Developing a Command and Control Application: Lessons Learned from FLEX

Albert G. Frantz, Howard T. Waller, Capt.,USAF, Patrick J. O'Neill,
Thomas A. Clark, David L. Marks, Robert M. Flo, David A. Griffith
Air Force Research Laboratory (AFRL)
Information Directorate (IF)
Information Systems Division (IFS)
Systems Concepts and Applications Branch (IFSA)
FLEX Program Management Office (FLEXPMO)

AFRL/IFSA/FLEXPMO
525 Brooks Road
Rome, NY 13441-4505

Phone: (315) 330-7764
FAX: (315) 330-2807

Email: <frantza, clarkt, wallerh, oneillp, griffithd, marksd, flor>@rl.af.mil

Abstract

This paper documents the lessons learned from the development of a command and control (C2) application, the Force Level EXecution (FLEX) software system. FLEX is the battle management software for monitoring the execution of Joint Air Operations in the Air Force Electronic Systems Center's Theater Battle Management Core Systems (TBMCS). FLEX, like other large software applications, experienced many well-known problems in software engineering such as requirements creep and early lack of domain expertise. FLEX also experienced integration pains as it relied on an evolving TBMCS data and service infrastructure for basic operations. This infrastructure supported migration from a traditional "stove-piped" system to a layered architecture. We will discuss these common problems, as well as FLEX-specific problems, and make recommendations to help ongoing and future software developments mitigate some of the same risks.

While targeted mainly for the DoD C2 development community, the lessons learned herein should also apply to the large-scale software development in industry. This paper starts with a discussion of the early years of FLEX, moves into the DoD mandated the use of Ada, and works its way to the current modern, lightweight, distributed, web-based and segmented FLEX. Architecture.

1 Introduction

Airpower modernization is moving ahead at a very rapid pace. In many instances, precision delivery of a single laser guided munition can do more effective damage than an entire wing of Vietnam-era bombers. Even so, the planners who task and direct those high-tech air resources still follow the same fundamental air battle management cycles as their Vietnam –era counterparts: 1) Target Selection, 2) Air Battle Plan (ABP) Generation, 3) ABP Execution and, 4) Battle Damage Assessment. While modern warfare is rapidly pushing this cycle to be done faster and faster, current systems and processes have not allowed operators to keep pace. This is particularly true during the execution phase of the cycle – the phase in which FLEX provides execution management support. FLEX operates as a distributed capability in the Air Operations Center (AOC) to provide direct support for the Combat Operations Division (COD) and other remote and unit level sites. This architecture and configuration allows direct support for the Air Force's centralized planning and decentralized execution concept of operations. FLEX also represents the very minimal in execution monitoring distributed architecture required to support the near real time operations of the Air Expeditionary Force concept currently being advanced in the Air Force.

FLEX was designed to operate in a highly dynamic and stressful environment, so it was fitting that its development phase would mimic its potential deployment environment. As FLEX is now nearing the completion of its development and integration phase we considered it important to document some of the lessons learned. FLEX software will become operational in 3Q CY99. The first TBMCS Joint Field Acceptance Testing (JFAT in Feb. 1999), including FLEX, was deemed a failure. Fielding decisions will depend upon additional test and the second JFAT.

1.1 *Background*

The Air Operations Center (AOC) operates as the operations command center for all joint (multi-service or combined multinational) air operations. As such, it is responsible for preparing, issuing, and managing detailed and coordinated ATOs, which may be adjusted as necessary to achieve the objectives and strategy of the Joint Force Air Component Commander (JFACC). The AOC also develops, maintains, and disseminates enemy force order of battle, threat information, target definition and weaponeering. To accomplish this role the AOC is organized into multiple divisions; those most related to FLEX are the Combat Intelligence Division (CID), Combat Plans Division (CPD) and COD. The CID and CPD primarily plan and develop the ATO for the next day's war. Once completed and approved, the ATO is disseminated for execution. From this point, the COD is responsible for the centralized tasking, monitoring and execution of that ATO. The FLEX system supports the COD personnel by providing the tools to monitor and track events, generate and distribute alerts and generate summary reports.

The FLEX system is an integrated, force level[1], air task management system that assists the Combat Operations Division in validating a pending ATO and monitoring the status of an executing Air Tasking Order (ATO). The ATO is the set of information in United States Message

[1] The control level as opposed to the unit or operating execution level.

Text Format (USMTF) that tasks the flying units for combat and combat-support missions, and maintains certain aircraft/aircrews at specified states of alert. FLEX is a component of the Execution Management (EM) Computer Software Configuration Item (CSCI) of the TBMCS, a hardware/software system composed of a variable number of local and geographically remote workstations which are networked together to support the wartime functions of the AOC.

1.2 *History*

The FLEX history begins in 1990 at the Air Force Research Laboratory, Rome. The existing state of practice in the AOC was the Computer Aided Force Management System (CAFMS) program, a textual form based interface to a central database. After success in demonstrating technology support for the Combat Plans Division of the Air Operations Center (AOC), an in-house program was started at Rome to demonstrate technology that could support the execution or COD side of the AOC.

The in-house program was the FLEX Advanced Technology Transition Demonstration (FLEX ATTD) which was supported with a contract to then Unisys at Eagan, MN. The Eagan team provided support for the reuse of code from the Advanced Planning System (APS) and changes to the database. APS was subsequently fielded and has evolved into the Theater Air Planner in the current TBMCS. The ATTD was based on three rapid prototypes with demonstrations and user feed back sessions to evaluate the results and make additional recommendations. The user feedback sessions were usually several days long and had up to 35 representatives of the Air Combat Command and it's combatant or numbered Air Force components mixing with the prototype developers. In the process, the in-house team became the domain as well as technology experts in air combat execution C2 automation.

The FLEX operational contract was awarded in 1994 to Logicon, Information Technology Group, in San Pedro, CA. Theater Battle Management Core Systems started in 1996 with Lockheed Martin in Colorado Springs, CO as the integration contractor. FLEX officially became part of TBMCS. In 1996 the coordinated retasking task in FLEX, as demonstrated in the ATTD, was taken from FLEX and became part of the Lockheed Martin Execution Management Replanning (EMR) program as part of TBMCS. The most recent major events were the trial of FLEX at the EFX Expeditionary Force Experiment in September 1998 and the TBMCS Joint Field Acceptance Testing (JFAT) in March of 1999. FLEX performance was operationally unacceptable at both events. However, FLEX's performance problems did not arise solely from software coding or execution problems. For example, while waiting for FLEX, users would start additional processes to occupy themselves and thereby exacerbating the problem of slow performance. ORDBEX (a commercial procedural database replication capability used in TBMCS) to be covered in detail in section 3.2 was also a significant problem to FLEX during the JFAT.

1.3 *Software Engineering*

"Software Engineering is concerned with software built by teams rather than individuals and uses engineering principles in the development of these systems and includes both technical and non-technical aspects" [Sommerville, 1992]. Software engineering factors played a significant role in the development of FLEX. Well-engineered software should be maintainable, reliable, efficient, and appropriate to the user. The most basic software engineering process involves five steps:

1. Requirements analysis and definition;
2. System and software design;
3. Implementation and unit testing;
4. Integration and system testing; and
5. Maintenance and upgrading;

The software process model used in FLEX would best be described as an Evolutionary Prototype approach. We will try to take a software engineering perspective as much as possible to the material presented in this paper.

1.4 *Organization of this Paper*

FLEX currently represents the state-of-the-art in operational distributed C2 software systems within the DoD; therefore, much of this paper will describe what makes the system and development unique. An overview of the FLEX functionality and architecture will be given in section two as a basis for understanding the lessons learned in section three and the conclusions drawn in section four.

2 **Overview of FLEX**

The Force Level Execution (FLEX) program objective is to develop software that will support the joint services ability to prosecute effective C2 of theater air resources during joint operations. FLEX will be used by Duty Officers in the AOC to monitor and manage those air resources being tasked by the Joint ABP during ABP execution. These activities will be automated using FLEX and other Theater Battle Management Core Systems (TBMCS) software applications, and improve upon current manual processes. Figure 1 represents the TBM process and some of the applications within TBMCS to support this process. Note that, FLEX was designed to support the execution management of the ABP. Current FLEX automated capabilities could substantially increase the timeliness of decision-making and help to maximize joint air operations effectiveness and efficiency during execution. When a problem arises during execution, i.e. an airbase is attacked/shut down or critical/pop-up targets arise, FLEX users can detect potential plan conflicts. This gives users an edge in preparing Air Battle Plan (ABP) modifications, which require the retasking of resources. Currently, FLEX does not recommend possible plan change alternatives, but will run complex constraints to determine the feasibility of proposed user changes.

FLEX's operational focus is the AOC where execution monitoring and control occur (Figure 2 shows where FLEX operates). Since FLEX is a joint program, the Navy and Marines also have FLEX clients. The Wing Operations Center (WOC) and Squadrons input the status of the

operations as they occur. FLEX is also present at the Control Reporting Center (CRC), the Army's Air Support Operations Command (ASOC) and, potentially, in airborne C2 aircraft.

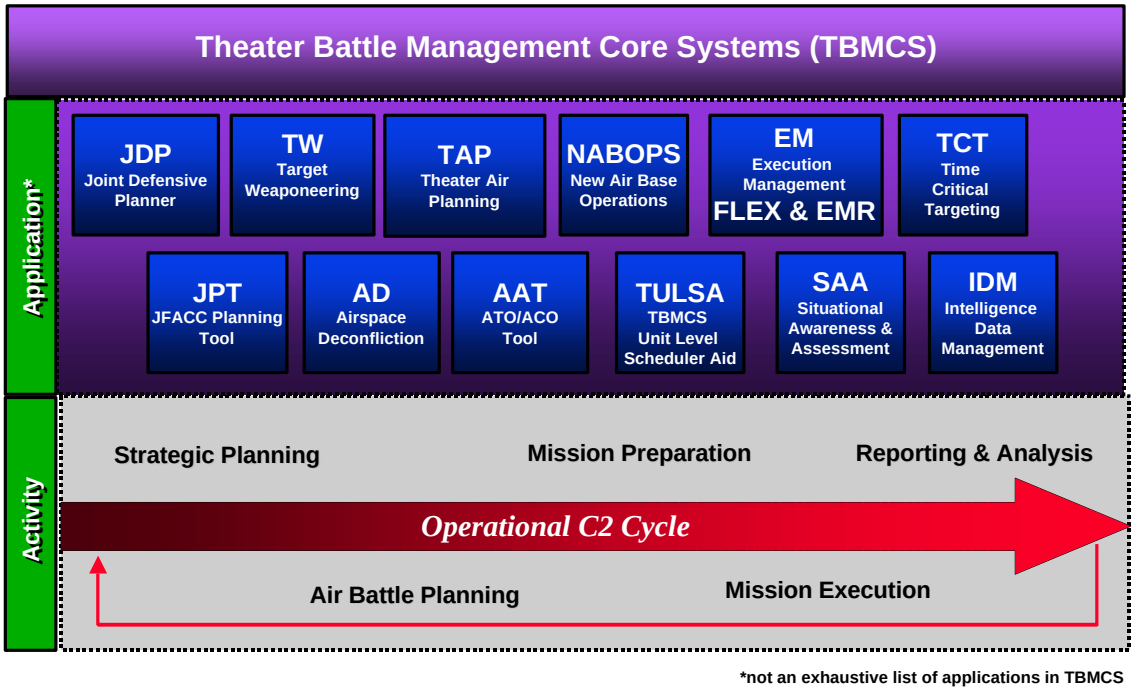

Figure 1 Theater Battle Management Core Systems

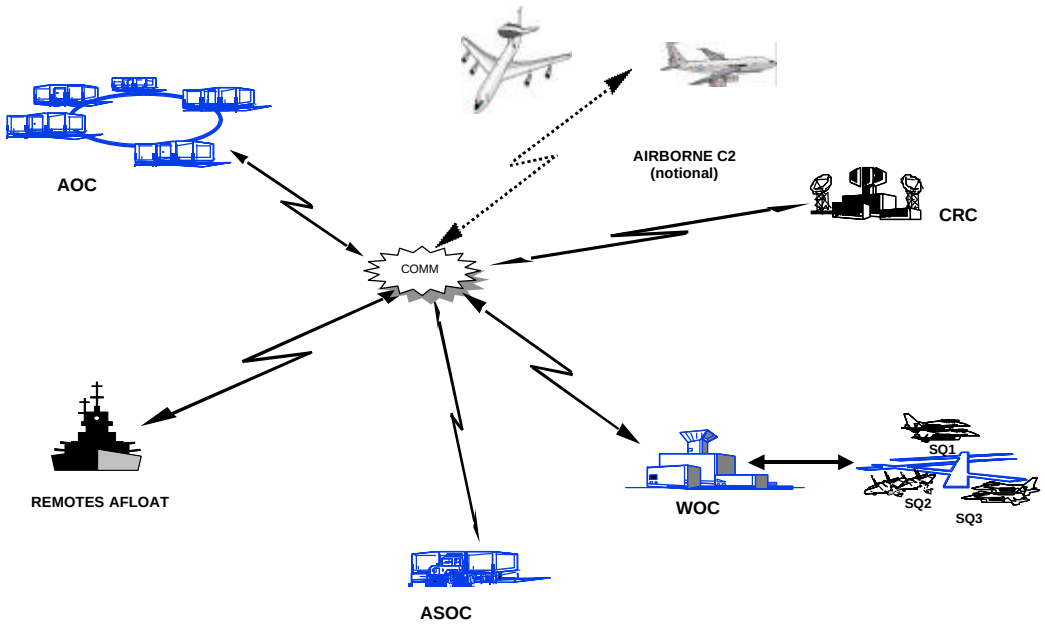

Figure 2 FLEX Sites of Operation

The development strategy for FLEX exercises an evolutionary prototyping process by which multiple builds of software are developed, with each successive release containing increased

functionality. A dual path approach is employed to develop both software functionality and interfaces. Users are involved in the beginning of the process to define requirements and continue to participate through review and critique of software prototypes. This process allows for a low-risk maturation of the operational requirements over the development life cycle.

FLEX was developed using a layered/object-oriented methodology and operates using the Common Object Request Broker Architecture (CORBA). This gives FLEX the ability to be distributed over a LAN (local area network) and WAN (wide area network) simultaneously and allows multiple users to share the same data. This capability allows AOC personnel to be physically disparate (i.e. moving closer to a limited footprint in theater with most personnel in CONUS), and provides an initial capability for collaborative replanning. FLEX relies heavily on a graphical user interface to communicate information in a time-constrained environment. FLEX is DII COE (Defense Information Infrastructure Common Operating Environment) compliant, fully segmented into the TBMCS system and operates using a single enterprise database developed for use by all TBMCS applications.

2.1 *FLEX Configuration*

Based on the current USAF and AC2ISRC visions, distributed systems will be the future for Command and Control software. FLEX is a true distributed client/server system, as shown in Figure 3, and may become the first operational C2 application resulting from Research and Development efforts at AFRL in Rome. FLEX runs on Sun computers using Solaris 2.5.1 operating system and Hewlett Packard platforms running HP-UX 10-20. As a set of DII COE segments, FLEX can be installed as both client and server segments, depending on the local requirements.

The FLEX clients create a local object store from data received from the FLEX database server, which in turn, receives information from the TBMCS Air Operations Database (AODB). The AODB is a common, enterprise database that services multiple applications including FLEX clients. The object store is based on the Common Object Request Broker Architecture (CORBA) compliant Interface Definition Language (IDL) using the commercially available product, Orbix to interface to Ada, and maintains all the information needed locally by the client. Changes on either the client or the server side are propagated to the correct location by the FLEX database monitor. Because of its distributed nature, operation can continue on the client even if the network goes down as long as only the local cached information is required. Updates will automatically propagate to the local cache from the FLEX server once network (or satellite) communication is restored. Currently client updates cannot be queued for entry into the server, but the architecture was designed to support this capability.

2.2 *The FLEX Graphical User Interface*

One of the easiest ways to understand the FLEX application and its use is to look at its graphical user interface (GUI). The interface is primarily based on Motif-windows with data entry and typically tabular display. The FLEX reports are based on a web-to-database query capability.

The FLEX reports are shown in Figure 3 as an arrow directly from the database cylinder to the clients.

Interfacing the Ada applications with Motif bindings produces the graphical user interface (GUI) in Motif. The GUI was developed based on years of working with FLEX users and is very user friendly showing extremely complex information concisely in its GUI.

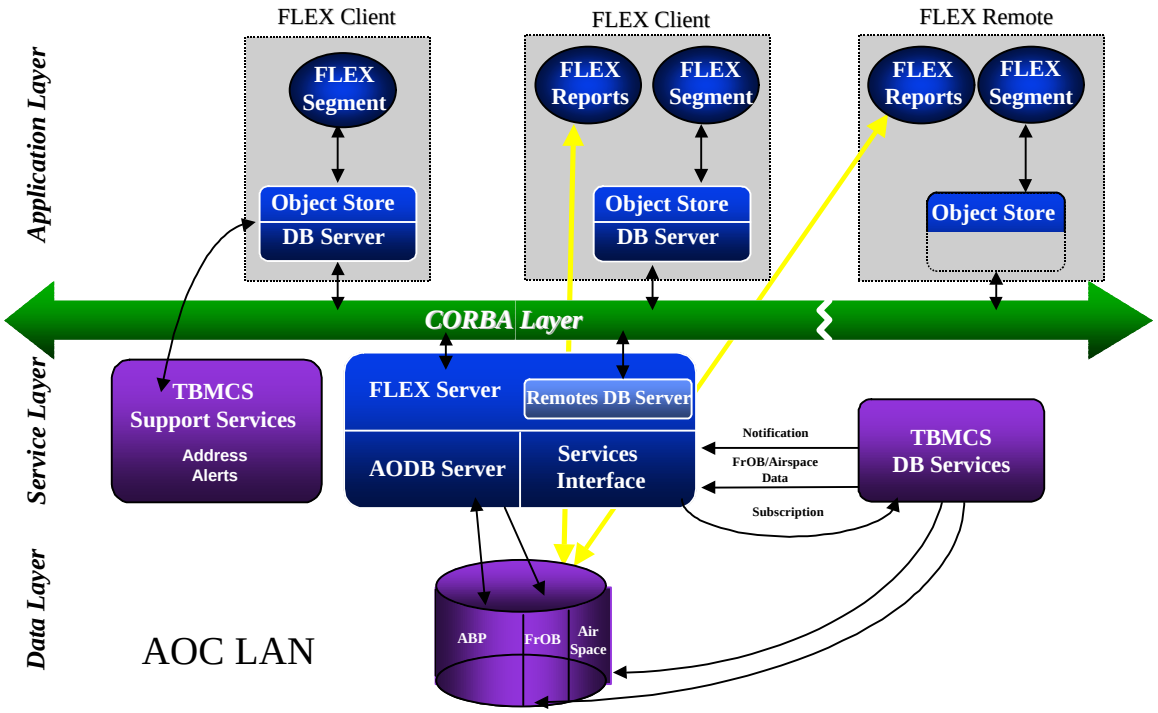

Figure 3 The FLEX Architecture

2.2.1 *The FLEX Marquee*

Much of the following information is taken from the FLEX Software Users Manual still in draft form. Figure 4 shows the FLEX Marquee, the flagship of the FLEX GUI.[2] Not only does the Marquee show tabular formats like other FLEX status display windows, it also presents missions graphically. The Marquee is the result of many person-months of mapping requirements analysis between the many FLEX users and developers to an efficient meaningful graphical display. If used correctly, the Marquee Display is a powerful tool for monitoring the ABP. The graphical format allows easy understanding of missions and their status and relationships to each other.

At the top of the Marquee is a title bar, then a menu bar followed by a tool bar. The tool bar does common functions including complex sorting, column and row display control of the missions on the display and queries for the missions to display. On the Marquee graphical display, a time

[2] The GUI figures are screen dumps from a Sun running FLEX. Much of the detail and resolution is lost in the process of going to PC graphics.

line is drawn across the top of the area. The time line divides the graphical area into time-periods (denoted by vertical hash marks), with daylight hours in yellow and nighttime hours in black. Each time-period is labeled with the time format DDHHMMZ. A vertical red line through the display indicates the current time; this line automatically moves with the clock. The time scale of the time line can be adjusted to focus on certain periods of the ABP.

Each row in the table portion of the Marquee window represents a separate mission. Every major event in the mission is represented by a pictogram. Pictograms always have the same shape, but vary in position and color to indicate a change to the mission's time line. In the first column, the user can click on the plus to show all the sorties involved in that mission as well as additional details about the mission (like multiple objectives). Knowing the meaning of the pictograms and colors allows the user to understand a mission at a glance. The first two and last three missions on Figure 1 are close air support (CAS) missions. The remaining missions are aerial refueling (AR) tankers.

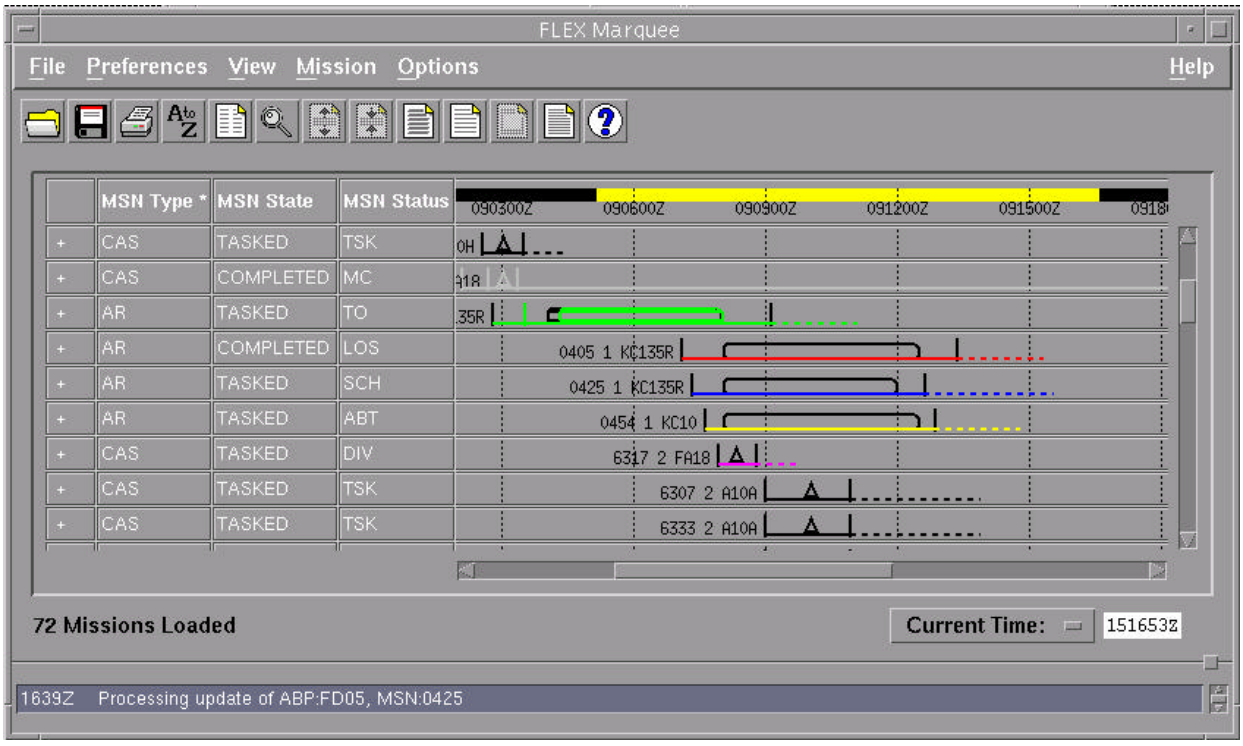

Figure 4 The Flex Marquee

3 **FLEX Lessons Learned**

The lessons learned from FLEX could probably fill a large book; so, in this section we will try to cover the key lessons learned, give some opinions and go into limited details about some of the problems and good things encountered in FLEX. In particular, the lessons apply to software

designed to operate over low-bandwidth satellite communications links or for those applications relying on the World Wide Web to move relatively large amounts of “real” data.

3.1 *Domain Expertise*

An essential element required for the successful development of operational software is experienced software developers who understand the problem domain. Although the FLEX contractor wrote an excellent proposal, the people assigned as developers did not possess sufficient domain expertise. Although Government engineers could provide significant insight, a more continuing input was required at the developers location. After much discussion, the contractor located and hired a former military planner to advise them, thus ensuring that developers would no longer be forced to make educated guesses on the implementation of requirements.

3.1.1 *User Sessions*

To enhance the on-site expertise, a strong user involvement was put in place. Excluding the last couple of years of the FLEX nine year program, periodic user sessions were held. It was not unusual to see 35 Officers from the Air Force, Navy and Marine Corps at these meetings representing all of the operational services. They were shown the latest FLEX prototype and ask for their feedback on it. These continuing interactions help to foster user support, understanding, and knowledge acquisition.

3.1.2 *Domain Expertise Lesson Learned*

Domain expertise is extremely important during software development and critical in the initial development. The contract management office should take any necessary steps to ensure that the contractor has the required domain expertise during proposal evaluation and throughout the program. Contracting Officers may be able to write a clause in the contract to help enforce the maintenance of proper domain expertise on the contract.

3.2 *FLEX as an Integrated Component*

FLEX relies on an enterprise TBMCS database called the AODB (Air Operations Database) and is designed to access the AODB through Data Access Agents (DAA) services provided by the DII COE (Defense Information Infrastructure Common Operating Environment). Services are also used for access to the Airspace, Enemy Order of Battle, and Target data. The TBMCS integration contractor created these services. The purpose of these agents is to abstract the applications away from the database and force them to use a common set of APIs. The desired effect is that database changes will only require recoding in the API not each application that uses it.

As of the writing of this paper (27 Apr. 1999) there have been 656 change requests to the AODB since its inception in 1996. The concept of a centralized AODB is a good one, but management

and implementation of the AODB and its services represents significant risk to FLEX. FLEX became directly reliant on other TBMCS services and when they break, FLEX suffers performance degradation or complete loss of functionallity. The services, in general, are not extensive enough to provide everything that's required by all of the applications. For example, the services do not provide friendly order of battle information, forcing local solutions to be implemented in each of the EMR, TAP and FLEX applications (see figure 1).

In an integrated system where services are provided in common, application elements like FLEX must be able to determine if critical services are working and available. The FLEX mission status board location name will not display if the Airspace Service is not available. Determining the cause of such problems cannot be properly resolved unless the status of common services is available. In our case, such problems were incorrectly attributed to FLEX. A similar problem occurred during recent testing because of changes to the database management design.

Having an enterprise database like the AODB keeps information consistent among applications but also causes compromises. The AODB can't be tuned for optimized processing of both online transaction like processing and "data mining" like analytical processing like that used in the "end of day debrief" FLEX report.

A major problem at the last Joint Field Acceptance Testing (JFAT) was database replication. FLEX automatically updates the local object store through database notification and database services. The FLEX replication is required for execution monitoring. If the FLEX server goes down the clients can continue to work on the local object store. Oracle database replication had failed to work in TBMCS in the past, so in the last JFAT they introduced ORDBEX a procedural database replication capability that would run every two hours to update each of the replicated databases over the wide area net. ORDBEX locks large portions of the AODB while it runs for a considerable time and sends out huge updates of the AODB. FLEX could not make updates to the AODB during this lock time and interprets the ORDBEX update as a huge update it needs to propagate to the FLEX clients. This immediately clogged the communications and brought FLEX to it knees. Had the FLEX development team been aware that ORDBEX was going to be used, changes could have been made to FLEX to compensate. Hopefully this will be rectified in the second JFAT. Overall, procedural replication, although suitable for fairly static planning databases, flies in the faces of database consistency and the near real time database requirements of execution monitoring.

As we prepare for the second, and most likely last, Joint Field Acceptance Testing we are told that the FLEX delivery must occur one week before the AODB changes are finalized. In the past tables existed with no data in them which prevented evaluation of FLEX without first populating the AODB. An example of this was with the Combined Air Operations (COMAO) data where the TAP (see Figure 1) data format did exist in the AODB.

3.2.1 *Component Services Lessons Learned*

Communication and cooperation between contractors creates additional risk for a program. The contract managers have to understand the potential vulnerabilities of the arrangement and make

sure that the highest levels of management understand and support these vulnerabilities. The component-based services should not be allowed to become a single point of failure. Complete end-to-end testing with the components in place is required before application acceptance testing. The lesson learned in component programming will be magnified as we move to agent-based programming.

3.3 *Remotes Requirements*

FLEX defines remotes as any FLEX client that is not on the FLEX servers local area net (LAN). Initial versions of FLEX did not address remotes, and when they did become a requirement it was desired that the remote software be identical to that which was present in the AOC, the typical site of the FLEX server. The first engineering change to address remotes had no performance requirements and mentioned up to thirty remotes. Later, Air Combat Command/DRC expressed the need to support up to one hundred remotes and the Navy weighed in with its timing requirements over a 9.6 kilobit per second (Kbps) circuit over a three-hop geo-synchronous satellite link.

The remote architecture consists of a distributed object-store directly implemented via CORBA compliant Object Management System, Orbix and OrbExpress. We did testing of the remotes starting in November 1998 using a Ka-Band "suitcase" satcom link. Link data rates of 9.6,32 and 1,024 Kbps were tested for one-hop. We also compared the baseline FLEX module performance with that of the first tweaked version where the object server connected directly to the database server.

From our experiments, we learned that over high-latency communication links bigger packets are better than small packets. TCP makes you wait until an acknowledgement from the first packet is received before sending a second packet. The LAN round-trip latency may be 2-10 milliseconds while the WAN latency may be 2-10 seconds or three orders of magnitude greater. However, bigger packets are not always better. If a packet must be retransmitted, a bigger packet means sending more data multiple times. Links with a high bit error rate (BER) can cause significant retransmits. Software that is configured to expect a response from a distance host within a specified period may retransmit a packet before the distant end has received the original packet.

For FLEX many small packets are sent because small objects are referenced. Bigger packets were obtained by referring to aggregate objects. With a single-threaded FLEX database server for remotes, if one remote object server hangs, then all the remotes have to wait until things clear up.

3.3.1 *Remotes-Technical Challenges Lessons Learned*

The FLEX remotes represented a significant technical risk to the program. The 9.6Kbps communication connection is a prime example. Rather than address this risk up front, it was not addressed until near the end of the program, making it a showstopper. In software engineering these risks need to be clearly identified early and steps taken to mitigate the risk. Requirements, which are clearly unrealistic or extremely costly, should be discussed so that compromise

solutions can be identified and put in place. A prime tenet of software engineering is that the sooner a problem is addressed, the less costly and time consuming the solution will be.

3.4 *The Contracting Vehicle*

Back in 1994 when FLEX was awarded, the standard contracting vehicle used was a CPFF Completion or cost plus fixed fee completion type contract. A CPFF is very ridged according to the original Statement of Work (SOW). Over the life of the FLEX contract, there were 19 engineering change proposals (ECP) made to the contract quadrupling its value to around 19 million dollars[3]. These changes have to be properly justified to fit within the scope of the original SOW. We were told that each ECP costs roughly 30K of manpower and three months to add, implying that ECPs to FLEX may have cost the Government $570K in primarily administrative manpower to implement. Keep in mind these changes were required mainly due to evolving changes in user requirements and software standards. The unfortunate reality is that we are forced to use contracting vehicles in the DoD that conform to the Government contracting laws passed by congress.

Although CPFF Completion was undoubtedly the best contracting vehicle in 1994, we never anticipated the level of changes required for FLEX. Currently if we started over again in todays acquisition environment, we are advised that we should use a modified IDIQ contracting vehicle or indefinite delivery, indefinite quantity contract. IDIQs are significantly faster and more flexible to changes and seem more appropriate to a dynamic software development like FLEX. Although an IDIQ contracting vehicle would be a significant improvement, it is by no means the complete answer to the problem of keeping up with the dynamic requirements of software developments since sizable effort is still involved in adding new tasks

3.4.1 *Contract Vehicle Lesson Learned*

Choose the most flexible contracting vehicle available for software developments.

3.5 *Programming Languages*

When the FLEX contract was awarded Ada was required for use in the DoD for operational software. Ada turned out to be a double-edged sword: on one side supporting excellent software engineering practices; on the down side the software bindings and interface libraries to commercial software packages like Motif and Oracle often did not exist and had to be written. We were particularly vulnerable because there was only one source of Ada bindings to CORBA, OrbExpress, which was maturing during the FLEX development. The conversion from Ada 83 to Ada 95 also represented a significant change and cost for FLEX.

Initial requirements led to certain design decisions. A distributed system was required because of the reliability and redundancy it would offer. Getting Ada to work in a distributed CORBA environment required substantial effort.

---

[3] Large for Rome, small for DoD software.

Given the current options we would likely use Java or C++ as the primary language and the web as the primary vehicle to implement modern FLEX.

3.5.1 *Programming Language Lessons Learned*

Now that there is a choice of programming languages to use, careful examination of the COTS (commercial off the shelf) support libraries and tools available for these languages is essential. Long term maintainability and supportability depend on mature tools and support.

3.6 *Re-tasking*

Another requirement for FLEX was the ability to replan parts of the Air Battle Plan dynamically. The desire was to allow FLEX to do most of this automatically while under control of the user. This requirement resulted in a design including a local object server for the client that would reason with the objects. This automatic replanning requirement was later removed when it was decided that another Theater Battle Management Core System component (EMR) would perform replanning.

There remains one major problem. The current Air Force planning process does not capture planning rationale in a machine-usable format that would support replanning. Replanning would require the reallocation and rescheduling of air missions based upon a previous plan that logically decomposed goals into missions to achieve certain goals. Although the Defense Advanced Research Projects Agency's (DARPA) JFACC Planning Tool program is working in the area of capturing "strategy-to-task" decomposition, tools developed as part of that program are not interfaced with the Air Operations Database (AODB). In the interim, FLEX users must know the overall plan rationale and FLEX could support them in computationally determining the feasibility of re-tasked missions by running constraints. The FLEX replanning problem combined with the JFACC Planning Tool could be adapted to various size planning problems to challenge even the state-of-the-art in artificial intelligence planning. The current FLEX design should support any immediate requirements to add dynamic replanning.

3.6.1 *Re-tasking Lesson Learned*

Design software to allow for logical future additions, so it won't be obsolete upon completion.

3.7 *Software Management*

A significant risk on FLEX was the management structure. The consistency of the management on FLEX was a problem. We were fortunate to have a single individual on FLEX here at Rome for the full nine years of its existence but had four different managers at the FLEX contractor. Roughly two years into the program, an experienced software engineer became the FLEX Logicon program manager taking the undisciplined FLEX and whipping it into shape with his software engineering background. He implemented a very disciplined software development approach and claimed the program was operating at a Software Engineering Institute maturity

level of three. The feeling at Rome is that had he not done this at that time, FLEX would not have come as long as it has. Unfortunately as pressure has mounted there has been a tendency to throw the software engineering processes out the window and resort to software artistry again.

The FLEX contractor reported to us at AFRL. Half way through FLEX, we became part of Electronic Systems Center's (ESC) TBMCS program with the integration contractor playing a significant role in FLEX. Two Government organizations and two competing contractors represented significant risk to the program. We used to be under the command of ESC until we were reorganized into the Air Force Research Laboratory. Participation by high level management is essential to ensure that a close cooperative environment is maintained. The alternative is chaos and failure.

3.7.1 *Software Management Lessons Learned*

Enforce consistent management with a disciplined approach to software engineering. In a Joint program like FLEX, the involvement of high-ranking leadership helps solve the disagreements that can destroy a program.

3.8 *FLEX Reports*

Two members of the Rome FLEX team have been working in-house on FLEX reports which is a web Java script based interface directly into the Air Operations Database. The FLEX reports addressed deficiencies in FLEX that CAFMSX (the existing operational capability) was able to provide. The FLEX reports are now so powerful that they can often replace many of the capabilities of FLEX. An example of this is when FLEX remotes go down, reports can be used to monitor the ongoing air battle execution. In the event of failure of FLEX the reports are likely to become very useful.

3.8.1 *In-house Lessons Learned*

The best way for the contracting management team to stay abreast of a program and technical issues is for them to be actually involved in the development process. This was very effective in FLEX.

3.9 *The Requirements Avalanche*

Requirements creep helped drive FLEX from a $4.4 Million program to a $19 Million program. Most of the requirements changes were not in functionality, but in keeping up with the latest TBMCS standards such as the AODB, ATO98 message format, Ada 95 and four versions of DIICOE.

3.9.1 *Requirements Creep Lessons Learned*

Requirement changes are going to happen on any large software development. Be prepared for them. As the parent system changes (e.g., TBMCS), wide distribution of requirements changes is essential for risk management.

4 **Conclusions**

To summarize the lessons learned on FLEX:

1. Enforce good domain expertise.
2. Mitigate the risk of cross contractor software component development.
3. Address engineering challenges early.
4. Use the most flexible contracting vehicle available.
5. Look carefully at the COTS available to support your choice of design and programming language.
6. Design software for logical additions.
7. Enforce consistent management with a disciplined approach to software engineering and have high-ranking support.
8. In-house development can support the development and maintain individual currency in software engineering.
9. Be prepared for requirement changes.

Acknowledgements: We would like to acknowledge the support of our on site contractors who have played such a significant role in the FLEX program: Maria Cappelli, Naomi Dyer, Scott Lingley, and Greg Paul. Thanks to Carl DeFranco for his "Able Toastmaster" improvements to this paper.

5 **Reference**

[Sommerville,1992] Ian Sommerville, *Software Engineering, Fourth Edition*. 1992 Addison Wesley Publishing