Agent Building and Learning Environment

ABLE 2.0 Overview

Joe Bigus, Ph.D.
IBM T.J. Watson Research Center
October, 29 2003
Outline

• ABLE Project overview
• Intelligent Agents
• ABLE Agent Framework
• ABLE Component Library
• ABLE Rule Language
• ABLE Agent Platform
• ABLE Development Tools
• Example Applications
• Tutorial: Differentiated Quality of Service
What is ABLE?

- A Java framework and toolkit for constructing hybrid intelligent agents (1999)
- A productivity toolkit for adding intelligent software functions to Java applications (2000)
- A toolkit for building multiagent intelligent autonomic systems (2002)
- A business rules and policy run-time framework
- “A RAD environment for agents”
- A commercial multi-agent platform
ABLE Project Overview

Data Mining
- Classification
- Clustering
- Prediction
- Optimization
- Neural Networks
- Decision Tree
- Naïve Bayes
- k-NN
- Genetic algorithms

ABLE Framework

Rules
- Scripting
- Simple if-then rules
- Rete’ net
- Prolog
- Fuzzy systems
- Planning
- Event Correlation
- Templates
- Author/Debug tools

Agents
- Autonomous
- Asynchronous event processing
- Timer-based processing
- Distributed multi-agent platform and admin console
- Conversational support
- Author/Test/Debug tools

5/12/2005 Able Overview
What Are Intelligent Agents?

 Agents are active, persistent software components that perceive, reason, act, and communicate. (Huhns and Singh)

- Software that assists people and acts on their behalf
- Agents can help people and processes
- Agents are used for automation and control

- finding and filtering information
- personalizing your environment
- negotiating for services
- automating tedious tasks
- taking actions you delegate
- learning about you over time
- collaborating with other agents
- capturing individual and organizational knowledge
- sharing knowledge

- finding and fixing problems
- automating complex procedures
- finding "best fit" procedures
- pattern recognition and classification
- predictions and recommendations
- negotiate and cooperate with other organizations' agents
Intelligent Agent - Abstract Architecture

Environment

Sensors

Perception

Attention

Decision-making

Plan

Domain Knowledge

Reasoning

Learning

Memory

Planning

Inference

Intelligent Agent - Abstract Architecture
What can ABLE do?

✓ Describe autonomic manager behavior using rules
✓ Learn from experience and predict future states
✓ Analyze sensor data using classification and clustering algorithms to detect complex states and diagnose problems
✓ Have stateful conversations with other autonomic elements
✓ Interface with other autonomic components via web services
✓ Reason using domain-specific application objects
✓ Use boolean forward and backward chaining, predicates, fuzzy
✓ Have autonomous (proactive) behavior and goals
✓ Provide tooling to build, test, and debug these kinds of agents
✓ Plan sequences of actions for an autonomic manager
✓ Correlate events into situations, reason, and take actions
ABLE
Agent Framework
AbleBeans – Java Agent Building Blocks

AbleBean, AbleRemoteBean: a Java interface (local and remote)
AbleObject: AbleBean instantiation with autonomous thread
Bean interactions: Direct method calls and event passing
AbleEvents: Notification and Action events with synchronous and asynchronous event handling
AbleBeanInfo and Customizer required for use in Agent Editor
Set of core data access and algorithm beans supplied
AbleAgents – Intelligent JavaBeans

**AbleAgent, AbleRemoteAgent:** a Java interface (extends AbleBean)

**Composable:** can contain other AbleBeans and AbleAgents

**Sensors and Effectors:** Allow agents to interface with apps

Can be distributed, synchronous or asynchronous (autonomous)
AbleBean Source File

import com.ibm.able.*;
public class SampleAbleBean extends AbleObject implements Serializable {

    public SampleAbleBean() throws RemoteException {
        this("SampleBean");
    }

    public void init() throws RemoteException {
        // need to initialize state of this bean, algorithm vars, etc. -- do ONE TIME initializations
        // initialize asynchronous Timer (if used) and define Event processing behavior
    }

    public void process() throws RemoteException {
        // perform synchronous processing on caller's thread
    }

    public void processTimerEvent() throws RemoteException {
        // perform autonomous (asynchronous) processing on own thread
    }
}

5/12/2005 Able Overview
ABLE
Component Library
ABLE Component Library

Agents
- Classification
- Clustering
- Prediction

Machine Learning
- Back propagation
- Self organizing maps
- Radial Basis Functions
- TD-Lambda
- Decision Trees
- Naive Bayes

Machine Reasoning
- Script (procedures)
- Forward / Backward chaining
- Predicate logic (Prolog)
- Rete'-based pattern match
- Fuzzy systems
- Planning (STRIPS)

Data Access/Analysis
- Text/DB read/write
- Cache, Filter, Transform
- Statistical routines
- Genetic algorithms
- other math analysis

Autotune (closed loop control)
Storage manager (multiple QoS)
ABLE Application Design

ABLE Library

ABLE Core Beans

Custom Beans (domain-specific)

Application

Agent
AbleBean Wrapper Design Pattern

- Allows easy integration of existing Java algorithms into the Able environment
- Requires creation of 3 Java classes, Bean wrapper, BeanInfo and Customizer
- Bean contains an instance of the algorithm and calls methods on it
- No (or minimal) source changes required in the algorithm class
ABLE  Rule Language
Ingredients of a Modern Rules Environment

- Authoring Tools
- Testing Tools
- Debugging Tools
- Templates
- Rule Logic Partitioning
- Powerful Inference Algorithms
- Procedural Scripting
- Easy to Integrate
- Optimization Algorithms
- Machine Learning
- Flexible: Choice of Rule Engines
- Scalable Run-Time
- Tracing/Logging
- Extensible
ARL Design Objectives

- A scripting language for AbleBeans and AbleAgents
- A business rule and policy specification language
- Very tight integration with Java objects and code
- Allow mix of procedural scripting and declarative inferencing as required
- Rich set of inferencing algorithm support
- Pluggable inference engines – optimized for run-time requirements
- Java-like text syntax and XML Schema representation
Language Concepts

• import and use Java classes, fields, methods
• Domain-specific user-defined function libraries
• Global variables, input/output variable lists
• Rules: assertions, if-then, if-then-else, when-do, while-do, for-loop, predicates
• Rule blocks – sets of rules and inference engines
• “Pluggable” Inference Engines – control algorithms
• Working Memory
• AbleRuleSet bean is run-time object
**ARL RuleSet Structure**

<table>
<thead>
<tr>
<th>DATA</th>
<th>RULES</th>
</tr>
</thead>
<tbody>
<tr>
<td>import &lt;package&gt;.class;</td>
<td>init() ruleblock</td>
</tr>
<tr>
<td>library &lt;package&gt;.class;</td>
<td>processTimerEvent() ruleblock</td>
</tr>
<tr>
<td>variables {} ;</td>
<td>preProcess() ruleblock</td>
</tr>
<tr>
<td>predicates{} ;</td>
<td>processAbleEvent() ruleblock</td>
</tr>
<tr>
<td>functions{} ;</td>
<td>process() ruleblock</td>
</tr>
<tr>
<td>inputs{}; outputs{};</td>
<td>catch() ruleblock</td>
</tr>
<tr>
<td></td>
<td>finally() ruleblock</td>
</tr>
</tbody>
</table>

<myName>() ruleblock...<myName..N>() ruleblock
ABLE RuleSet Example

ruleset SimpleBusinessRuleExample1 {
    import com.ibm.able.examples.rules.Customer ;
    variables {
        Customer customer ;
        Boolean seniorCitizen = false ;
    }

    inputs { customer };
    outputs { seniorCitizen };

    void process() using Script {
        SeniorTest: if (customer.age > 55) then seniorCitizen = true ;
    }
}
ABLE Inference Engines

• Script - procedural
• Fuzzy – forward chaining using Fuzzy logic
• Forward – data-driven inferencing
• Backward – goal-driven inferencing
• PatternMatch - Forward with working memory
• PatternMatchRete - Forward with Rete’ network
• Predicate – backchaining with backtracking
• Planning – classical AI planning engine (STRIPS)
Built-in Data Types

- **Boolean** – Java boolean
- **Byte/Short/Integer/Long** – Java byte/short/int/long
- **Character** – Java char (unicode value)
- **Double/Float** – Java double/float
- **String** – Java String
- **Object** – java Object
- **Categorical/Discrete** – set of Java Strings/doubles
- **Continuous** – double with min/max limits
- **Fuzzy** – linguistic variable with FuzzySets
- **TimeStamp** – Java Calendar date/time object
- **TimePeriod** – IETF compliant TimePeriod Conditions
- **Selector** – query with constraints
- **Expression** – holds a logical or math expression for use in rules
Rule Blocks

\[ \text{<type> <name>() using <engine> { ruleList } ;} \]

- Semantically equivalent to Java methods
- Can specify a return data type
- Can use pre-defined or user-defined name
- No formal parameter lists, use global vars
- Specify inference engine via using <engine> clause
- <engine> can be any AbleInferenceEngine Java subclass
- Body of ruleblock contains one or more Rules
- Use setControlParameter() built-in function to set goals, options, etc.
- Ruleblock can have local or shared working memory
ARL Rule Syntax

<ruleLabel> { preConditions } [priority] : <ruleBody>;

- ruleLabel – unique identifier in ruleset
- preConditions – list of Java objects (e.g. TimePeriods)
- priority – used in conflict resolution during inferencing
- Rule body must be one of the ARL rule types
- myRule { weekdaysOnly } [ 3.0 ] : println("wow");
ABLE Rule Templates

- Allow IT Developer or Programmer to create rulesets and templates using WSAD editor
- Minimize external meta-data or artifacts
- Business user can create rules from templates using web-based UI
- Allow easy parameterization of rules and rule logic, with constraints on parameter values
- Reuse existing ABLE data types and ARL syntax

- Allow users to customize rule templates and create new rules
- Variable values are constrained based on ruleset author constraints
- Can generate individual rules or entire rulesets via templates
- Can edit generated rules using same authoring environment
ARL Rule Template Syntax

Ruleset myRuleTemplateExample {
  import com.ibm.myclass.Customer;

  variables {
    Customer    customer = new Customer() ;  // myclass type
    template Categorical customerLevel = new Categorical("gold", "silver", "platinum");
    template String salesMsg = new String("Thank you for shopping IBM");  // example msg
    template Continuous discountValue = new Continuous(0.01, 0.50); // allow range from 1% to 50%
    Double        discount = new Double(0.0) ;
  }

  inputs {  customer } ;
  outputs { discount } ;

  void process() {
    Rule1: if (a > b) then println("regular old rule") ;
    Rule2: if (a <= b) then println("another regular old rule") ;

    template myRuleTemplate1: if ( customer.level == customerLevel )       // NOTE: Rule is a template
      then { discount = discountValue ;
        println( salesMsg ) ; } 

  }
}
ABLE Rule Templates Demo

Select an ABLE Rule Language file to load:

Load File  TemplateExample1.arl

Reset  View Source  View XML

Rule Templates

discountTemplate Discount rule template, style 1

discountTemplate

Rules Generated From Templates

discountTemplate Discount rule template, style 1

Edit

Done
Web-based UI for ABLE Rule Templates

Create Rule From Template

Template Description: Discount rule template, style 1

Rule comment: generated from template
Rule label: nowRule2
Select type of item: Wine
Specify the customer’s age: 21
Select the customer’s nationality: German
Desired discount (percentage): 15.0
Create a new Rule instance

Rule text:

```java
/** A discount rule template */
template discountTemplate[1.0]:
    if ((shoppingCart.contains(Item) && (customer.age > Age)) &&
        (customer.nationality == Nationality))
        then discount = Percentage;
```

Done
Template: discountTemplate successfully created new rule from template

/** generated from template */
newRule2[1.0]: if ((shoppingCart.contains("Wine") && (customer.age > 21)) && (customer.nationality == "German")) then discount = 15.0;

Back to Main Page
ABLE
Agent Platform
Example agent platform

Key
- Physical system
- JVM (separate job/process)
- Platform service
- Agent

System1
- Host1: 5001
  - Agent 1
  - ADS
  - LCS
- Host2: 5002
  - Agent 2
- Host3: 5003
  - Agent 3

System2
- Host4: 5001
  - Agent 4
  - Agent 5
  - Agent 6
  - Agent 7
  - ALS

System3
- Host5: 5001
  - EJS
  - MTS
  - ANS
- Host6: 5002
  - Agent 8
Administrative Console

• Manage agents running on remote systems

• Tool for administrator to manage or oversee many agents running on the distributed platform.

• View all agents by type, system, or domain

• View agent requests

• Set level of autonomy for each agent
Autonomic Agents

Hierarchical

Peer-to-Peer
Autonomic Agents

Hierarchical

Peer-to-Peer
ABLE
Development Tools
ABLE Rule Language Source Level Debugger in Eclipse
Swing Agent Editor

Bean palette
Canvas contents
Canvas area
Bean
Container agent name
Bean tree
Status line
State indicator
Connection
BackPropogation
OutFilter
Import
Connection
InFilter
TestImport
Import
NeuralClassifierAgent
Untitiled
Untitled
File Edit View Tools Windows Help
Step Cycle Run Halt
Process buttons
Serialized file name
Able Overview 40
ABLE Agent Editor in Eclipse using GEF
ABLE
Applications
System Health Monitoring using Fuzzy Rules
Agent-Based Diagnostics

- New Case
- Agent-Vehicle Dialog Mgr
- Case Retrieval
- Case Modification
- Case Learning
- Diagnostic Manager
- Diagnostic Agents

Case Base
Autotune Agent Web-Tuning Scenario

Agent Properties
- Flexible
- Autonomic
- Generic

AutoTune Agent
- Modeling
- Run-time Control

KeepAlive
MaxClients

CPU
MEM

Users

Desired Utilization Level

Apache Web Server
Design Phase I: System Modeling
eSupport Intelligent Agent - phase n (IA_n)

IBM Service Engineer (or customer)

iSeries

Manager (PTF analysis)

Task (PTF Finder)

Task (PTF Installer)

Task (Comm Trace Taker)

Task (Gross Comm Trace Analyzer)

Task (Fine Comm Trace Analyzer)

Manager (comm failure)

Manager (job failure)

Director

Monitor

Resource

Task (event analysis)

Task (msg analysis)

Task (System Data)
iSeries System Administration using ABLE

SysAdmin Agent

SysAdminActions RuleSet

SysAdminBrain RuleSet

Task/Info Agents

CPUWatcher

DiskWatcher

DiskPredictor

NOJWatcher

FindRunawayJobs

Cleanup

FindLargeObjects

findDuplicateJobs

Action Agents

Able System Admin Client

Client
Performance Prediction using Neural Networks

- WebServer running on Windows 2000
- Hit with variable workload, seasonality
- Capture Performance Monitor Data
- Train neural network to predict future response time
For More Information

• ABLE 2.0.1 driver posted to alphaWorks in October
• ABLE toolkit drivers and on-line info available at IBM alphaworks: www.alphaworks.ibm.com/tech/able
• IBM external project page at: www.research.ibm.com/able
• Java Agent Services (JAS) webpage: http://www.java-agent.org/
• FIPA webpage: http://www.fipa.org/
ABLE
Tutorial
Differentiated Quality of Service

- Silver Policy
- Gold Policy
- Platinum Policy

SAN Manager

Silver Customer

Gold Customer

Platinum Customer

SAN Storage
SAN Manager Scenario Overview

Uses new AbleRuleAgent as rules-based policy manager
Models multiple quality of service levels (represented by rule sets)
$N$ systems are defined, each with associated QoS levels
Requests include system identifier and current utilization
The SAN Manager:
  Looks up QoS for that system
  Invokes the corresponding QoS rule set
  Rule sets make recommendations that allocations are either unchanged, increased or decreased
  SAN Manager evaluates recommendations and changes allocations
    based on total capacity limit
Platinum QoS RuleSet

// Low allocation
: if Allocation is Low and Utilization is Low
  then RecommendedAction = NoAction;
: if Allocation is Low and Utilization is Normal
  then RecommendedAction = NoAction;
: if Allocation is Low and Utilization is High
  then RecommendedAction = IncreaseAllocation;

// Normal allocation
: if Allocation is Normal and Utilization is Low
  then RecommendedAction = DecreaseAllocation;
: if Allocation is Normal and Utilization is Normal
  then RecommendedAction = NoAction;
: if Allocation is Normal and Utilization is High
  then RecommendedAction = IncreaseAllocation;

// High allocation
: if Allocation is High and Utilization is Low
  then RecommendedAction = DecreaseAllocation;
: if Allocation is High and Utilization is Normal
  then RecommendedAction = DecreaseAllocation;
: if Allocation is High and Utilization is High
  then RecommendedAction = Send.Warning_LowMem;
: if Allocation is positively High and Utilization is positively High
  then RecommendedAction = Send.Warning_CritMem;