

# Understanding intelligent agents: analysis and synthesis

John Fox, Martin Beveridge, and David Glasspool

*London Research Institute,  
Cancer Research UK,  
44 Lincoln's Inn Fields, London WC2A 3PX*

*{john.fox,martin.beveridge,david.glasspool}@cancer.org.uk*

Current views of intelligent agent technologies are reviewed with respect to (a) their general cognitive capabilities and (b) the classic *Belief-Desire-Intention* (BDI) model. A benchmark agent model is developed as a basis for analyzing and comparing agent systems. *PROforma* is an agent technology that has grown out of work in modeling medical expertise and the benchmark is used to carry out a case study analysis of this technology, looking at it from three contrasting points of view: logic programming, object-oriented programming and agent-oriented programming. These viewpoints yield different insights into the strengths and weaknesses of *PROforma* and lead to a clarification and consolidation of the benchmark agent features. The consolidated model offers a useful framework for analysis and comparison of other agent systems in medicine or other domains.

Keywords: Agent programming; Object oriented programming; Logic programming.

## 1 INTRODUCTION

Research on “intelligent agents” has diverse motivations, ranging from general theoretical interests in AI to practical objectives of building flexible distributed systems in specific application domains. As agent technologies mature so also does the interest in applying them in medicine. Agent technologies offer advanced platforms for building expert systems to assist individual clinicians in their work [20] and distributed agent systems have the potential to improve the operation of healthcare organizations, where failures of communication and coordination are important sources of error [6].

As computer science advances, however, there is a danger that older technologies are neglected. Worse still the advocates of different computing paradigms sometimes engage in unhelpful disputes about the

relative weaknesses of their preferred approaches, rather than looking for ways of combining their different strengths. Historical examples include the debates about logic programming versus functional programming, object-oriented databases versus relational databases and rapid prototyping versus formal software engineering. No one view has a monopoly on insight, however, and it would be desirable if novel paradigms did not simply displace old ones but that new understanding should accumulate on top of the old.

This paper presents a case study in analyzing and understanding intelligent agent systems. This case is *PROforma*, an agent language and technology which has developed out of research in logic-based decision making and autonomous systems [12], decision-support and workflow agents [9] and plan and process modeling [15]. The paper examines *PROforma* from three points of view: logic programming, object-oriented programming and agent-oriented programming. Each perspective provides a different way of understanding the technology, and reveals ways in which it might be improved.

The paper is organized as follows. First, we briefly review the “agent” concept, illustrating it with the classic BDI model but also considering a wider range of cognitive capabilities that the AI community has traditionally been interested in. We define a benchmark model that summarises many of these capabilities, and then turn to the case study. The general conclusions from the study are that although *PROforma* can be viewed as an instance of the benchmark agent, it also satisfies standard criteria of being logic-based and object-oriented, and that all three perspectives provide useful insights. The paper closes with a proposal for an improved benchmark that synthesises the insights from the three paradigms, and provides a general model with which to compare proposals for agent systems.

## 2 WHAT IS AN AGENT?

According to Webster’s dictionary an agent is:

1. *a person or thing that acts or brings about a certain result*
2. *one who is empowered to act for another*

The first part of the definition could include almost any software or hardware device and is not restricted to anything that the AI in Medicine community is interested in. The second part is more suggestive though the concept is not at all precise. Because the idea of an “agent” is somewhat vague it is now common to define agents in terms of their typical characteristics.

Caglayan and Harrison [2] identify the characteristic features of agents of the second type as entities that:

- perform tasks (on behalf of users or other agents)
- interact with users to receive instructions and give responses
- operate autonomously without direct intervention by users, including *monitoring* the environment and *acting* upon the environment to bring about changes
- show *intelligence* – to interpret monitored events and make appropriate decisions.

The first three criteria seem straightforward, but the requirement that an agent should “show intelligence” is more troublesome. Psychologists, educationalists and many others have found it difficult to come up with a precise definition of this everyday concept. As recently as 1999 Richardson writes “among scientists as well as among people in general, [intelligence] seems to be a remarkably flexible and ‘handy’ concept: a concept of convenience for filling in an argument at an intuitive level, but an embarrassment when brought out into the daylight”. Mugny and Carugati [29] describe human intelligence as a “polysemous” concept – many meanings serving many purposes, so presumably the idea of an intelligent agent is similarly multi-faceted.

Jennings and Wooldridge [24] avoid the ambiguity of terms like “intelligence” by limiting their definition to the *behavioural features* of agents. They suggested that agents are characteristically:

1. *Proactive* – showing the ability to exhibit goal-directed behaviour. “Proactiveness rules out entirely passive agents who never try to do anything.”
2. *Reactive* – having the ability to be able to respond to changes in the environment, including detecting that the agent’s goals have been thwarted in some way.

3. *Autonomous* – making decisions and controlling its actions independent of others.
4. *Social* – interacting with other self-interested agents through organized conversations, including conversations about cooperation and negotiation.

Wooldridge and Jennings [41] presented a further list of features, this time emphasizing *cognitive functions* that an agent should implement, including:

1. Maintaining an explicit *model* of the state of its environment, and perhaps its own “mental state” as well;
2. Raising and pursuing *goals* about the state of its environment, or its knowledge of the state of the environment;
3. *Perceiving* events that occur in its environment, recognizing when its goals are not currently satisfied;
4. Establishing *plans* to acquire information about the environment and to create changes which will be consistent with its goals;
5. *Implementing* these plans by acting upon the environment in order to bring about the desired change.

Rao and Georgeff [34] are usually credited with the first full implementation of an agent technology of this kind. A “Belief-Desire-Intention” (BDI) agent is able to monitor its environment, and maintain a database that symbolically represents the state of that environment (its *beliefs*). It can operate under the influence of a set of data-structures that refer to states of the agent’s environment, or the state of the agent’s knowledge of the environment, that it would like to bring about (its *desires*). Lastly, BDI agents can assess whether their beliefs are consistent with their desires and, if not, adopt plans of action (*intentions*) that are expected to bring its environment or its knowledge into line with them.

We conclude this brief overview by bringing together the behavioural and cognitive capabilities discussed above into a general definition of a *Unified Autonomous Agent (UAA)* that is intended to provide a benchmark model for the present discussion. At the time of writing the collection of capabilities covered by the *UAA* benchmark cannot all be found in any single technology implementation, but most if not all the capabilities have been individually demonstrated.

### 2.1 Benchmark agent definition

A *UAA* is an entity that exists in some sort of physical or informational environment that behaves at least in part independently of the actions of that agent.

The *UAA* interacts with the environment through various channels, viz:

1. *Perception*: acquisition and interpretation of data about environment states and events;
2. *Action*: output, including actions that change the state of the environment;
3. *Communication* with other agents in the environment through “conversations”;

A *UAA* maintains a *representation* of the state of its environment and its situation in the environment (sometimes called the agent’s “mental state”) that may include:

4. *Beliefs*: what the agent holds to be true about its environment and itself;
5. *Desires*: the agent’s current goals for itself and the environment;
6. *Intentions*: actions, plans or other tasks to which the agent is committed;
7. *Knowledge*: a special class of beliefs referring to general laws, ontologies, rules etc, rather than specific situations;

*UAAs* can implement *cognitive functions* over their mental states, such as:

8. *Reasoning*: making inferences on the basis of its beliefs, knowledge etc.;
9. *Decision-making*: making rational choices, often involving uncertainty;
10. *Planning* (constructing collections or sequences of actions to achieve its goals);
11. *Scheduling* (adaptively controlling its plans and actions as the environment changes);
12. *Learning* (storing and recalling past situations and solutions to problem goals).

*UAAs* also need to maintain flexible *control* of their behaviour in response to circumstances, achieving a balance of

13. *Deliberative control*: goal-directed, coordinated actions carried out over time;
14. *Reactive control*: responding to situations and events, even if unexpected.

Finally of course a *UAA* should be able to demonstrate

15. *Autonomy*: the capacity to plan, make decisions and act based on its own goals and beliefs, and independently of other agents.

The absence of agent systems that implement all 15 capabilities is not critical to the benchmark; given the polysemous nature of intelligence there is no single capability or combination of capabilities that is critical. We take the view that the notion of an intelligent agent cannot be defined categorically, rather the more features from the above set a system

demonstrates the more we are entitled to claim that it is an intelligent agent.

### 3 THE DOMINO AGENT MODEL AND THE PROforma LANGUAGE AND TOOLKIT

Our work in this area has been driven by attempts to understand clinical expertise and build autonomous systems for use in medicine. By 1996, when a consensus about the nature of “agenthood” was beginning to emerge, the AI in Medicine community had accumulated a body of experience in the use of AI and knowledge-based systems, such as expert systems in clinical settings [5], and we had built a range of experimental applications for clinical diagnosis [20, 17], treatment selection [17, 40] and management of cancer treatment protocols [39, 22]. By this time we were able to identify a set of general functions that appeared to be common across decision-making domains [14, 23] and process management applications [23, 9]. Several central challenges for building flexible agents for use in clinical applications were identified.

1. The creation of a *unified model* of the decision-making, planning and other tasks involved in a representative range of clinical processes, from patient monitoring and situation assessment to management and control of complex therapy protocols.
2. A *formal language* with which to specify clinical processes, facilitating interoperability of applications across healthcare sectors (primary care, hospital care, clinical research etc) and automatic verification of applications.
3. Design of *software development tools* based on such languages that would support rapid specification, verification, implementation, testing and dissemination of agent applications.

The first task was to develop a general model of clinical processes as a foundation for designing a formal process specification language. The “domino model” of clinical processes shown in figure 1 (a) was developed as a general summary of the cognitive processes that appeared to recur across a range of medical tasks. The left hand side of the domino deals primarily with decision making, while the right deals primarily with plan enactment.

The nodes of the domino can be viewed as *state variables*, while the arrows are *inference functions* that derive the data of the type at the head of the arrow from data of the type at the tail (and other state information such as general medical knowledge and

domain-independent knowledge). For example, suppose we have a patient who has inexplicably lost weight then the first step is to raise a clinical goal to diagnose the cause of the weight loss (arrow [1]). According to the model we then propose a set of possible causes (arrow [2]), and then assess the arguments for and against the competing explanations [3]. Once a detailed history has been taken we may commit to a specific diagnosis [4] such as cancer.

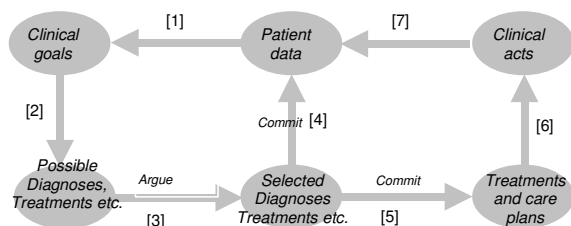


Figure 1 (a): the domino model of clinical processes

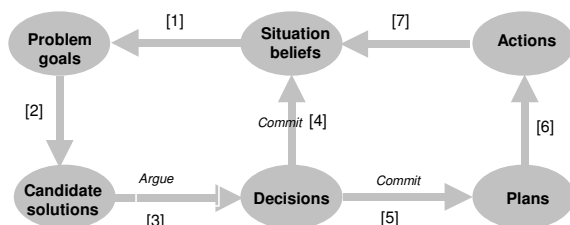


Figure 1 (b): generalized domino model

The diagnosis is a new item of patient data [4] and according to the model may lead to a new goal, to treat the cancer. Again several treatment options may be identified (arrow [2]), such as chemotherapy, surgery and radiotherapy, and once again there are likely to be arguments *pro* and *con* the options constructed by [3].

Deciding on the preferred option(s) in this case entails a commitment to an appropriate therapy plan [5]. The component steps of the plan will be scheduled [6], resulting in execution of actions. Action will often produce additional information about the patient's state, response to treatment and so on [7]. This new information may result in additional goals being raised (e.g. to manage unexpected side-effects or failures to achieve original goals) that will initiate further cycles of decision-making and planning.

Despite the simplicity of the model we have found that it captures and formalizes the expertise involved in a wide range of clinical tasks and domains, addressing requirement 1 above. Furthermore it can

be generalized as a domain-independent framework for modeling agents [7, 9]. In this general form (figure 1b) the domino model has been the focus of a variety of pieces of formal work. We first proposed a logical framework for decision-making under uncertainty based on an argumentation approach [14] and formalized the central components (arrows 3, 4 and 5) in terms of a *logic of argument* [19, 27] and a *modal logic of commitment* [8]. Later work built on these results in an interval based temporal logic for formalizing clinical scheduling [7]. The value of this formalisation effort was that it made it possible to design a formal representation language for general decision-making and plan management, which was called R<sup>2</sup>L [9]. This addresses the second requirement listed earlier.

In order to address requirement 3 we needed to develop software tools that could support the design, verification, testing and delivery of systems based on the language. An experimental interpreter for R<sup>2</sup>L called the RED agent language was developed [7] but in the present paper we focus on PROforma, a later derivative that supports a systematic software engineering method [18].

#### 4 AGENT SPECIFICATION IN PROforma

A PROforma process is specified by composing tasks into collections of prepared plans that are to be carried out under particular contingencies. Tasks can be enacted sequentially (explicitly scheduled), in parallel, or reactively in response to events, or a mixture of these modes. A simple example is ERA (Early Referrals Application) which is being used in the UK National Health Service to assist doctors in deciding whether patients with suspected cancer should be referred for urgent investigation. ERA can be found at [www.infermed.com/era](http://www.infermed.com/era).<sup>1</sup> A graphical representation of the ERA task structure is shown in figure 2. The PROforma specification for these tasks is shown in figure 3.

This ERA referral guideline of figure 2 consists of a single plan containing a set of component tasks: a decision and two alternative actions that follow the decision.

<sup>1</sup> To run one of the examples select a specific cancer by clicking on "referral form" for one of the cancers indicated. A patient data form will be displayed; once this has been completed click on the OK button to submit the data for processing by a PROforma server on the web site. ERA will return its recommendations on the basis of the data provided. Please note this is a demonstration and is not appropriate for practical clinical use.

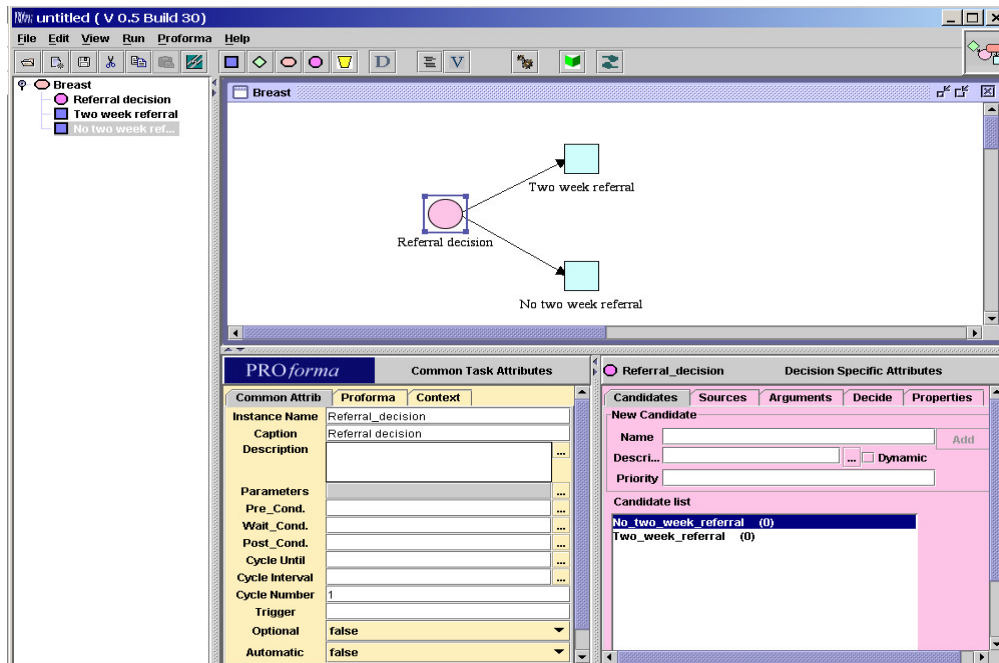


Figure 2: Task structure (a medical referral guideline) in the Tallis specification composer tool. The structure of the application is summarised in the left panel, showing that this simple application consists of a single plan containing a set of data acquisition, decision and action tasks. The sequencing of these tasks is visualised in the plan area, and the properties of each task are defined using the editing tools at the bottom. Tallis automatically generates the PROforma specification for the agent's tasks (see text), and includes a variety of verification and deployment tools, including a web browser interface for interacting with and controlling the agent.

ERA is based on "Referral Guidelines for Suspected Cancer" published by the UK Department of Health<sup>2</sup>. In this document the decision rules are set out in informal English as in the following conditions for "urgent referral for suspected breast cancer":

- Patients with a discrete lump in the appropriate age group (e.g. age > 30).
- Signs which are highly suggestive of cancer such as:
  - Ulceration
  - Skin nodule
  - Skin distortion
  - Nipple eczema
  - Recent nipple retraction or distortion

The behaviour of a PROforma agent is captured as a set of tasks that are modeled as object-attribute-value structures. The PROforma specification of figure 3 consists of a plan that has a set of component tasks whose enactment sequence is defined explicitly by scheduling constraints: an enquiry to collect patient data, a decision to determine urgency, and two alternative actions, one of which will be executed

depending on the result of the referral decision. The decision is made by evaluating a set of "arguments" for and against referring the patient, based on the criteria listed above. The "Netsupport" function in this example counts the number of arguments that evaluate "true".

This is a simple example but PROforma has proved to be a versatile and expressive language capable of implementing a wide range of processes in medical and other domains.

The generalized domino model was initially viewed as a logical framework for describing clinical processes. However, PROforma can also be seen as having object-oriented and agent-like features. We describe these different perspectives in more detail in the next section together with the insights they provide.

NB our objective is not to promote this particular approach to agent-oriented programming - as we shall see the analysis reveals weaknesses as well strengths of the PROforma approach - but to use the different paradigms to provide a better understanding of the model, and agent systems generally.

<sup>2</sup> <http://www.doh.gov.uk/cancer/referral.htm>

```

/** PROforma Guideline for Suspected breast cancer */
/** 25/10/2000, (simplified for presentation, 1/5/2001 **)
plan :: Breast ;
  caption :: 'Breast' ;
  component :: Clinical_information ;
  component :: Referral_decision ;
    schedule_constraint :: completed(Clinical_information) ;
  component :: No_two_week_referral ;
    schedule_constraint :: completed(Referral_decision) ;
  component :: Two_week_referral ;
    schedule_constraint :: completed(Referral_decision) ;
end plan .
decision :: Referral_decision ;
  caption :: 'Referral decision' ;
  choice_mode :: single ;
  support_mode :: symbolic ;
  candidate :: Two_week_referral ;
  argument :: for, ( tissue_changes includes 'Discrete lump' and age
>= 30 ) ;
  argument :: for, ( skin_changes includes Ulceration ) ;
  argument :: for, ( skin_changes includes Nodule ) ;
  argument :: for, ( skin_changes includes Distortion ) ;
  argument :: for, ( nipple_changes includes Eczema ) ;
  argument :: for, ( nipple_changes includes
'Retraction_or_distortion' ) ;
  recommendation :: Netsupport( Referral_decision,
Two_week_referral )>=1 ;
  candidate :: No_referral ;
  recommendation :: Netsupport( Referral_decision,
Two_week_referral ) < 1 ;
end decision .
action :: No_two_week_referral ;
  caption :: 'No two week referral' ;
  precondition :: result_of( Referral_decision ) = No_referral ;
  procedure :: 'No referral' ;
end action .
action :: Two_week_referral ;
  caption :: 'Two week referral' ;
  precondition :: result_of( Referral_decision ) = Two_week_referral ;
  procedure :: 'Referral, though not necessarily urgent' ;
end action .

```

Figure3: PROforma specification for the task structure of Figure 2.

## 5 THREE PERSPECTIVES ON THE DOMINO AGENT MODEL

### 5.1 The logic programming perspective

Traditional procedural programming requires that a programmer tell a computer what to do in order to get the desired output for a range of defined inputs using a suitable algorithm. In his classic book on logic programming Kowalski [25] argued the importance of distinguishing between determining *what* the results of a program are intended to be (the declarative reading) and *how* the result is obtained (the procedural

reading). The “Kowalski Doctrine” views all computer programs as being decomposable into two separate elements, expressed in the famous rubric

$$\textit{Algorithm} = \textit{Logic} + \textit{Control}$$

The AI programming language Prolog was the first attempt to capture this idea in a practical programming system. Syntactically Prolog is based on the Horn Clause subset of the first order predicate calculus. A Prolog program typically consists of a set of unconditional “facts” together with “rules” of the form *conclusion if premises*, where the conclusion is a positive literal and the premises are Boolean expressions (with the usual logical connectives *and*, *or*, *not*). For historical reasons conclusions are viewed as logical theorems and program execution is based on Robinson’s resolution theorem proving method (a depth-first, goal-directed search technique).

Prolog is a succinct, expressive and versatile way of writing general computer programs. A “pure” Prolog program can be read declaratively, as describing the logical relations that hold between the terms in the program, but it can also be read procedurally, because it can be used to control the execution of the theorem prover. AI languages in general, and logic programming languages like Prolog in particular, have a high-level of meta-level expressiveness (a feature which appears to have been lost from many modern procedural languages).

Prolog proved to be a particularly good language for AI programming due to its many built-in symbol-processing features (including search, pattern recognition and deduction) and the clear separation between knowledge of *what* and *how* which promoted reusability of knowledge. Another important feature of the language addresses a major concern for software developers who are building safety-critical systems such as medical systems; the language is constructed on well-understood mathematical foundations, which facilitates formal analysis of programs (e.g. to prove that process specifications are consistent with certain integrity constraints and are incapable of entering dangerous states). Nealon and Moreno [30] note the importance of legal issues and professional trust in taking up agent technologies in healthcare; we believe that in order to seriously address this it will be important to ensure that such technologies will have well understood formal foundations (see also [13]).

Our requirement for a medical agent language that can capture clinical processes first led us to a logic programming approach for implementing the domino

model. The first interpreters for both R<sup>2</sup>L and PROforma were both implemented in Prolog. In [9] we showed how the model can be formalised in the language R<sup>2</sup>L, which extends the standard syntax and semantics of Prolog to support agent-like capabilities, including decisions, plans and actions, temporal reasoning and constraints.

Like Prolog, R<sup>2</sup>L and PROforma preserve both a declarative and a procedural reading and R<sup>2</sup>L retains many of its advantages like meta-level expressiveness and formal soundness. However, the interpreters differed from the standard Prolog interpreter in their control structure for enacting tasks. Decisions, plans and other tasks are interpreted as parallel processes unless explicitly serialized by scheduling constraints. In addition processes can be invoked and terminated asynchronously by situation events and data.

From a formal perspective the extension of Prolog to include constructs like decisions and plans is acceptable if the semantics of the associated logics is provably sound and complete, which Das and others have demonstrated [15]. Furthermore, although Prolog adopts a backward chaining control structure this does not mandate this particular control regime for all logic programming systems; so long as the interpretation of the logical terms of the task specification does not violate axioms of logic we are free to apply more or less whatever control regime will be appropriate for the demands of the application. PROforma specifications can be interpreted under a number of control regimes. For example, as situations change tasks can be invoked, inferences made and databases (beliefs) updated accordingly. PROforma agents frequently operate more like a forward chaining rule-based system than a backward chaining theorem prover, yielding a strongly “reactive” mode of execution. Other reactive control mechanisms are discussed below.

From a logic-programming point of view existing PROforma interpreters are unsatisfactory in some respects however. Some first-order logic features have been implemented (e.g. logical conditions can include variables which will be bound at run-time) but a general ability to reason over logical classes is not available in existing interpreters. Furthermore, despite the potential for providing meta-level programming capabilities, which is fundamental to Prolog, and preserved in R<sup>2</sup>L, current PROforma development environments limit the logical capabilities available to application developers to a predefined set of predicates which include only a small set of meta-logical features. Although this simplifies

programming - and makes it easier to understand the meaning of a PROforma specification for non-technical developers such as clinicians - it limits flexibility for experienced programmers.

## 5.2 The object-oriented perspective

Despite the simplicity and uniformity of rule-based systems logical rules may not be the best level of description for complex processes. The common belief about early expert systems was that rules were “cumulative”, each rule representing an additional fragment of expertise. This is not always true. For example in a project to develop a system for the differential diagnosis of leukaemia the addition of a rule could produce a *reduction* in performance rather than the expected improvement, due to unforeseen interactions with existing rules [15]. We concluded that this was because particular rules are often only relevant in particular clinical contexts that could not be easily encoded in the rules, so they could “fire” inappropriately.

Elsewhere we explored the use of first-order rules for describing decision-making and process control in medical applications [14, 17, 23] with encouraging results. The meta-level features of logic languages can describe different kinds of decision-making, including statistical methods as well as symbolic approaches, but simple, recursive control structures like forward and backward chaining are inflexible when applications require both reactive and deliberative capabilities.

Results like this led us to the view that first-order logic is a powerful way of modeling reasoning and decision-making in clinical processes, but more complex agent processes need to be modeled in terms of the situations in which they are applied. These are frequently associated with high-level clinical goals, such as “decide the diagnosis of weight loss in middle-aged men”; “classify the risk-level for family history of breast cancer”; “control the blood pressure for a patient with proven hypertension”; “eradicate middle-ear infection in an infant” and so forth.

These observations suggested that a coarser-grained representation was needed to formalize complex tasks, such as decisions and plans. R<sup>2</sup>L was designed around these two classes of task. As we have seen tasks are represented using object-attribute-value triples, suggesting that R<sup>2</sup>L and PROforma might be viewed as object-oriented programming (OOP) languages. OOP was developed to support code reuse, so that a programmer does not in principle need to know what an object does, only that it has a particular

behaviour if you send it an appropriate message. The OOP approach speeds the development of new programs, and can improve the maintenance, reusability, and modifiability of software.

Using the style introduced by Kowalski for logic programming we can summarise the definition of an object with the following rubric:

$$\text{Object} = \text{Class properties} + \text{Behaviour}$$

Objects are instances of classes that define the properties and behaviour that are common to all members of the class (and a class can inherit these from its super-classes). R<sup>2</sup>L retains logical constructs like rules for formalizing inference, argumentation and scheduling of tasks, but in addition it introduces the idea of a decision class and a plan class. Each instance of a decision or a plan inherits part of the specification for its decision-making or enactment method from the class definition.

The PROforma language takes the R<sup>2</sup>L language forward by embedding the decision and plan model in a class structure that includes four classes of task (figure 4). The root class of this structure is the “keystone” task. Intuitively the keystone is any task that is designed to achieve a goal, such as a clinical treatment goal, but as yet lacks an associated method. The primary attributes of the keystone class include:

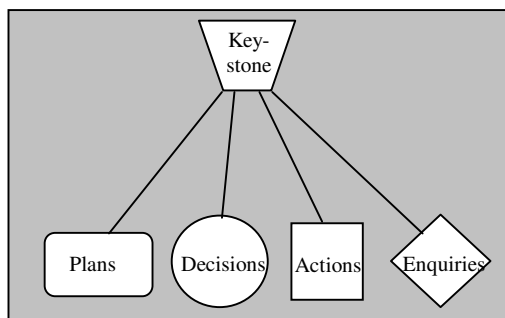


Figure 4: The PROforma task ontology

<i>Goal</i>	a term expressing the goal that the task is intended to achieve
<i>Trigger</i>	an event that can asynchronously cause a task to be considered for enactment
<i>Preconditions</i>	any conditions that must be true before the task may be enacted
<i>Post-conditions</i>	any conditions that will be true after the task is enacted (e.g. side effects)
<i>Autonomy</i>	Boolean property indicating whether the task can be enacted

without authorization by another agent.

All PROforma tasks inherit these attributes. Actions and enquiries provide the means by which an application interacts with its environment or communicates with other systems; decisions have attributes that allow application developers to specify decision options and rules and other procedures for making choices under uncertainty, and plans provide a way of packaging collections of tasks that serve a common goal (and may include sub-plans).

PROforma development tools can exploit the object (task) structure with CASE tools to assist the programmer in specifying task definitions. CASE tools can support visual programming by laying out clinical processes as task networks and providing specialized editors for defining logical conditions, data formats etc. In figure 2 the task attributes are shown in the editing panels below the task diagram. Here the “referral decision” decision has been highlighted, and the attributes for this task are made accessible in the editable windows at the bottom, keystone attributes on the left and decision-specific attributes on the right.

Since PROforma is a declarative language it is also possible to provide software tools for automatically checking and verifying the consistency, completeness and other logical properties of task definitions<sup>3</sup>. A range of object-oriented tools for authoring, testing and delivering applications is available for research use (see [www.openclinical.org/kpc](http://www.openclinical.org/kpc)).

The principal strength of the PROforma approach is in representing and enacting complex task structures. The object-oriented perspective also reveals limitations in the current tools however. For example, although there is a well-defined class structure no mechanism has been implemented for sub-classing over tasks in order to extend the built in task ontology. In medicine, for example, we might want to create reusable classes for diagnosis, drug prescribing, risk assessment or treatment-selection decisions, by creating them as sub-classes of the generic decisions, and pre-populating the attributes of the specialized classes with appropriate rules (e.g. for arguing the pros and cons of alternative drugs, or the possible causes of symptoms). Similarly, plans could be sub-classed into different types of process, such as research protocols, best-practice guidelines, care

<sup>3</sup> The Arezzo® suite, a PROforma toolset available from InferMed Ltd ([www.infermed.com](http://www.infermed.com)) incorporates a range of syntax-directed verification capabilities.

pathways and the like. There is also a need to support the creation and integration of other kinds of application ontology, such as drug databases and disease ontologies, for reuse across PROforma systems, though this requires further development.

Formal soundness has also been somewhat compromised by the translation of the R<sup>2</sup>L language into the more object-oriented structures of the PROforma development system. The introduction of the task model led to a much more usable application development environment, but some compromises have been made in that the language semantics are not as clear. For example, the concept of a task *trigger* provides an effective way of introducing a mechanism for asynchronous initiation of tasks but its formal semantics have not been defined (though there is a published operational semantics; [37]). Perhaps the introduction of such features is unavoidable in a practical engineering tool. Prolog was similarly compromised by the famous “cut” operator. This provided the programmer with practical control over the proof process in order to gain greater efficiency but at the risk of compromising completeness of the execution.

### 5.3 The agent-oriented programming perspective

The classic BDI agent model is summarized with a rubric that emphasizes the agent’s “mental states”:

$$\text{Mental state} = \text{Beliefs} + \text{Desires} + \text{Intentions}$$

Where “beliefs correspond to information that the agent has about the world ... desires represent states of affairs that the agent would, in an ideal world, wish to be brought about [and its] intentions represent desires that it has committed to achieving” [42]. The general idea of “mental states” is associated with the programming paradigm called agent-oriented-programming (AOP) by [36]. An AOP system is characterized by:

1. A formal language with a clear syntax (such as the predicate calculus) for describing the mental state of the agent in terms of concepts like beliefs, goals, etc.
2. A programming language in which to define agents. The semantics of this language should be closely related to those of the formal language.
3. A method by which the agent can communicate with other agents by means of messaging activities from a restricted class of “performatives”, such as *informing* and *requesting*.

In these terms R<sup>2</sup>L and PROforma are agent programming languages. Both languages represent

mental states like beliefs, goals, commitments (decisions) and plans, together with logical constructs for formalising inference rules, arguments and task constraints. Like logic programming languages they have a procedural as well as a declarative reading thereby satisfying the requirement for a programming language as well as a declarative format for specifying an agent’s behaviour.

An important feature of the benchmark agent’s behaviour is that it is “deliberative”, as it seeks to achieve its goals by constructing plans, choosing among competing actions, scheduling tasks to reflect particular timing or resource constraints, and so on. However, an agent may also need to operate in challenging environments in which events occur which are outside its control and whose occurrence may be unpredictable (e.g. a patient who is apparently stable may unexpectedly relapse). The agent will also therefore need to be “reactive”.

Experience in medical applications suggests that this broad distinction is too limited and designers require more differentiated control options. PROforma supports three features that naturally support deliberative processing:

<i>Scheduling constraints</i>	Impose sequencing and synchronization constraints on the steps (tasks) in a plan
<i>Temporal conditions</i>	To parameterise a task schedule in time
<i>Situation preconditions</i>	Must hold before a task can be enacted; task may wait until conditions are true.

In addition PROforma supports features that are naturally interpreted as reactive:

<i>Triggers</i>	Can asynchronously initiate a task, over-riding any existing scheduling constraints.
<i>Termination and abort conditions</i>	If the condition becomes true at any time the plan (or complete agent) will terminate or abort task process enactment.

The last control construct supported by PROforma is the goal, which has both deliberative and reactive characteristics. Goals are the “glue” that bind the parts of processes together, bridging an agent’s beliefs and intentions. An agent can use its goals deliberately to identify appropriate methods (e.g. by matching a current goal with a candidate task’s

Paradigm	PROforma features	Opportunities for improvement
Logic programming	<ul style="list-style-type: none"> <li>a. Logical inference</li> <li>b. Declarative and procedural reading of programs</li> </ul>	<ul style="list-style-type: none"> <li>a. First-order and meta-logical operations</li> <li>b. Clarify language semantics</li> <li>c. Exploit potential for formal verification of specifications</li> </ul>
Object-oriented programming	<ul style="list-style-type: none"> <li>c. Class structure with inheritance over tasks</li> <li>d. Object-oriented development tools</li> </ul>	<ul style="list-style-type: none"> <li>d. User-defined sub-classing on tasks</li> <li>e. Reusability of user-defined classes</li> </ul>
Agent-oriented programming	<ul style="list-style-type: none"> <li>e. Mental states</li> <li>f. Flexible control</li> <li>g. Agent programming language</li> <li>h. Simple performatives</li> </ul>	<ul style="list-style-type: none"> <li>f. Introduce a larger repertoire of cognitive functions (e.g. learning, perception)</li> <li>g. Extended range of communication performatives</li> </ul>

Table 1: Summary of features of PROforma and opportunities for strengthening them

postconditions). Goals can also be used reactively, in that if a task's goal has already been achieved the task can be omitted or terminated if it is in progress (whether or not the task itself brought the goal state about).

Turning to a critique from the AOP perspective PROforma provides limited support for inter-agent communication, though this is an important feature of the UAA benchmark. Currently, only simple messages can be implemented by means of action and enquiry tasks, e.g. allowing an agent to "tell" other agents things and "ask" them questions. This communication repertoire is weak as compared with, say, the KQML [10] and FIPA agent communication languages. For example, the FIPA set of performatives includes "propose", "request", "inform", "confirm", "agree", "cancel" and "not understood" among its message types<sup>4</sup>. However Nealon and Moreno [30] note the lack of a universally accepted communication language, and lack of methods for handling the interactions between software agents and humans, and with pre-existing healthcare systems.

There are ways in which we could enrich the communication capabilities of the simple actions/enquiries approach. From a logic programming perspective communication performatives can be viewed as meta-programs. The object level proposition that "Rosemary has breast cancer" becomes a message such as *inform(self, <recipient>, "Rosemary has breast cancer")* where *inform* is a meta-predicate, and the messaging process is implemented by a suitable meta-interpreter. Huang et al [23] demonstrated this technique showing how a wide range of performatives can be implemented in

the context of multidisciplinary shared care of breast cancer patients.

Work on robust speech-based communication between software agents and human users is also suggestive [1]. Recently there has been some consensus that at least three types of structures are required to support such communication [28]: an *intentional* structure (representing the intentions of a speaker in making certain utterances), an *informational* structure (which specifies relations between the propositional content of utterances), and an *attentional* state (which specifies which discourse objects, intentions etc are salient at a given point in a discourse). The PROforma task model is proving to provide a practical platform for modeling the intentional structure of a dialogue, with the informational structure being provided by a medical concept ontology (developed in collaboration with *Language and Computing N.V.*<sup>5</sup>; [4]).

In this work we employ a notion of dialogue which is based on concepts from conversational game theory [26]. For example, Carletta et al [3] have identified a set of twelve conversational moves that are sufficient for coding task-oriented dialogues in which an instruction-giver has to communicate a route on a map to an instruction-follower. Six moves may be used to initiate a conversational game: Instruct, Explain, Check, Align, Query-YN, Query-W; five are considered to be response moves: Acknowledge, Reply-Y, Reply-N, Reply-W, Clarify; and one simply prepares for a new game to be initiated: Ready.

In our own work in generating task-oriented dialogue we currently make use of eight of Carletta et al.'s moves. Four of the initiating moves: Instruct and Explain (which derive from PROforma Actions) and Query-YN and Query-W (which derive from PROforma Enquiries); and four of the response

<sup>4</sup> See <http://www.objs.com/agility/tech-reports/9807-comparing-ACLs.html> for a comparison of agent communication languages.

<sup>5</sup> [www.LandC.be](http://www.LandC.be)

moves: Acknowledge (which confirms an Action) and Reply-Y, Reply-N, and Reply-W (which satisfy Enquiries). In addition we use a move called Open, which is similar to Carletta et al.'s Ready move and is used to open a conversation (at the initiation of a PROforma process) and a move called Close which is used to close a conversation (at the termination of a PROforma process).

The current PROforma authoring tools provide some *ad hoc* support for specialising Actions and Enquiries along the lines described above, but a more principled way of extending the range of performatives from an object-oriented perspective would be to sub-class actions and enquiries into a task ontology for representing the different types of performatives required. This would greatly simplify the application development process and seems to be a promising line of future development.

#### 5.4 Discussion

It is apparently profitable to analyse technologies from multiple points of view. We have sought to demonstrate this using a case study of the PROforma language and development tools in terms of three established programming paradigms, *viz* logic programming, object-oriented programming and agent oriented programming. The analysis has been valuable, if only because it reveals opportunities for improving the PROforma technology (summarized in table 1).

The lessons are not all one way. PROforma was developed to address a set of problems in medicine that do not necessarily come up in basic research in AI, programming language design or software engineering. There are a number of areas where a multi-paradigm analysis could help us to understand the concept of an intelligent agent more deeply.

## 6 WHAT IS AN AGENT? (REPRISE)

As we have discussed the concept of "intelligent agent" has been used by different communities to mean different things. One leading school of thought seeks a compact, formal definition, emphasizing mental states and social interaction between agents (e.g. [36, 42]). A second approach takes a more eclectic (arguably *ad hoc*) view, seeing intelligent agents in terms of a set of practically important capabilities including a large repertoire of cognitive functions and flexible behavioural control.

Recognising that intelligence is a polysemous concept we have adopted the approach of setting out the features that are typical of (though not

individually critical to) the agent concept. In the light of the present analysis the benchmark list of agent characteristics introduced above is both liberal, in that it subsumes the principal capabilities identified by both schools, and strict in that each of the capabilities listed can be realized computationally.

The multi-paradigm analysis presented here has the added benefit that it might be used to provide a framework within which to seek a broader understanding of the UAA benchmark. Returning to the three rubrics introduced above:

*algorithm = logic + control*

*object = class properties + behaviour*

*mental state = beliefs + desires + intentions*

We propose to integrate these in a new benchmark model, UAA/2, consisting of a single overall rubric *viz*:

*Intelligent agent = Mental state + Behavioural repertoire + Control process*

Drawing on the above discussion each component of this rubric can be given its own definition.

The definition of a *mental state* begins with those states that have been studied in the BDI literature. Beliefs, desires and intentions are formalised primarily in the context of particular situations (e.g. the beliefs desires and intentions of a doctor dealing with a specific medical case). However, the agent's mental state is also taken to include its *general knowledge* about the domain in which it is operating (such as knowledge of medicine), hence:

*Mental state = Awareness + Knowledge*

Although "awareness" includes a representation of an agent's beliefs, goals and plans we have seen that in a practical implementation the current state of awareness will need to be considerably more differentiated. For example a PROforma agent routinely reasons about the preconditions and post-conditions of its tasks, whether or not the goal of a task is currently satisfied, whether any task is in progress or completed etc. *Knowledge* refers to general conceptual knowledge, such as ontological and "common-sense" knowledge, as well as domain-specific and technical knowledge. In medicine, for example, the latter includes knowledge of symptoms and diseases, anatomy, drugs and so forth.

An agent's mental state can only have utility if the agent has a repertoire of functions or capabilities with

which to respond to that state. We define an agent's *Behavioural repertoire* to be:

*Behavioural repertoire* =

*Task structure + Logics + Communications*

A *task structure* is the collection of cognitive and perceptual methods available to the agent. The *PROforma* task structure includes the basic classes of decisions, plans, actions and enquiries but a number of other medical process representation languages are currently in development which support different, though quite similar, task structures ([33] and see also [www.openclinical.org](http://www.openclinical.org) for overviews of a range of systems in development). Logical capabilities are typically embedded within these task structures (e.g. for decision-making and reasoning) and generally include standard deductive logic (for interpreting clinical data for example) as well as non-standard logics like argumentation and temporal logic. Finally there is the communication element of the behavioural repertoire which is fundamental to distributed and multi-agent systems. *PROforma* only provides support for the most basic communication performatives (*tell*, *ask* and *request authorization*) but extended communication languages, including complex dialogue systems, may be introduced within this general rubric.

The last element of the revised benchmark agent is the *Control process*; which is concerned with how tasks are enacted. As we have seen, a generally accepted feature of an agent system is that it can operate in deliberative and reactive modes, so we might define this with a simple rubric such as:

*Control process* = *Deliberative control* +

*Reactive control*

A collection of tasks can be enacted in a synchronized or coordinated manner, but individual tasks can also have the capacity to be triggered asynchronously, when the demands of unexpected events or situations need to override prepared plans. As with *Mental state* and *Behavioural repertoire*, however, this basic distinction may be too simple. *PROforma* and other technologies reviewed by Peleg et al [33] have shown benefits of more developed control structures, and we anticipate that agent research will benefit from research into more sophisticated control mechanisms.

Clearly many different agent technologies could be designed that fall within this benchmark scheme; that is in fact our intention. The benefit of having such a

general benchmark is that diverse approaches can be compared and contrasted within a common framework. Something like this has already taken place in the medical informatics community. The Intermed consortium's Guideline Interchange Format [31] has a different set of task primitives than those offered by *PROforma*, for example, and Huang et al [23] describe a richer set of communication performatives than *tell* and *ask* to meet the needs of interdisciplinary patient care. The Peleg et al study (op cit) reports a systematic comparison of a number of task-oriented modeling systems but without a benchmark model this tends to be limited to a comparison of features at the syntactic level. We believe that the revised benchmark model UAA-2 represents a consensus summary of features of current agent technologies, and provides a useful basis for understanding and comparing their cognitive and behavioural capabilities.

## 7 SUMMARY AND CONCLUSIONS

We have discussed the nature of an "intelligent agent" and proposed the UAA benchmark model to summarise the characteristics currently being discussed in the agent literature. The *PROforma* agent is viewed as an instance of this benchmark agent and has been used as a case study for critical analysis. The analysis itself has been carried out from three perspectives; logic programming, object-oriented programming and agent-oriented programming. The analysis appears has been productive, revealing weaknesses and strengths of *PROforma*. It also suggests that our first benchmark agent may be improved in the light of the insights gained from the different perspectives and we have proposed an improved agent benchmark UAA-2 as a general schema for analysis and comparison of agent systems in healthcare and further afield.

## References

- [1] M. Beveridge and D. Milward "Definition of the high-level task specification language" Deliverable D11, EU 5<sup>th</sup> Framework Programme, HOMEY project IST-2001-32434.
- [2] A. Caglayan and C. G. Harrison. *Agent Sourcebook : A Complete Guide to Desktop, Internet, and Intranet Agents*, John Wiley, 1997.
- [3] J. Carletta, A. Isard, S. Isard, J. Kowtko, G. Doherty-Sneddon. "*HCRC dialogue structure coding manual*", Technical Report HCRC/TR-82, Human Communication Research Centre, Edinburgh University. 1996
- [4] W. Ceusters, P. Martens, C. Dhaen, B. Terzic "LinkFactory: an advanced formal ontology management system" *Proc. Interactive tools for knowledge capture workshop*, KCAP 2001, Victoria BC, Canada, October 20<sup>th</sup> 2001.

- [5] E. Coiera *Guide to medical informatics, the internet and telemedicine*, London: Chapman and Hall, 1997.
- [6] E. Coiera, "Mediated Agent Interaction" in Quaglioni, Barahona and Andreassen (Eds). *Proceedings of the 8th Conference on Artificial Intelligence in Medicine Europe*, AIME 2001, Cascais, Portugal, 1-15. Springer Lecture Notes in Artificial Intelligence No. 2101, Berlin (2001).
- [7] S. K. Das "Managing tasks using an interval based temporal logic" *Journal of Applied Intelligence*, 6, 311-323, 1996.
- [8] S. Das and P. Hammond "A Logic for conditional recommendation" Technical report, William Penney Laboratory, Imperial College London, 1994.
- [9] S K Das, J Fox, D Elsdon and P Hammond "A flexible architecture for autonomous agents" *Journal of Theoretical and Experimental Artificial Intelligence*, 9 (4), 407-440, 1997.
- [10] T Finin, R Fritzon, D McKay, R McEntire "KQML as an Agent Communication Language" (1994) *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*
- [11] FIPA Foundation for Intelligent Agents, see web site at <http://www.fipa.org/>.
- [12] J. Fox "Decision theory and autonomous systems" Keynote lecture at *IMACS International Workshop on Decision Support Systems and Qualitative Reasoning*, M G Singh and L Trave-Massuyes (eds), Amsterdam: North Holland, 1991.
- [13] J Fox "Quality and safety of clinical decision support technologies: a discussion of the role of formal methods" in *Proc .Int. Conference on Integrated Design and Process Technology*, IDPT-2002, Pasadena, June 2002 and also at [www.openclinical.org](http://www.openclinical.org).
- [14] J. Fox, D. Clark, A. Glowinski and M O'Neil "Using predicate logic to integrate qualitative reasoning and classical decision theory" *IEEE Trans. on Systems, Man and Cybernetics*, 20 (2), 347-357 (1990b).
- [15] J. Fox and S.K. Das, *Safe and Sound: Artificial Intelligence in Hazardous Applications*, AAAI and MIT Press, 2000
- [16] J. Fox, S. K.Das, D.Elsdon "Decision making and planning by autonomous agents: theory, technology and applications" *Workshop on Decision Theory for Distributed AI Applications, European Conference on Artificial Intelligence*, Amsterdam, 1994.
- [17] J. Fox, C. Gordon, A. Glowinski and M O'Neil "Logic engineering for knowledge engineering: the Oxford System of Medicine" *Artificial Intelligence in Medicine*, 2 (6), 323-340 (1990a).
- [18] J. Fox, N. Johns, A. Rahmazadeh, R. Thomson "PROforma: a method and language for specifying clinical guidelines and protocols", *Proc. Medical Informatics Europe*, Amsterdam: IOS Press, 1996.
- [19] J. Fox, P. Krause and P. Ambler "Arguments, contradictions and practical reasoning" *Proc. European Conference on Artificial Intelligence*, Vienna, John Wiley: Chichester, 623-626, 1992.
- [20] J. Fox, C. D. Myers, M. Greaves, and S. Pegram "Knowledge acquisition for expert systems: experience using EMYCIN for leukaemia diagnosis" *Methods of Information in Medicine*, (24), 65-72 (1985).
- [21] J. Fox and R Thomson "Decision support and disease management: a logic engineering approach" *IEEE Transactions in Biomedicine*, 2 (4), 217-228, 1998.
- [22] P. Hammond, A L Harris, S I Das, J C Wyatt "Safety and Decision Support in Oncology" *Methods of Information in Medicine*, 33(4): 371-381, 1994.
- [23] J. Huang, N. R. Jennings, and J. Fox "An agent based approach to health care management" *Int. J. Applied Artificial Intelligence* 9 (4) 1995.
- [24] N. R. Jennings and M. J. Wooldridge: "Software Agents" *IEE Review*, January, 1996, 17-20.
- [25] R Kowalski *Logic for Problem solving*, Amsterdam: Elsevier, 1979
- [26] J. Kowtko, S. Isard, and G. M. Doherty. (1991) "Conversational Games within Dialogue", *Proceedings of the DANDI Workshop on Discourse Coherence (M. Caenepeel, J. L. Delin, L. Oversteegen, G. Redeker and J. Sanders, eds)*.
- [27] P. J. Krause, S J Ambler and J Fox "A logic of argumentation for uncertain reasoning" *Computational Intelligence*, 11.1,113-131, 1995.
- [28] J. Moore. (1995) "The Role of Plans in Discourse Generation", In *Discourse: Linguistic, Computational, and Philosophical Perspectives*, Daniel Everett and Sarah G. Thomason (Eds.).
- [29] G. Mugny and F Carugati *Social representation of intelligence* Oxford: Pergamon Press, 1989.
- [30] J. Nealon, and A Moreno "The application of Agent Technology to Healthcare", *1st International Joint Conference on Autonomous Agents and Multiagent Systems*, Bologna, July 2002.
- [31] L. Ohno-Machado, JH Gennari, SN Murphy, et al. The guideline interchange format: a model for representing guidelines *Journal of the American Medical Informatics Association* 5(4):357-372, 1998
- [32] S Parsons and J Fox "Argumentation and decision making: a position paper" in D Gabbay and H J Ohlbach (eds) *Practical Reasoning*, Berlin: Springer, 1996.
- [33] M. Peleg, S. Tu, J. Bury, P. Ciccicarese, J. Fox, R. A. Greenes, R. Hall, P. D. Johnson, N. Jones, A. Kumar, S. Miksch, S. Quaglioni, A. Seyfang, E. H. Shortliffe & M. Stefanelli. Comparing Computer-Interpretable Guideline Models: A Case-Study Approach. *Journal of the American Medical Informatics Association*, vol. 10, No. 1, Jan-Feb 2003, pp. 52-68 2002
- [34] A S Rao and M P Georgeff "BDI Agents: from theory to practice" *Proceedings of the first international conference on multi-agent systems (ICMAS-95)*, San Francisco, 1995
- [35] K. Richardson, *The making of intelligence*, London: Weidenfeld and Nicholson, 1999
- [36] Y. Shoham. "Agent-oriented programming" *Artificial Intelligence*, 60(1): 51--92, 1993.
- [37] D. Sutton and J. Fox "The syntax and semantics of the PROforma process modeling language", submitted for publication (2002).
- [38] P. Taylor, J. Fox and A.T. Pokropek The development and evaluation of CADMIUM: a prototype system to assist in the interpretation of mammograms, *Medical Image Analysis*, Volume 3, Issue 4, 1999, pp. 321-337.
- [39] R. T. Walton, C. Gierl, P. Yudkin, H. Mistry, M. P. Vessey and J. Fox Evaluation of computer support for prescribing (CAPSULE) using simulated cases *British Medical Journal* 315 791-5, 1997.
- [40] R. Taylor and J-L Renaud-Salis "The BOSS cancer protocol management system" described in Fox J (1990a above).
- [41] M. Wooldridge and N Jennings, Intelligent agents: theory and practice *The Knowledge Engineering Review*, 10(2), 115-152, 1995
- [42] M. Wooldridge, *Reasoning about rational agents*, Cambridge: MIT Press.