when 001 is used to define a brand new system.

The later phase of the OTD project (the period during which the requirements were evolving) is an example of how 001 supports the maintenance of an existing system. The growth of the system in size, compared to the preliminary phase, was not significant. However, several sections of the system were altered or even completely re-written to accommodate the modified requirements. 001's Effects Analysis features automatically ensured that modifications entered into the system would not conflict with other portions of the system, making the job of maintenance much easier and the safe completion of modifications much swifter.

Section IV: The Development Process in terms of 001

The 001 tool suite was found to aid the development of the OTD system in a variety of ways, some of which were powerfully direct, others subtle (but often, even more important). The use of the 001 Analyzer frequently uncovered several weaknesses (ambiguities, inconsistencies and contradictions) in the given requirements. This section first discusses examples of errors in the original requirements. A far more subtle class of errors found by 001 (a class that plagues any extended development effort) were the result of evolving requirements; changes to the requirements, "midstream" as it were, are extremely prone to conflict with the premises of the previous state of the requirements. Examples of this are also discussed.

---

³ Final Report: Object Tracking and Designation (OTD), Architecture Independent Operating System (AIOS) and Run-Time Ensemble Benchmark Language (REBEL), February 1990, prepared for Los Alamos National Laboratory.

Library Name	# Modules	# lines 001AXES	#lines (raw) C generated
OTD	19	433	1889
CTS	93	1318	6588
OBJECT_SORTING	47	836	4039
OBJECT_SCREENING	19	336	1704
TRAGFIT	1	7	49
METRICDISC	11	156	742
LAPLACE	48	803	3636
KALMAN	1	26	116
PREDICTION	102	1056	6536
TRACKUPD	13	240	1041
ALGO_SUPPORT	3	49	180
GEN_SUPPORT	55	524	2908
KNEWTKMSG	13	254	1118
STAT_DEBUG	14	327	1433
STATREPORT	6	123	544
Totals :	445	6488	32523

Adjusted C: 22,876
TMap Generated C: 6,535
Adjusted Total Generated C: 29,411

001 CoreLibrary Contribution: (≥) 10,000
Total # lines Equivalent (Basis 1*) (≥) 39,411

* Basis 1 is used as comparison standard and represents the expected equivalent number of C-code lines that would have been written by a programmer attempting to produce the same functionality. Basis 1 assumes the programmer starting from scratch with a standard VAX/VMS environment, a C compiler and Linker, but without a powerful application specific support library like the 001 CoreLibrary.

Table II.

IV.1 Errors in the Original Requirements

Through the feedback provided by automated analysis, the original requirements were discovered to be incomplete, requiring that the missing parts of the requirements be provided to HTI. For example, several pages from the requirements of the Laplace curve fitting algorithm were only provided after a set of FMaps was defined and analyzed using the partial requirements. It became clear that large parts of that algorithm had to be rewritten to satisfy the newly revised and more complete requirements. In the original requirements provided, pages 127, 133 and 134 were missing. Page 127 contained critical initialization values for objects needed by the Laplace algorithm. The method that was used to compensate for that missing data was to assign dummy initialization values for those objects. A first level of integration testing was then performed with those dummy values. This allowed for continuity in testing always emphasizing integration. Once the correct initialization values for these objects were provided, the Laplace module went through a final set of tests.

Pages 133 and 134 contained algorithmic details for the threshold test filtering decisions, which once provided, required changes to several FMaps. In this case, not having the full specifications from the beginning forced rewriting and forced adding several new FMaps. The impact of this rewrite was noticed to affect modules in the Laplace algorithm only. The modular and hierarchical organization of the FMaps allowed for very efficient modification of the specification. Most of the low level modules were reused without any modifications; this is also true of most of the high level modules of the Laplace algorithm. The most affected areas were those involved in controlling decision flow, since the missing requirements pertained to threshold results. In the Laplace algorithm, a track is validated on the basis of some test threshold results. A few selected modules were then modified to make the threshold decisions, the system was brought to a consistent state using the 001 Analyzer, new code was generated and a new wave of performance testing was initiated.

The use of 001 revealed another interesting weakness in the original requirements document: the notation utilized was not consistent throughout the document. Several conceptually different objects were given the same symbol, complicating the understanding of the specifications. The converse was also found to be true, several

conceptually identical objects were given different symbols in the requirements again complicating the understanding of the requirements. A typical example of an object being given different symbol is the symbol used for the object state vector. In the Prediction algorithm, the symbol for the predicted state vector is $\hat{x}_0$, the same object in metric discrimination is given the symbol x_0 . The coordinate transformation matrix is always referred by the symbol C. This identical symbol is used in for both the *Earth Centered Inertial to Local Level* frame of reference transformation matrix and *Local Level to Earth Centered Inertial* transformation matrix, even though these two matrices are the transpose of each other. If conventional development techniques had been employed, instead of 001, a profusion of extremely subtle run-time errors would have resulted from this inconsistent usage of symbology; these errors would only have been found well after the implementation and integration phases had been completed, and the corrections would have to be made, after the fact, to an already "finished" system.

Finally, the use of 001 located several areas where ambiguous interpretations were possible. Some involved sections of the requirements that were blackened out due to their classification sensitive nature. Additionally, the original specifications made reference to documents which were not provided. This made the reading and interpretation of these requirements areas difficult, and increased the tendency for the English to be ambiguous in interpretation. Each ambiguity had to be resolved, in order to satisfy the Analyzer.

IV.2 Evolution of Original Requirements.

Several aspects of the original requirements underwent changes, some of which were quite involuted, in the midst of the development process. Examples of these are listed below.

- The original requirements assumed a scanning type of sensor, the new requirements a staring type of sensor. Several totally unspecified algorithms had to be developed to support the staring type of sensor. The Object Sorting algorithm needed modification, in this case, a linear time shift was applied to all scanning sensor data. This algorithm was only provided during the second phase of the OTD effort.

- Some fundamental equations were not provided until very late in the testing phase. For example the transition matrix from a local level frame of reference to an earth centered inertial frame of reference was only provided late in the second phase of the OTD effort. This resulted in modification of several algorithms in the specifications in order to accommodate the computation of the transformation matrix.
- Some of the fundamental assumptions about the form of the data was very different than in the original requirements. The Navigation History File in the original requirements was to include the coordinate system transformation matrix. This matrix was not provided in the Navigation History File; new requirements had to be added to account for this discrepancy in the data formats. The Prediction, Metric Discrimination and Laplace modules were affected by the new data format.
- Certain simplifications (see section II), which originally were the basis of turning the original requirements document into a viable prototype project, resulted in having to rewrite or derive equations to complete the gap opened by cutting out the algorithm to be simplified. For example the decision not to do the Kalman smoothing precision fit algorithm as part of trajectory fitting required the original specification to be modified to integrate the simplified system. The consequence of this class of modification is that non local areas of the specification were affected, but from an object and behavior point of view, they were functionally connected. To complete the integration of the functional thread all of the different areas effected has to be updated and resolved by the Analyzer.

As was mentioned in section III, the 001 tool suite performs Effects Analysis upon modifications that are performed upon an existing system of 001AXES specifications. This feature is invaluable during the maintenance/alteration of any system, and OTD was no exception.

IV.3 Problems that 001 did not directly cure.

There was a category of problems encountered while developing OTD that 001, for the most part, did not provide direct assistance in solving. It is interesting to contrast these problems against the kinds that 001 does directly address; a given developer, knowing that there cannot be any interface errors in his system, can

concentrate on these other areas instead. In this case, the kinds of problems that 001 does not directly concentrate on can be categorized as "system engineering" problems. Here, we make the distinction of "development" and "system engineering". The act of system engineering (in this context) is really the act of making design *choices*. 001 makes certain your ideas are logically complete and consistent, but it does not presume to think for you. Rather, it supports the thinking process that the designer goes through. Thus, when it comes to problems of this kind, solutions require expertise in the application domain (e.g., physics) itself, instead of expertise in the development technique.⁴ Below is an example list of some system engineering problems that occurred in the OTD development effort.

- Several equations were not provided at all requiring system engineering derivation to complete the requirements. For example in the Laplace algorithm, the least square fit equations were not provided as part of the requirements. The sequent iterate algorithm equations were also not provided as part of the requirements.
- Several constants, tables for bias, and correction factors were not provided at all. For example, the Refraction Correction tables (RR) to compute new value for EL' (elevation), was never provided. The refraction table parameter constants $e1$ and $e2$ needed to compute EL' were never provided. RR contains refraction correction values which are a result of functions on $e1$ and $e2$ as well as the altitude of the object. Another example is that the window size growth parameter in Object Screening was never provided. A trial and error procedure was employed to fine tune the parameters to acceptable values.
- HTI did not have a FORTRAN compiler available to compile the FORTRAN program to recover the raw sensor data. A consequence of this was that the sample true state vector computation FORTRAN algorithm was never usable. Instead, creative substitutions for the verification mechanisms that this program provided had to be engineered.

Section V: Further Evolvment and Conclusions

The current implementation of the OTD system has a variety of options available as probable "next-step" rapid-prototyping evolvments. One important step would be

---

⁴The converse is also true; usage of 001 allows experts in the application domain to no longer have to worry about trusting their ideas to be realized through expertise in an unrelated implementation technique (like FORTRAN). Instead, 001 allows experts, like physicists, to worry only about the problems that they are experts on (like many body interactions).

to verify solutions to the "system engineering" kinds of problems that have already been encountered, correcting any inappropriate assumptions found. Additionally, general categories of implementation choices could be re-examined for possible improvements. For instance, some of the low level objects could have been defined in term of more refined data types. The azimuth (AZ) and elevation (EL) objects could have been defined as data type AZ and EL instead of data type Rational number.

Along these lines, another possibly fruitful improvement may come from reducing the degree of information hiding. The current implementation of the OTD system uses information hiding to various extents. For example, the top most map of the OTD system carries the *Global Data Base* object in and out of the major functions called by OTD. One recommended modification to the top of the system is to make explicit the data flow of certain key objects. The *Predicted Window File*, for example, would flow explicitly from the Prediction function to the Object Screening function where it is needed. The *Tracks* object would also flow in and out of Prediction and in and out of Object Screening. The various message sets (Candidate Track Messages, New Track Messages) would also be made to flow explicitly from the various functions where they are generated to the functions where they are needed. Showing the data flow of these key variables would make the understanding of the system easier. The advantages of hiding the explicit data flow is that a system can be integrated very quickly. This style is very useful when an overview is emphasized of which functions are called and what decisions affect which functions.

Another possible "next-step" involves repeated functional structures. During the modeling phase of OTD several common structural patterns were identified. Those common patterns can be made into 001 AXES Defined Structures. In the Prediction subsystem, several such common patterns have been isolated. The structures in the Prediction subsystem relate to the symmetrical operations performed on the covariance matrix, and would be powerfully expressed in terms of Defined Structures.

One more probable improvement would be a consolidation of the TMap. Several objects which have been defined in the TMap and were needed during an earlier phase are no longer really needed. This consolidation will make understanding the TMap that supports the OTD system more natural.

VI: Retrospective and Conclusions.

Several observations were made during the development of the OTD system. We realized that in the process of developing the OTD system that we made use of aspects of all the major categories of properties of *development before the fact* requirements.

Requirements in this category are implemented in terms of systems which inherently support their own development and operation. These types of requirements produce systems that are more reliable with much higher levels of productivity than are obtained from conventional requirements. All 001 systems are defined and developed with properties of development before the fact. We demonstrated during the process of developing the OTD system that with 001, OTD categories of systems can be defined to:

Capture and Understand Integrated Real-World Definitions. The OTD system was defined in terms of an integration of FMaps and TMaps and with a powerful set of integrated and generic data types provided by the 001 tool suite. The TMap facility alone helped to lead the way in supporting the process in "thinking out" or organizing the system both when the system is understood already or when parts of it are not well understood. Objects were defined to reflect real world OTD definitions such as the abstract types of tracks, initiators and continuators. With this approach, integration happens with the definition, not during implementation and testing as is the case on conventional developments.

Support the generation of reliable and predictable systems. Because the OTD system definition was always defined ultimately in terms of the three primitive control structures of 001, its definition is unambiguous and contains necessary and sufficient information (no more, no less). Interface errors are eliminated (including data flow, priority and timing errors both at the highest and lowest levels of a system). Once we defined and analyzed the OTD system, the 001 analyzer found approximately 75% of the errors that would have still been around at the beginning of performance testing with a conventional system. Some of the other errors which traditionally fall into the user intent area, or behavioral area, were non-existent due to the fact that a set of integrated object oriented types were used to define and develop the OTD system. Still others were found by the tool suite during execution and reported back to the user. For those remaining errors we have found so far, we noted that the tool suite was able to suggest what type most of them were, so that the

"error detection and recovery" process during development was expedient and relatively speaking, obvious. These included errors like forgetting to initialize and values which were out of range. This was a demonstration of approaching many aspects of reliability during the requirements phase instead of during performance testing as is commonly done with conventional developments.

Support Reuse. Once we defined the system, the properties of the 001 definition supported us in finding abstractions and common patterns throughout the system that could be formally defined. The result is that the OTD system could be collapsed into a system definition with much of its detailed information hidden when it is appropriate to do so. It is possible, therefore, to shorten the definition of the system from both a requirements and specification point of view. These patterns fell into four main categories: patterns of initialization, computation, symmetry of data and use of common abstract types. We also noticed that the generic types, with primitives that were designed to have cooperative behavior, accelerated the current definition process at the higher levels for this kind of system. We realized, however, that even higher level types will support an engineer in accelerating this process even more on the next go-around. This was a demonstration of reusability during the requirements phase rather than during implementation or after testing.

Capitalize on parallelism. A 001 AXES definition can define all inherent parallelisms in a system. This provides the basis to judge alternative specification solutions in terms of these inherent parallelisms. We observed that those algorithms which lend themselves to parallelism, including the particular types of categories of parallelism, were formally delineated. No additional work needs to be done to uncover this information "after the fact" after implementation or testing, since it was already uncovered during the requirements specification phase.

Support Flexibility and Handle the Unpredictable. We experimented with change in several categories and were not disappointed. First, we attempted to experiment with many forms of our models both at higher, middle and lower levels of the system. The powerful integration and generic object oriented features of the tool suite allowed us to change the models numerous times and to create prototypes at the rate of our demands. The system will be more flexible to optional resource architectures after we have the opportunity to create higher level types in some areas. The controlled definitions resulting from the use of the 001 technology and from the associated analysis of the analyzer provided us the courage to safely change at will. The system was clearly easy to change and maintain because of these properties. Alterations to the specifications that occurred in the later phases of the project fall into the category of system maintenance, which is one of the most

expensive aspects of standard software development (both in terms of man hours spent completing a given maintenance-related change, and in terms of dollars). The automated Effects Analysis and change propagation facilities of 001 have a profound effect upon this; the usual nightmare of a small change suddenly causing a perfectly running system to fail in thousands of unexpected places (i.e. the usual result of conventional maintenance techniques) is practically eliminated.

Concentrate on Maximizing Automation of the Life Cycle. The OTD system was defined and developed with the 001 tool suite which automated the software development phases as described above. As was demonstrated, automation does not have to wait until after implementation.

Maximize Productivity. Productivity on this project was significantly higher than it would have been with conventional techniques because of the properties which inherently exist in a 001 defined system and the degree to which the user and/or the 001 tool suite takes advantage of these properties.

Support run-time performance analysis. We did not take advantage of the Where Statement capability (the feature in 001 whereby constraints such as execution time and memory on a potential architecture can be defined) on this project, since we concentrated on demonstrating an automated development on a fixed architecture. We noted, however, where we could use this capability in the OTD system. Towards this end, the parallelism property (see above) is highly relevant, however, and this capability was demonstrated on this project. Indirect impacts of other 001 properties were noted during the development of the OTD system that are strongly associated with aspects of run-time performance testing. They include the use of all the properties in the above categories in being able to create designs, scenarios and test cases that are more reliable with significantly more productivity than is the case with conventional techniques. 001 provides the properties which support the design engineer to accelerate and more accurately perform run-time studies.

Conclusion

We exercised the use of many of the development before the fact properties of 001-defined systems when we defined and developed and redeveloped the OTD system. It is for this reason that our productivity in the development of OTD was in a category of its own.

The further development of the OTD system with 001 demonstrated again that 001 is an ideal prototyping and development environment for systems in the OTD class. The next recommended step is to have HTI develop or assist in the development of a set of required SDIO related software based systems with 001.

**DATE
FILMED**

2 / 10 / 94

END