RC 20305 (12/05/95)
Computer Science

IBM Research Report

Reusable Architecture for Embedding Rule-based Intelligence in Information Agents

Benjamin N. Grosof, David W. Levine, Hoi Y. Chan, Colin J. Parris, Joshua S. Auerbach

IBM Research Division
T.J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598
(914) 784-7100 main
Internet: {grosof,levine,hchan,cjparris,jsa}@watson.ibm.com
(alt.: grosof@cs.stanford.edu)
WWW: http://www.research.ibm.com

IBM Research Division
Almaden · T.J. Watson · Tokyo · Zurich

Publication Information: *This paper is a reformatted version of the paper appearing in the Proceedings of the ACM CIKM-95 Workshop on Intelligent Information Agents, held Dec. 1–2, 1995, Baltimore, MD. Workshop held in conjunction with the ACM Conference on Information and Knowledge Management. Co-chairs of Workshop: Tim Finin and James Mayfield. Information and on-line proceedings are available via the World Wide Web: http://www.cs.umbc.edu/iia/*

Permission to publish for the above Proceedings is non-exclusive; copyright for the version of this paper appearing in it has been retained by the authors.

Abstract

We identify practical software design requirements for rule-based intelligence in the next generation of commercial information agents. Besides basic inferencing, these include embeddability, reusability, user-friendly authoring of rules, communicability of rules, flexibility especially of inferencing control strategy and performance, and extensibility of representation and reasoning.

We develop an architecture that fulfills these requirements to a substantial degree: RAISE (Reusable Agent Intelligence Software Environment). RAISE provides building blocks for embeddable agent smarts. It is founded upon a declarative representation and clean semantics, equipped with a simple yet powerful approach to procedural attachments. This results in highly pluggable components for inferencing, authoring, and communication, embodied in a fine-grained object-oriented class library. We have found RAISE to enable high reusability of both code and knowledge while embedding rule-based intelligence enhancements in three prototyped information agent applications: personal messaging, newsgroup filtering and handling for customer service support (the Globenet system), and collaborative news service in Lotus Notes.

Keywords: intelligent agents, rule-based, architecture, reusable, embeddable, information retrieval, artificial intelligence, software engineering, WAN, network newsgroups, collaboration support.

Version Note: This is a **preliminary** report in the form of an **extended abstract**, quite condensed and still somewhat rough.

Revised Versions and Follow-ons of this paper can be found via the World Wide Web: http://www.research.ibm.com . Navigate to on-line Research Reports ("CyberJournal") and to Project Pages under "Computer Systems".

1 Architectural Requirements for Commercial Settings

The problem we address is practical design and architecture of rule-based intelligence in information agents, for commercial settings in the not-too-distant future (e.g., within the next five years or less).

Information access is currently one of the commercially most important application realms for intelligent agents. Most industry forecasters expect it to continue to be for at least the next five or ten years.

Another, emerging, dominant view among industry forecasters is that the main business value of intelligent agents in the short- to medium-term time horizon will be as parts of overall applications, rather than as stand-alones. That is, the value of an intelligent agent will typically be as an augmentation (differential, delta) upon the value of an application which it enhances. This is the case in today's commercial intelligent agents scene.

1.1 Embeddability

This view about the structure of *business value* implies that for intelligent agent *technology*, there will be a premium on embeddability.

Embeddability includes several aspects. One is the importance of the attachment of inferencing on the one hand to procedural invocations on the other hand, e.g., invocations that perform actions. Another is the ability to symmetrically interoperate with other software: both to call, and be called by, other procedures. A third is the importance of a lightweight option, and more generally, of flexibility with respect to performance and scaling.

1.2 The Useful Roles of Rules

We observe that rules are useful as a complement to search and retrieval techniques, in particular to conventional free-text information retrieval techniques and their underlying (relatively shallow) natural language analysis (e.g., keywords and phrases). Rule-based reasoning is especially helpful for controlling filtering and handling actions (we will list these towards the end of section 4), based on structured (e.g., mail headers which are essentially relational data) aspects of information items as well as unstructured (e.g., text bodies) aspects of those items. Rules and reasoning are more accessible to users, especially non-technical users, than scripting or macro languages, yet more powerful than menus and direct manipulation in terms of the complexity of behavior that they can specify.

Machine learning offers the hope that the process of a user instructing her agent can be changed from explicit to implicit. As a pragmatic first step, however, the capability for explicit instruction is very important. It enables confirming, editing, and augmenting the results of any available learning mechanisms. A historical engineering lesson in knowledge-based systems and machine learning is that it is usually advisable first to work out how to explicitly instruct a performance element before hooking up a learning element.

As a first step within explicit instruction, yes/no (as opposed to probabilistic-flavor) rules are very important. These are well-understood in terms of both knowledge-based applications and fundamental theory. Yes/no rules enable the control of tasks with predictability. Probabilistic-flavor knowledge will, of course, be very important in the future with the wider use of inductive learning mechanisms, since they yield statistical-flavor knowledge as output.

1.3 Additional Requirements

Next, we discuss additional practical software design requirements for rule-based intelligence in the next generation of commercial information agents, beyond basic inferencing plus the requirements (notably, embeddability) that we discussed above.

A premium on embeddability implies a premium on **reusability of code**. From an investment viewpoint, **reusability of knowledge bases** as well is vital because of the difficulty of knowledge acquisition. This implies a premium on the **communicability of rules and knowledge**. Communicability implies a need for **clean semantics and standards**.

Also, communicability of facts in particular is important for basic interoperability of multiple agents, so that one agent can output facts (perhaps derived) as input to another agent.

In large part, the short- to medium-term commercial vision for the value of intelligent information agents is that they will enable non-technical end-users to personalize the behavior of their applications. Therefore, more specifically with respect to knowledge acquisition: **user-friendly authoring of rules** is crucial, so as to enable those users to specify the behavior they desire.

While chaining as a requirement is an article of faith for many in the AI or logic programming communities, it demands a rationale in this context. Many current rule-based commercial intelligent information agents do not enable chaining of rules.

We observe that **chaining** is important for conciseness of rule sets and for reusability in rule authoring, even in relatively small or shallow rule sets. Intermediate conditions, such as Importance (high vs. low) or Topic (topical categorization), enable multiplicative factoring of a set of rules. For example, consider a rule set about filtering and handling e-mail. There are, say, 5 rules about which e-mail is important based on its internal attributes. That is, these 5 rules fan in to Importance as their consequent. In addition, there are, say, 4 rules about what actions to do with mail based on its importance. That is, these 4 rules fan out of Importance as their antecedent. Alternatively, one might equivalently represent the mapping from internal attributes to actions via a more "compiled-form" rule set without chaining. This would take 20 rules. 9 rules with chaining thus serve equivalently to 20 rules without chaining, More generally, even one level of chaining enables one to reduce the number of rules from $m \times n$ to $m + n$ instead.

We observe that some applications require forward inferencing, some require backward inferencing, and some need the combination to achieve efficiency. **Flexibility of inferencing control strategy** is thus important.

Finally, **extensibility of knowledge representation and reasoning** is important. As time goes on, learning and probabilistic knowledge will become increasingly common, community knowledge bases will evolve, and more ambitious applications will be undertaken.

Contrast with Expert Systems, Logic Programming, etc.:
The above collection of requirements contrasts with what previous rule-based intelligence technology offers. The software previously available for expert systems, Prolog-type logic programming, OPS5-type production rule systems, and mathematical theorem-provers each have major weaknesses, especially with regard to meeting the conjoined requirements of: embeddability, flexibility of inferencing control strategy, weight, combining clean declarative

treatment with powerful procedural attachments, and pluggability to interoperate or to extend reasoning functionality.

2 The RAISE Architecture and Applications, Prototyped

We develop an architecture that fulfills these requirements to a substantial degree: RAISE (Reusable Agent Intelligence Software Environment). RAISE provides building blocks for embeddable agent smarts. At its heart, RAISE's functionality includes:

- Inferencing and reasoning, including both forward and backward chaining of yes/no rules, as well as the ability to mix forward and backward inferencing. Forward inferencing is especially important for applications in which an information agent is triggered by events such as arrival of new information items. Backward inferencing is especially important for mediator-type functions such as intelligent directory services and brokering. Mixing forward and backward inferencing is especially important to enable computing or obtaining information only upon demand in the course of forward inferencing. E.g., in newsgroup filtering and handling, it is inefficient to compute the presence of all possible interesting keywords (i.e., those mentioned in some rule) in each incoming newsgroup item. Rather, it is better to compute the presence of keywords only after being constrained by already having tested some conditions in rules, in the midst of primarily forward inferencing.

- Knowledge representation and reformulations of representations. The end-user's representation of knowledge is distinct from the operational form used by engines for the inner loop computations of inferences. E.g., we find it helpful to permit an end user to specify a "extended rule" syntactic form of belief in which there is a top-level implication connective, but the antecedent may contain both logical and's and or's that connect atoms, and the consequent may contain and's that connect atoms. In our experience with users, they have found this to be relatively natural and concise. RAISE includes the capability to reformulate such extended rules into Horn rules.

- Procedural attachments ("reflexes") for not only actions ("effectors") but also special evaluations ("sensors"). A **reflex** is treated as a special primitive entity, akin to a purely declarative (i.e., belief) rule in the form of a binary Horn clause. The reflex associates a logical atom (often containing free variables) with a procedure (e.g., method) call.

  - An **effector** reflex invokes an attached procedure for the sake of its side effect. E.g., an effector reflex might associate the belief-status atom $should\_keep\_in\_folder(\$message, AutoStuff)$ with the attached procedure call $SAVE\_To\_Folder(\$message, /Userid/autos/mail)$ that, when invoked, actually saves the message in the folder. Effector reflexes are "run" after rule consequences are derived.

- A **sensor** reflex invokes an attached procedure for the sake of its return values. It is used to answer a query, returning either a boolean or, more generally, an answer set (binding lists). Sensor reflexes are "run" when rule antecedent conditions are being tested. E.g., a sensor reflex might associate the belief-status atom *body_contains_keyword*($\$message, \$string$) with the attached procedure call $KEYWORD\_IS\_PRESENT(\$message, \$string)$ which returns True exactly when the message's textual body contains the string.

RAISE is founded upon a declarative knowledge representation and clean semantics, equipped with a simple yet powerful approach to procedural attachments. This results in highly pluggable (substitutable, recombinable, extensible) components for inferencing, authoring, and communication. RAISE has the form of a fine-grained object-oriented class library, of which a first phase of components have been implemented in C++. RAISE has been designed to be object-oriented from the ground up. Its design has a deep, fine-grain decomposition into objects, as opposed to only shallowly wrappering large "black boxes" as objects. For example: Horn rule, predicate, term, and atom (we mean these in the first-order logical sense) are each a class. The RAISE architecture not only enables reusability of code among different applications, but also reusability and sharing of knowledge bases between multiple agents, e.g., within a single application. Inter-agent knowledge sharing and communication capabilities support the ARPA Knowledge Sharing Effort and extend simply to other emerging communications standards such as HTML (Hyper-Text Markup Language) and OMA (Object Management Architecture, the extension of CORBA).

RAISE includes a number of features that ease authoring:

- cleanly declarative semantics that facilitates predicting rules' behavior, as well as maintaining, updating, and merging rule sets
- the concept of a rule *set* as a first-class notion; support of multiple rule sets per user
- extended rules (discussed above)
- "canned" rule schemas and rule-set schemas which a user can simply select and parametrize in order to create his rule set, rather than having to deal with the often more difficult mode of free-form rules or rule sets
- interfaces to graphical rule editors

The components in the RAISE library can be combined and extended to form many different overall configurations.

We discover one particular configuration of the RAISE architecture to be useful for a group of several different application areas within the overall arena of information agents. In this configuration, a dynamically-supplied user rule set drives the processing of an input stream or collection of mail-like information items. The processing steps include not only search and retrieval but also other kinds of filtering and **handling functions**: e.g., categorization, attentional prioritization, personal information management, and selective dissemination (routing, forwarding, sharing). Inferencing is primarily forward, triggered by incoming items; however, query-answering is interleaved upon demand, e.g., to perform keyword analysis via a sensor reflex. An application-specific adapter represents each incoming item as a set of facts. This set of facts is asserted to the inferencing engine. A knowledge library, fed by the user via an editor, supplies the working knowledge base, which is updated by derived facts. The code is relatively lightweight, less than 10,000 lines of C++.

We have found this configuration to be nearly completely reusable with only minor modifications across three different intelligent information agent applications that we have prototyped. Each of these applications existed in another prototype version, written by others, before we enhanced it with embeddable rule-based agent intelligence using RAISE. The first application is a personal messaging (mail) service. The second is a network-newsgroup filter and handler. This extends a system, Globenet, that has been deployed for customer service support within IBM. The third is a collaborative news service for Lotus Notes.

The appended Figure 1 illustrates this configuration as used in Globenet.

3 Future Directions

Future directions we are exploring for applications include collaborative information sharing and workflow in Lotus Notes, matchmaking consumers and suppliers of computational or economic services, and network systems management.

Our major current effort to augment RAISE itself is in the area of rule authoring. We are pursuing several avenues in combination: graphical and forms-based representations and interfaces, simple natural language processing for textual representations and interfaces, and conflict management.

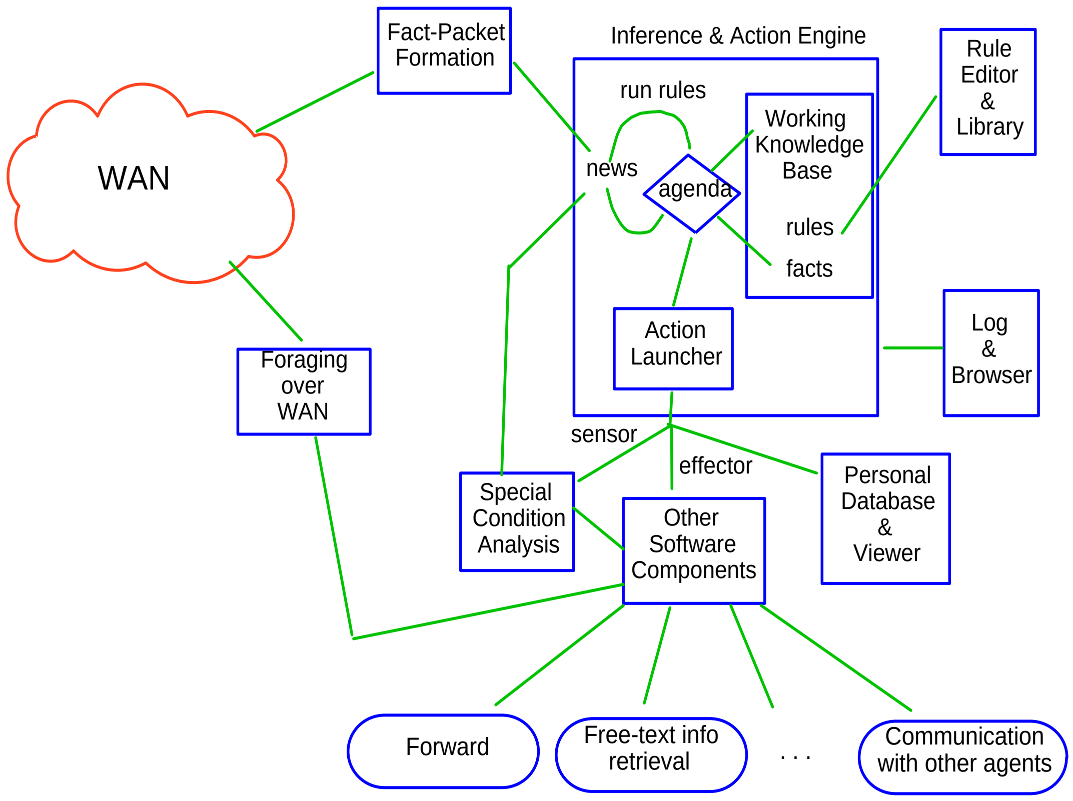

Figure 1: RAISE Architecture — First Configuration

Acknowledgements

Thanks to all the other people in our past or present group who have contributed to RAISE, especially Stephen Brady, Colin G. Harrison, and Yannis Labrou. Thanks also to our collaborators, especially Davis Foulger and Terry Heath, as well as Manny Aparicio, Tad Davis, and Rick Spagna.