Ontogenetic Reasoning System for Autonomic Logistics

Joel R. Bock* and Tom W. Brotherton* and Doug Gass†
*Intelligent Automation Corporation
13029 Danielson Street, Suite 200, Poway, CA 92064
Tel: 858-679-4140, Fax: 858-679-4144
E-mail: {joel.bock,tom.brotherton}@iac-online.com
†NAVAIR-JSF Program Office, 22195 Elmer Road, Bldg. 106, Rm. 228, Patuxent River, MD 20670
Tel: 301-757-0477, E-mail: frank.gass@navy.mil

***Abstract*— Joint Strike Fighter Autonomic Logistics will minimize operational and support costs by increasing system reliability, while reducing maintenance requirements to essential levels. Using Prognostics and Health Management, parts and service are ordered or performed only when needed, obviating costly routine scheduled maintenance, and reducing aircraft downtime.**

Realizing this vision requires communication between the aircraft, industrial contractors and suppliers, and the maintenance and support team. Management of interactions between these entities is challenging, characterized by uncertain information, and conflicting demands for resources. A system is needed to encode the knowledge of maintenance personnel, suggest corrective actions in fault conditions, and learn from previous decisions. Integration with the supply chain would free human resources for more critical decision making tasks.

IAC is developing an intelligent software infrastructure to manage these complexities. This *Ontogenetic Reasoning System* features an adaptive knowledgebase of maintenance information, and autonomous software agents which (1) analyze on-board sensor and model data, and past behaviors; (2) recommended actions under dynamic and uncertain conditions; (3) manage knowledgebase evolution; (4) connect maintenance activities to the supply chain; and (5) perform various communications, security and support functions.

This paper presents the architectural design of the system, describes an example application scenario, and concludes with an assessment of the technical challenges in developing such a system using the multi-agent systems approach.

CONTENTS

I. INTRODUCTION

The Joint Strike Fighter (JSF) Autonomic Logistics Program aims to minimize operational and support costs by increasing system reliability, while reducing maintenance requirements to essential levels. Using Prognostics and Health Management (PHM), components and maintenance are ordered or performed only when needed, obviating costly routine scheduled maintenance, and reducing aircraft downtime.

This vision of efficiency and responsiveness requires asynchronous communication between the intelligent air vehicle, industrial contractors, their suppliers, and the aircraft maintenance and support team. All of these entities utilize prognostics from on-board sensor data or life cycle models, as well as historical maintenance information, in order to optimize PHM decision making. It is a considerable challenge to manage the interactions between the entities comprising this dynamic aggregation; by nature, the logistics are fraught with missing or ambiguous information, and conflicting demands for resources. Flight line maintenance logs are often observed to contain incorrect or suboptimal diagnostics and corrective actions in the face of specific fault conditions.

There is a need for a system which encodes the applied knowledge of engineering and maintenance personnel, autonomously recommends actions to correct fault conditions, and is capable of learning from previous decisions, both erroneous and correct. Integration with the supply chain would facilitate planning, ordering and logistics, freeing human resources for more critical decision making tasks. A system providing some or all of this functionality would make a

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
01 MAR 2004	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Ontogenetic Reasoning System for Autonomic Logistics	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Intelligent Automation Corporation 13029 Danielson Street, Suite 200, Poway, CA 92064**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**UU**	**8**	

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

significant contribution towards realization of the goals of Autonomic Logistics.

Intelligent Automation Corporation (IAC) is developing an intelligent software infrastructure to manage these complexities. This *Ontogenetic Reasoning System* (ORS) features an adaptive knowledgebase of maintenance information, and autonomous software agents which serve a variety of functions, including (1) analysis of on-board sensor and model data, and past behaviors; (2) recommended actions under dynamic and uncertain conditions; (3) management of knowledgebase evolution; (4) connection of maintenance activities with the supply chain; and (5) a variety of other communications, security and support functions.

This paper presents the architectural design of the system. A practical example application scenario demonstrating the system follows, involving learning from a maintenance and logistics database currently in use on U.S. Army aircraft. The mechanisms of learning and automatic adaptation of the knowledgebase will be demonstrated. The paper concludes with a realistic assessment of the major technical challenges in development of the system associated with the multi-agent systems approach.

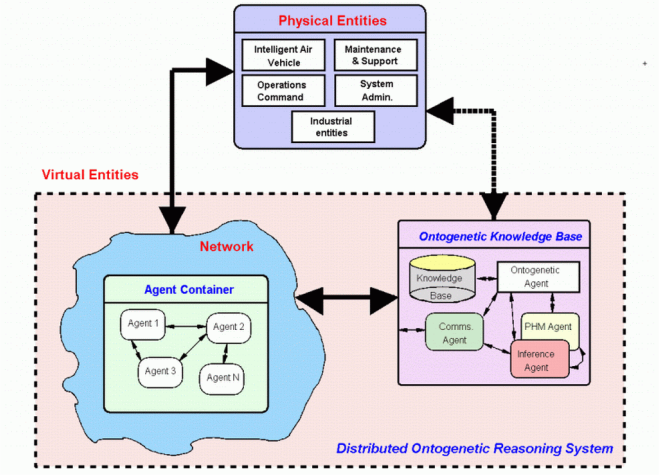

Fig. 1. Conceptual overview of the Ontogenetic Reasoning System. These include a Knowledge Base of maintenance information, an Ontogenetic Engine to manage evolution of the knowledge base, and a number of autonomous Software Agents performing analysis, decision making, communication and other support functions. These agents are distributed both locally and across the World Wide Web. This system offers a novel solution to management of the complex dynamic relationships and information flows characterizing JSF Autonomic Logistics. Under Phase I SBIR funding, IAC has concentrated on components surrounded by the dashed line in the figure.

II. System Description

A. System overview

The architecture of the distributed Ontogenetic Reasoning System software system is shown schematically in Figure 1. At this abstract level of generalization, the system comprises two types of entities: physical entities (air vehicles, human operators, industrial contractors, etc.) and virtual entities, which include runtime containers, agents, and databases, for example. ORS elements are virtual; entities which interact with ORS may be either physical or virtual in general. Autonomous *Agents* are runtime processes that execute within a *Container*, and may be distributed across a *Network*. A *Knowledge Base* is a repository for data and information, and other entities may create, read, update or delete records if they possess the proper privileges to do so.

B. Information flows

The problem domain of the *Ontogenetic Reasoning System* is established by describing information between entities within the system, as introduced in Figure 1. This view is high-level–the entities shown subsume a variety of functional groups, perhaps not physically co-located; for example, *Operations and Planning* or *Supply Chain Management*. Figure 2 presents a diagram summarizing typical data flows between ORS and client entities soliciting services. Inputs to ORS are denoted by the blue arrows, while outputs from ORS are colored in red. Note that this view is neither a complete nor definitive depiction of entities and data flows to and from ORS. In particular, the linkages connecting entities within the system should not be taken to necessarily reflect actual data transmission channels; some of the data flows may well be mediated by the *Autonomic Logistics Information System*, for example. Furthermore, practical boundaries defining these conceptual groupings are not sharp; it is easy to identify overlapping functions that could be characterized as belonging to both *Logistics* and *Operations and Planning*. This view of the system will be continually modified and updated as the program evolves over time, and architectural implementation details become more concrete.

C. Web-enabled architecture

A detailed view of the current architecture of the Ontogenetic Knowledge Base (OKB) component of ORS in client-server mode is illustrated in Figure 3. Clients gain access through nodes connected to the internet. The user interface is constructed from Java Server Pages (JSP) and are rendered in conventional web browser applications. Any number of clients can be distributed across the network; each client may host agents of varying levels of responsibility and embedded “intelligence”. The multi-agent architecture enables these distributed components to communicate with one another asynchronously, while providing a way to incorporate learning algorithms which implicitly handle uncertainty, and may be programmed to automatically evolve the knowledge base over time. In the early stages of system development, graphical displays enable human experts to directly oversee the initial knowledge base development, by encoding facts and knowledge specific to the aircraft maintenance domain, and providing immediate feedback regarding the correctness of decisions. These decisions might be made by the system in response to fault conditions detected by PHM hardware, or in response to queries posed by maintenance personnel, for example.

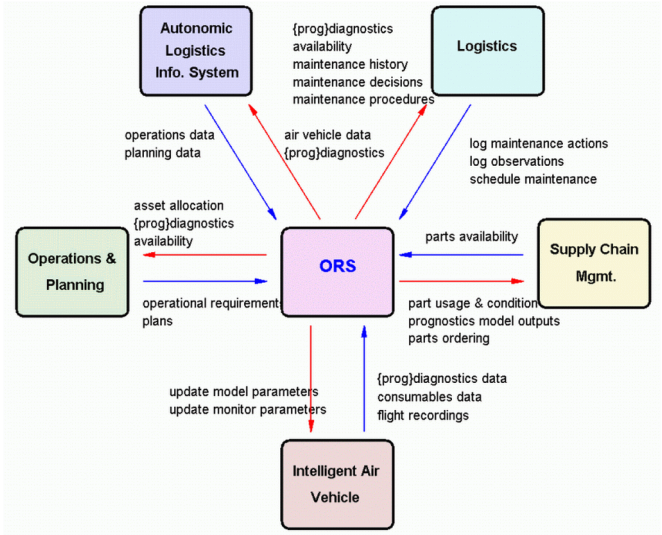

Fig. 2. Notional data flow diagram for the Ontogenetic Reasoning System (ORS). This view depicts interactions between external entities and the system under development, shown in the center of the figure. Inputs to ORS are denoted by blue arrows, while outputs from ORS to external entities are colored red. This view will be continually updated as the program evolves over time.

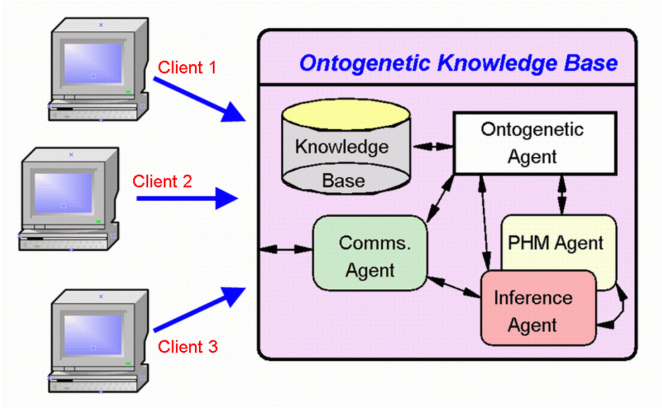

Fig. 3. Current architecture of the Ontogenetic Knowledge Base component of ORS in client-server mode. Clients gain access through nodes connected to the internet. The user interface is constructed from Java Server Pages (JSP) that are rendered in conventional web browser applications.

D. Agents and Components

In this section, we summarize Java package hierarchy of components that have been designed IAC during Phase I SBIR development. Java packages are a convenient means to group software components according to their conceptual or functional similarities, or to enforce data encapsulation requirements. Presently, six packages constitute the ORS software system:

1) Package `com.iac.agent` contains software agents and peripheral classes;
2) Package `com.iac.gui` has a preliminary set of components for graphical user interfaces to agents and the knowledge base;
3) Package `com.iac.okb` implements the persistent knowledge base, including data access objects and database connectivity components;
4) Package `com.iac.properties` contains various configuration files used throughout the system;
5) Package `com.iac.rules` holds a set of components implementing propositional logic for the rule base;
6) Package `com.iac.www` consists of JSPs, HTML and image files designed to facilitate distributed human interaction with the system components.

The most fascinating elements of this system are *Agents*. Multi-agent technology is perfectly compatible with the adaptive knowledge base infrastructure described here, for a number of reasons. Agent-based technologies are appropriate in applications with some or all of the characteristics summarized in the following [8], the first three of which are highly relevant to the Autonomic Logistics Program:

- The environment is open, highly dynamic, uncertain, or complex.
- Distributed data, control or expertise are hallmarks of the system.
- Agents are a natural metaphor to model interacting entities collaborating (or competing) to solve a complex problem or achieve a goal.
- Use of legacy systems requiring ”wrapping” for compatibility is mandated.

At this stage of development, IAC has integrated several fundamental software agents within the system. The most important of these are

1) The *OKBAgent*, responsible for managing knowledgebase access, ontogenesis, and security tasks.
2) The *FaultReasoningAgent*, responsible for oversight of the analysis of fault conditions recorded in the knowledgebase, and for recommending actions by querying its rule base in uncertain conditions.

Package `com.iac.agent` includes auxiliary agents that facilitate decomposition of processing tasks according to the “Model-View-Controller” (MVC) software engineering design pattern. MVC creates three distinct groupings of components. The Model implements business logic (here, the knowledge base, intelligence in the form of propositional logic, and Agents); the View is obviously the user interface for posing queries and displaying results; and the Controller logic mediates communications between clients (JSP or other agents) and the Model. While adherence to the MVC design pattern introduces some programming complexity early in a system design cycle, once put in place it promotes component reuse, and overall system maintainability and testability. Furthermore, it is our experience that new client applications are more easily developed within the existing software infrastructure. Figure 4 presents the Phase I system components considered in the MVC context.

All agents have been designed to extend the class `jade.core.Agent`, found within the JADE Agent Frame-

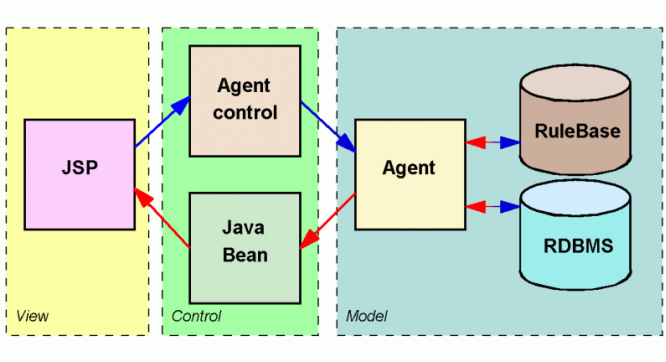

Fig. 4. Phase I system components, from the Model-View-Controller perspective. The Model implements business logic (the knowledge base, intelligent agents); the View provides a user interface for querying the knowledge base and displaying results; the Controller mediates communications between distributed clients and the Model.

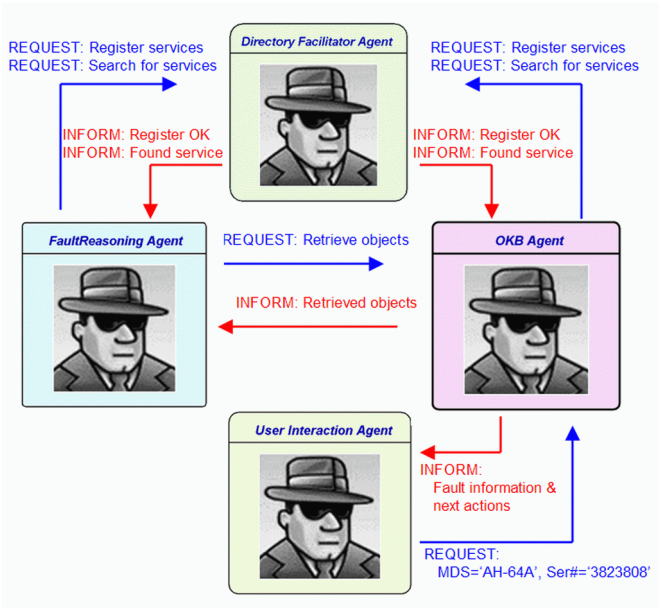

Fig. 5. Example PHM scenario as a conversation between agents.

work [1]. JADE is both an open-source software framework to write agent applications conforming to the Foundation for Intelligent Physical Agents (FIPA) specifications, and a runtime execution environment (Container) for the agents developed using the JADE application programming interface (API). FIPA defines a reference model of an agent platform, and a set of services that should be provided. Adherence to FIPA standards ensures that JADE agents can communicate with other agents in compliance with these specifications.

III. Example PHM Scenario

A. Agent and user interaction

This section presents a hypothetical PHM scenario involving the ORS system in one mode of operation. This scenario is considered as a "conversation" between agents in Figure 5.

In this figure, a symbolic *UserInteractionAgent* is shown. This agent is a proxy for any client application, and is presently embodied as a series of JSPs and their composite logic for navigation, querying by the user, and rendering of results retrieved from the OKB. The agents are represented using cartoon graphics to underscore the idea that they have beliefs, desires and intentions (BDI) [2], and carry out actions autonomously according to these humanistic characteristics.

The *Directory Facilitator Agent* maintains a directory of services ("yellow pages") for other agents to access, such that they might contact purveyors of services of interest. This agent is part of the JADE installation.

The conversation suggested in Figure 5 does not necessarily proceed in any deterministic sequence. Agents communicate by sending and receiving messages asynchronously; if a message of the appropriate type appears in an agents' message queue, he responds with a behavior that is consistent with his preprogrammed BDI characteristics. The timing of this response depends upon other tasks that have been previously scheduled in the agent behavior queue. From the client's point of view, the only temporal relationship perceived in this situation is that a query is posed to the OKB system, followed by a response some time later.

The principal events in this PHM demonstration scenario can be summarized in the following script.

1) *Login to OKB.* The user navigates to the main client page in a web browser. He logs into the system, supplying a valid username and password.
2) *Select Vehicle for PHM.*
   - Aircraft Type: The user specifies an aircraft type for analysis. Upon this selection, an *OKBAgentController* agent is added to the JADE platform. This agent contacts the *OKBAgent*, which retrieves relevant information on the designated aircraft from the knowledge base. This data is rendered within a dynamically-populated table for the user.
   - Aircraft Serial Number: User selects a row from the Aircraft Types table. Agents *OKBAgentController* and *OKBAgent* are called again, this time returning information on all serial numbers in the knowledge base corresponding to the specified vehicle type. A second table is displayed, showing all available tail numbers. Once choosing a row from this table, the user constrains subsequent processing to a single aircraft.
3) *Display Fault History.* The designated aircraft identifiers are passed to this JSP. Agent *FaultReasoningAgent* and its proxy *FaultReasoningAgentController* are created, and proceed to access fault condition information in the knowledge base via *OKBAgent*. Faults are grouped according to date of occurrence, severity, etc., and are displayed in a scrollable table for the user's review. The user now selects an individual fault or fault cluster; the corresponding information is used to initiate reasoning by *FaultReasoningAgent*.

4) *Maintenance Reasoning.* This page presents information comprising guidance on appropriate maintenance action(s) to correct the currently-selected fault condition. The source of the recommendations (from maintenance manual, reasoning algorithms or models) is be displayed for the user's information. Currently, the reasoning is carried out by accessing the rule base of the *FaultReasoningAgent*.
5) *Order Parts or Service.* This page provides form for entering part numbers, selecting vendors, etc., to order parts or servicing that may be needed based upon the fault condition at hand. The OKB is updated to record all actions carrried out by the user and agents during the current PHM session.

B. ELAS maintenance database

To demonstrate the concepts and technical objectives set for Phase I, IAC chose to use a Logistics Management Database satisfying United States Army requirements for systems to control and manage aircraft, aviation-associated equipment, mission related equipment, and maintenance [7]. That document provides instructions for the use, preparation, and disposition of forms and records used to control and manage aircraft, aviation-associated equipment, mission related equipment, and maintenance. The specifications described therein apply to the Active Army, U.S. Army National Guard of the United States (ARNGUS), U.S. Army Reserve (USAR), Department of Defense (DoD), and other U.S. Government agencies that operate and maintain Army aircraft. Also covered are aircraft and aviation-associated equipment operated, maintained, and stored by DoD contract support maintenance activities.

The Enhanced Logbook Automation System (ELAS) is an attempt to encode the requirements and guidelines for maintenance and logistics as set forth within [7]. The back-end of ELAS includes a database providing for information storage in a loosely-relational set of tables. The ELAS database schema is highly relevant to the objectives of this SBIR project, which aims to develop a Self-Evolving Maintenance Knowledgebase to support JSF Autonomic Logistics. Records contained within the ELAS database represent actual field maintenance logs, work orders, and other information. In particular, this database

- is an abundant source of free-text entries, for example describing faults or maintenance actions, containing redundant information due to lack of ontologies prescribing bounds for descriptive information;
- has annotations on errors in maintenance judgments and actions;
- includes multiple-record histories of actions relating to a particular fault over the course of time.

These characteristics can be found in most maintenance databases in field application on real vehicles. Therefore, the ELAS database is an ideal data resource to provide the foundation for development of prototype agents representing the core autonomous processing entities described in Section II.

C. Aircraft fault conditions

Fault conditions within ELAS are noted by maintenance operators according to the U.S. Army Maintenance Management System-Aviation (TAMMS-A) [7]. The fault conditions may be detected by on-board sensor hardware, by computations that predict component degradation based on hours of use, or perhaps may be first noticed during routine inspection or service of non-related components or subsystems. Each fault condition is assigned a priority score, the numerical value of which is inversely proportional to the severity of the condition. The highest-priority fault conditions are considered to be those that require immediate maintenance or service to correct. Such conditions are annotated in the knowledge base using a "Red X", signifying a serious deficiency that may endanger flight crew, maintenance personnel, or the asset itself. In the TAMMS-A nomenclature, a deficiency is defined as "a fault, defect, or problem so severe that it causes an item, system, or subsystem to be inoperative or inaccurate".

In ELAS, once corrective actions have been taken to address identified faults, these actions noted in the same `FAULT` table record as the associated fault. For the *FaultReasoningAgent*, this presents a technical problem–i.e., unravelling the temporal relationship between a fault declaration and its correction. It is not known whether or not this type of maintenance logging is performed in other aircraft maintenance systems. The immediate application assumes that fault and correction are coincident, clearly an incorrect assumption that is however necessary, given the current ELAS database maintenance logging procedures.

Maintenance actions are encoded by an alphanumeric code and associated with a textual description; this ontology of actions is defined in the `ACTION_CODE` table, reproduced here in Table I. Note that many descriptions listed in the table use past-tense verbs, illustrating the problem with fault-corrective action timing alluded to above. It is further evident that semantic gradations and redundancies are present in this ontology, allowing for imprecise documentation of maintenance performed.

IV. Methods of Inference and Self-Evolution

The *FaultReasoningAgent* developed during Phase I SBIR research uses an internal rule base to map fault conditions to appropriate maintenance actions. This section outlines the propositional logic used by this agent to perform inference.

A. Knowledge base engineering

There are five basic procedural steps that must be followed in the construction and practical use of a knowledge base [5]:

1) First, the knowledge engineer must determine which objects and facts within the problem domain are important, and which are not.
2) Second, an ontology appropriate for the problem domain must be conceived and implemented. This formalizes a specification of the important objects and facts such that they may be subjected to propositional condition-action logical statements for inference production.

TABLE I

MAINTENANCE ACTION CODES AND THEIR DESCRIPTIONS. THESE ARE USED IN THE ELAS DATABASE, AND IN THE CONSTRUCTION OF RULES USED BY THE FAULTREASONINGAGENT.

Action Code	Description	Action Code	Description
1	Svc.-sched.	I	Corrosion rem.
2	Svc.-unsched.	J	Tested
3	Prev. maint.	K	In-process insp.
4	Maint. test flt.	L	Remvd./Reinst.
5	Prev. maint.-insp.	M	Checked NRTS
6	Spec. insp.	N	Checked-can't repair
7	Grd. handling	O	Ovrhld./Reblt.
8	Maint. can't do	P	Checked-Svcable.
9	Modif. by repl.	Q	MWO Removal
10	No action	R	Removed
A	Replaced	S	Installed
B	Adjusted	T	SOF compliance
C	Repaired	U	Decontam.
D	Manuf./Fab.	W	Hr. meter Chg.
E	Not used for A/C	X	Gun Change
F	Initial insp.	Y	Spec. mission chg.
G	Final insp.	Z	Safety/Slipmark
H	MWO applied		

3) The delineation of such sentences or axioms is the third step in knowledge base engineering. Once programmed, these logic statements provide for automated evaluation of consequences associated with a predicate state as represented in the knowledge base.
4) The fourth step is to encode a specific problem instance using the ontological framework that has been developed previously.
5) Finally, the system may be queried given this particular problem instance, and the knowledge base will produce an answer using the semantics encoded during its construction.

B. Propositional logic

Propositional logic is constructed from sentences that are admissible under the defined syntax. These sentences comprise symbols (perhaps joined by connective operators) representing logical propositions that evaluate to TRUE or FALSE. For example, an expression such as

$$P \wedge Q \Rightarrow A \tag{1}$$

in the present context is interpreted to read

$$\text{IF CONDITION } (P \text{ AND } Q) \text{ THEN DO } A \tag{2}$$

to accentuate the fact that we are connecting a conjunctive antecedent fault condition ($P \wedge Q$) with its consequent recommended maintenance actions (A). Of course, any number of predicates may be combined to create more complex expressions as needed.

C. Fault condition and maintenance rules

Inference within the first-generation *FaultReasoningAgent* is carried out using the propositional logic, proceeding from a *Knowledge Base* of rules using forward and backward chaining algorithms [4]. Rules construction is the third step of the knowledge base engineering process described in Section IV-A.

Fault conditions described in the FAULT table in ELAS are combined with the action codes listed in Table I, provide a minimal but sufficient set of data to construct a rule base using the propositional logic. As the Phase I research comes to a close, IAC is in the process of constructing a rule base to associate descriptive fault text with corrective actions. The process is labor-intensive, and involves analyzing records in the FAULT and ACTION_CODE tables, and the articulation and programming of expressive rules (of the form shown in Equation 2) to execute this mapping. These rules are in turn added to the rule base of the *FaultReasoningAgent*, and tested for accuracy by comparing the observed result upon rule firing to the anticipated result. As each rule is programmed, debugged and tested, the knowledge base continues to grow. All previous rules must be continually evaluated to ensure accuracy of the forward chaining algorithm with the newly-added rule.

An example display of intermediate results used in the rule construction procedure appears in Figure 6, which presents the results of an SQL query of the ELAS database. All faults and corresponding action codes for a specific aircraft have been returned. The FAULT field is expanded to display more detail of the fault descriptive text. The reader should note the heterogeneity of content exemplified in this field. This characteristic of the data presents significant technical difficulties for the knowledge engineer.

D. Ontogenesis

Strictly speaking, we consider an ontogenetic component as responsible for "managing the evolution of the knowledge base", and one aspect of that responsibility is embodied within the *OKBAgent* that was designed, developed and integrated during Phase I research. Ontogenesis implies much more than database connectivity, however–the vision of this project is to create a system that is self-evolving, that is, capable of augmenting its base of knowledge by consumption of new information that accumulates over time. Certain classes of learning algorithms are relevant to such an endeavor. In particular, we anticipate that reinforcement learning algorithms [6] will play an important role towards realizing ontogenesis of the *Ontogenetic Knowledge Base* in software. IAC has developed and tested prototype reinforcement learning components as stand-alone applications; these are currently being incorporated into the JADE Agent Architecture developed in Phase I and reported herein.

Reinforcement learning is intriguing because it is data driven (like forward chaining for propositional logic). "Data-driven" implies automation, after adequate training and eval-

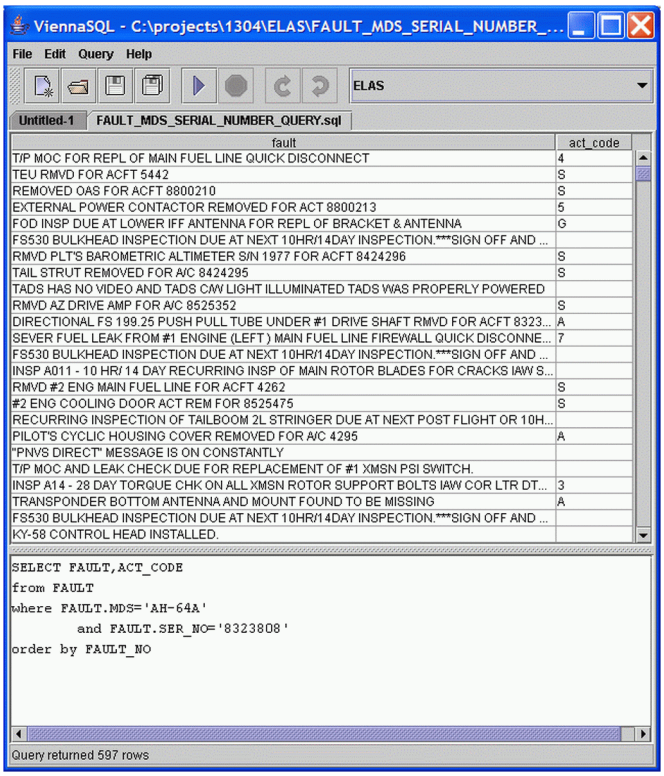

Fig. 6. . Results of specific SQL queries are analyzed to construct the OKB rule base. The FAULT field is expanded to display more of the returned text. Notice the heterogeneity of content in this field. The corresponding ACTION_CODE is shown.

uation of the system by human experts. Autonomy obviously is a focus of Autonomic Logistics for JSF.

The application of reinforcement learning to induction from databases is an exciting topic of machine learning research that naturally integrates with autonomous agent-based learning. For example, in [3] the idea is advanced that this type of learning may have utility in situations where the effects of the actions of agents are unknown.

V. Technical Challenges

Rule base development proved to be the most demanding technical task of this Phase I SBIR program, due primarily to several factors. Looking to future development, list the most significant challenges below. It is noteworthy that most of these are related to the area of natural language processing (NLP), generally considered to be among the most difficult sub-disciplines within the broader field of artificial intelligence. We anticipate that many of the technical challenges identified here are instrinsic to most aircraft maintenance databases in real-world use.

1) We have noted that *faults* and *corrective actions* are coincident within ELAS database records. This presents a minor technical problem to unravel the temporal relationship between a fault declaration and its correction.
2) The ELAS database was observed to contain many instances of redundant information, due to weak or non-existent ontologies setting bounds for descriptive information. For example, the ACTION_CODE table presents one ontology that maps action code to description (shown in Table I). The ACT_CODE field in the FAULT table provides a foreign key to this table. However, FAULT also contains an ACTION field with descriptive information that may be completely different than the ontology ostensibly represented by the ACTION_CODE table. Lack of referential integrity may invalidate assumptions underlying the propositional logic.
3) Many of the so-called "faults" were in fact notations that inspections were due in some time in the near future. This creates the need to parse the descriptive fields for certain keywords or word fragments that are semantically associated with inspections, in order to discriminate between inspections and faults.
4) ELAS includes multiple-record histories of actions relating to a particular fault condition over the course of time.
5) Faults are not an associated FAULT_CODE, which would simplify the conceptualization and programming of rules significantly. Faults are described by free-text annotations made by multiple maintenance technicians at different times, with apparently no formal rules for production of the descriptive content. This is a major technical obstacle.
6) It is likely that ELAS records contain erroneous information. Careful, retrospective analysis by domain experts must be carried out to identify errors in annotation, including misdiagnoses that may not be recognized until some time after the presumed corrective action (repair, replacement) has been completed.

VI. Conclusions

Under Phase I SBIR funding, IAC designed, developed and integrated software components of the *Ontogenetic Reasoning System*. The system design includes adaptive knowledge base of maintenance information; autonomous software agents which evaluate and interpret data, model outputs and past behavior to recommend actions under uncertain conditions, and handle external communications; an ontogenetic engine which manages the evolution of the knowledge base; and web-based agents to perform security functions and broker immediate communication between entities in the system.

We offer these conclusions drawn from our experience on this project.

1) The *Ontogenetic Reasoning System* architecture presented in this manuscript addresses many of the requirements and goals for Autonomic Logistics. Preliminary efforts have been concentrated on designing a system that will scale, and provide for the worldwide distribution and access of components and services without compromising security. This architectural design of the includes implicit support for internationalization. Therefore, international partners of JSF will have access to

the system functionality in their native language, with little additional programming required.

2) The system design and prototype components developed in Phase I will be immediately applicable to other related systems in Phase II. In particular, our plans include applying this system to a turbofan faults database with direct relevance to the Joint Strike Fighter.
3) Although not discussed herein, a central focus of future development is to encompass the supply chain side of the system. A copious supply chain management design is included within the Phase I Knowledge Base schema.
4) The system design presented in this report is sufficiently detailed to enable continual development by a team of software engineers, working in collaboration with domain experts in aircraft maintenance, logistics and supply chain management.
5) The ELAS database used for system development was an excellent resource for Phase I development, because its records contain free-text entries, redundant information, lack of ontological rigor, and errors in corrective actions for certain fault conditions. These are likely characteristics found in most maintenance databases, which are updated by diverse groups of individuals.
6) Software agents are a sensible technical solution for Autonomic Logistics. They operate within dynamic and uncertain data environments, are trivially distributed over a network, and may be programmed to carry out complex behaviors that are goal-oriented, providing a potentially high degree of autonomous processing. For self-evolving knowledge bases, the use of some variant of agent technology appears to be inevitable.
7) We anticipate that reinforcement learning algorithms [6] will play an important role towards realizing ontogenesis of the *Ontogenetic Knowledge Base* in software. These algorithms are "data-driven", facilitating automation, after adequate training. Autonomy is a focal point of Autonomic Logistics for JSF.

ACKNOWLEDGMENT

This work was supported by the Naval Air Warfare Center in Patuxent River, MD, under SBIR contract No. N683335-04-C-0179. JRB and TWB gratefully acknowledge the support and encouragement provided by our co-author, Mr. Doug Gass, and by Mr. Michael Begin of the NAVAIR Joint Strike Fighter Program Office.

REFERENCES

[1] F. Bellifemine, A. Poggi, and G. Rimassa, "Developing multi-agent systems with a FIPA-compliant agent framework," *Software Practice & Experience*, vol. 31, no. 2, pp. 103–128, February 2001.

[2] M. Bratman, D. Israel, and M. Pollack, "Plans and resource-bounded practical reasoning," *Computational Intelligence*, vol. 4, no. 4, pp. 349–355, 1988.

[3] S. Dzeroski, L. De Raedt, and H. Blockeel, "Relational reinforcement learning," in *Proceedings of the 8th International Workshop on Inductive Logic Programming (ILP '98)*, D. Page, Ed. Springer-Verlag, 1998, pp. 11–22.

[4] A. Goodall, *The Guide to Expert Systems*. Oxford, England: Learned Information, 1985.

[5] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Upple Saddle River, NJ: Prentice-Hall, 1995.

[6] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[7] U.S. Army, *Functional Users Manual for the Army Maintenance Management System–Aviation (TAMMS-A)*, United States Army, Washington, DC, March 15 1999.

[8] M. Woolridge, *An Introduction to MultiAgent Systems*. Chichester, England: John Wiley, 2002.