Web-Based Health Care Guideline Agents: The Case of <u>R</u>eminders of <u>T</u>o<u>Do</u>s-<u>II</u> (R2Do2)

by

Barry G. Silverman, Ph.D.
Christo Andonyadis
Alfredo Morales
In-Kuk Song
Institute for Artificial Intelligence
George Washington University
Washington DC 20052
http://www.seas.gwu.edu/seas/institutes/iai
barry@seas.gwu.edu

February 1997

KEYWORDS
Intelligent Agents; Computer-based Health Care Guidelines; Enterprise Architecture

ABSTRACT

This paper describes efforts to develop and field an adaptive health planner and parallel agents that securely connect distributed users and data sets in a collaborative approach able to anticipate health ToDo items and to remind and alert about these items over the web. The planner and agents, reasoning temporally and non-monotonically, are able to handle changing health conditions, unknown health facts, user wellness and lifestyle preferences, and updates about each user's situations. Usage of and extensions to argumentation, qualitative decision making, situational calculus, and temporal algebras are described. Defeasible assertions, default arguments, support sign dictionaries, closed world assumptions, and anytime algorithms are used to derive feasible health plans that are later updated as more information unfolds. A set of robust, open standards-based client server approaches (e.g., CORBA, ODBC, Java, and ANSI HL7 Common Object Model) are used to encapsulate and plug all these components into the web, and to handle the distributed agent and object communications. Extensions to these emerging standards are at times made due to engineering and domain considerations, however, the goal is to maximize reuse of published guidelines and integration of R2Do2 by any patient record system that also adheres to the standards. From this perspective, R2Do2 is an experiment in an open standards framework for middleware in the healthcare field. This research also tries to reflect lessons learned about the extensions needed in these standards if healthcare middleware frameworks are to transparently support users over the web.

1) Introduction

Several related paradigm shifts are occurring in the health care field to produce an opportunity for a dramatically more cost-effective delivery system. First, medical practice is shifting toward more prevention, health promotion, shared patient-provider decisionmaking, and managed-care. This shift is creating a greater demand for online services and information. Second, distributed, heterogeneous patient data and isolated applications are becoming less tolerated in view of progress in health informatics standards, the information highway, and a proliferating set of access devices. This implies health system users want a shift toward usable interfaces, and easily accessed, integratable information sets. Third, due to cost and quality control concerns, inspection, rework, and paper intensive processes are giving way to reusable electronic information sets that are machine-checked for omissions and to get the decisions right the first time. Lastly, to meet these demands, applications developers are attempting to shift formerly closed, proprietary applications toward open, rapidly customizable software. This, in turn, will further democratize health information delivery.

To enable these types of paradigm shifts, one would like to utilize the Internet and the World Wide Web ("the web"), since it is widespread, supports cross platform interconnection, promotes heterogeneous information systems, and has an installed base of open standards based software browsers and distributed servers. Yet, the web alone suffers several drawbacks. Among others, the current obstacles to using the web include:

1) It is largely document-centric and ignores the relational technology that is so central to much of the patient record and personal health information systems of the medical sector. It is important to explore how best to utilize plugins, object resource brokers, and other open database interfaces to integrate the document- and data-centric worlds,

2) The web is largely a browse and surf environment in which users search for information and then "pull" it out. An alternative is for the web to proactively anticipate information needs of the users and to "push" it toward them. This is the agent approach, in general. In the health care field, an important type of proactive agent is computer generated reminders and alerts. The advent of the Java programming language and the inclusion of applets in the hypertext markup language opens up the prospects of agent push approaches. An open question is how can the methods of artificial intelligence, software agents, and Java be scaled to the needs of large communities of health care users.

3) The web, being a document-centric environment, is often devoid of active social presence. Yet both consumers and providers require a high degree of human contact for health care services, consults, support groups, and the like. This is beginning to shift, particularly as web-enabled browsers begin to extend support for cooperative environments such as email, graphical bulletin boards, electronic white boards, chat groups, and interactive video. This is a broad topic, and this paper will only briefly examine it from the view of how agent push approaches can be integrated.

4) The web and the Internet grew to prominence largely on their openness and how they fostered democratic values, freedom to roam, and barrier elimination. Yet health care involves sensitive data, private information, and “firewalls” that assure greater security and confidentiality. The latter must be incorporated while sacrificing as little of the former as possible.

Given the paradigm shifts in health care, progress is needed to reduce each of these four sets of obstacles. At present they present a collective challenge to any health-related project, and this article reviews how one project is trying to address them with particular emphasis on 1 and 2. Specifically, this paper describes a reusable intelligent agent project that seeks to unify the data- and document-centric views and to provide a push approach in a collaborative and secure environment. This agent approach is being built with open standards-oriented technology that is maturing at present, but is expected to be widely implemented in the near future.

The system we are developing, called Reminders of ToDos-II (R2Do2), offers a model for how to meet the four challenges in the area of health care planning in general, and for reminders and alerts, in particular. Medical reminder and alert systems organize and display machine-triggered ToDo items, reminders, and alerts. Studies show these machine generated reminders and alerts significantly increase performance of clinicians as well as consumers/patients, and if widely deployed they hold the potential to save the health industry billions of dollars annually through better disease/error prevention and health promotion: e.g., see Shea et al. (1996), Einarson (1993). In the health care arena, this approach also holds the ability to empower consumers to better manage their own wellness efforts, and to enable clinicians to reduce omissions and unintended medical errors: Safran & Rind (1996). Further, it should enhance communications and collaboration between health care consumers and providers. More generally, this paper illustrates an approach to adaptive planning for personal assistant agents.

This article reviews these innovations and identifies how R2Do2 serves as a model for further unification efforts. Specifically, Section 2 presents the architectural view of R2Do2 and shows how the four web-related challenges are addressed for both institutional and personal versions of R2Do2. This raises a number of challenges for scaling the AI algorithms and agents and for incorporating numerous open software and data standards as are discussed in the next two sections of the article, including the adaptive planner (Section 3), and the temporal reasoning, reminder agents (Section 4). Finally, Section 5 returns to the topics of the Introduction with lessons learned and future research needs.

1.1) **Healthcare Guideline Agents**

The idea is not new of online alerting and reminding from electronic practice guidelines able to interact with a medical record and with providers’ orders. Early examples may be found in McDonald (1976) and Warner(1972), while more current examples are described in Musen et al. (1996), Safran & Rind (1996), Shea et al. (1996), and many others. However, for a variety of reasons, very few of these systems have made it out of the academic center, hospital, or clinic where their developers have piloted and tested them.

The major obstacles to portability seem to be severalfold: (1) the reminder methods and engines are not generic, but are written in a variety of languages and according to arbitrary, undocumented standards so that the code is difficult to modify and maintain; (2) the reminder methods and engines are hard-wired to “legacy” or institution-specific record systems with local lexicons and interface code; (3) the user or screen interfaces are similarly hard-wired and non-standard; (4) the guideline KBs are specific to the home institution that authored them and may not be desired by others without modification; and (5) the programmers who created the code are frequently unavailable to modify it later.

Some vendors offer new, turnkey patient record systems that try to overcome such obstacles. For example, Healthpoint ACS (anon. 1996) is a patient record system for small private practices that includes an integrated module that scans all prescription orders against other drugs on the patient’s medication list as well as against the patient allergy list. Right out of the box, its conditions rulebase reminds the user when contra-indications and adverse interaction effects will arise. One can add to or edit these rulebases to make them institution-appropriate. Unfortunately, such vendor-specific systems are a necessary, but not sufficient, step to overcome the obstacles. This is because most information systems already exist, and they won’t easily be converted to a given vendor’s implementation, nor do their institutions wish to do so.

Another answer is to connect applications to middleware so that others may view the application as a component that they may embed and or plug in by also connecting to the middleware. One manual approach to this is where component or object vendors have taken to “wrapping” legacy information systems and adding their applications via integration interfaces. An example is Multum Inc’s MediSource expert system for drug therapy and order-entry (Schrier, 1994). Originally a stand-alone expert system, MediSource now offers a consulting service that manually embeds their software as an agent obtaining order entries from the legacy system records, and submitting its alerts to the clinician through the familiar legacy interface. This is a solution that could be duplicated with practice guideline knowledge bases (rather than just drug therapy), however, it fails the test of open standards-based software, and is once again vendor-dependent. This solution does not scale well or quickly when trying to expand to each new site in a given healthcare community, when adding other potential knowledge bases or agents, or when extending over the distributed, longitudinal, heterogeneous “record.”

The alternative pursued by R2Do2 is to build towards a middleware architecture, called HOLON (Silverman et al., 1996), that adopts interoperability and open-software standards. R2Do2 is a test application of the HOLON project meaning both R2Do2 and HOLON are being evolved in parallel. HOLON uses a three tier middleware architecture. This idea is an “any-any” model to connect any client to any database system via an open-standards middleware component. An editable/extendible reminder/alert agent such as R2Do2 can plug into this middle tier. Specifically, to integrate guideline agent capabilities (or other applications) with legacy systems via a three tier approach, HOLON is using:

- a semantically rich, open, object-oriented, standard for specifying and accessing heterogeneous, longitudinal, distributed information (It is not intended to provide new information, nor to replace existing information standards as far as they go, only to adopt an open, standards-based meta-model where others can plug in their datasets and applications. To do this we explore the HL7 messaging standard as an object oriented model of the clinical data repository, and extend it at the information content level via use of ICD 9 and 10 codes, among many other content standards),
- the wrapper, lexicon translation, vocabulary server, and integration engine capabilities to query/manipulate and plug-together new and legacy systems,
- traditional middleware services such as communications, transport, security, message queuing, etc., as well as emerging standards for application mobility and object brokering (e.g., CORBA, OLE), and

The “middleware” field partially addresses many of these items: e.g., see Bernstein (1996), Lewis (1995). Yet sector- and industry-specific frameworks sitting above traditional middleware are also being developed to complete this list and to open up the full agent potential. In the health care sector, a number of standards bodies and consortia are working in these directions: e.g., see Fitzmaurice (1994), Halloran et al. (1996), and Rischel (1996). Our research investigates how a reminder engine can exploit some of these advances. In this way R2Do2, sitting atop HOLON, is a model of a middleware framework for the healthcare field (i.e., a vertical facility in CORBA terms).

This is a point also made by Musen et al.(1996) in their EON system. EON is an environment for authoring electronic protocols and connecting them to patient record systems so as to automate the recommending of therapeutic interventions. Musen et al. repeat the point that cross-industry middleware frameworks like CORBA and OLE are vital advances. Yet “for clinical software to be reusable and maintainable in a proficient manner, higher level software abstractions are necessary…” to promote the clinical-specific middleware field. R2Do2 shares that objective of a more healthcare oriented middleware framework with EON, however, there appear to be several differences between the two systems: (1) EON seems to be oriented toward supporting a user at a clinical workstation, whereas R2Do2 is concerned with supporting a distributed group of users; (2) EON seems to focus on clinical recommendations, whereas R2DO2 covers both the clinical and consumer perspectives; (3) EON offers fine-grained recommendations for both the diagnostic and care tasks, whereas R2Do2 to date provides only coarse-grained support for prevention and wellness promotion tasks; (4) there are internal algorithmic differences for accomplishing roughly equivalent clinical tasks such as the heuristic propose-revise-plan algorithm of EON as compared to the nonmonotonic, adaptive argumentation logic of R2Do2; and (5) EON provides a robust framework that middleware standards need to be connected to, whereas R2Do2 is currently attempting to conform with and utilize a number of middleware standards such as CORBA, HTTP, IIOP, Arden, HL7, and the like. These differences reflect the fact that no single framework is sufficient to address the entire field of middleware for healthcare, and that multiple frameworks should be encouraged. However, it is important that all the healthcare middleware frameworks be integratable if application users are to make use of them all.

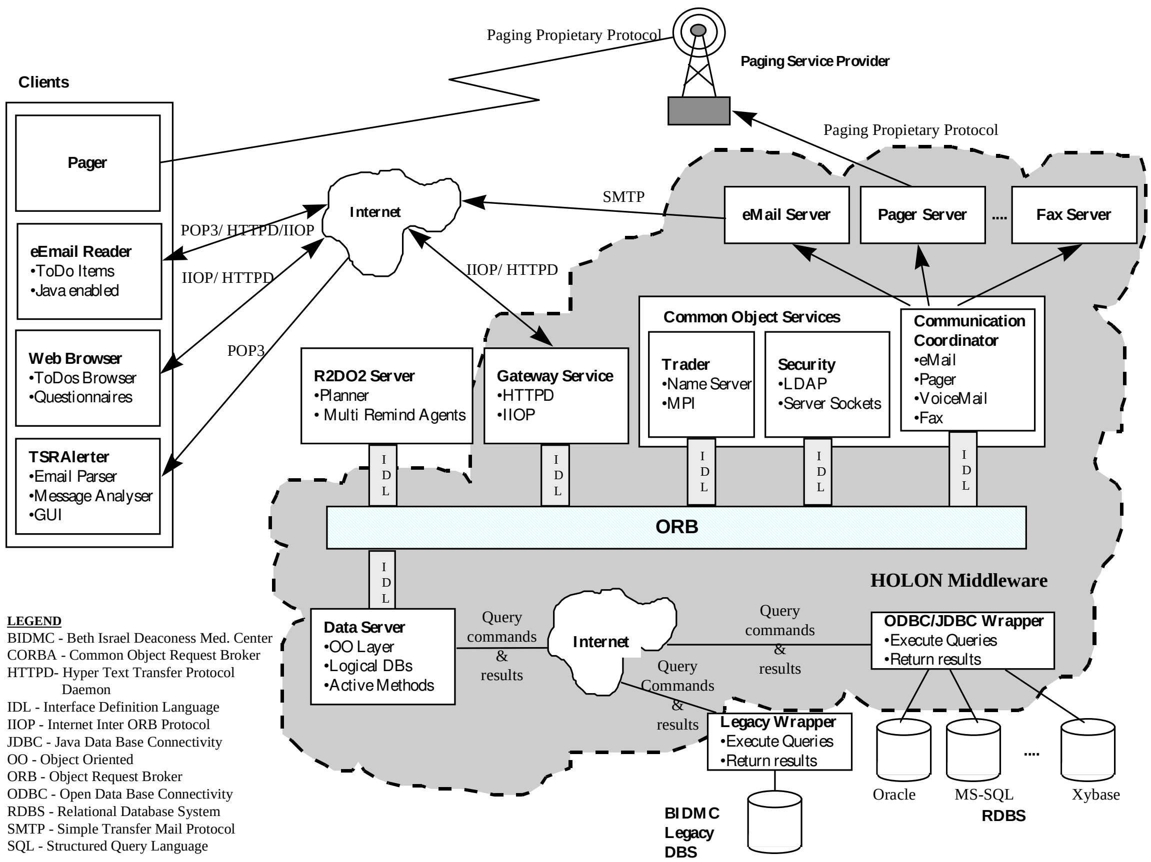
2) The R2Do2 System

The R2Do2 architecture makes use of several principles of modern software engineering such as object-oriented design (in its modules, communications, and databases), open software standards, portability of interfaces, and three tier client-server configuration. In Figure 1 one can see multiple clients on the left, the servers on the right (and bottom), and middleware connecting the two. Before we discuss the details of Figure 1, several items are worth pointing out. First, this architecture avoids the "stovepipe effect" or closed, proprietary, non-portable interfaces and databases found in much legacy software in the health care field. By contrast, the architecture of Figure 1 permits virtually any vendor's hardware to be seamlessly substituted for either the client or server stations. R2Do2 is platform independent. Likewise, any legacy patient record system is potentially connectable once the wrapping is effected. In the same vein, browsers, collaborative environments, personal information managers, and the like can be readily substituted just by registering them with the common object services. Finally, the human notification process is not limited to computer clients, but is extensible to pagers, interactive TV, fax, voice synthesis, and other forms of reminding and alerting.

Returning to the details of Figure 1, there are six major modules that warrant discussion: clients, groupware, middleware ORBs, middleware services, reminder server, and data server. Perhaps more important than any module, though, are the users who consist of healthcare consumers and providers, each of whom have differing needs from the modules. We will mention these range of needs and how they are supported in the discussion of the modules.

Clients -- The initial client design target is a desktop PC, although our current collaboration with HOLON eventually requires interactive TV and beeper service. There are two basic types of clients -- consumer clients that support personal health management and provider clients that help a provider manage multiple consumers. The differences between these two types of clients is largely a matter of screens and access rights that are variously on or off. In both cases the client includes a web browser (Figure 2), web-enabled email viewer, and a terminal stay-resident (TSR) alerter component (Figure 3) that parses email and alerts users when high priority health items have arrived. The web browser is the main route through which applets elicit "user preferences" for health plans, recurring ToDo items, and reminder schemes, and through which the database is browsed. The email viewer is the primary place for receiving and perusing ToDo item reminders and for altering their settings, forwarding them to others, and so on. All three elements of the client make use of the Java virtual machine for executing applet bytecodes, and thereby maximize platform independence, multi-threading, and security of local resources.

Figure 1 - Architecture of R2Do2 Showing Its Use of HOLON's Three Tier "Client-Middleware-Content Server" Model



Groupware -- R2Do2 is compatible with many forms of groupware, and is intended to embellish the conversations that pairs and groups of individuals hold. For example, if an HMO, hospital, or clinic adopts R2Do2, the persistent store will include a clinical data repository of all member information, and it will comprise a rich, dynamically updated model of the consumers. This helps the consumer. They can receive R2Do2-generated reminders about preventive care and wellness activities they and their providers deem important and relevant. And, consumers with chronic conditions can set R2Do2 to remind them about ongoing management, while letting the providers know that progress is OK or not. The clinical providers, in turn, benefit from R2Do2 as it can help them remind patients about appointments, proactively keep track of what a patient needs, suggest what to do for people with appointments today based on institutionally-approved guidelines, monitor patient self-management of chronic conditions and report back on patient difficulties, and so on. To support such activities, R2Do2 explicitly incorporates an email router with a forward and CC list for relevant consumers, providers, and surrogate caregivers. R2Do2 can be incorporated into any open standard groupware environment, such as a web browser that recognizes CORBA and Java applets. Thus it can be used alongside whiteboards, chat spaces, and videoconferencing facilities that are similarly enabled. In one such testbed, we are engaged in exactly such a collaborative environment currently being deployed at numerous homes, clinics, Beth Israel-Deaconess Medical Center in Boston and Norwalk Hospital, Connecticut: e.g., see Silverman et al.(1996).

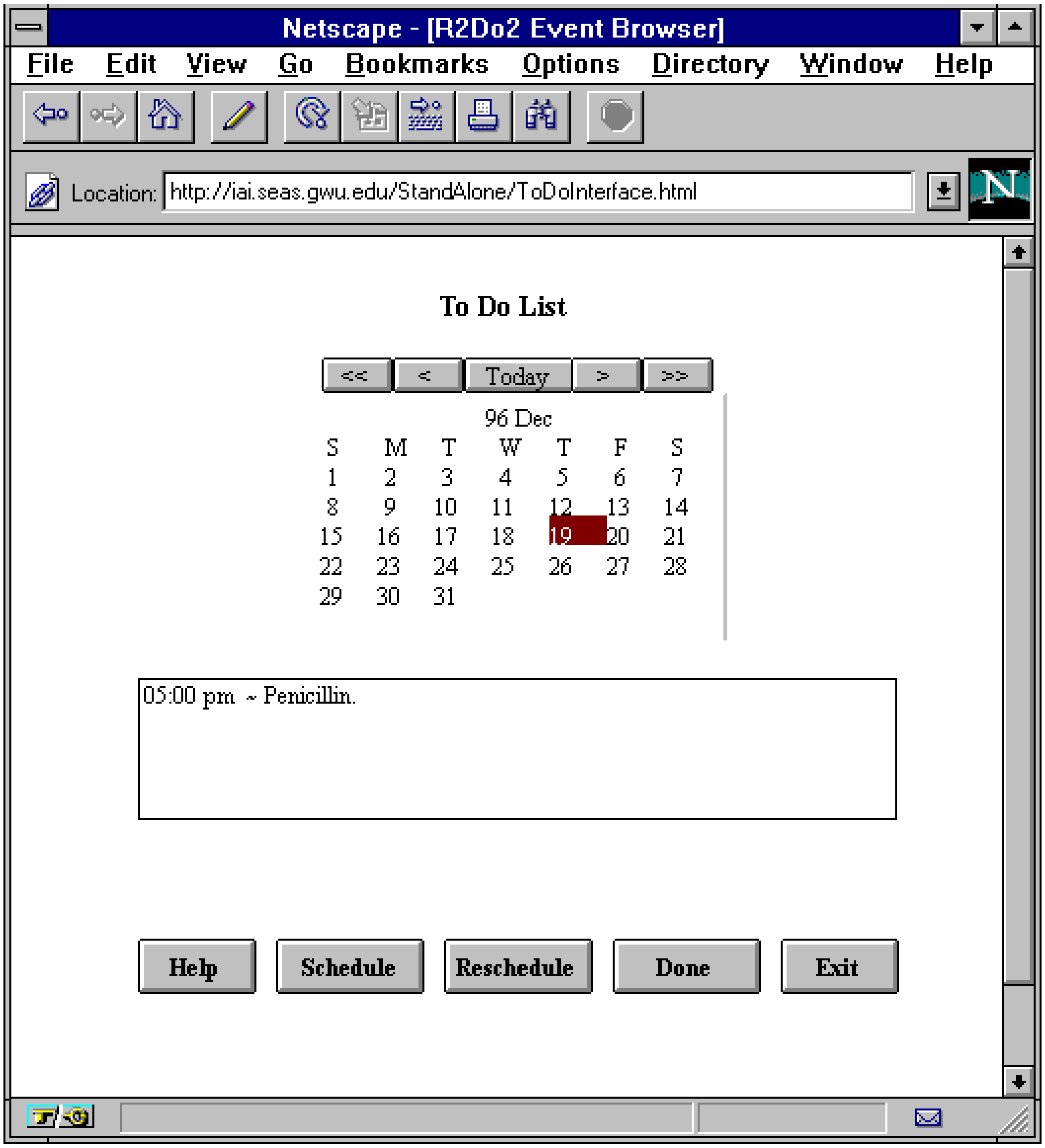


Figure 2 - ToDo Interface Main Screen

Middleware ORBs -- Rather than connecting to servers, the clients connect instead to open standards middleware, or more specifically via HTTP and the Internet to Object Resource Brokers (ORBs) and gateway services that comprise a middle tier. This is what permits the portability and replacability of the components in the architecture. The ORBs effectively encapsulate the various client and server modules, and turn them into object-like components that can message each other transparently, through a common object model. At present, HOLON subscribes to the model of the Common Object Request Broker Architecture (CORBA), an industry sponsored standard that eventually will be extended to provide handshakes with Microsoft and other common object models and ORBs (e.g., see Otte et al., 1996). At this writing, the CORBA standard is supported by several web browsers (e.g., Netscape, Sun's Hot Java, and others), and is available on many Unix-, Apple-, and Microsoft-based desktop platforms. The ORB and Basic Object Adaptors (BOA) are installed on each client and server in the architecture. They handle basic functions of listening, activation, dynamic invocation, dispatching, marshaling, and so on. In HOLON and R2Do2, we use the VisiBroker ORB from Visigenic, Inc.

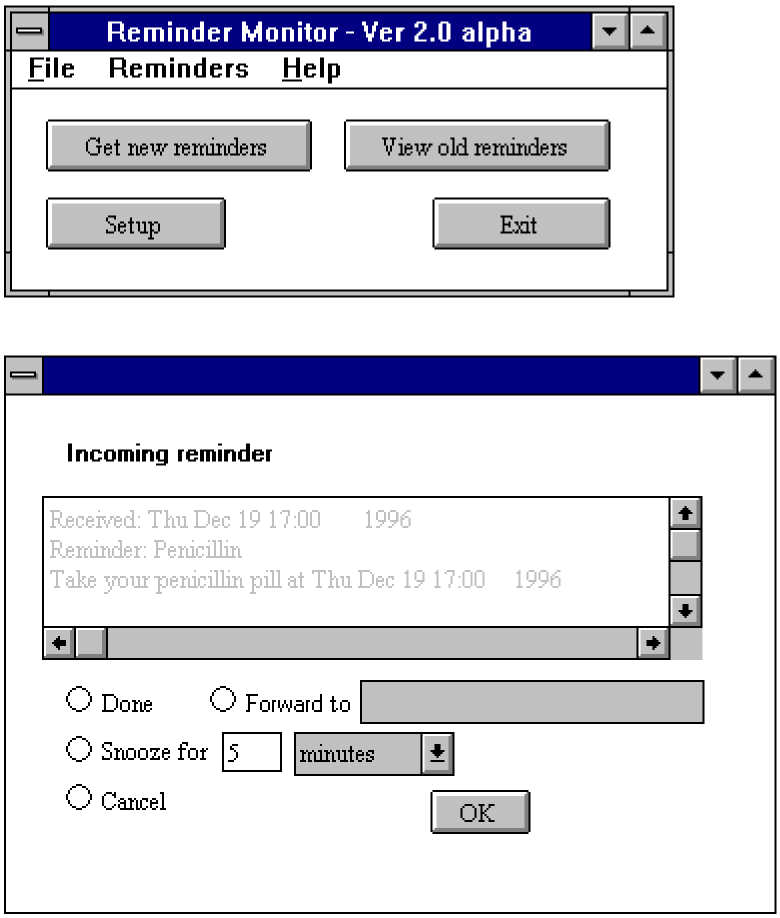


Figure 3 - Reminder Monitor Control Screen & PopUp Reminder

Middleware Services -- HOLON embellishes several CORBA services such as name serving, security handling, communications coordination (provided by R2Do2 to HOLON), and the like. New users are registered with these services to provide Master Patient-Member Indexes; security authorization, access, and authentication information; and notification alternatives. In the personal version of R2Do2, many of these services are omitted, while those that remain are automated. In the institutional version, this activity is partially supported by applets collecting data from users, but is highly dependent on a system administrator to verify, update, and maintain the various registrations.

Data Server -- This server contains computerized patient records, personal health information not normally found in patient records, and PIM type activity information generated by the consumer from their screen. The persistent store is, in effect, a dynamic model of the consumer's health status and activities that is jointly maintained by the consumer and by health care providers. Because there is the potential for a vast warehouse of sensed input, longitudinal data, and heterogeneous information, we are working jointly with the HOLON team to specify and deploy an object layer sitting at the top of the persistent store. The objects in this layer monitor the input streams, effectively detect trigger conditions, and transmit these items to the planning component and reactive agents. A few example transmissions include (1) users whose birthday is today (R2Do2 uses a 1 year planning horizon starting on a consumer's birthday), (2) users whose records have been updated, and (3) consumers scheduled to visit a provider today.

If a consumer interacts with R2Do2 privately (without institutional backing), the persistent store will reside on an R2Do2-dedicated data server. Still, software for consumer-managed health is growing like personal money management packages started to about a decade ago. So, this could be a large store as private users of R2Do2 become increasingly willing to fill out R2Do2's questionnaires and maintain its sensory input without clinical or institutional support. If, on the other hand, there is an institutional or community usage of R2Do2,more of HOLON's components are invoked and the data sever becomes a distributed system potentially with gateways and interfaces to the various clinics, labs, and offices that maintain records on the member consumers. In practice, connecting institutions to R2Do2 is a non-trivial task due to cross-institution guideline differences, lexical and semantic ambiguities in diverse patient record terminologies, and legacy storage system wrapping and translating constraints. Fortunately, the HOLON consortium is piloting more generic ways to connect and wrap such institutional resources including use of the ANSI HL7 healthcare data model, open database connect (ODBC) standard, and the UMLS vocabulary server.

Reminder Server -- R2Do2 copes with the "external world" via environmental sensors, an adaptive planner, and a set of effectors called personal assistant agents (one for executing each plan). The external world includes the users who have dynamic, continually shifting ToDo items and reminding preferences covering wellness activities, prevention efforts, health guidelines advice, disease recovery steps, chronic condition management, and so on. These items are "sensed" via the user interface applets and via a persistent store called the data server.

The adaptive plan generator is sent this input and then dynamically generates the appropriate new plans to respond to it, and instantiates and launches agents to execute those plans. Specifically, the plan generator produces plans for each user registered in the environment. Numerous plan intervals and actions may need to be generated at various times during the year for each user such as required by, among others, a 6 day medication prescription (2 doses per day); a course of weekly shots; a pregnancy carried to full term; the current year's set of routine preventive checkups, screenings, and immunizations; and ongoing glucometer readings and chronic insulin injections. Alternatively, in the event of changes, the plan generator module adapts existing plans and advises currently deployed agents of the updates. As new, sometimes unexpected, events occur (e.g., reaching the next birthday, a miscarriage, a diabetic complication, etc.) the original plans will need to be revised, and the agents carrying them out, modified. The planner module is written in Java, and makes use of a Java version of a shareware forward chaining inference engine, CLIPS, for processing health guideline KBs against consumer data to generate the agent plans.

When the agents are launched with new or updated plans, they are given both personal assistance and limited self-planning capabilities. These capabilities allow the agents to interact directly with the users and with the client server environment. Thus agents can conserve resources by putting themselves to sleep until needed; alter their reminding schemes in response to user preference; postpone, cancel, or re-assign ToDo items and reminders also upon user request; coordinate alternate modes of communication (e.g., TSR alerter, email, and beeper); and contact providers and inform surrogate care givers when critical ToDo items are neglected.

2.1) The Architecture From A Planning Perspective

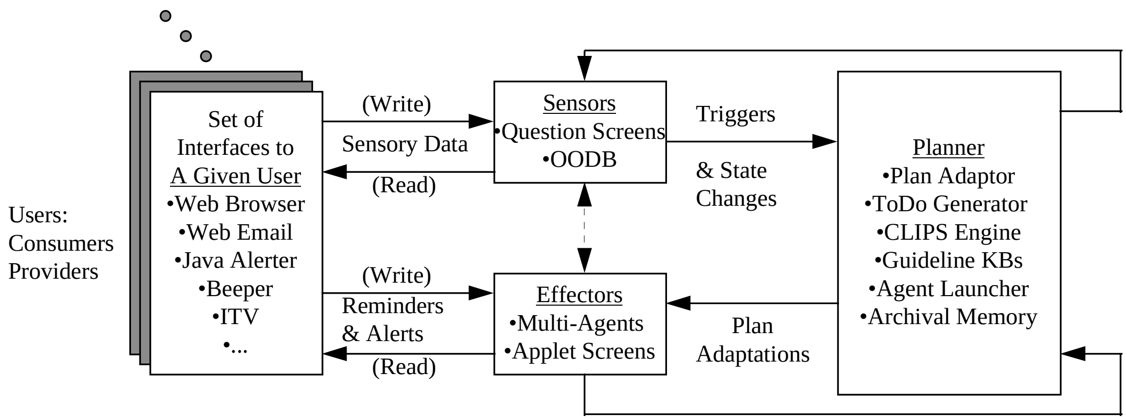
The architectural view just presented reflects a software engineering perspective intended to address several of the web challenges plus significant scalability and implementation issues regularly confronting both AI and reminder engine projects. That view of R2Do2 is important, and it shows how data and documents can be co-mingled (the guidelines and advice sent to users are documents), and how multiple components can be combined. Thus one can integrate AI components, collaborative groupware, web browse and surf, and database transactions.

However, in this view, the adaptive planner is just another CORBA application to provide communication services for. Such a view potentially obscures an important point. That point is, the planner is the computer's "brain" as concerns the health reminders and alerts. Recall, R2Do2 copes with an external world, and includes environmental sensors, an adaptive planner, and a set of effectors called personal assistant agents. These are key extensions of the middleware framework into the healthcare domain.

The classical definition of a planner is a program that generates a sequence of actions (the plan) that will move an entity from its initial to its goal state. A plan is first generated and then sent to an execution agent to carry out each of the (ToDo) actions. A number of researchers have extended this approach so it responds to a dynamic, uncertain environment where the initial plan may need to be revised before execution is completed: e.g., see Bresina & Drummond (1990), Das et al. (1997), Lyons & Hendriks (1992), Weld (1996). In general, these approaches include (1) a reactive agent that first senses and then tries to effect and/or respond to the environment according to the original plan (or within tolerances of it) and (2) a deliberating component capable of modeling the state of the environment, adaptively replanning, and transmitting the revised plan to the agent for altering its behavior. These approaches differ in terms of, among other things, varying degrees of world knowledge in their planner's model of the environment, sophistication of adaptive reasoning, and degree of separation of sensor, planner and effector/executor. The best way to handle these and other differences are open questions in the literature.

In R2Do2's case (Figure 4), the Planner generates a number of ToDo items for each consumer and provider. These are turned over to the Effectors or agents who provide personal assistance by reminding and alerting the users of their ToDo items and by posting procedure and user advice documents. The users read these emails and documents, and may alter their plan directly (e.g., snooze, cancel, or reassign an item). Also, users can add new ToDo items either directly through applet screens, or indirectly via their answers to health profile and preference questionnaires. In this manner R2Do2 derives an initial health plan that it continually refines and adapts as the users age, change their practices, develop new conditions, and so on.

Figure 4 - Diagram Showing How R2Do2 Acts Like a Software Robot With A Brain (The Adaptive HealthPlanner), Sensors (the Active, Distributed OODB and user interface applets), and Effectors (The Dynamic, Health Reminder & Alert Agents and Applets)



Important to R2Do2 and in the literature is the notion of a "universal plan" (Schoppers, 1989) that guarantees the eventual achievement of a planner's goals no matter what the initial (or subsequent) states of the environment. Often, this universal plan is impossible to specify given all the uncertainties and dynamic shifts in the environment. Instead, a feasible plan is rapidly derived and turned over for execution. In parallel, varying adaptive heuristics and algorithms are utilized to try and incrementally drive this toward a more robust plan over time and as the environment shifts. This two step "initial plan derivation/plan refinement" process is a continually repeating cycle. Of interest here is how to scale this idea and integrate it into the R2Do2 system.

3) The HealthPlan Generator

The planning space just described is potentially vast, fraught with unexpected state shifts, and dynamically evolving. Keeping up with it requires reactive execution of robust plans coupled with parallel plan-refinement that produces adaptive plans. The challenge is to continually (re)derive the most robust plan possible while using the least amount of derivational and execution resources in keeping with users' preferences.

To assure R2Do2 meets this challenge, it is important to adopt a framework for organizing the distributed reasoning and processing elements. The framework settled on here is a set of 5 mathematical equations. In essence, R2Do2 must continually strive to solve a set of 5 equations that comprise its objective function. The five equations push the system to:

1. maximize consumer health by recommending prevention and care plans for each situation that arises
2. search a guidelines KB in order to construct a robust plan (of recurring ToDo items) for a given situation
3. map the recurring plan elements (ToDo items) onto the consumer's (and provider's) schedule and launch agents to help carry out the scheduled plans
4. utilize personal assistant agents to isolate and carry out a user-acceptable scheme to notify and remind the consumers of the ToDo items likely to maximize their health
5. alert providers and surrogate caregivers when consumers seriously deviate from the objective function to their self-detriment

Much of the remainder of this paper describes these 5 equations from both a mathematical and executable logic perspective. The former perspective focuses on a notation and set of formalisms that allow us to try and maximize the objective function. In a managed care and litigious environment, maximizing the objective function becomes more than just a noble ideal. It concerns real costs and losses in both money and lifespan reduction. So, the discussion will point out numerous issues in the effort to maximize the validity and correctness of the system of equations. The latter (execution) perspective, in turn, concerns how these equations can be reliably implemented in the R2Do2 modules and agents. At best, the executable logic could only be shown to be provably correct subject to both guideline KB validity and consumer lifestyle choices (e.g., don't maximize my smoking health area). Further, this domain is inherently heuristic and constructive, rather than analytic and mathematically deductive. In lieu of a mathematical proof of correctness, our efforts thus far focus on careful delineations such as this paper, and extensive testing. Nevertheless, a more formal approach would offer numerous benefits (maximizing health, assuring cost objectives, satisfying user preferences, etc.) and it warrants further research.

The first equation of this system is, as already stated, to maximize consumer health by recommending prevention and care plans for each situation that arises, subject to user preference for situational resolution (or not), and to prior plans that may need adapting:
Maximize:

$$HP(Consumer_s) = \sum_{\sigma=1}^{S} \sum_{k=1}^{K} \{(Pnk[Check_s(\sigma) \rightarrow Pk \#Mk] \mid Pnk_{s-1}, wi, \alpha k) * UP(\sigma,k)\} \qquad (1)$$

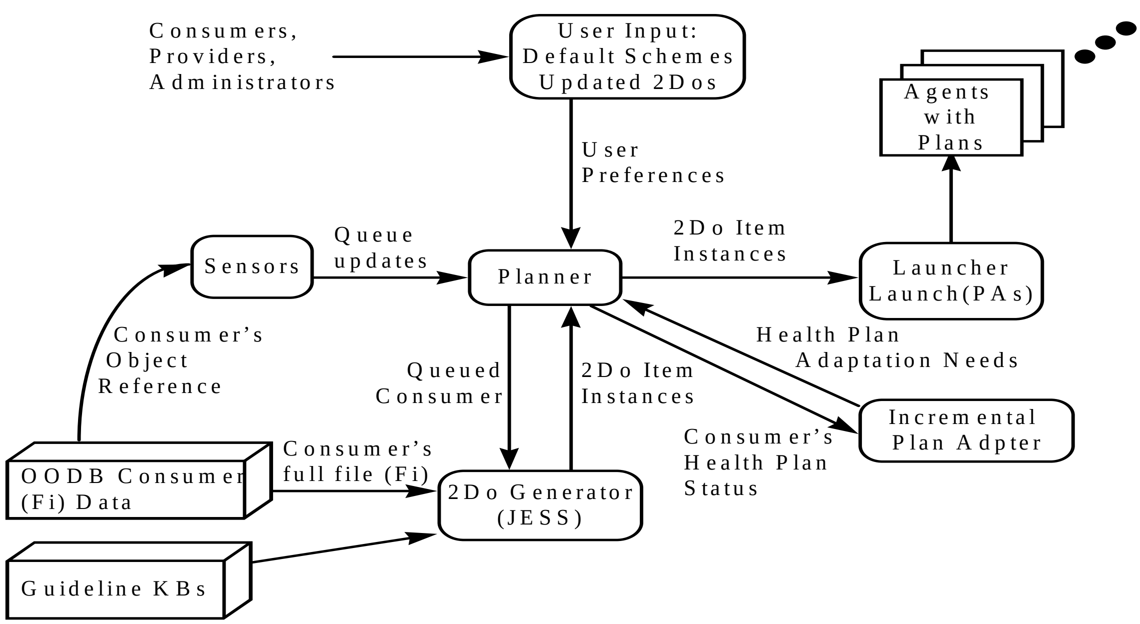
subject to,

$$UP(\sigma,k) = \begin{cases} 0 & = \text{ignore Pk for } \sigma \text{ OR ignore } \sigma \\ 1 & = \text{include Pk for } \sigma \end{cases}$$

Mk = EXECUTE(UUP(αk), αk)
where,

HP = list of subplans (Pnk) in the overall health plan for all the situations in the current state of the consumer. HP keeps pointers to the valid plans that apply to a consumer and that are acceptable to that consumer.
s = current state
s-1 = prior state
σ = situation counter 1,S
k = plan counter 1,K
$Sit_s(\sigma)$ = set of situations (1, S) argued to exist during state s, where situations are conditions to which one or more plans may be applied.
$Sit_{s-1}(\sigma)$ = store of situations assumed during the prior state
Pk = kth plan where a plan is a sequence of actions intended to alter a given $Sit_s(\sigma)$. Any Pk may exist across states.
Pn = nth recurrence of a plan. Pn also may exist across states.
Pkn = nth recurrence of kth plan
# = "is defeated by" operator
Mk = plan monitor that detects when plan k needs adapting and that executes those adaptations
wi = constraint relaxation operator
αk = machine generated message to adapt Pnk or Pk
UP(α,k) = initial user preference message for plan k under situation σ
UP(α,k) = updated user preference: message to adapt Pnk or Pk
$Check_s(\sigma)$ = monitor that detects all the situations σ = 1,S in state s

Figure 5 - Overview of the Planner



To satisfy these goals, the plan generator incorporates the modules illustrated in Figure 5 and described in the following subsections. Figure 5 shows how transmissions are sent as object references by the sensors (OODB, user query screens) to the plan generator's queue. As long as the queue is nonempty, the process depicted in Figure 5 continues. When it is emptied, the generator returns to idle state, monitoring for incoming environmental changes. The process may be viewed algorithmically as:

1) If detect consumer object references in queue OR detect user input plans, then proceed, else wait,
2) Invoke Incremental Plan Adaptor to determine plan adaptation needs,
3) If plan adaptation needs include generating a new plan or revising an existing plan, OR user input plans are unspecified, then continue, else skip to 5.
4) Invoke ToDo Item Generator to run Guidelines KBs over designated consumers (retrieve full consumer object) and infer new or modified plan steps,
5) Invoke Agent Launcher to delete or modify current agents and/or to assemble and initiate new agents,
6) Delete processed object references from the queue
7) Return to 1.

As can be surmised by inspection, this algorithm runs the first three equations as will be described in the remainder of Section 3. The last two equations are described along with the agents in Section 4.

3.1) Incremental Plan Adaptor

Equation 1 called for adapting plans previously derived for prior situations Sit_{s-1}(sit). Here we define the adaptation process (α) as some combination of two operators:

AddPlan(k), Ak -- AddPlan is the operator that calls for the ToDo Item Generator to create a new plan unless a user-generated plan is provided. In the latter case, there is no need to generate the plan. In the former case, the ToDo Item Generator must create a new plan either when one never existed, when further recurrences of an existing plan are needed, or when plan extensions are called for. The output of AddPlan includes a place-holder for the revised plan sequence, HP' as follows:

$$HP \xrightarrow{Ak} HP'(Pnk, Pk\#Mk)$$

DeletePlan(k), Dk -- DeletePlan is the operator that sends a message to the agent to set its Pk to the DONE state and terminate all processes. The Mk operator inside the agent receives that message and takes the necessary action to effect the deletion. It should be noted that plans are of a given type (e.g., TakePill, HaveExam, GetShot, etc.) and that R2Do2 elicits user preferences for reminder and notification schemes for each class of plan. The Dk operator does not destroy R2Do2's memory of these learned preferences. They are separately filed in the persistent store and are accessed anew by any replacement plan and agent of the same class. Also, the Dk does not delete the entire HP, just specific Pnk within that HP.

$$HP(Pnk, Pk\#Mk) \xrightarrow{Dk} HP'$$

Adaptation, α(Ak, Dk) -- Adaptation is thus a process that merges various combinations of Ak and Dk operators. The result is an improved health plan, HP'. The maximization of Equation (1) is affected by machine- and user-generated αk as follows:

$$HP \xrightarrow{\alpha\kappa} HP' \tag{1.1}$$

At present α includes various combinations of two operators: delete and add. Specifically, when a consumer object reference is transmitted to the queue, that consumer's records must be checked for the existence of a previous plan. This is done by the Incremental Plan Adaptor. If a prior plan is found, it must be marked for deletion, and a replacement plan generated by the ToDo Item Generator or obtained from the user. This provides the intelligence to detect what part of the HP has changed, and to adapt that HP into a new theory of maximal health for the consumer. Detecting what part of an Pnk or Pk that needs to be updated, however, is a nontrivial, future research item. This is related to what the planning literature calls the "frame problem" e.g., see McCarthy & Hayes(1969). So, for now we just replace the entire Pnk or Pk. If no prior plan is found, the Adaptor needs to return the message to add a new plan for that consumer.

α1: [CheckPlan(Pnk) → Pnk] ==> Dk ^ Aj
α2: [CheckPlan(Pnk) → nil] ==> Ak

Consider an example that R2Do2 had to cope with during the Fall of 1996. A 42 year old consumer's HP consists of a set of immunizations and a routine physical for the coming year, but excludes a prostate cancer screening (not needed until age 50). The consumer contracts a bacterial flu and, after getting a prescription from his doctor, he uses the R2Do2 screens to input a plan for medication (penicillin twice daily for 6 days). This is a user-generated Pnk that triggers the α2:Ak to tell the Planner to bypass the ToDo Item Generator and formulate a message to the Agent Launcher to add the medication Pnk to this consumer's HP. Also, the consumer spent some time updating the family history section of his medical record to reflect the fact that his father had prostate cancer several years ago. Due to an object oriented database-transmitted object reference on the Planner's queue, the Adaptive Incremental Planner ultimately inferred α1 with a Dk of the current routine physical (Pk) and an Aj of the physical with a screening for prostate cancer (Pj). Thus simple deletes and adds of Pk and Pnk lead to continual adaptations of HP.

Given that people continually age, develop new unexpected medical conditions, and succumb to various conditions, there is a legitimate need to replace current plans with new ones. The interactions between treatments for various conditions render it difficult to update existing Pnk, and easier just to replace them. The scale up of R2Do2 also suggests the need for simple adaptive operators. As a result, the two adaptivity operators are felt to be a reasonable starting point.

One last issue in the scale up of this adaptation method concerns how best to identify the Pk in the current HP for a given consumer. The approach currently in use is to maintain an index of the list of Pk for each consumer. As consumer plans grow and as more consumers are added, it is not clear that this is the optimal approach. An alternative we hope to explore later is to add agent conversational abilities. Thus agents could answer questions such as, among others, “Who has plan P34?” or “Has plan P18 been deleted?”

3.2) ToDo Item Generator

The ToDo Item Generator integrates the data-centric, Fi, with the document-centric Kbs to derive the plan, Pk, for a given consumer who has one or more situations, Sit_s(sit). In situational calculus (McCarthy & Hayes, 1969), the Pk are the edges of the state transition graph, the operators that transform a given situation (nodes of the graph) into a new situation. In our implementation, a state has many situations, each of which may have a Pk or Pnk associated with it. When one or more situations change, or a new one arises, a new state of the world is said to arise. Since Pk is a sequence of actions or ToDo items in the domain, as derived from inferences over the Guidelines KB, a given Pk may persist across one or many states.

The ToDo Item Generator has a queue for incoming consumer object references. For each consumer object reference in the queue, this module retrieves the full consumer file, Fi, and processes the active Guideline KBs against the Fi values. The Fi hold the full state of the consumer including all plan-addressable situations, non-changeable facts, and so on. Let,

s = Fi = F(sit0, ... , sit(S), fact1, ..., fact(F))

By way of example,

s = F(sit(flu), fact(prostate-history), f(male), f(d.o.b:7/5/54),
fact(med-list: tetanus boosters on 5/13/79, 1/2/90), ...)

To bind this data to the KB, R2Do2 first converts it to the format required by each of the j= 1,J guidelines or KB partitions. A few examples include: date of birth is converted to age in years, a list of tetanus booster events is converted to a numerical count plus a duration or interval since the last vaccine, and so on. Once the Fi are prepared, a shared working memory is maintained while processing rules for a particular person. Also, the multi-threading option in Java is used up to the limit of threads supportable on the server. So, the algorithm for the ToDo Item Generator becomes:

1) If detect consumer object references in queue proceed, else wait,
2) If maximum number of Inference Engines has not been reached proceed, else wait,
3) Retrieve consumer object file (fi) into memory and convert to rule-ready form
4) Invoke Inference Engine with fi, in memory, and guideline KBj: IEng(fi, KBj)
5) Ak(Pk) indicated by KBj firings
6) If guideline KBj+1 exists then j = j + 1, go to 4, else continue
7) Delete processed object references from the queue
8) Return to 1.

Given a number of assertions of facts and situations, Fi, about a given consumer, each inference engine binds to Fi. The engine then utilizes forward chaining over a rule base and an argumentation logic variant of heuristic construction (see below) to fire each relevant plan, Pk and any recurrence schemes, Pnk while assembling the current HP.

Ideally, we may stipulate this process as the second equation that searches a guidelines KB of the form (Fi $\rightarrow$ Pk) to construct a robust plan (of recurring ToDo items) for a given situation and set of user health preferences (UP):

$$[Fi \wedge (Fi \rightarrow Pk)]_{ik,Sit(\sigma),UP} \implies Pnk\#(Mk, UP(\sigma,k)). \qquad (2)$$

In practice, though, when implementing this equation there are three sources of complication we must attend to:

1. Usage Constraints --There may be resource and action constraints inherent to Consumer, HMO, and Hospital deployment and usage. At present, we attempt to mitigate these via distributed client-server processing, via processing limits (e.g., a one year at a time horizon for each consumer's Sit(s).), and via use of Java's multi-threading to attempt "parallel" execution on multiple consumers simultaneously. This latter feature and related efforts are attempts to improve scale up to institutional levels. However, several limits remain and institutional settings will not have the capacity to process the R2Do2 on every patient every day, only on days they are likely to be seen, on birthdays, and the like.

2. Omissions in the health status data, Fi. -- A well known obstacle in the Fi is that consumers rarely fill out the entire patient history interview at one sitting. Further, the primary caregiver shifts over time, and the electronic patient record may be likewise incomplete, containing only some of the facts and situational assertions. Still other example omissions in the Fi are due to unknowns, unexpected events, lags in reporting, sensor misinformation, missing information in the persistent store, 1 year planning horizon limit, and processor resource constraints. So, the Ideal Consumer Model (Fi*) that reflects the consumer's true health situations, status and history can only be incrementally approximated by Fi' an internally consistent set of assumptions about the state of the world. In other words, conclusions must be reached and plans formed nonmonotonically, so they can later be adapted when new knowledge about the consumer is added. To address this set of obstacles, we utilize a nonmonotonic, default and defeasible logic described below. R2Do2 also maintains record completeness checks, and issues numerous prompts and messages to users to fill in missing items as they can.

3. Issues in the Knowledge Bases -- Further, due to issues discussed below about the guideline KBs, we only have a reasonably good approximation of the ideal rule set. Let us call that approximation (Fi $\rightarrow$ Pk)'

This is not to say that the resulting plans are useless or even nonusable due to these complications. To the contrary, through the use of a “closed world assumption,” the resulting plans are feasible, workable, and positive. The types of usefulness sacrifices that occur are that consumers encounter items that can’t be planned yet. Instead, they receive more messages at the screen asking the user to collect omitted data. Thus many of the HP(Pk)’ ToDo items involve data collection by the user and/or suggestions for the user to consult the provider, while the HP’ ToDo items include more health-related actions of direct value to the consumer (and provider). Put another way, the value and usefulness of the plans output by the Planner increase as the users provide more of the missing information. Likewise, HP’ grows closer to the ideal HP* as this occurs, and as items 2 and 3 above are addressed.

Given there may be some upper limit before reaching HP*, we are still interested in adaptations that lead to the following type of “initial plan derivation/plan refinement”. Let, us further clarify the process of maximizing Equation (1) as follows:

wi = constraint relaxation or simplification operator that permits HP to utilize simple (more data collection and consult provider) ToDo actions initially, with greater enhancements and accuracy to HP as wi is tightened. Also, wi invoke the closed world assumption in which the KB rules for that wi are consistent with the state of the world asserted to date.

w1>w2>w3>...>0

$$HP_{w1} \xrightarrow{\alpha} HP'_{w2} \xrightarrow{\alpha} \dots \xrightarrow{\alpha} HP^*_0 \qquad (1.2)$$

The wi may lead to default rule firings. For example, in the absence of family history the consumer is asked to fill in family history when possible, but for the time being, is assumed to be healthy and prostate cancer-free. This is a form of “anytime algorithm.” (e.g., see Zilberstein, 1996) where the rules and resulting HPs are defeasible. As more facts become available (e.g., the father had prostate cancer), the theory of how to maximize HP must be extended. We retract positions of Hp_{w1} and, at least temporarily, argue for HP'_{w2}. Thus over time, the ToDo Item Generator accumulates better assertions that trigger more of its health relevant rules, and less of its “collect data” and “consult provider” ToDo items show up in the resulting plans.

3.3) Guidelines Knowledge Bases

The heart of R2Do2 lies in being able to bind machine readable health care guidelines documents to patient information, to discover and adapt healthcare plans. This section reviews the logic for arguing from guidelines to assert/retract plans (Section 3.3.1), default reasoning assumptions (Section 3.3.2), requirements for English level editors that can parse guidelines into Arden syntax as well as machine readable, open standard software (Section 3.3.3), and the temporal algebras required by the plans (Section 3.3.4). Throughout this discussion we stress the use of open standards, extensible guidelines, and reusable code.

3.3.1) Logic for Guideline Argumentation

Health care guidelines may be thought of as temporal and situational arguments with varying degrees of support that may be asserted, modified, and/or retracted for a given consumer health plan: e.g., see Das, Fox, Elsdon et al. (1997). R2Do2 contains a growing number of knowledge bases. The KB covers several logical subspecialties, j, for which we were able to find a published and accepted guideline. For example, there is a KB for immunizations (Dickey 1996), a KB for screening exams (Dickey 1996), a vital signs KB, a diabetes mellitus KB (Feld 1995), and so on. Let us define a line of argument A_T based on some theory Tj, about the truth of a plan, Pk in a subspecialty, j. The following equation governs how the argumentation process works.

IF: For all i, Tj, k in a KB,
OCCUR[Fi(Consumer, Tj(Preconditions)) or
Ask-User(Consumer, Tj(Preconditions))] (3)
ARGUE(Tj, A_{Tjk}, Pk, $Sign_T$)
HOLDS(A_{Tjk}(Warrant, Citation), $Sign_w$)
COMBINE([$Sign_T$, $Sign_w$], Sign)
THEN:
AddPlan Pk: TODO[(action, time parameters, event-class, defaults,
Hp_s(consumer, provider)), Sign]
where,

OCCUR = A predicate that uses Fi to test the existence of the jth T for the consumer (or failing to find information in Fi, it asks the user about each of the j=1,J theories directly)

Fi(Consumer, Tj) = The set of facts, situations, etc. asserted in the current state, including the diagnosis or symptom set and other preconditions for asserting Tj

ARGUE = a predicate that asserts or retracts a given Pk based on guideline arguments, A_{Tjk} and with qualifier $Sign_T$

Sign = a qualifier of support for the argument drawn from some dictionary (e.g., if the dictionary is first order logic then the signs are True or False, if the dictionary is probability then the signs vary from 0 to 1.0, if the dictionary is qualitative reasoning then the signs might be asserted, supported, unknown, doubted, retracted.)

HOLDS= A predicate that examines situational, temporal, and user-preference warrants, W, to test what degree of support exists for the argument, A_{Tjk}, in the current Fi context

COMBINE = A predicate that merges the support for the theory and the warrants in the current context for the current consumer, and depending on the dictionary in use (e.g., if a probability dictionary is in use, this may be a Bayes Formula).

AddPlan = The operator that builds the Pnk, as described earlier

TODO(.) = The Pk is a ToDo item or sequence of ToDos of the following parameters

action = the action (or sequence of actions) for the user to take along with any advising or warning messages

time parameters = action scheduling information described in Section 3.3.4

event-class = the type of event this action falls under (e.g., TakePill, HaveExam, GetShot, etc.) and which is later used to organize reminder and notification schemes

defaults = set of closed world assumptions and qualifications inferred about the consumer

Hp_s(consumer, provider) = identifier of the overall HP this ToDo action is part of, including reference to the consumer and the providers of this ToDo. The latter is inferred from the event class and user preferences if they are filled in.

Note that the last two items, defaults and HP(.), are not written into each rule, but rather are inferred dynamically by the inference engine using event-class, UP, and other Fi.

A central element of Equation 3 is the ARGUE predicate, and the related items that test for support sign. An example may clarify this notation. For that example, part of the immunization KB for the DPT vaccine is illustrated in Table 1. Here the theory asserts DPT as the goal: i.e., SignT = DPT-goal(supported). Within that theory is a schedule of vaccination plans, Pnk (n=1,5 and repeating B) as Table 1 illustrates. Each plan along the schedule requires an argumentation process before it can be asserted. Table 2 illustrates the arguments and warrants the guideline imposes for P2k, the 2nd shot in the series. Column headings are guideline arguments, A_{Tjk} , for or against various plans, Pnk. Row headings are the conditions of warrants that support or defeat the various arguments. Finally, the cells of the table reflect the degree of support each warrant provides to the various column arguments. Thus, for example, the top left cell supports the argument that if any of the contraindications are supported, the 1st plan affecting the 2nd shot, P21(= get 2nd shot), should be retracted. Equation 3 ensures the inference engine will bind to the Fi data and iterate through all the cells of this table to determine which of six possible plans related to the 2nd shot should be asserted.

3.3.2) Dictionaries of Support Signs and Default Reasoning

In order to use the COMBINE predicate, we provide dictionaries based on first order logic (true/false statements) and qualitative decision theory (asserted, supported, unknown, unsupported, retracted, NA). We also hope to but have not yet implemented a Bayesian function for updating beliefs about warrants and the Fi evidence. The dictionary is depicted in Table 3. Here one can see that Theories which are supported by Fi or direct user input are generally asserted unless there is a warrant that directly leads to "unsupported." Similarly, if the Theory is unsupported (usually by provider decision or user preference), no amount of warrant support will overturn that conclusion. This is a feature that may be changed, of course, if one wants the guidelines to critique provider decisions. Lastly, where unknowns exist about the theory (e.g., we don't know if the consumer has had any DPT shots), if the guideline warrants support the action, we adopt it. Thus we utilize a default reasoning concept that assigns support unless direct rebuttal is observed. This means that individuals are assumed healthy and compliant in the presence of unknown Fi, although warning messages are issued, as are ToDos that try to discover the missing Fi. In these cases, the adaptive algorithm is relied on to remove any incorrect default assumptions during subsequent iterations and as new Fi arrive.

Table 1 - Overview of Immunization Schedule in the Guidelines KB *

Age	At birth	**2** months	**4** months	**6** months	**6-18** months	**12-15** months	**12-18** months	**15-18** months	**4-6** years	**11-12** years	**14-16** years	Every 10 years	Over **65**
Diphtheria Pertussis Tetanus	-	1	2	3	-	-	-	4	5	B	-	B	-

- \- n = 1,2,3,4,5 (order of primary series) or n = B (booster)

Table 2 - Warrants (Conditions→Arguments) of the Guideline KB for the 2nd Immunization under the Theory about DPT Vaccines.

	Guideline Arguments, A_{Tjk} about T=DPT, j=immunizations, k=DPT plans*				
Conditions (left sides) of the Warrants that support the A_{Tjk}	Retract P21 Assert P22	Assert P21	Assert P21 & P23	Assert P24	Assert P21 & P25 & P26
Contraindications: • anaphylactic reaction • encephalopathy within 1 wk • neurologic disorder • convulsion history Citations: Dickey(1996),PPIP PDR(1996), Redbook	Supported	Unsupported	Unsupported	Unsupported	Unknown
Compliant: • schedule • vaccine type Citations: Dickey(1996),PPIP PDR(1996), Redbook	NA	Supported	Unsupported	NA	Unknown
Precautions: • fever > 104.9 • screaming > 3 hours • convulsions w/i 3 days • collapse w/i 48 hours Citations: Dickey(1996),PPIP PDR(1996), Redbook	NA	Unsupported	NA	Supported	Unknown
Other					

* NOTE1: P21(get next DPT shot), P22(get next shot as DT), P23(Display Noncompliance Warnings), P24(Ask-User for Missing Fi), P25(Warn about Compliance Schedule, Contraindications, and Precautions), P26(Warn about Precautions and Recommend Provider Consult)
NOTE2: NA = Not Applicable (doesn't affect the support)

Table 3 - Dictionary of Support Signs and a Qualitative Decision Theory for Combining Warrant Support with Theory Support Signs

SignT	SignA	Sign
Supported	Supported	Asserted
	Unsupported	Retracted
	Unknown	Asserted
	NA	Asserted
UnSupported	Supported	Retracted
	Unsupported	Retracted
	Unknown	Retracted
	NA	Retracted
Unknown	Supported	Asserted
	Unsupported	Retracted
	Unknown	Unknown
	NA	Retracted

So, to continue the DPT example rule, , if prior shots = 0, the THEN clause becomes a statement about seeing the provider and getting the first DPT. In this case, the default is set to healthy, but non-compliant with respect to the current KBj. If on the other hand, the consumer had an adverse reaction to the first shot (precautions = supported), then the advice is to consult the physician about altering the dose of the second shot, and the default is set to compliant, but not-healthy with respect to the current KBj and a situation is inferred.

Messages to consumers and providers, such as called for by P3, P4, and P6 in the vaccine example, parse and display the default assumptions (in English) with a caveat warning that if default assumptions are in error, then the ToDo items might not apply and the provider should again be consulted or the missing information should be added. Users can suppress such messages after the first display, provided they accept the risks. Several ongoing research issues here are: to assess whether this is a complete taxonomy of default types, to study performance issues of inferring and retracting defaults in separate rule sets, and to derive improvements over the currently manual methods for enumerating and testing defeaters of each default.

One of the defaults is unknown Fi, leading to a ToDo of "ask-user." In order to handle this default, the content of the KBs are organized accordingly. This looks like:

$$\left.\begin{array}{l} -Fi \wedge (-Fi \rightarrow Pk1) \\ Fi' \wedge (Fi \rightarrow Pk2) \end{array}\right| \implies Pnk$$

where

-Fi = data items noticed to be missing by OCCUR
Pkn1 = "collect data" and "consult provider" ToDo items
Pkn2 = direct health-related ToDo items

The example rule just provided about DPT vaccines illustrates Pkn1 and Pkn2 type assertions. Also, to continue the earlier prostate example, failure to specify family history would precipitate a question about family history (Pkn1) and a plan assuming the default healthy family history (Pkn2). Once the family history is updated, other Fi→Pkn2 rules defeat the earlier plan, update the default assumption parameter to be inserted in the consequent, and infer the new ToDo item that calls for immediate cancer screening.

3.3.3) Guideline KBs Representation Language and Editors

The argumentation and default reasoning formulation presented thus far utilizes an extensible, yet generic logic for representing guideline KBs that is well suited to today's environment for authoring guideline documents. In particular, like the ARDEN syntax originators, Fridsma et al. (1996), Shahar et al. (1996), Fox and Johns (1997), and others, we are concerned that users can reuse generic guideline standards, yet tailor them to their particular experience, organization, and setting. We are convinced that the guideline representation language presented in Sections 3.1 and 3.2 can support these objectives.
To further support the use of this language, we are currently evaluating several options for constructing a structured English editor/KB wizard that will interview authorized users for new guidelines and/or allow them to graphically display and edit the existing guidelines. Some of the options we are investigating are an existing ARDEN editor, Musen et al.(1996)'s Protégé II, Fox (1997)'s PROMPT, and our own COPE question asking engine (Silverman 1992).

Tables 1 and 2 illustrate many of the types of elements that we need to collect from users. Others are indicated in Moidu et al. (1996). They must be able to specify new KBs, theories within KBs, plans to implement the theories, arguments to support the plans, warrant-types and specific warrants for the arguments, and so on. We are constraining the argumentation taxonomy and lexicon, where feasible, to pulldown menus listing open standards that we are currently CORBAfying, such as HL7 object names, well known coding systems such as ICD9 & 10 diagnostic codes, and ARDEN syntax items, among others. This framework will have to parse the guidelines into ARDEN BNF syntax, as well as parse it into the machine readable KBs that will bind to the electronic records (Fi) and generate the agent plans. A primary difference between the editor needed here and the editors mentioned above lies in the requirements to author/update guidelines in a world wide web browser and, more importantly, to produce CORBA compliant objects. Thus the guidelines must be immediately reusable by any patient record system that complies with open standards for software (e.g., CORBA) and data encodings (e.g., HL7). This approach could serve as an extension of the data section of the Arden Syntax Standard for medical logic modules, a point further discussed in Section 5.1.

3.3.4) Temporal Algebra

A final feature of equation 3 concerns its time parameters and how R2Do2 reasons about time. Much of the early situational calculus literature models time as passing only when a new state or situation arises. A continuous process is modeled primitively as a succession of discrete states. Fortunately, though, time is a central topic of much of the nonmonotonic literature and numerous proposals exist for how to implement it (e.g., see

McDermott 1982, Allen 1984, Shohan 1986, Lee et al. 1993, Fisher 1996). This literature distinguishes the discrete from the interval calculi, but we utilize elements of both. Situations hold over intervals, but arise and terminate at discrete points of time. Similarly, plans may involve sequences (Pk) that involve numerous discrete actions over an interval, such as prepare glucometer, measure blood glucose, report blood glucose, inject insulin if glucose incorrect, report injection. Chronic conditions, such as diabetes among others, have recurring plans, Pnk, that stipulate the recurrence schedule of the sequence of plans. Vaccines involve recurring plans as well.

In order to explain the time parameters of the TODO item and how they subsequently get managed by the agents, it is necessary to introduce a notation for R2Do2's intervals and time points. An important concept in Allen's algebra is the interval, I, which is represented as a tuple of real numbers (IB, IE) with IB < IE, denoting the beginning and end points of the interval, respectively. In dealing with events that appear to be points in time, Allen considers such events to be implicitly referring to another event's beginning or ending. An example interval is the interval of action, I_{action} or I_A, during which a plan's actions are carried out.

At this point we depart from Allen in that we define several kinds of intervals. The first is as just described. A second form of interval used in the R2Do2 system as a period of time defined by the number of units in the period and the size of the unit of time. This unit of time we refer to as a Chronon, borrowing from Wade et al. (1994). We represent this type of interval as the tuple (Unit, Chronon). Wade defines a chronon is a nondecomposable unit of time, multiples of which are concatenated to form intervals. The most natural duration of a chronon should be the time resolution of the data. Wade adopts the calendar day as the duration of a chronon. We found that this resolution is insufficient for the needs of R2Do2, so have adopted a model of chronons with varying resolution from a quarter hour (or 15 minutes) to a year, depending on the event. R2Do2 needs to be able to establish temporal relations between intervals in terms of chronons of different sizes (from multiple medication doses per day to multiple year intervals between immunization boosters).

In addition to the chronon and the action interval, I_A, we need to define the separation interval, I_S, that indicates how long one must wait before a recurring action may be repeated. Using these concepts, we are now in a position to fully specify the "time parameters" (TP) of earlier Equation (3). Specifically, for any Pk, let the time parameters of the ToDo of Equation (3) include the following, many of which are optional settings:

$$\text{Time Parameters} = [(I_A B,\ I_A E,\ \text{Chronon}_A),\ (I_S B,\ I_S E,\ \text{Chronon}_S),\ (\text{RC or } (R_D E,\ \text{Chronon}_D))] \qquad (3.1)$$

subject to,

$\text{MEETS}(I_A E,\ I_S B)$
$\text{MEETS}(I_S E,\ I_{A+1} B)$

where,
$Chronon_A$ = units of the action interval
$Chronon_S$ = units of the separation interval
RC = repeat or recur count
R_DE = recur duration endpoint (the start of the recur duration is I_AB)
$Chronon_D$ = recur duration units
MEETS(X,Y) = a basic relation of two discrete time points X and Y where X < Y, but there is no interval between them.

The time parameters thus consists of up to three tuples, one for the action interval, one for the separation interval, and one for the recurrence information. The end of the action interval equals the beginning of the separation interval, the end of which is the start of the next action interval. In the case of one-time events, the third tuple will be empty. When the interval is one chronon in length, the interval endpoint may be omitted. Also, if the action is ToDo something "now", both the chronon and the action interval endpoint may be omitted. Recurring events, in turn, may specify either RC, the number of recurrences expected, <u>or</u> the time at which repetitions should be ended. Three examples should illustrate some of the flexibility of this approach.

Yearly Screening: Perform routine screening once a year on January the 10^{th}.
Due Date (I_AB): January 10, 1997
Separation Interval (I_SE, $Chronon_S$) : (1, year)
Recur Count(RC): infinite, while consumer lives

Medication Course: take pill twice a day for 7 days
Action Interval (I_AB, I_AE, $Chronon_A$): (11/25/96, 11/25/96, date)
Separation Interval (I_SE, $Chronon_S$) : (12, hours)
Recur Count (RC) : 14
or
Recur Until Duration (R_DE, $Chronon_D$) : 7, days

Vaccination Sequence: Follow a varying schedule of DPT shots depending on age
Action Interval (I_AB, I_AE, $Chronon_A$): (2, 2.5, months)
Separation Interval: (data recomputed by Kbj for irregular intervals)
Repeat Count (RC) : (empty until periodic booster begins)

The first two examples contain all the information a recurring plan, Pnk, needs for the life of the consumer and should be self-explanatory. It is worth pointing out though, that for Example 1 the separation interval begins when the action actually occurs. So, if the consumer procrastinates on any given year, the next appointment will be 12 months after that. By contrast, the event-class for pills ties the separation interval to the ideal schedule, and the actions are set for every 12 hours regardless of user behavior (this can be overridden by user preference). The last example (drawn from Table 1) shows that the first vaccine administration occurs between 2 and 2.5 months. Subsequent repetitions are at variable intervals and of varying type (booster after a certain age). So the next recurrence must be computed by firing the rules once again after the first administration. This firing is precipitated once the prior vaccine is entered into the OODB, which pushes the update message onto the Planner's queue. Once the boosters begin, they repeat every ten years until the end of life. At that iteration, the KB provides a regular rule where separation interval and repeat count are filled in more like Example 2.

The results of the KB firings are a set of Pnk for each consumer. These are passed from the ToDo Item Generator to the Agent Launcher who must prepare the Pnk and the agents that execute them.

3.4) Agent Launcher

The Agent Launcher receives Pnk not only from the ToDo Item Generator but also from several other sources. One source is from the users via R2Do2's user screens. These user-generated Pnk may come from a given consumer or that consumer's providers. A final source of new Pnk are Dk and Mk messages from the Incremental Plan Adaptor (or user) which, as may be recalled, do not need to be routed through the ToDo Item Generator. The Agent Launcher, collects all these Pnk, and instantiates numerous agents to carry them out. Since each agent carries out one Pnk, a consumer's health plan, HP(Consumer), may contain many agents at any given point in time.

As there may be numerous agents carrying out a given HP, the agent launcher carefully performs the following functions:

1) Perform final computations on the Pnk received since the last iteration.
    - convert action and separation interval data to actual calendar dates,
    - instantiate event class attributes (priority, default reminder schemes, etc.)
2) Optimize across the Pnk of a given consumer's HP (i.e., using the 13 relations of Table 2, detect and combine any visits to same clinician that occur within an interval tolerance and subject to the action priority obtained from the event-class. Update the ToDo time parameters accordingly)
3) Instantiate a new agent for each Pnk or Mk from the generic-agent and generic-agent-message objects, respectively. Fill in its plan slots, method slots, and so on.
4) Launch the agents using a CORBA agent factory, or broadcast the Mk or Dk message object references.
5) Return to step 1.

The present version of the Agent Launcher performs most of these five steps reasonably well. The exception is that Step 2 is still primitive and few types of optimizations can yet be carried out. This is an important area for future research with potentially high payoff in increased user satisfaction. At present R2Do2 too often relies on the users to aggregate visits to the same clinician for different Pnk.

4) Personal Assistant Agents

The Personal Assistant Agents (PAAs) take over control of the 2Dos launched by the Agent Launcher. In describing the temporal reasoning of the PAAs, we have adopted some further elements of Allen(1984)'s interval algebra. Allen's interval algebra is based on the idea of relations between pairs of temporal intervals. The most fundamental predicate is HOLDS, which states that a property or plan is true during a time interval. HOLDS(P,I) is true if and only if plan P is true during interval I.

We adopt a simplification of this predicate that excludes the reference to the interval I, rather HOLDS(P) is true if and only if plan P is true at the present time. Recall that we represent an interval I_{Action} as a tuple ($I_{Action}B$, $I_{Action}E$) of points in time, with $I_{Action}B < I_{Action}E$ denoting the begining and ending points respectively. The following table shows the predicates that represent the relations of interest to R2Do2.

Relation	**Representation**
MEETS(X,Y)	XXXYYY
BEFORE(X,Y)	XXX YYY
AFTER(X,Y)	YYY XXX
DURING(X,Y)	XXX YYYYYY

We introduce additional predicates DONE, where DONE(Pk) asserts that plan action Pk has been taken, and NULL which asserts that the referenced variable has no value.

The PAA performs two basic functions in support of a given plan -- remind about a ToDo item, and determine the next recurrence. These two functions correspond to the lower and upper loops, respectively, of the state transition diagram of Figure 5. Specifically, the Launcher sends the instantiated agent a plan or an Mk. If its sent an Mk, the agent awakens and checks if the Pk no longer HOLDS. If instead its sent a plan, the agent goes to sleep until the action interval is to begin. The agent is responsible for reminding the consumer and/or the provider about upcoming actions that need to occur as part of a given plan. This reminding takes place at specified Lead Time Intervals before the Interval of Action. The consumer can provide the PAA with the Lead Times, or the system defaults for the different event classes can be used. Second, the PAA is responsible for iterating recurring actions within a plan.

In the algorithm below, line 1 represents the sleep state. Once awakened, if the plan P_NK no longer holds, the plan is terminated and the PAA exits operation (line 3). If the individual action Pk is done and the Interval of Separation is not null, the PAA computes the next occurrence of the plan P_NK (line 4) and skips the remind section of the algorithm. If the Pk is not done, lines 5-7 determine the type of action (Remind or Alert) that the PAA will take, depending on the temporal relation between the Interval of Action and the current point in time. Lines 8-10 determine the next Interval of Sleep End point, depending on the user preferences (UP) or the system defaults.

Variables used in the PAA algorithm are defined as:

- I_{Action} is the interval during which an action should take place, with endpoints $I_{Action}Begin$ and $I_{Action}End$
- I_{Remind} is the interval between reminders about an action, units in terms of $Chronon_{Remind}$. I_{Remind} is not used when the interval between reminders is variable.

- Lead Time(LT) and Lead Chronon(LC) are variables whose values specify the timing of reminders for the 2Do. LT is an integer array where each item describes the number of chronons of time before I_{Action}Begin to remind. LC is an array of chronons (hour, week, year) cooresponding to each LT. When the interval between reminders is fixed, the LC and LT arrays will contain only one item each, describing the interval between the first reminder and the I_{Action}Begin, all subsequent reminders will occur at I_{Remind}*$Chronon_{Remind}$ intervals.
- I_{Sleep} is the interval during which the agent sleeps. It has endpoints I_{Sleep}B which is generally the current point in time, and I_{Sleep}E which is determined by default or user specified schemes (LT, LC, and/or I_{Remind}, $Chronon_{Remind}$).
- ι is the number of reminders created for a given Pk.

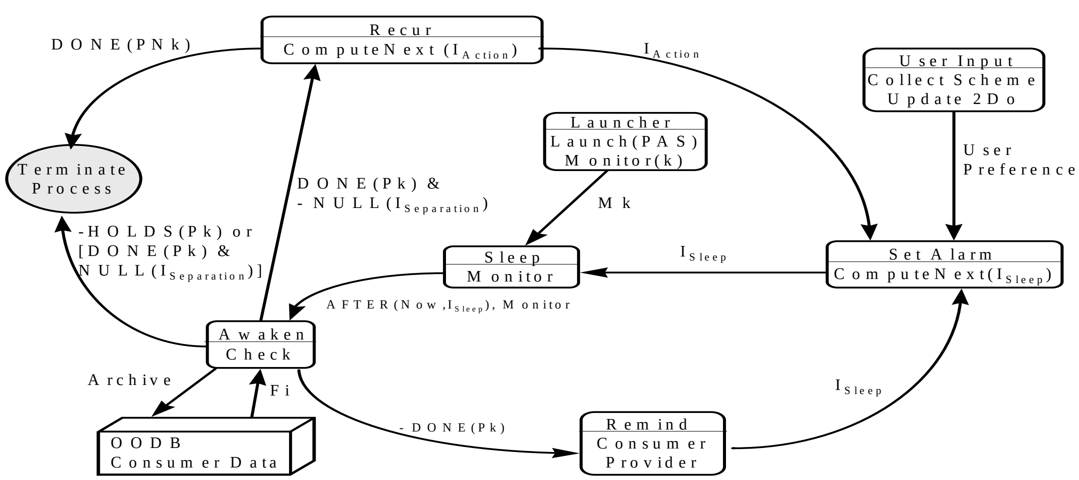
Using event-class information, user preferences, and screen input such as in Figure 6, for each Pk in P_Nk during Sit(s) the PAA algorithm must assign and execute one of two basic reminder schemes:

$$\forall k,n \quad \text{REMIND}\ \{\text{Pkn}\left(\left| \begin{array}{l} \text{RS1 [(LT1, ...LT5), (LC1, ... , LC5)]} \\ \quad \text{or} \\ \text{RS2 [LT1,LC1, } I_{Remind}\text{, } Chronon_{Remind}\text{]} \end{array} \right| \right)\} \qquad (4)$$

Example Reminder Schemes:

- Appointment: Remind 30, 14, 7, 1 and 0 days before Appointment Date (Due Date);
  Lead Time: 30,14,7,1,0
  Lead Chronon: days
- Medication: Remind every 2 hours starting 2 days ahead
  Lead Time (LT): 2
  Lead Chronon(LC1): day
  Remind Interval (I_{Remind}): 2
  Remind Chronon ($Chronon_{Remind}$) : hour
- Yearly Screening: Perform screening once a year, Remind 5 times;
  Recur Interval: 1
  Recur Chronon: year
  Recur Count: 0 - forever
  Lead Times: 2,1,2,1,1
  Lead Chronon: month, month, week, day, hour

Figure 5 - State Transition Diagram of a Personal Assistant Agent



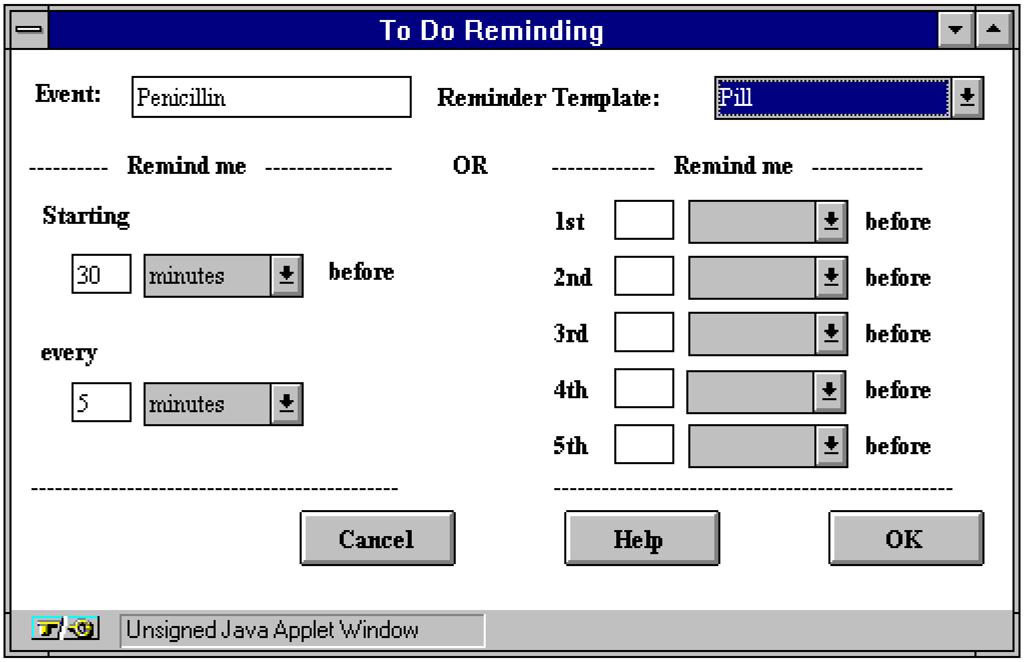


Figure 6a - ToDo Reminding Screen

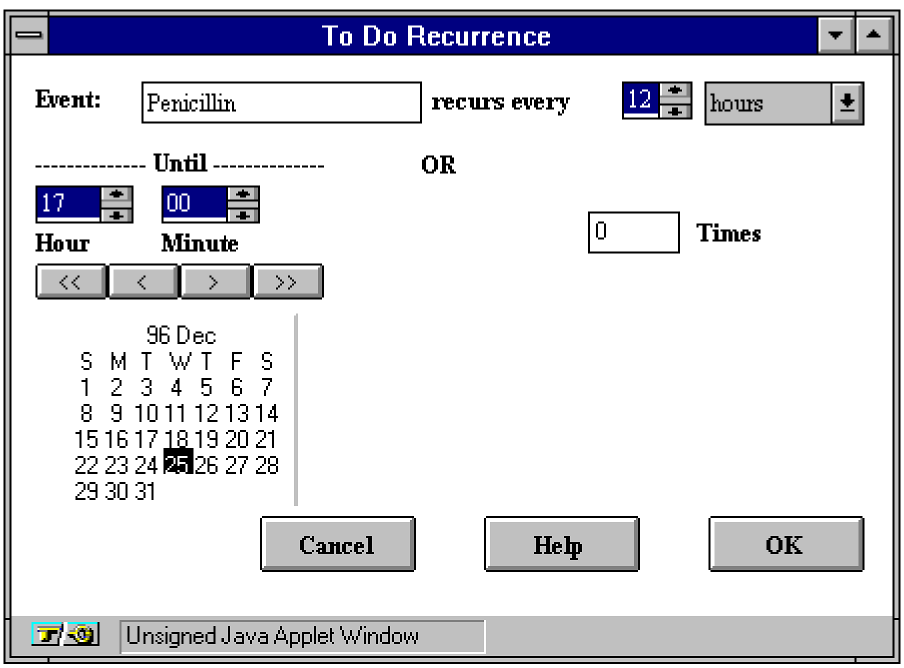


Figure 6b - ToDo Recurrence Screen

The PAA Algorithm executes the schemes, sending web reminders and email alerts over the CORBA bus while conserving computing resources during separation intervals. It does this via the following steps that implement the state transitions of earlier Figure 5:

1) While BEFORE(Now, $I_{Sleep}E$), sleep
2) Wake
3) -HOLDS($P_N K$) $\rightarrow$ Terminate $P_N K$ {DeletePlan($P_N K$)}, Exit
4) DONE(Pk) & -NULL($I_{Seperation}$) $\rightarrow$ ComputeNext2Do, Go to 8
5) DURING(Now, I_{Action}) $\rightarrow$ Alert(Consumer, Action, Message, Scheme), Continue
6) BEFORE(Now, I_{Action}) $\rightarrow$ Remind(Consumer, Action, Message, Scheme), Continue
7) AFTER(Now, I_{Action}) $\rightarrow$ Alert(Consumer, Provider, Action, Warning, Scheme), Continue
8) AFTER(Now, I_{Action}) $\rightarrow$ $I_{Sleep}E$ = Now + Min Sleep Interval, Return to 1
9) -NULL(I_{Remind}) $\rightarrow$ $I_{Sleep}E = MIN[I_{Action}E, (I_{Sleep}E + (I_{Remind} * Chronon_{Remind}))]$, Return to 1
10) NULL(I_{Remind}) & ($\iota < 5$) $\rightarrow$ $I_{Sleep}E = MIN[I_{Action}E, (I_{Sleep}E + (LT\iota * LC\iota))]$
11) $\iota = \iota + 1$
12) Return to 1

The ComputeNext2Do algorithm mentioned above, is responsible for calculating the Interval of Action for the next occurrence Pn+1 of the recurring plan $P_N k$, and the initial Interval of Sleep. Line 1 assures that the current occurrence Pn is in fact done, and line 2 checks for the variables required to proceed. Lines 3-4 calculate the Interval of Action for Pn+1 by adding the Interval of Separation to the Interval of Action of Pn. In line 5, the number of reminders ι is reset to zero, since the reminders for Pn+1 need to follow the same schedule used for Pn. The beginning of the Interval of Sleep is set to the current point in time in line 6, and its end is set in line 7 to the point in time of the first reminder of Pn+1. If the recur count is equal to 1 it is set to null (line 8), since this $P_N k$ has run its course and will not recur again. Otherwise, the recur count is decremented by 1 in line 9. Additional variables used in the ComputeNext2Do algorithm are defined as:

- $I_{Seperation}$ is the interval of separation between recurring actions.
- $Chronon_{Seperation}$ is the unit of time referenced by $I_{Seperation}$
- $Recur_{Count}$ is one of the variables that determine the recurrence behavior of an R2Do2. When $Recur_{Count}$ is a positive integer (>0), it represents the number of times the 2Do should recur. When $Recur_{Count}$ is zero, it indicates that the 2Do could recur infinitely. When $Recur_{Count}$ is null, the 2Do does not recur.
- $Recur_{Duration}E$ is the endoint of the interval of duration. Its starting point is $I_{Action}B$(P1), where P1 is the first action.

The ComputeNext2Do algorithm is as follows:

1) DONE(Pnk)
2) -Null($I_{Seperation}$) & [BEFORE($I_{Action}E$, $Recur_{Duration}E$) or -Null($Recur_{Count}$)] $\rightarrow$Continue, else exit

3) $I_{Action}B(Pn+1) = I_{Action}B(Pn) + I_{Seperation}(P_NK) * Chronon_{Seperation}$
4) $I_{Action}E(Pn+1) = I_{Action}B(Pn+1) + I_{Action}(P_NK)$
5) $\iota = 0$
6) $I_{Sleep}B = Now$
7) $I_{Sleep}E = I_{Action}B(Pn+1) - LT1_{PNK} * LC1_{PNK}$
8) $Recur_{Count} = 1 \rightarrow Recur_{Count} = null$
9) $Recur_{Count} > 1 \rightarrow Recur_{Count} = Recur_{Count} - 1$
10) Return to 1 of PAA algorithm

5) Lessons Learned

This paper described efforts to develop and field an adaptive health planner and parallel agents that securely connect distributed users and data sets in a collaborative approach able to anticipate health ToDo items and to remind and alert about these items over the web. The planner and agents, reasoning temporally and non-monotonically, are able to handle changing health conditions, unknown health facts, user wellness and lifestyle preferences, and updates about each user's situations. Usage of and extensions to argumentation, qualitative decision making, situational calculus, and temporal algebras are described. Defeasible assertions, default arguments, support sign dictionaries, closed world assumptions, and anytime algorithms are used to derive feasible health plans that are later updated as more information unfolds. A set of robust, open standards-based client server approaches (e.g., HOLON, CORBA, ODBC, Java, and ANSI HL7 Common Object Model) are used to encapsulate and plug all these components into the web, and to handle the distributed agent and object communications. Extensions to these emerging standards are at times made due to engineering and domain considerations, however, the goal is to maximize reuse of the guidelines by any patient record system that adheres to the standards.

At the start of this paper, four challenges to web-based applications were introduced that we return to in the next four subsections.

5.1) Merging Data- with Document-Centric Computing

Web pages with their hyper-text markup language (HTML) are a document-centric mode of computing that has revolutionized usage of computers. In R2Do2, guideline KBs, questionnaires, preference elicitation screens, and ToDo item messages and advice utilize this document modality. In trying to extend this approach to the data-centric side of health information management systems, several difficulties arise that are not easily overcome. These difficulties center around the collection, storage, retrieval, and maintenance of each user's health information and situations, what is earlier defined as Fi. For a new user, the Fi is a large set of questions to answer. This is often called the health profile or patient interview, and it may involve collecting hundreds of answers. Ideally, one would like to use a question engine that can branch over interview rules and collect those answers most relevant to a given user's health situation. Unfortunately, the web standard inhibits reaching such an ideal if one seeks to adhere to HTML standards such as cgi forms and Java virtual machines. For example, one cannot download the Java inference engine and run it within an applet since local memory resources cannot be written to. Waiting for the engine to read and write assertions on a remote client is impractical. Likewise downloading a plugin interview system can be discouraging to users. The simplest alternative is to use fixed questionnaires about each health area and situation, and to stream these to the user as cgi forms or Java applets. This means that Fi are collected in a more exhausting, less user-sensitive manner than one would like. As a result, critical Fi may remain unanswered longer than would otherwise be necessary.

At the start of this paper, several paradigm shifts in the health care field were mentioned, one of which was the idea of empowering users by connecting them to access and manage their own health data. Using the web browser, Internet Inter-ORB Protocol (IIOP), an ORB bus, and the ODBC standard we have demonstrated that this is feasible, as have others before us through comparably open standards (e.g., as did Kohane et al., 1996 with HTTP). However, there are numerous, serious problems to this approach. Here we will concentrate on the connectivity problems, while subsequent paragraphs will address programming and security aspects. The connectivity problems straddle the syntactic and semantic levels. At the syntactic level, there is the basic problem of connecting the "pipes." The main obstacle to point out here is that the CORBA standard has not yet matured to the point needed. In particular, the ORB services include, among others, a trader, a name server and a database query server. The name server standard, however, is still underspecified so that it is not possible to locate a given patient and their records -- that is being addressed as of this writing by request for proposals for a master patient index standard by the CORBAmed committee. The query server standard within CORBA was adopted in the summer of 1996, but that standard in turn adheres to the SQL3 and Object Query Language (OQL) standards that are not yet widely available in current database products. For these reasons, we found we had to implement our own name and query servers. In the former case we are depending on a HOLON consortium partner (Concept Five), while in the latter case we replaced the OMG Query Server with a Java ODBC-standard based connection outside the ORB. That is why earlier Figure 1 shows the distributed databases in our current testbed connecting through the data server, rather than directly to the ORB.

At the information content level, there are more obstacles to the connectivity problem. Our goal is to utilize site-independent software and lexicons via use of common object standards such as ANSI's HL7, among others. With this approach, each institution that wishes to connect its legacy systems to R2Do2 must go through a translation and wrapping effort as is currently ongoing at our Beth Israel-Deaconess and Norwalk Hospitals testbed sites. Once they conform to the accepted standards, they can use the reminder engine since R2Do2 attempts to restrict its guideline lexicon and patient data references to the same accepted vocabulary standards. However, the vocabulary sets that need to be referenced are not under the purview of any single standard setting body. There are a variety of authoritative or de facto standards such as, among others, ICD-9 diagnostic codes, LOINC laboratory order codes, Observation Ids, and so on that are not part yet of the HL7 standard. We are trying to stay with the more well established of these coding schemes as embellishments where HL7 is underspecified. The ultimate settling of these vocabularies is a challenge the standards groups need to settle for this approach to work. Another challenge for standards setting is that HL7 tables do not currently exist that support hierarchical inferencing, such as , for example, “has the patient taken any type of pertussis vaccine?” At present one must manage all such processing within R2Do2. It would be useful if the standards would recognize decision support needs as well as information retrieval.

A final issue concerns the Arden Syntax standard for medical logic modules. Our guidelines KBs conform to this documentation standard in an effort to promote their reusability and local modification. A future version of R2Do2 will also be providing an HTML viewer of the guideline KBs in Arden form. We also would like it to permit English level editing of the KBs. One major obstacle to this idea is the Arden Syntax’s inability to standardize how a medical logic module gets access to data. It needs both events and the ability to retrieve from the patient's "record" or from the data repository. The Arden Syntax Standard leaves the precise access syntax unstandardized, with the idea that it would be written locally because of the different IS environments. This inhibits portability and reuse of logic modules since the local rewriting of that section is time consuming, error prone, and requires person to person consultation between someone at the author's institution and the person attempting to reuse it. An alternative we are trying to provide for this section is to embed a full set of CORBA interfaces directly connected to HL7 and other (finer grained) standard code "events" (e.g., ‘a new DT booster was administered’) pushed to it by the object layer of the medical record data repository (Fi). To date, our approach seems feasible and we think it would prove a viable way to extend the Arden Standard. It would also allow the creation of structured-English level editors of the guideline KBs and the various rule elements (events, actions, etc.) in which users could avoid the need to code access connections or to deviate from approved repository vocabulary.

5.2) Integrating Personal Assistant Agents

Solving the AI side of a class of applications, such as R2Do2 does, is no easy task. It involves working out theories, algorithms, engines, typical planning problems (qualification, ramification, frame, closed world, defaults, etc.) and KBs. At the end of such an effort, the typical AI person expects to be done with the project. However, as this article amply illustrates, integrating the AI into the Internet and the web adds an entire new layer of issues and effort. The costs of that level included about a person-year-equivalent of added effort, above and beyond the AI- and KB-related effort. The advantage of this effort is that as soon as it is done, the AI environment is immediately available to the world.

A related issue is that no matter how much effort one puts out, there are some obstacles that can’t be eliminated. With enough effort, one can overcome CORBA and IDL learning curves, database connectivity obstacles and so on, but some aspects of the agent-web integration are immovable. For example, in the field of reminding, the PIMs have set a standard for the metaphors and screen look and feel. Our screen widgets to date are restricted to those shown in Figures 2 and 6. Without resorting to PIM plugins, the screen interfaces are relatively plain. While this will change in the near future, as of this writing, we have few widgets for calendar presentation, day and weekly planners, alarm bells, clocks, recurrence sliders, and so on. Using PIM plugins would solve this dilemma, but would violate the web ideals of easy portability, rapid streaming, and local resource security. Also, installing the plugin on low end machines, those with low baud rate modems and limited extra memory, might be unappealing to the users who own them.

Whether embedding the R2Do2 planner and agents in PIMs or directly in the web, this is a good place to bring up an important issue we have been grappling with throughout this article -- that of reliability of the agent and its advice. There are two levels of reliability of concern here, logical and engineering reliability. At the logical level, R2Do2 uses a mathematical framework for organizing its distributed processing and temporal reasoning. The discussion at numerous points has attempted to draw parallels between this framework and the more traditional frameworks of the nonmonotonic logic literature. It would be interesting to explore a more formal framework of logic for R2Do2, one that addresses satisfiability and correctness proofs of the HealthPlan. However, in the interest of practicality, R2Do2 did not start with such a rigorous goal. There are numerous concerns that such a framework must overcome, many of which have been mentioned such as the frame and qualification problems, the lack of a complete KB of rules to defeat all defaults, the computational difficulty of satisfiability tests required in proof procedures, and others. Nevertheless, a more formal approach or alternate formalisms may offer numerous benefits (maximizing health, assuring cost objectives, satisfying user preferences, etc.) and they warrant further research.

From the engineering reliability perspective, no matter how reliable a plan or plan defeater may be, it has no value if it does not reach its user properly. This brings up a number of issues we have so far ignored about Internet communication failures. At present, we have not ensured recurring reminders cover all the scenarios for communication failure that may arise. One of the reasons CORBA works is that it is essentially stateless. It may be that our planner and agents are too stateless as well, particularly in terms of tracking the outcome of their conversations with one another. What happens if R2Do2 launches an Mk that is not received by the agent or sends a health alert email about a consumer that is never received by the caregiver? These and other engineering reliability issues have not yet received sufficient attention that we can adequately answer such questions. If R2Do2 is to graduate from a web curiosity to a serious tool for institutions and individual consumers, we will need to conduct extensive engineering studies and reliability hardening efforts.

5.3) Web-Interactive Social Presence

One can get numerous reminder and alert benefits from a personal version of R2Do2 that doesn't need to connect to anyone else. Provided the user inputs their health profile, Fi, the guideline KBs will fire and generate ToDo items and plans. Also, R2Do2 offers useful templates and default reminder schemes for user-specified ToDo items such as pills, recurring shots, a few areas of chronic care, and the like. All that is needed to get these features is a Java-enabled web browser.

Yet the institutional version of R2Do2 seeks to extend these features. When R2Do2 is connected to institutional databases and patient record systems, it can give the caregivers reminding and alerting messages about patients coming in for an appointment that day, patients needing appointments, and consumers who have situations requiring attention. Consumers in turn get connectivity to their caregivers both via email reminders, and for chronic care triggers. For this connectivity to happen, in addition to a Java- and email-enabled web browser, each user must install a terminal stay resident alerter and email parser on their machine. This is a Java application. Since they are installing these base items, one could bundle this with other collaboration features useful for a remote care environment, such as desktop videoconferencing, electronic whiteboard, and support group chat space. Our current testbed for the HOLON consortium includes such items. Results to date show they do not increase the complexity or effort for the R2Do2 developers.

5.4) Security and Privacy on the Web

Although these are critical issues, our approach to security and privacy thus far has been purposefully low key. For the personal version of R2Do2, security and privacy would be preserved by the consumer if they could install the entire system on their local machine. Thus one could price a commercial version that would maintain security and privacy relatively easily. This would be unaltered if we embed R2Do2 within a PIM.

If we maintain a central server for supporting personal usage, however, this perception of security and privacy is compromised. Most users would probably be reluctant to place personal data on a quasi-public server, even if we assured them our current version has been implemented within the Java virtual machine to prevent foreign code reading/writing on their local disk, with secure sockets during ORB transmissions, with access password-controlled by the name server, and with authentication and auditing provided by a HOLON security module: i.e., see Harris et al. (1997). Still, we have taken no further effort on the belief that the web in conjunction with vendors like Visigenic (ORB), Concept Five (HOLON Security Module), and others will ultimately produce a solution that is widely perceived as allaying security and privacy concerns. For institutional intranets that adopt firewalls and pursue vigorous security management procedures and policies, our current configuration may be close to what they will find acceptable. This latter issue is being explored by the Beth Israel-Deaconess Medical Center in the current HOLON testbed and will be the topic of a future report.

5.5) Concluding Comment

R2Do2 is an experiment in extending middleware in the healthcare field. This framework makes use of open standards for architecture, software, guideline KBs, clinical repository models, information encodings, and intelligent system modules and agents. By pursuing the use of such standards we hope eventually to maximize immediate reusability of the R2Do2 framework by others who also adhere to these open standards. This research also tries to reflect lessons learned about the extensions needed in these standards if healthcare middleware frameworks are to transparently support users over the web.

ACKNOWLEDGEMENT
The financial support of NIST/ATP, the HOLON Consortium, KOOP Foundation, and Beth Israel Deaconess Medical Center/Harvard Division of Clinical Computing are gratefully acknowledged.

REFERENCES

Allen, J.F., "Towards a General Theory of Action and Time," Artificial Intell, v. 23, 1984, pp.123-54.

Anon.1, Introducing Health Point ACS, Durham: HealthPoint Inc., technical specification, 1996.

Anon.2, vCalendar: The Electronic Calendaring and Scheduling Exchange Format, versit Consortium, Spec. vers.1.0, Sept. 1996.

Dickey, L.L. (editor), Guide to Clinical Preventive Services (2nd Ed.), Washington DC: US Public Health Service (http://text.nlm.nih.gov/cps/about.html), 1996.

Bernstein, P., "Middleware: A Model for Distributed System Services," CACM, v.39, n.2, Feb. 1996, 86-98.

Das, S., Fox, J., Eldson, D., et al., "Decision Making and Plan Management by Autonomous Agents: Theory, Implementation, and Applications," Agents'97 Conference Proceedings, New York: ACM, 1997.

Drummond, M., Bresnia, J., "Anytime Synthetic Projection: Maximizing the Probability of Goal Satisfaction," AAAI-90 Conf. Proc., San Mateo: M. Kaufman, 1990, pp. 138-144.

Einarson, T., "Drug Related Hospital Admissions," The Annals of Pharmacology, v. 27, 1993, pp. 832-40.

Feld, S. (chair), et al., Guidelines for the Management of Diabetes Mellitus, American Association of Clinical Endocrinologists, Diabetes Guidelines Task Force, 1995 (http://www.aace.com/guidelines/diabetes_guide.html)

Fisher, M. "An Introduction to Executable Temporable Logics, Knowledge Engineering Review, v. n. 1996, pp. 43-56.

Fitzmaurice, J., Putting the Information Infrastructure to Work: Health Care and the NII, Washington DC: DHHS, AHCPR Pub. No. 94-0092, June 1994.

Fridsma, D.,B., Gennari, J.H., Musen, M.A., "Making Generic Guidelines Site-Specific," Stanford: SUSM/Medical Informatics, tech report, 1996.

Friedman-Hill, E., Java Expert System (JESS), Livermore: Sandia National Lab, http://herzberg.ca.sandia.gov/jess, 1996 (vers. 2.2.1).

Giarratano, J.C., CLIPS Users Guide, Houston: NASA/JSC, JSC-25013 (vers.6), 1993.

Haddawy, P., A Logic of Time, Chance, and Action for Representing Plans, Artificial Intelligence, v. 80, 1996, pp. 243-308.

Halloran, M., et al., "White Paper: Accelerating the Movement Toward Standards-Based Interoperability in Health Care," New York: IEEE-USA Medical Technology Policy Committee (Clinical Information Systems Subcommittee), May 15, 1996.

Harris, L., et al., "Personal Security Object: Status Report," HIMMS Conference Proceedings, San Diego, 1997.

Jonsson, P., Backstrom, C., "A Linear Programming Approach to Temporal Reasoning, National Conference on AI Proceedings, Menlo Park: AAAI Press, 1996, pp. 1235-40.

Kohane, I.S., et al., "Building National Electronic Medical Record Systems via the World Wide Web," JAMIA, v.3, n.3, May/Jun 1996, pp. 191-207.

Lee, H. Tannock, J., Williams, J.S., "Logic Based Reasoning About Actions and Plans in Artificial Intelligence," Knowledge Engineering Review, v.8, n.2, 1993, pp.91-120.

Lewis, T.G., "Where is Client/Server Software Headed?" Computer, April 1995, pp. 49-55.

Lyons, D.M., Hendricks, A.J., "A Practical Approach to Integrating Reaction and Deliberation," in J. Hendler (ed.), Artificial Intelligence Planning Systems, San Mateo: Morgan Kaufman, 1992, pp. 153-162.

McCarthy, J., Hays, PJ, "Some Philosophical Problems from the Standpoints of Artificial Intelligence," Machine Intelligence, v.4, 1969, pp. 463-502.

McDermott, D., "A Temporal Logic for Reasoning about Processes and Plans," Cognitive Science, v.6, 1982, pp.101-55.

McDonald, C., "Protocol-Based Computer Reminders," New England J. Med., v. 295, 1976, pp. 1579-81.

Moidu, K., Falsone, J.J., Leehy, M., et al., Guidelines On-Line for HOLON: Concepts and Challenges, Norwalk: Norwalk Hospital/Hinds Research Center, white paper, 1996.

Otte, R., Patrick, P., Roy, M., Understanding CORBA: The Common Object Request Broker Architecture, Englewood Cliffs: Prentice Hall, 1996.

Pollock, J.L., "How to Reason Defeasibly," Artificial Intelligence, v.57, 1992, pp.1-42.

___________, "Justification and Defeat," Artificial Intelligence, v.67, 1994, pp.377-407.

Rishel, W., "Software Components, the Clinical Workstation, and Healthcare Networks," in Proceedings of the Healthcare Information Management Support Systems (HIMSS) Conference, pp. 1-13, Feb. 1996.

Safran, C., Rind, D., et al., "Effects of a Knowledge Based Electronic Patient Record on Adherence to Practice Guidelines," MD Computing, v. 13, n.1, Feb. 1996, pp. 55-63.

Schoppers, M., "Representation and Automatic Synthesis of Reaction Plans, Urbana-Champaign: U. of Illinois PhD Thesis (uiucdcs-r-89-1546), 1989.

Schrier, R., Improving the Quality of Healthcare Through Better Information: MediSource Progress Report, Denver: Multum, technical progress report submitted to Geo. Wash. Univ., Sept. 1994.

Shahar, Y., Miksch S., Johnson, P., "An Intention Based Language for Sharing Clinical Guidelines, AMIA Annual Fall Symposium Proceedings, Oct. 1996, pp.

Shea, S., DuMouchel, W., Bahamonde, L., "A Meta-Analysis of 16 Randomized Controlled Trials to Evaluate Computer-Based Clinical Reminder Systems for Preventive Care in the Ambulatory Setting," JAMIA, v. 3, n.6, Nov/Dec. 1996, pp. 399-409.

Shoham, Y., "Chronological Ignorance: Time, Non-Monotonicity, Necessity, and Causal Theories," Proceedings AAAE, 1986, pp. 389-93.

Silverman, B.G., Critiquing Human Error: A Knowledge Based Human-Computer Collaborative Approach, London: Academic Press, 1992a.

______________, "Computer Reminders and Alerts," Computer, v.30. n.1, Jan.1997.

Silverman, B.G., Owens, J., "Enterprise Information Infrastructure," Systems Engg Journal 1996

Silverman, et al., Health Object Library Online (HOLON), Gaithersburg: Koop Foundation Inc., White paper, 1996.

Tu, S.W., et al., "Ontology-Based Configuration of Problem Solving methods and Generation of Knowledge Acquisition Tools: Application of Protégé-II to Protocol Based Decision Support," Artif. Intel. In Med., v.7, 1995, pp. 257-289.

Wade, T.D., Byrns, P.J., Steiner, J.F., et al., "Finding Temporal Patterns - A Set Based Approach," AI in Medicine, v. 6, 1994, pp.263-271.

Warner, H.R., Olmstead, C.M., Rutherford, B.D., "HELP – A Program for Medical Decision-Making," Computers in Biomedical Research, v.5, 1972, pp. 65-74.

Zilberstein, S., "Using Anytime Algorithms in Intelligent Systems," AI Magazine, 17/3, Fall 1996, pp. 73-83.