AFRL-RI-RS-TR-2008-322
Final Technical Report
December 2008

DISTRIBUTED EPISODIC EXPLORATORY PLANNING (DEEP)

Rome Research Corporation

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

NOTICE AND SIGNATURE PAGE

REPORT DOCUMENTATION PAGE	*Form Approved* **OMB No. 0704-0188**

1. REPORT DATE *(DD-MM-YYYY)* DEC 2008	**2. REPORT TYPE** Final	**3. DATES COVERED** *(From - To)* Jun 07 – Sep 08

4. TITLE AND SUBTITLE DISTRIBUTED EPISODIC EXPLORATORY PLANNING (DEEP)	**5a. CONTRACT NUMBER** FA8750-07-C-0176
	5b. GRANT NUMBER N/A
	5c. PROGRAM ELEMENT NUMBER 62702F
6. AUTHOR(S) Kurt Lachevet, Daniel Kaczynski, and Geraldine Rogers	**5d. PROJECT NUMBER** 558S
	5e. TASK NUMBER CP
	5f. WORK UNIT NUMBER 12
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Rome Research Corporation 314 South Jay Street Rome, NY 13440-5600	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFRL/RISB 525 Brooks Rd. Rome NY 13441-4505	**10. SPONSOR/MONITOR'S ACRONYM(S)** N/A
	11. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-RI-RS-TR-2008-322

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# 88ABW-2008-1255

13. SUPPLEMENTARY NOTES

14. ABSTRACT
DEEP is a mixed-initiative decision support system that utilizes past experiences to suggest courses of action (COAs) for new situations. It was designed as a distributed multi-agent system, using agents to maintain and exploit the experiences of individual commanders, as well as to transform suggested past plans into potential solutions for new problems. The commander, through the agent, can view and modify the contents of the shared repository. Agents interact through a common knowledge repository, represented by blackboard, selected because of its opportunistic reasoning capabilities and implemented in Java for platform independence. Java was chosen for ease of development and integration with other projects. Research also included investigations into various scalability software suites and frameworks, as well as different database management systems. Hibernate, an object/relational persistence and query service, was chosen for interaction with the database. Comprehensive testing revealed the Java Distributed Blackboard was limited only by system resources and network bandwidth. Thus, its architecture is well suited for dealing with ill-defined, complex situations such as military planning.

15. SUBJECT TERMS
Case-Based Reasoning, Planning, Distributed Computing, Episodic Planning, Blackboard, Agent

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Dale W. Richards
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UU	20	**19b. TELEPHONE NUMBER (*Include area code)*** N/A

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

Table of Contents

1. EXECUTIVE SUMMARY

The initiation of a Global War on Terrorism has brought to the forefront unique challenges in identifying adversaries, their plans, motivations, intentions and resources, and opportunities for successfully engaging them. Technologies enabling the right people to discover the right information at the right time, anywhere in the world, are of vital importance. The September 11, 2001 attacks and subsequent violence around the world proves that failure to develop such technology can contribute to the unacceptable loss of life, degradation of military readiness, and placing the U.S. population at unacceptable risk.

The Air Force Research Laboratory has initiated research within its Commander's Predictive Environment (CPE) program and the associated Distributed Episodic Exploratory Planning (DEEP) project have take on the challenge of developing technology to enable commanders to rapidly develop better and more robust plans, and collaborate with other Air Operations Centers and other command centers. These efforts seek to improve the understanding of the operational picture (past, present and future), learn from past failures and successes, and be able to characterize and predict likely future events within the planning and execution process.

Advanced decision support systems provide a core capability required to align resources and increase effectiveness on the battlefield of today and prepare for the future. By effectively integrating and coordinating proactive measures and dynamic responses assisted through advanced technology, commanders will be proactive in meeting the dynamics necessary to deal with the myriad of roles commanders will face, and win the war on terrorism. The United States defense strategy requires continuing information superiority to secure an advantage over adversaries. In order to support the goal of information dominance and the focus on network-centric warfare, it is necessary to develop tools that can assist in predicting combat environments and building plans in response. Such a predictive environment must provide an ability to develop proactive Courses of Action (red, blue and gray) tailored to potential or pending crises, and be able to adjust them over time.

The objective of this effort was focused on developing a software capability to assist a Joint Force Commander (JFC) and/or Joint Force Air Component Commander (JFACC) to dynamically build and adjust combat plans and execute decisions based on a predictive battlespace environment, drawing from past experiences and apply that knowledge to present situations. This robust decision support environment will enable the commander to develop better, more robust plans to meet strategic objectives.

The JFC/JFACC must have an understanding of the operational picture (past, present and future), be able to characterize and predict likely future events within the planning and execution process, generate options, comprehend the impact of decisions made today on the battlespace of tomorrow; and do this reliably within the pace of modern combat. The JFC/JFACC must be able to anticipate plausible end states (based on "red" and "blue" actions), evaluate possible courses of action, have rapid, easy access to shared battlespace information, and interactive with planning tools in an intuitive mixed-initiative manner.

This effort designed and implemented a prototype decision support capability. This includes technology to: (a) gain greater understanding and awareness of the battlespace through reflection, imitation and experience; (b) develop proactive course of action tailored to potential or pending crises; (c) develop and adjust projections temporally, spatially and across friendly (blue), enemy (red) and neutral (grey) forces; (d) provide mixed initiative planning; and (e) support distributed and collaborative planning.

This report provides an overview of the work performed by Rome Research Corporation (RRC) in support of the DEEP project at AFRL's Rome Research Site. DEEP is a mixed-initiative decision support system that utilizes past experiences to suggest courses of action (COAs) for new situations. It was designed as a distributed multi-agent system, using agents to maintain and exploit the experiences of individual commanders, as well as to transform suggested past plans into potential solutions for new problems. The commander, through the agent, can view and modify the contents of the shared repository. Agents interact through a common knowledge repository, represented by a blackboard in the initial architecture.

The blackboard design pattern was selected because of its opportunistic reasoning capabilities. The initial design called for an Open Source blackboard written in LISP. The intent was to extend the blackboard to be a distributed environment; however, detailed examination revealed difficulties integrating LISP with Java, as well as noting that LISP does not have a network standard – each LISP implementation has its own networking implementation. Consequently, the combined RRC and AFRL team decided to develop a true distributed blackboard, using Java for platform independence. Java was chosen for ease of development and integration with other projects. A Java Distributed Blackboard (JDB) had the benefits of a generic shared memory space which could be molded and extended to fit the exact needs of the DEEP project. Research also included investigations into various scalability software suites and frameworks as well as different database management systems. The team decided to use Hibernate – an object/relational persistence and query service – to interact with the database.

Comprehensive testing revealed the JDB was limited only by system resources and network bandwidth. Thus, its architecture is well suited for dealing with ill-defined, complex situations such as warfare.

2. INTRODUCTION

The objective of this contract was to investigate, design, prototype, test, evaluate and demonstrate a capability for providing a Joint Force Commander (JFC) and/or Joint Force Air Component Commander (JFACC) with a set of tools to cooperate with other Air Operation Centers (AOC) and command centers. This end goal was to assist the JFC/JFACC in dynamically building and adjusting combat plans and execution decisions based on a predictive battlespace environment, and draw from past experiences to apply to present situations. The result is a highly robust decision support environment, enabling the commander to develop better, more robust plans that meet strategic objectives by:

- Understanding the operational picture (past, present and future)
- Characterizing and predicting likely future events within the planning and execution process
- Generating options for the commander
- Comprehending the impact of decisions made today on the battlespace of tomorrow
- Blending Commander’s intent and situational awareness/understanding into a predictive environment with a veracity level that would allow operational domain plans to be developed within the pace of modern combat
- Developing an environment that provides the JFC/JFACC the ability to:
    - Anticipate plausible end states (based on “red” and “blue” actions)
    - Evaluate possible courses of action
    - Provide rapid, easy access to shared battlespace information
    - Allow for intuitive like mixed-initiative planning

A major element of DEEP was to support the cognitive domain within the strategic layer of the Office of the Secretary of Defense Command and Control (C2) Conceptual Framework. Many earlier research and development programs provided the technology underpinning for the DEEP project, such as:

- Computational behavioral modeling
- Intelligent agents/enhanced machine-to-machine collaboration
- Immersive interfaces
- Game theory
- Real-time learning
- Episodic memory
- Multi-agent systems
- Collaboration tools

- Blackboard systems

The scope of this effort included:

- Defining opportunities to enhance the DEEP system
- Designing software components to fulfill definitions
- Implementing designs
- Integrating interoperable software modules within existing CPE baseline
- Testing components
- Documenting tests and experiments results
- Documenting integration steps, system configurations, and installation plans
- Providing software documentation, code listings, and user manuals for developed capabilities

The delivered software components included defined service components; graphical user interfaces, where applicable; and installation, configuration and User’s Guide documentation. Presentations and demonstrations on advanced technologies were conducted at the AFRL Rome Research Site and elsewhere.

3. METHODS

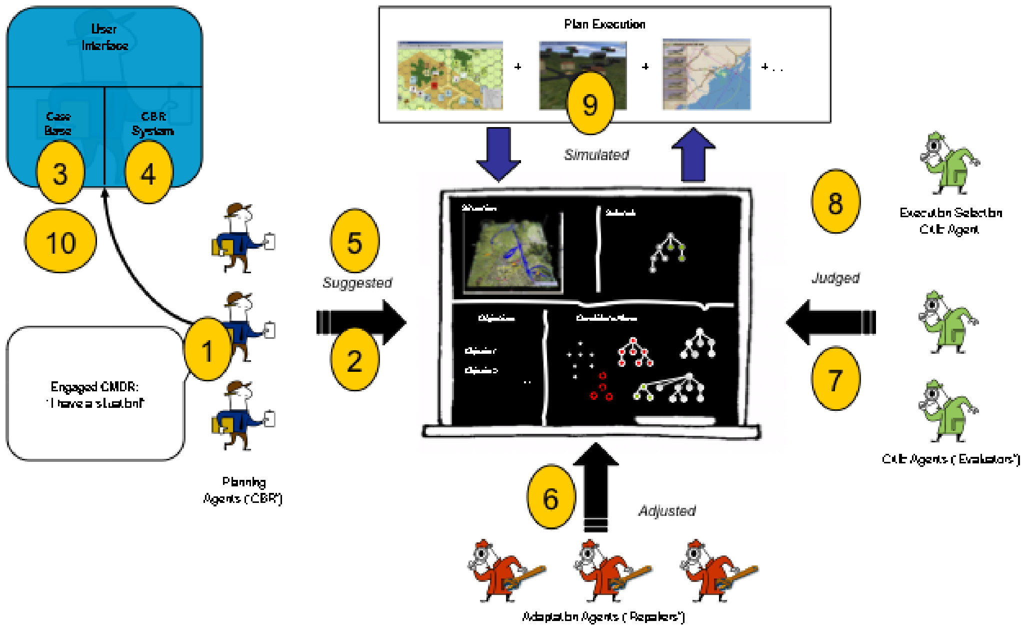
3.1 Statement of Work Tasks

For this effort, the Statement of Work tasks were to:

- Research and identify enhancements to CPE Technology Program; define and prototype DEEP capabilities
- Design, develop and prototype DEEP tools
- Design, develop and prototype software to support prediction and assessment of probable COAs using opportunistic reasoning
- Design and develop tools for knowledge bases using blackboard and multi-agent systems technologies

3.2 DEEP Architecture Overview

The scope of this effort was confined to building tools for DEEP. Development of the core components of DEEP itself was not part of this effort. While no discussion of the DEEP architecture is included in this report, Figure 3-1 is provided for reference in the following Java Distributed Blackboard discussion. Detailed descriptions of the DEEP architecture, as well as the underlying aspects of mixed-initiative and episodic planning are documented elsewhere. (Ford and Carozzoni, 2007) (Carozzoni and Lawton, 2008) (Ford and Lawton, 2008)



User Interface
Case Base
CBR System
Plan Execution
Simulated
Suggested
Judged
Execution Selection CtE Agent
Engaged CMDR: "I have a situation"
Planning Agents ("CBR")
CtE Agents ("Evaluators")
Adjusted
Adaptation Agents ("Repairers")
1
2
3
4
5
6
7
8
9
10

Figure 3-1 DEEP Architecture

3.3 Blackboard

The DEEP system requires a method to communicate and interact. To achieve this, a blackboard mechanism was selected. The blackboard design offers opportunistic reasoning capabilities (Corkill, 1991). It serves as a repository easing communication between systems.

Due to DEEP requirements, the blackboard needed to be extended from a monolithic to a distributed environment; however, research revealed existing commercial blackboards were not distributed. The DEEP team designed and implemented the distributed environment using the design patterns described in *Parallel and Distributed Programming Using C++* (Hughes, C., & Hughes, T. (2003).

The initial design called for using an Open Source blackboard written in LISP which would be extended to become a distributed environment. Issues soon became evident, including:

- LISP does not integrate well with Java
- LISP does not have a network standard – each LISP implementation has its own networking implementation

Consequently, the Team decided to develop a true distributed blackboard using Java. Java was chosen for ease of development and integration with other projects. A Java Distributed Blackboard had the benefits of a generic shared memory space which could be molded and extended to fit the exact needs of the DEEP project. The Java implementation included the ability to easily choose a port; as a consequence, the operator can run as many blackboards as desired on a single machine.

3.3.1 Java Distributed Blackboard Architecture Overview

The architecture is composed of a hierarchy of Java classes and is illustrated by the diagram depicted in Figure 3-2. The diagram illustrates that a blackboard starts up as a 'server' on a machine. Multiple blackboards connect to this server as clients using the Java Remote Method Invocation (RMI). When a client connects, the server propagates everything on the blackboard to the new client(s). When the server has something new on it, it updates its clients. Conversely, when a new object is placed on a client blackboard, the object is given to the server which then updates all of the clients. This is done to avoid the synchronization issues inherent to a distributed environment.

Agents, or knowledge sources, connect to a blackboard through Java RMI as well. A BBProxy (BlackBoard Proxy) server exists for each blackboard application to facilitate these connections. This proxy provides the interface for the knowledge sources to interact with the blackboard.

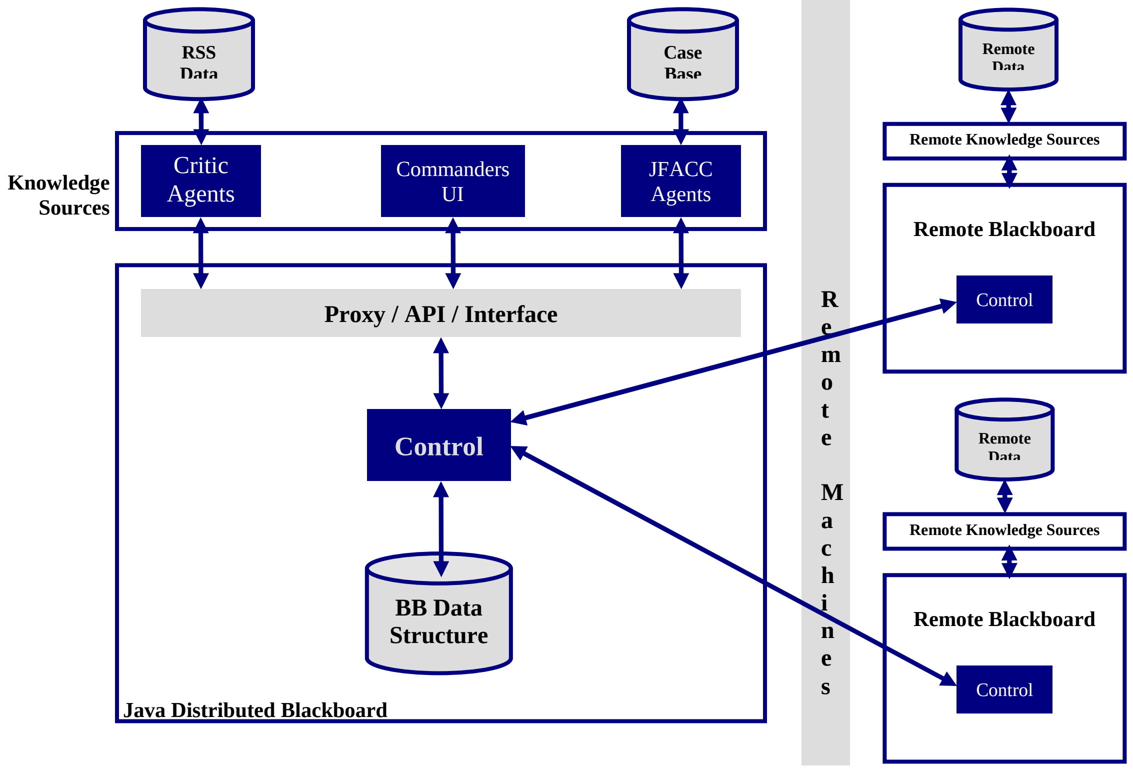


Figure 3-2 Java Distributed Blackboard Architecture

3.3.2 Blackboard Components

A Blackboard architectural pattern, traditionally, is composed of three components: (1) knowledge sources, (2) a core data structure; and (3) a control component.

3.3.3 Knowledge Sources

By connecting to the blackboard, an application has the ability to become a knowledge source. Figure 3-2 shows example knowledge sources from the DEEP project. Also shown is how a knowledge source could contribute external information to the Blackboard. Further, there is a mechanism in place to notify the knowledge sources of Blackboard events.

3.3.4 Core data structure

The current Blackboard core data structure contains "partitions" to allow objects to be sorted and stored based on a unique string. These partitions store objects mapped by their unique identifier (UID). The Team designed and implemented the data structure in the Java Distributed Blackboard such that it could be extended to become a wrapper to a database or other high performance data store.

3.3.5 Control

The Java Distributed Blackboard (JDB) contains a controller to handle the flow of information between the knowledge sources on the same machine as well as separate machines; however, as far as control in a traditional blackboard sense, the paradigm used is that the knowledge sources are given the responsibility of contributing to the solution. To facilitate the contribution, the knowledge source registers with the JDB thus ensuring it is notified of updates.

3.3.6 Additional Components

In addition to a traditional blackboard components, the RRC Team developed components specific to the Java Distributed Blackboard. These additions include a proxy, blackboard objects, and blackboard utilities.

3.3.7 Proxy

The proxy is an interface used to connect the knowledge source to the Java RMI. Through the proxy the interface can put and retrieve objects on/from the client. Other actions supported by the proxy include retrieving an object by its UID and registering new blackboard listeners. Similar to the core data structure, the proxy can be extended to accommodate integration with other applications (new or existing) as needed.

3.3.8 Blackboard objects

For a Java object to be placed on the blackboard, it has to satisfy two requirements. First, it must have a universal identifier; second, the object must be serialize-able. To comply with these requirements, an interface was created to force the use of a UID. For convenience, the object inherits from the *java.io.Serializable* class so serialization is automatic. Essentially, for any Java object to be placed on the JDB, it must implement the JDB objects interface.

3.3.9 Blackboard utilities

Several utilities were developed for the DEEP Java Distributed Blackboard including the Packet, BBUID, Log writer, and Properties file parser. The main utility is the Packet which is used by the control to send information. The *BlackboardListener* receives the Packet when it gets an update event from the blackboard. Based on packet type, *BlackboardListener* will contain information or blackboard objects. The *BBUID* is an identifier which is unique across a network. The remaining utilities provide convenience functions (log writer and properties file parser).

3.4 Research

Research performed for this effort included investigations into various scalability software suites and frameworks, as well as databases. The following subsections provide a brief overview and the reasons why or why not an item was used.

3.4.1 Terracotta

Terracotta is an open-source infrastructure software product used to scale a Java application to as many computers as needed. Terracotta was considered briefly based on its usefulness to the distributed aspect of DEEP. Although a useful application, the Team decided against tying the project to a specific application early in the development process. Further, DEEP's portability requirements called for platform independence; Terracotta lacked support for the Macintosh platform.

3.4.2 Hibernate

Hibernate is an object/relational (O/R) persistence and query service. Queries can be stated in Hibernate's portable SQL extension (HQL), in native Structured Query Language (SQL), or with an object-oriented criteria and example application program interface (API). For DEEP, Hibernate offered the following advantages:

- Abstracts SQL by utilizing HQL so any database with a Java Database Connectivity (JDBC) connector can be swapped out with ease
- Returns a collection of objects, eliminating the need to build these objects from a result set returned by using SQL+JDBC
- Increases performance by using 'prepared' query statements and caching
- Allows a mapping to a data store without modifying existing project code
- Externalizes object-relational mapping to an XML file

While not necessarily considered disadvantages the Team realized that:

- A Data Access Object (DAO) is required for each object to be mapped to the database
- A JDBC driver has to be installed as Hibernate uses it as connection to the database

For DEEP, the main issue would be the notification system. This system could be built by one of two means: either build/use a notification system inside the database, or build a notification system as a wrapper to the database. To utilize Hibernate to persist an object, four steps were necessary:

1. The to-be-persisted object had to be acceptable to Hibernate – Hibernate prefers persistent objects be in 'Java-bean style' and be serialize-able. Time was spent reviewing Hibernate's documentation to find ways to minimize the changes that would need to be implemented in the DEEP code

2. An XML mapping file for the object was necessary – the mappings needed to be simple while functional. Significant time was working out how to map objects within objects.

3. The object had to be persisted either Stored or Retrieved using HQL.

4. The database had to be configured properly to accept these objects – the Team learned to incorporate certain tags into the XML mapping file to minimize or eliminate a database configuration.

Based on the analysis performed, the DEEP Team selected Hibernate.

3.4.3 Database Management Systems

The DEEP team compared and contrasted the differences between Object-Oriented (OO) and Relational (R) database management systems (DBMSs) as related to DEEP. OO DBMSs reviewed included: Versant, Objectivity, Matisse, EyeDB, Ozone, and Cache.

The Team researched O/R mapping, its relevance (it masks the task of mapping objects to tables from the programmer) and identified the following relational DBMSs as having O/R Features:

- DB2
- Greenplum
- Intersystems Cache
- OpenLink Virtuoso
- Valentina
- PostgreSQL
- Hibernate3
- GigaBASE
- Informix
- Oracle
- UniSQL
- VMDS
- Zope / ZODB

3.5 Development

The JDB was designed and modeled using Unified Modeling Language (UML) and developed code using Java for platform independence. Support for agents was considered. On examination, the adaptation capabilities agent was decomposed into a capabilities critic and a capabilities adaptation agent. The capabilities adaptation agent creates an adapted plan out of an instantiated plan, taking into account the critic's scores. It was designed and developed using requirements provided by the DEEP team. The capabilities adaptation agent was integrated and tested. A third agent was developed - the execution selection agent. The RRC team researched, designed, implemented and integrated it into the JDB.

The DEEP team tested put and retrieval latencies to and from the JDB with various numbers and size objects. After the initial testing concluded, the DEEP team extended the JDB, implementing the following capabilities:

- Blackboard User Interface
- Agent Support
- Additional Proxies and Methods
- Messaging Support
- O/R Mapping to Oracle Database
- Hibernate

3.6 Other Tasks

Additional tasks performed under this effort included:

- Assisting in the evaluation of the potential for interfacing a commercial military simulation software package, Modern Air Power, to the DEEP system.

- Assisting with testing, debugging and demonstration of the DEEP system.

- Organizing, co-authoring the draft and submitting a paper to the 13th International Command and Control Research and Technology Symposium. (Destafano, 2008) (Note - final submission was undertaken by the author after leaving Rome Research Corporation and becoming an Air Force employee.)

4. RESULTS

This effort implemented a functional Java distributed blackboard for the AFRL DEEP project. After confirming the initial design and implementation, the DEEP team researched and implemented a major upgrade to the JDB by installing the Hibernate database. Hibernate replaced the majority of the JDB's database and a connection to the blackboard. The implementation was effective while maintaining complete functionality.

Formal testing of the JDB revealed the following:

- A local blackboard is not necessary, an agent (or any knowledge source) may connect to a remote blackboard as if it is on the local machine.

- Put and retrieval latencies to and from the JDB demonstrated that one hundred (100) million objects (500 MB) could be placed on the JDB without slowdown

5. CONCLUSIONS

DEEP is a mixed-initiative decision support system that utilizes past experiences to suggest COAs for new situations. It was designed as a distributed multi-agent system, using agents to maintain and exploit the experiences of individual commanders as well as to transform suggested past plans into potential solutions for new problems.

The commander, through the agent, can view and modify the contents of the shared repository. Agents interact through a common knowledge repository, represented by a blackboard in the initial architecture. The Blackboard design pattern was selected because of its opportunistic reasoning capabilities. Comprehensive testing revealed the Blackboard was limited only by system resources and network bandwidth. Thus, its architecture is well suited for dealing with ill-defined, complex situations such as warfare.

5.1 Discussion of Findings

The Java Distributed Blackboard is a generic distributed data structure which may be utilized by any application requiring a distributed framework. The DEEP researchers concluded that the Blackboard was limited only by system resources and network bandwidth.

5.2 Issues and Concerns

Issues and concerns experienced during the course of the contract included:

- AFMC computer policy restrictions –
  - Software/frameworks under consideration could not be installed until permitted by AFMC. One specific example is that Oracle10g is not (was not) on the approved list for installation. This affected the choice of databases.
  - Loss of local administrator privileges on development computers.
- Personnel changes –
  - The initial researcher/developer changed employment status from contractor to Government civilian.
  - The replacement researcher/developer changed employers and contracts.
- Technical –
  - The team experienced setbacks after learning two repositories cannot operate on the same machine.
  - Machines on different subnets (*rome-2k* and *jbi-dev*) within the AFRL network were unable to be cannot connect to the same repository.

6. REFERENCES

Carozzoni, J., Lawton, J., DeStefano, C., Ford, A., Hudack, J., Lachevet, K. & Staskevich, G. (2008). AFRL Technical Report AFRL-RI-RS-TR-2008-279, *DEEP: Distributed Episodic Exploratory Planning*.

Corkill, D. (1991, September). “Blackboard Systems.” *AI Expert , 6* (9), pp. 36-38.

Destefano, C., Lachevet, K., Carozzoni, J. (2008). “Distributed Planning in a Mixed-Initiative Environment.” *Proceedings of the 13th International Command and Control Research and Technology Symposium*.

Ford, A., & Carozzoni, J. (2007). “Creating and Capturing Expertise in Mixed-Initiative Planning.” *Proceedings of the 12th International Command and Control Research and Technology Symposium.*

Ford, A., & Lawton, J. (2008). “Synthesizing Disparate Experiences in Episodic Planning.” *Proceedings of the 13th International Command and Control Research and Technology Symposium.*

Hughes, C., & Hughes, T. (2003). *Parallel and Distributed Programming Using C++*. Boston: Pearson Education.

7. LIST OF ACRONYMS

AFMC	Air Force Materiel Command
AFRL	Air Force Research Laboratory
API	Application Programming Interface
AOC	Air Operation Center
BB	Blackboard
C2	Command and Control
COA	Course of Action
CPE	Commander's Predictive Environment
DAO	Data Access Object
DBMS	Data Base Management System
DEEP	Distributed Episodic Exploratory Planning
FTR	Final Technical Report
HQL	Hibernate SQL
ICCRTS	International Command and Control Research and Technology Symposium
JDB	Java Distributed Blackboard
JDBC	Java Database Connectivity
JFACC	Joint Force Air Component Commander
JFC	Joint Force Commander
LISP	LISt Processing
OO	Object-Oriented
O/R	Object-Relational
PM	Program Manager
R&D	Research And Development
RISB	Information Systems Research Branch
RMI	Remote Method Invocation
RRC	Rome Research Corporation
RRS	Rome Research Site
SQL	Structured Query Language
UI	User Interface
UID	Unique Identifier
UML	Unified Modeling Language
XML	eXtensible Markup Language