Loads–n-Limits and Release-n-Sequence: The "Brains" behind WEPS

Paul S. Cerkez

DCS Corporation
46641 Corporate Drive
Lexington Park, MD. 20653
pcerkez@dcscorp.com
pcerkez@acm.org

Abstract

The Loads-n-Limits (LNL) and Release-n-Sequence (RNS) modules are rule-based sub-systems designed to validate various configurations of aircraft stores loading and weapons release planning. LNL, in conjunction with verifying and validating the loads presented, returns usage restrictions and limitations of the aircraft and the weapons load. The RNS module evaluates user-selected weapons employment planning. Between the two modules, the planner can create legal and safe loads, determine flight restrictions, create safe weapon release sequences and determine weapons delivery restrictions throughout the entire flight scenario. This paper is a description of the two modules, the development environment, system growth, change management problems, and the scope of changes incorporated between the first release of Weapon Employment Planning Software (WEPS) in 2001 and the latest certified version.

Acronyms and Terms

AI - Artificial Intelligence
ATACS - Automated Tactical Manual Supplement
CLIPS - C Language Inference Production System
E/F – the F/A-18E/F Super Hornet
F-18 – the F/A-18A/B/C/D Hornet
LNL – Loads-n-Limits
MRI - Minimum Release Interval
NATOPS - Naval Aviation Tactical Operations Manual (NATOPS)
NAVAIR - Naval Aviation
RNS – Release-n-Sequence
SEDA - Safe Escape and Delivery Application
SLIC - Stick Length Interactive Calculator
SLIM - Stores Limitations Manual (SLIM)
Store – any item mounted on an aircraft
Suspension - the interface mounting hardware between a store and the aircraft
TACMAN - Tactical Manual (TACMAN)
WEPS -Weapon Employment Planning Software

Background

During the 1990's, a software program called the Automated Tactical Manual Supplement (ATACS) was developed to automate the load planning process of the F/A-18 Hornet (F-18) aircraft used by the US Navy and Marine Corps. The ATACS system was originally DOS-based but was later updated to a Windows environment.

ATACS is based on the F-18 aircraft's Tactical Manual (TACMAN) and Naval Aviation Tactical Operations Manual (NATOPS). The TACMAN is the source document for the authorized stores and load configurations for the aircraft with the associated flight characteristics, limitations, and restrictions. ATACS implements these data as a listing of all store loading configurations authorized on the aircraft. This information is primarily stored in a large FoxPro database. The restrictions associated with the aircraft loads are coded into the application. Flight limitations are also stored in both the database and the system's code. The primary database associated with the system is approximately 8 megabytes in size and there is a very tight coupling of code and data.

The tight coupling between data and code led to maintenance issues. As the capabilities of the F-18 were updated in the TACMAN and NATOPS, changes in authorized load configurations and stores also had to be updated in ATACS. Every time ATACS needed updating, decisions had to be made as to where the change needed to be made: in data, code, or both. As ATACS results are safety-of-flight critical, every time ATACS was updated, the system was recertified for use. Under the ATACS architecture, virtually all changes required re-compiling the entire application. On average, the time it took to implement even a simple change was 7 to 10 days for the software effort alone.

ATACS has proven to be very successful and is highly desired by pilots for mission planning. Without ATACS, a single mission planning session is a couple of hours work consisting of lots of page turning, data look ups, and filling out various forms to facilitate calculations. With ATACS that same mission can be planned in less than 1 hour, typically, 30 minutes or less from start to finish. With the introduction of the F/A-18E/F Super Hornet, (E/F) the desire was to produce a tool functionally similar to ATACS but significantly easier to maintain.

Colloquially, our overriding requirement was to match the TACMAN in all output data. Our tasked objects were simply stated as (Cerkez, 2000):

- For a given load condition, an indicator that flags the load as legal/valid or not
- If illegal/invalid, an indication of the cause of the failure condition(s)
- Platform total weight returned as three data elements: total Platform weight (as loaded), total Stores Weight and Fuel
- Platform wing moments with stores loading
- Individual store drag indices and restrictions as loaded (altitudes, airspeed, G's, etc)
- If legal/valid:
  - total platform restrictions in an as-loaded-condition (altitudes, airspeed, G's, notes, etc)
  - drag indices for the platform will be provided for the four (4) "True Mach Number" data points listed on the "Weaponeering Checklist" for both the DASH and CRUISE conditions.

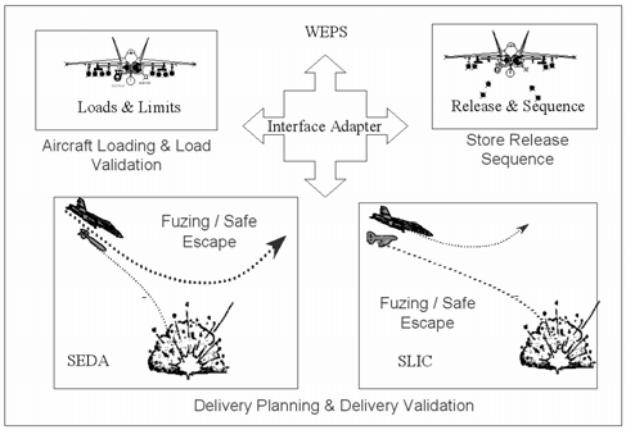

Figure 1: WEPS Components

Problem Description

A significant concern with the introduction of the E/F was the number of updates that would be required to any automated system. Like the F-18, the E/F has the capability to perform both traditional bombing missions and air superiority missions during the same flight sortie, but with two additional stores mounting points. Due to the two additional mounting points, the process for planning store loads is more complex. Since its introduction, the E/F has been undergoing continuous flight testing to expand its operating limits, to validate load configurations and to conduct performance evaluations. From the outset, it was a known development risk that changes to the TACMAN and the type and quantity of authorized loads were going to happen and happen often. In the case of the E/F, the TACMAN was not published until 2002, and was delivered to operational squadrons in mid-2002. Prior to that, an engineering test and evaluation document, the Stores Limitation Manual (SLIM), was used. The SLIM evolved to become the first TACMAN. (From here on, unless specifically called out, the use of the word TACMAN will refer to both the SLIM and TACMAN.)

In 1998, we proposed an AI-based approach to the problem of aircraft load planning to the NAVAIR Ballistics office (AIR 4.11.2). After a number of discussions, it was decided to develop the tool with this concept.

Weapon Employment Planning Software (WEPS) is the stores planning tool for the E/F. The most significant design difference between ATACS and WEPS is the decoupling of data and code. In WEPS, the GUI collects and displays data, with minimal knowledge of load planning or weapons release. The intelligent modules that are part of the application contain all of the pertinent knowledge of the aircraft and the authorized loads. There are basically three sub-divisions in WEPS: (refer to Figure 1) Loads-n-Limits (LNL), Delivery Planning and Validation, (SEDA and SLIC), and finally, Release-n-Sequence (RNS). WEPS versions 1.0 and 1.1, with only LNL and SEDA, were subjected to a very rigorous certification process due to safety-of-flight requirements and were delivered to the fleet in November 2001 and June 2002 in conjunction with the first TACMAN. WEPS 2.0, with an updated LNL, updated SEDA, SLIC, and the new RNS completed safe-for-flight testing in November 2004 and was published to the user base in December 2004.

During the development effort leading to WEPS 1.1, the source baseline data (SLIM and TACMAN) was changed ten times, and subsequent to WEPS 1.1 certification testing, ten more baseline (TACMAN) updates have been released with more pending. From 1998 through 2002, there were eight releases of the SLIM, each either adding or deleting weapons or modifying the restrictions and limitations of the aircraft. From June of 2002 through September 2003, there were four major changes to the TACMAN. From September 2003 to May 2004, there were six more. Comparatively, in the same time frame (1998 –2004), the TACMAN for the F-18, an aircraft in production since the early 1980s, only had two major changes.

LNL is the rules based sub-system that contains all of the rules and facts necessary to both generate and subsequently validate a load configuration. While the numerous changes to the baselines were a frustration, the structure of LNL allowed most changes to be made relatively easily.

RNS is an application developed to explicitly capitalize on the WEPS 1.1 work of LNL and provide the users with additional planning capabilities. RNS is a knowledge-based module that validates store releases from the aircraft. Because of the dynamics of flight and the weight and aerodynamics of stores, the stores have numerous limitations associated with them. RNS addresses these limitations and provides the planner with the final tool to complete mission planning.

The front end of WEPS is a graphical interface that allows the user to interface with the various sub-system modules. The SEDA and SLIC components are software implementations of the physics models used to calculate the flight path of a released weapon, impact points and fragmentation balloons.

Because CLIPS is a script language, a significant advantage of LNL and RNS is that changes do not require recompilation of the WEPS code. De-coupling the load planning and release rules from the user application was a success. LNL and RNS can be run independently of the WEPS GUI application.

What follows is a discussion of the technology of LNL and RNS. While our conceptual approach was simple on the surface, implementation was tedious. We were successful in our approach. Barring major exceptions to the existing rules, it currently takes about 2 weeks to fully implement complete new baseline data into LNL and RNS. The majority of the time is spent analyzing the change itself. To implement the simple change of adding a single new store to LNL is in total, about 1 hour worth of software effort. 'Removing' a store takes less than 5 minutes. Testing however, may take many weeks more. LNL and RNS can be maintained by a single developer with domain knowledge.

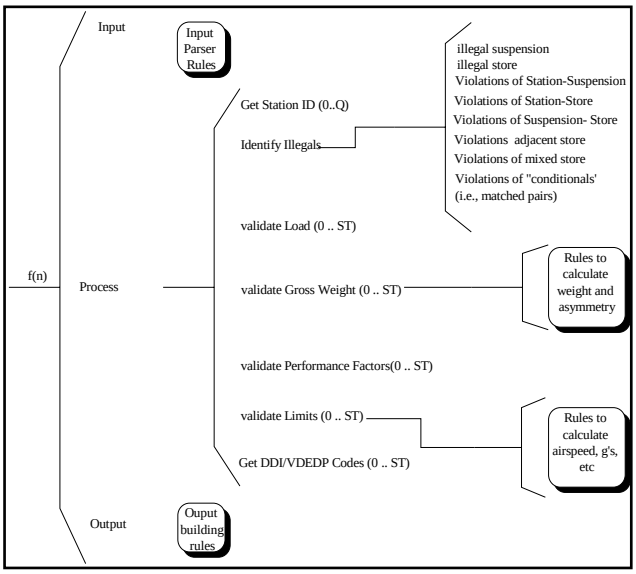

Figure 2: Overview of LNL operations

Our Approach

While ATACS utilized a brute force database of every possible combination of stores and loading locations, the concept behind LNL was to utilize an AI approach that looked at the rules and relationships between the stores and where they are loaded on the aircraft. A number of implementation languages were discussed and CLIPS was finally selected. A detailed analysis of the TACMAN was performed with the express notion of finding patterns in the data. One goal was to abstract the pattern data to the point that any representation chosen for the stores and suspension would be irrelevant. The LNL systems was designed and built around the concept that it did not matter how the stores are represented. LNL only cares about relationships that apply to the store code (more about these codes later). This goal was achieved and the same abstraction was later carried over into RNS.

The next goal was to identify rules that affect the data. This proved to be the most difficult effort of the project. The biggest obstacle to overcome was all of the exceptions or modifications to supposedly standard rules. Many of the exceptions and modifications are store driven while others are driven by load configurations that have been flight tested as safe in only one configuration or had not completed full flight testing. Over the five years of development and maintenance, as flight clearances were approved, many rules were modified or deleted because the earlier condition driving the rule's existence was no longer valid. Figure 2 is an overview of the LNL process developed from the extracted rules used to guide the development effort.

Just the Facts

The Ballistics office produces a standardized representation of all stores, aircraft and suspension items used in Naval Aviation software planning tools, including a representation for no store or suspension. These are referred to as either "Schanck Codes" (after the developer) or simply 'store codes'. From these store codes, a person with knowledge of the code's structure can tell the configuration of the store it represents. For example, a store code of 11201023 represents a Mk-82 500lb bomb with a specific type bomb body, nose fuze and tail fin.

With nothing mounted on the aircraft, the aircraft has baseline performance and flight characteristics. Everything that can be mounted to the wings or body of the aircraft has an impact on airflow over the aircraft. Each item has its own aerodynamic characteristics. As each store and suspension item is mounted on the aircraft the baseline performance and flight limitations are affected. Many limits and restrictions are detailed in the TACMAN's textual loading notes.

The very first configuration management requirement of the system was to apply all notes identified in the various reference manuals and to track any changes to the system via *certifier's notes*. All items in the fact bases in the LNL and RNS systems contain a slot to enter and hold a note identifier and a slot to contain a string representing the certifier's notes. A certifier note typically contains the date, the initials of the person making the change, the source document directing the change and the affected data (ex: 10/2/04 PSC TACMAN IC 5: changed airspeed knots value from …).

As result of the analysis of the TACMAN and other manuals, a basic methodology was laid out. We determined that the entire aircraft load could be represented in its simplest form as a tree. The root node of the tree is the aircraft itself. The first level of the tree's branches represents all of the possible mounting locations on the aircraft. At each of these branches, even on an empty station, there is always a suspension code followed by either another suspension code or a store code. By using this particular structure, the system can easily handle those cases where there may be 3 or more levels of suspension before reaching the store leaf node. Additionally, as flight testing continued, this structure allowed us to adapt to new mounting locations approved for the aircraft. A case in point occurred during the later stages in the development of version 1.0. A requirement was added to account for mounting points internal to the aircraft that support specialized systems. The effort required to accommodate these new stations was to simply give the locations unique identifiers and identify the Schanck codes for the stores.

To support the tree structure approach, we developed a simple method of defining what constituted a legal pairing. To accomplish this we defined and developed the *relations* file. This is a CLIPS fact file containing all applicable relationships between any one item and another. The fact `(relation [A8234900] [S11201231])` states that store `[S11201231]` can be mounted on the suspension item `[A8234900]`. Because this simple approach can allow unwanted mountings, a second part of this methodology incorporated another fact file defining explicitly illegal mounting locations. The advantage of this approach was it enabled the system to be quickly modified by simply removing the illegal condition fact as new mountings became authorized. Also, by simply adding a fact to the illegal fact base, a previously approved mounting could be removed if necessary.

Now that the most basic part of the system had been described, the next effort was to extract default data about each item represented in the load. To support this, three separate fact bases were developed: PLATFORM, STORES, and SUSPENSION. Each of these fact bases contains simple default information about each item. All three contain the item's weight, nomenclature, Schanck Code, note identifiers and a certifier note. To support these default fact bases, a CLIPS super class structure was defined. Each subclass (platform, store, and suspension) inherited the basic slots that applied across all items. Following this, in LNL we created three additional fact bases containing load specific data such as airspeed limits, g-limits, and drag data. In RNS we created additional fact bases to hold release data and sequencing data,

The next two fact bases generated were in support of *mixed loading*. The first of these fact bases contains the information necessary to define authorized aircraft loads. Even though a suspension combination is authorized on the aircraft, there are combinations of stores not authorized on each wing. Additionally, there are combinations of left wing and right wing loadings that are explicitly authorized or not authorized. Furthermore, there are wing combinations that are authorized but are illegal in combination with certain fuselage station loads. This mixed load fact base contains all of the authorized loading patterns and explicitly illegal combinations. With each entry, there are the associated notes. The second fact base supplies the unique notes when specific combinations of wing loadings are evaluated.

A significant advantage of this approach is that it allowed for temporary flagging of certain loads as illegal. We accomplished this by creating a special note called a User Defined Error (UDE) note. By simply replacing the existing legal notes with a UDE note, the load configuration in question is made invalid and a detailed reason is returned. To reverse the flag, we simply remove the UDE note and the system returns to the way it was. This approach also allowed us to pass dynamic data through the GUI layer to the physics models by creating special performance notes.

Finally, the last set of facts stored in the system is the drag calculation factors interpolated from NATOPS flight performance graphs for drag. These facts are used to dynamically determine the overall drag effects on the aircraft.

Following the Rules

As stated earlier, determining the rules was the most troublesome effort on this project. The system was set up in a heuristically guided building block structure. As mentioned earlier, the E/F is a very robust weapon platform capable of carrying a vast array of stores. Weapons range from precision guided stores to missiles to general purpose iron bombs. The Navy's Office Of Information web site shows a sampling of the stores used by the E/F. (Navy) As with all US Navy aircraft, the TACMAN is the authoritative source describing what stores can be loaded where on the aircraft and in what quantities. These descriptions constitute the load patterns LNL must evaluate.

The flip side of the coin is the ability to deliver stores on target. The E/F is a flying computer controlled by an operating system developed by the aircraft manufacturer. Each operating system has its own set of rules regarding stores deliveries and has an associated manual detailing the rules of operation. (Greybook) RNS is constructed from these rules.

To evaluate a load for both loading and delivery, the approach we took was to start with the simplest conditions, validate them then move on to the more complex. Simply put, if the load can't pass the simple test, there is no need to check the more complex.

The first set of rules applied to a load pattern was to determine if the load exceeded any maximum parameters for the aircraft. After some housekeeping rules to check that the input load pattern is properly formatted, the first test is for maximum weight. The second is for asymmetry. If either of these parameter limits are exceed, the

load evaluation halts and an error message is returned to the calling system (WEPS) for display to the user.

Upon successful completion of the initial testing, LNL checks to ensure a legal wing loading exists followed by legal combinations of wings and fuselage loads. If the entire load combination is legal, notes applicable to the load are extracted and stored for final return to the calling routine. Once the load is validated as legal, airspeed and g-limits are then calculated followed by the drag data calculations. With both airspeed and g-limits, the output is not necessarily a simple retrieval from the appropriate fact base. It is an aggregation of the limits of the basic aircraft and all items loaded. For example, the limits from one wing's store combination can impose more restrictive limits on the rest of the aircraft. Additionally, in many cases, the presence of a single store will impact all other stores on the aircraft. However, a problem associated with calculating the limits is that some limits are published as numeric values (i.e., -1.9, 6.3, 575, etc) while others are mnemonic (i.e., P, NA, LON, etc.). To support the safe for flight requirement, the most restrictive is always used and so a function (Figure 3) was created that compared the numeric values and the mnemonics to determine the most restrictive limit.

```
(deffunction LNL::calc_as-limit (?IN1 ?IN2 )
;;; First check to see if IN1 is LON and IN2 is NOT LON, NA or P.
    (if (and
            (or
              (member$ NA ?IN1 )
              (member$ P ?IN1 )
            )
            (not (subsetp (create$ P NA) ?IN2 ))
             ) then
               (bind ?OUT ?IN2)
            else
              (if (or
                    (member$ NA ?IN2)
                    (member$ P ?IN2)
                ) then
                    (bind ?OUT ?IN2)
                else
                    (bind ?t1_1 (nth$ 1 ?IN1))
                    (bind ?t2_1 (nth$ 1 ?IN2))
                    (bind ?t1_2 (nth$ 2 ?IN1))
                    (bind ?t2_2 (nth$ 2 ?IN2))
                          ; returns the lower as limit
                    (bind ?out_1 (min_as ?t1_1 ?t2_1))
                          ; returns the upper as limit
                    (bind ?out_2 (max_as ?t1_2 ?t2_2))
                    (bind ?OUT (create$ ?out_1 ?out_2))
              );end if
            ) ; end if
     (return ?OUT)
) ; END deffunction calc_as-limit
```

Figure 3: Calculate Air Speed Limit Function

Finally, the LNL subsystem evaluates the MRI. The MRI refers to the authorized time increment for a pilot to release the stores from the aircraft. In order to return the proper release notes associated with the load, LNL performs a very rudimentary evaluation of the MRI data. This represents a slight crossover of functionality between LNL and RNS. A second cross over with RNS is the evaluation of the load with regard to proper downloading (i.e., removal of stores). The E/F drops its weapons in specific sequences based on the store and suspension. In order for RNS to perform its functions, LNL iteratively validates the load as the stores are released. Also, because the aircraft can fly in a 'downloaded' configuration, LNL validates these loads to ensure that the downloaded configuration is not an invalid condition.

When it is time to plan a stores delivery, the planner selects a store from the ones loaded and queries the system for the authorized quantities, multiples and priority stations. RNS is the engine that determines these outputs. RNS was developed separately from LNL but with many common concepts. RNS provides more detailed information and specific limitations related to releasing stores from the aircraft. The primary complexity of RNS was the source data. For any given legal load configuration, there is only one proper sequence as dictated by the rules in the Greybook. For a selected quantity, there are an approved number of stores (the multiple) that can be released in a single pulse. Then, based on the load configuration, a designated station to start release pulses from is determined. Included in the calculations are the final MRI determinations.

Because the E/F is so capable, there exists load configurations that can be supported, but because they have not been approved yet they must be disallowed by the system. Fortunately, the existing rules cover most situations. However, we did have to create a specific set of rules and facts to handle a peculiar condition. Any store mounted on the aircraft has an associated code used by the aircraft's armament computer to support releasing of the weapons as required. The problem is that many stores within a class can have the same aircraft armament code but cannot be on the aircraft together. The problem deals with store compatibility. An example is the Mk-82 (500 lb) bomb series. In the case of the Mk-82, there are about 120 different Schanck codes for the Mk-82 class stores to define different variations of the bomb, fin and fuzes. Of these 120 store codes, all have the same aircraft armament code. However, some are considered compatible with each other while others are not. An example is a live bomb and a practice bomb (inert). As far as the aircraft is concerned these are the same bomb. The only difference is one that does not explode on impact. Among other attributes, WEPS was designed to differentiate live and inert bombs.

To solve this problem, we brainstormed with the domain experts and came up with a mapping technique where we could identify in data what constituted compatible stores. All Schanck codes have a well defined structure so we decided to capitalize on those definitions. We developed a simple mapping of codes using 2 tokens (= and ~) and the list processing capability of CLIPS. We created a class instance for each type of store that we

called the STORE_MASK. Every store in the system is assigned a STORE_MASK key (e.g., k112). Within each STORE_MASK we defined the positional compatibility of each store code. For example, for the Mk-82 series, we used the following:

```
([k112] of STORE_MASK
       (pos1 "=")
       (pos2 "=")
       (pos3 "=")
       (pos4 "~"); don't care
       (pos5 "=")
       (pos6 "=")
       (pos7 "=")
       (pos8 "="))
```

All Mk-82's stores supported in LNL and RNS share this key. This example shows that for Mk-82 stores, all positions of the store codes being compared must be the same EXCEPT position 4 which is a *don't care*. So long as all 7 of the other positions are the same, the two store codes being compared are compatible.

For Mk-84's (2000 lb) however, the following mask exists for position 4:

```
(pos4 "1 5 6 9 A | 2 3 4 7 8");
```

In this case, we have two sets of compatible codes for position 4. A code for one set is NOT compatible with one from another, but within the set, all are compatible. This methodology allowed us to handle other store classes were we had to support 3 and 4 sets of equivalences. As flight testing progressed and more clearances were approved, compatibilities were updated. Utilizing this approach, we could change the compatibility of stores in the system via data without the need to re-build a new WEPS release. More importantly, there were no software code changes. The entire compatibility test for 674 store codes consists of a single rule, 3 functions and 30 STORE_MASK instances.

This approach proved it worth when two stores of the same class were previously considered non-compatible. In the later stages of the WEPS version 2.0 development, these two stores were now considered compatible. To implement the change, we simply adjusted the appropriate tokens in the applicable STORE_MASK and in less than 5 minutes the change was effected.

Talking the Talk

The LNL effort preceded the WEPS GUI effort, so LNL had to establish an interface protocol that any calling language could use. At the time, the decision for the WEPS GUI implementation language was still being studied and choices included Visual Basic, C, C++, and compiled FoxPro. The final decision was C++ as the development language. To allow LNL development to begin and to establish a universal parameter passing methodology a tokenized string structure was developed for passing all data in and out of the LNL subsystem. An implementation specific code block (class, procedure, etc) to build the data string going into LNL and parse the returning data string coming from LNL was the only requirement of the calling application. Typically, the string representing the loads going into LNL are about 1k long, however the return strings coming out of LNL are typically 10k or more. This tokenized string with the addition of four new tokens was later used in the RNS subsystem.

Keeping up to Date

The dynamics of the changes to the TACMAN and other source material for the LNL and RNS subsystems required close monitoring of the source data. Except for the MS-Excel spreadsheet containing the Schanck codes, all source material available was either a domain expert's input, a paper copy or a PDF file of the reference document. After surviving a few valuable hard learned lessons in the development efforts leading to WEPS 1.1, we initiated a process that allowed us to identify each data element change between versions of the source material, primarily the TACMAN. The procedure was time consuming but simple: a side by side, page by page comparison of the old document and the one replacing it. Each data element that changed in the new document was circled in red. As part of this procedure, we identified the impact of the changes to the systems. There were cases where a simple one word text change in a loading note affected every single load on the aircraft. Fortunately, in most cases changes were simply revising a limit or removing a restriction.

The second step in the process was to work through each fact base with the marked-up TACMAN and to make each required change to the appropriate individual facts. In support of those cases where an entirely new store or a new set of load patterns was being added to the system, we developed a series of flow charts and step by step procedures to walk a maintainer through the various fact files in the proper order. When the procedures were not followed, there was a risk of missing a data point.

The third step was the most troublesome: implementing new rules or modifying existing ones. The difficulty here was determining side effect interactions with other existing rules.

The problem with the third step is best illustrated by a situation in the RNS rules to determine the proper release sequence for a set of stores given a change in restrictions. Prior to the change, the rules and facts worked cleanly. When the restrictions for a new store were added, it caused unintended conflicts with other rules and resulted in wrong outputs. After significant analysis and effort (almost 70 hours), the whole section of 31 rules and 125 facts was removed. It was replaced by 3 new rules and 634 new facts that only took 12 hours to generate. The new implementation eliminated the possibility of interactions and was significantly easier to maintain. A positive side effect was a speed up in performance.

The WEPS software program is a certified safety of flight application and as such there is a traceability requirement to source documents. To that end, all changes

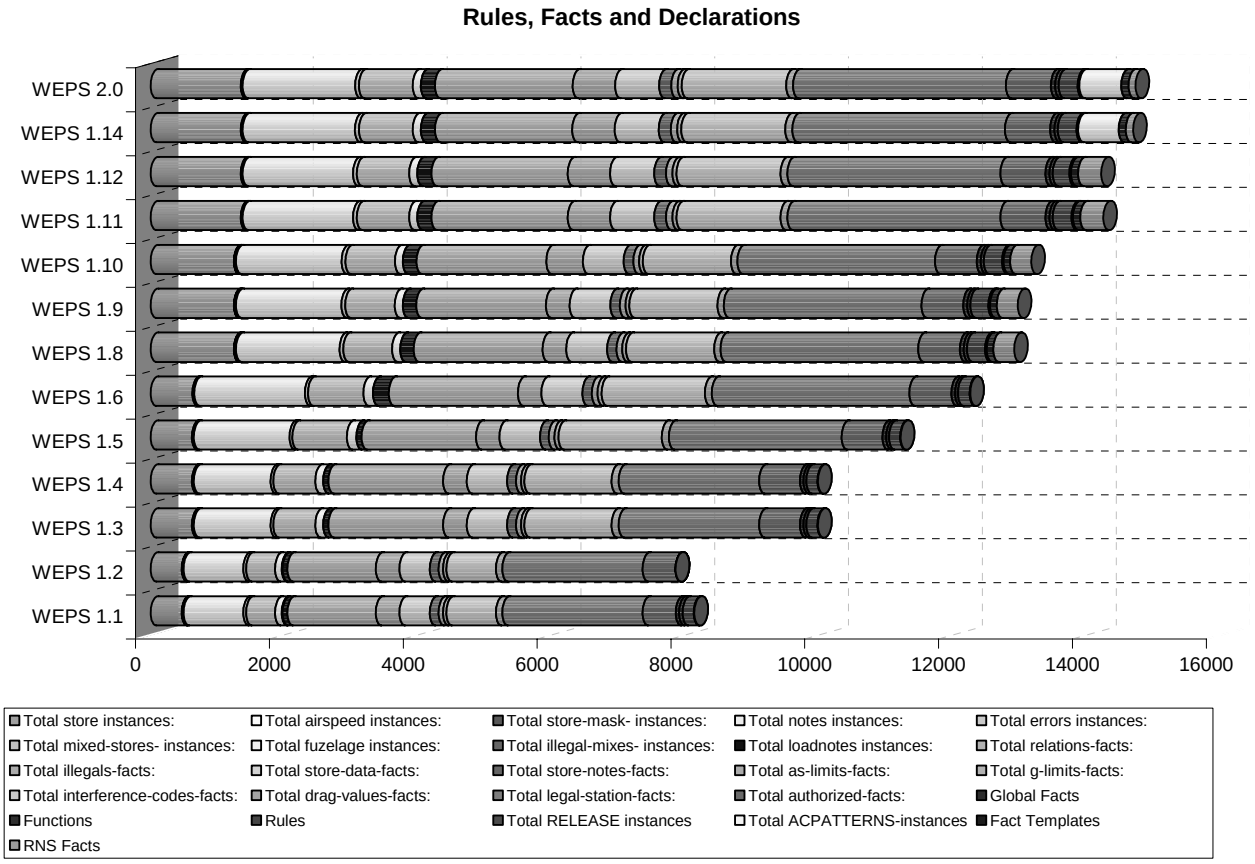

Figure 4: LNL and RNS Rules, Facts and Declarations Growth

were initiated, recorded, prioritized, coordinated, directed, implemented and documented per a formal configuration management program managed by the Ballistics Office.

Today

As of January 2005, WEPS 1.1 had been in use for approximately 2.5 years. Since its release some of the changes released in the TACMAN have had major impacts on the applicability of WEPS for the users. Most of the TACMAN changes resulted in many new stores being added, many new load patterns and expanded performance envelops. If an approved load configuration was not already in WEPS, the planning had to be done manually. E/F pilots enjoy the ease WEPS affords when planning compared to the manual method. The Ballistics office was constantly being asked for the next WEPS release so the pilots could do all planning work in the software. Figure 4 is a graphic summary of the additions to the system made in development builds between WEPS 1.1 and WEPS 2.0. The chart only shows those areas were there were changes. Note there is only partial data for WEPS 1.2. WEPS 1.7 was the first version to have RNS included but was not distributed so its data is excluded from the chart. WEPS 1.8 is essentially a distributed version 1.7 with some additions.

The initial goal to decouple the technical knowledge of the aircraft loads from the user-level application was a success. 99% of the knowledge related to validating an aircraft load and delivery planning is currently contained in the LNL and RNS subsystems. The simple, abstracted approach we used facilitates the rapid changes needed to support the E/F as more stores become authorized for the aircraft and the limits of the weapons become clarified. WEPS 2.0 is expected to be used by Super Hornet community for at least 2 years until it is replaced by a common aircraft mission planning tool.

Acknowledgments

This material is based upon work performed for Ballistics Office, Naval Aircraft Weapons Center, Aircraft Division, (NAWC-AD) Patuxent River, Maryland under Contract No. N00421-02-D-3176.

Figure 1: Permission to publish granted by K. Schanck, WEPS Project Manager, NAVAIR Ballistics Office.

References

Cerkez, P.S. June 2, 2000, System Design Specification to Aircraft Loads and Limits Module. DCS Corporation, Lexington Park MD 20653.

Greybook. MDC B 1984-13E. Operation of the F/A-18 Avionic Subsystem for F/A-18E/F Aircraft with 13E Operational Flight Programs. Boeing Aircraft Corporation.

NATOPS. A1-F18EA-NFM-000. NATOPS Flight Manual F/A-18E/F 165533 And Up Aircraft. CHIEF OF NAVAL OPERATIONS (N880). WASHINGTON , D.C. 20350-2000.

Navy.
http://www.chinfo.navy.mil/navpalib/aircraft/fa18/fa18ord.html

SLIM
Stores Limitations Manual for the Navy Model F/A-18EF

TACMAN. F/A-18E/F Tactical Manual. NWP 3-22.5-F/A18E/F Volume IV A1-F18EA-TAC-020. Chief of Naval Operations (N880). Washington , D.C. 20350-2000