JACK®

An Agent Infrastructure for Providing the Decision-Making Capability Required for Autonomous Systems
Introduction

The growth of interest in autonomy as a capability, and the potential for autonomous systems, is now considerable. In the United Kingdom, the national ASTRAEA program aims to have commercial autonomous Unmanned Air Vehicles (UAVs) operating in UK airspace without restriction by 2012. The oil industry sees the introduction of autonomous oil production systems and autonomous unmanned underwater vehicles underpinning exploration and extraction of oil and gas from the more hostile and ecologically sensitive areas of the world, such as the Arctic. This demand has arisen in the last five years and has now crystalized in the commencement of major capability programs such as ASTRAEA. Also, the UK Civil Aviation Authority recently published its latest policy document on UAV operations, CAP722 [1]. This policy contains for the first time a definition of autonomy and highlights the requirements for such systems to operate with high reliability.

Autonomous systems cannot be assumed to rely upon human intelligence to any great degree. Consequently an autonomous system, e.g. an autonomous farm machine or air vehicle, must do everything for itself. It must be able to follow a plan route and communicate with its supervisor and other vehicles or humans it co-exists with. It must also detect, diagnose and recover from faults, and operate at least as safely as a system with continuous human involvement and control.

Autonomy is distinguished by the need for decisions to be made at any time, with an appreciation of the context of the current situation (often referred to as situation awareness). This is a critical point. If we wish to emulate human intelligence in some simple respects, the system must take into account circumstances that may affect the decision about to be made. Autonomous systems should make a rational evaluation of the choices available plus...
consider possible courses of action that could be taken, in light of this situation awareness. We expect such a rational system to then make ‘good’ decisions in terms of a human’s assessment of those available choices.

To perform in this way, an autonomous system still accepts sensor and user inputs as an automatic system would. But it operates with more abstract concepts, rather than reacting only to inputs in a fixed manner irrespective of the current situation. As with humans, an autonomous decision-making system should be able to act in a proactive or reactive manner when making its decisions.

We should also acknowledge the sophistication of contemporary feedback control systems that adapt their responses by taking into account current circumstances, i.e. they do not have one fixed response to a set of input conditions, but may have a number of responses, determined by tertiary factors. Examples of such controllers are aircraft gust alleviation systems, FADECs (Full Authority Electronic Digital Engine Controllers) on gas turbines, or GPS-guided automated steering systems on automated farming equipment. They are highly adaptive but are not constructed to make decisions at the level of abstraction achieved by autonomous systems.

The decisions made by an autonomous system are made on a rational basis. Additionally, to ensure consistent behavior that will encourage human trust the system’s decision-making should be repeatable. That is, the system should exhibit the same behavior each time it is exposed to identical circumstances and it should not produce large changes in behavior for small changes in inputs. An obvious exception to this is where the input to the system results in a “yes/no” decision, such as a point of no return (e.g. deciding to return to the departure airfield instead of continuing to the destination. Such a decision might depend on a very small variation in the amount of remaining fuel). Such behaviors can be evaluated using sensitivity analysis, relating system inputs to output.

The autonomy concept encompasses systems ranging in capability from those that can operate without human control or direct oversight (‘fully autonomous’), through ‘semi-autonomous’ systems that are subordinate to a certain level of human authority, to systems that simply provide timely advice and leave the human to make all the decisions and execute the appropriate actions (sometimes referred to as ‘intelligent assistants’).

It is envisaged that the most cost-effective combination will be one where the human and the autonomous system work together as a team, with the human as the ultimate authority. If command and control communications between the human and the system is lost the system has to be able to reason independently of the human and must revert to a pre-determined safe scope of operation.
This demand for autonomy and the need for rational decision-making to enable this capability leads to a search for software concepts, technologies and products that can provide this. Early experimentation by the large aerospace groups has shown that traditional approaches to programming the interaction between sub-systems has proven to be time consuming and error prone. Typically, autonomous systems are embedded in, and must interact with, a changing environment. The interaction with the environment is particularly problematic where there are many external entities that must be controlled, serviced or modeled. In the past, implementing such systems has entailed explicit programming of the interplay between each external entity. This makes the application difficult to implement, maintain and change if it embodies more than a simple client/server interaction. Furthermore, the application may not have the required flexibility or responsiveness to its environment.

Agent-based approaches have proven to be well suited to applications requiring complex interaction with an ever-changing environment. Arguably, the most significant attribute of agent-based systems is that each agent is an autonomous computational entity. Autonomous systems differ from automatic ones because they make rational decisions based on knowledge of the current situation. Whereas an automatic system comprises a repertoire of simple stimulus/response pairs, an autonomous system has a high-level appreciation of the situation and can evaluate the available courses of action in a context-sensitive manner.

A lift is a good example of an automatic system. It has a set of sensors including an infrared-beam/detector in the doorway. If the beam is interrupted, the doors will not close. The beam/sensor is there to ensure that the door does not close on a person or object in the doorway. Note that the system does not reason about the situation, it simply reacts to the interruption of the infrared beam. Given richer sensor data (e.g. cameras coupled with a face recognition capability), an autonomous system could react in a more context-sensitive manner. For example, if a person is continually blocking the door with a box so that he can load the lift up with goods, an autonomous system could recognize this situation and take appropriate action. If it is the weekend and no one else is using the lift, the autonomous system would take no action. However, if there are many people waiting to use the lift, the autonomous system could issue a verbal warning and then ignore the user’s floor selection, thereby nullifying his attempts to commandeering the lift.

Like an automatic system, the autonomous system still accepts sensor and user inputs, but operates with more abstract concepts instead of a knee-jerk response to inputs. As with humans, such an autonomous, decision-making system will balance proactive (goal-directed) and reactive (responsive)
aspects when making its decisions. Consequently, the behaviors exhibited by an autonomous system are far more sophisticated than those of a simple automatic system.

Autonomy, coupled with an ability to perceive the environment, act upon it and communicate with other agents, provides system builders with a very powerful form of encapsulation. A given agent can be defined in terms of its goals, knowledge and social capability, and then left to perform its function autonomously within the environment it was designed to function in. This is a very effective way of building distributed systems – each agent in the system is responsible for pursuing its own goals, reacting to events and communicating with other agents in the system. There is no need to explicitly program the interactions of the whole system, rather, the interactions emerge as a by-product of the individual goals and capabilities of the constituent agents.

This paper provides an overview of the JACK autonomous agent platform. The BDI (Beliefs/Desires/Intentions) paradigm underlies JACK and has been extensively studied [2], [3], [4].
JACK®

JACK is a mature implementation of the BDI paradigm, written as an extension of Java™. It has a sophisticated graphical development environment that supports the design, implementation and monitoring of BDI agents. Based on work by [5] on situated rational agents, the BDI approach has been applied to a wide range of problems, including fault diagnosis for Space Shuttle missions [6], and simulation of the tactical reasoning of combat pilots [7].

The BDI paradigm was developed to address a problem with existing Artificial Intelligence (AI) approaches to automated planning. Automated planning systems generate a sequence of actions that achieve the desired goal. Research on this problem was successful in developing general-purpose techniques (e.g. [8]). However, these approaches assumed infinite time and resources. They did not address the temporal pressures that apply when trying to achieve the goal within the context of a fluctuating environment that presents a multitude of interacting, conflicting and changing opportunities. Yet, this is the rule rather than the exception. Agents are typically situated in a dynamic environment and must constantly review their goals and activities, and need to be aware of the resource-bounded nature of their reasoning.

JACK is a compact and efficient BDI implementation that runs on any system on which Java is available, from personal organizers to high-end multi-CPU enterprise servers. JACK is extremely lightweight and is designed to handle thousands of agents running on relatively low-end hardware. It provides a simple communications model for communicating with other agents, back-end systems and GUIs. All these are layered on top of industry standard communications protocols. JACK extends the BDI paradigm to deal with inter-agent co-ordination, and includes new constructs and tools that support modern software engineering practices. JACK’s key programming constructs are outlined in Table 1.
JACK has been applied to a number of domains that require agents to take on functions that, because of their complexity, have only been handled by humans. These include decision support applications such as meteorological alerting. JACK is used in simulation environments to implement synthetic human entities, for example, military commanders and air combat pilots. Military models require a comprehensive knowledge acquisition phase, and this is greatly facilitated by JACK’s graphical plan representation (see Figure 1) and the JDE (JACK Development Environment). The JDE’s graphical tools support the design,

<table>
<thead>
<tr>
<th>Programming Construct</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>Events are the central motivating factor in agents. Without events, the agent would be in a state of torpor, unmotivated to think or act. Events can be generated in response to external stimuli or as a result of internal computation. The internal processing of an agent generates events that trigger further computation.</td>
</tr>
<tr>
<td>Plan</td>
<td>Plans are procedures that define how to respond to events. When an event is generated, JACK computes the set of plans that are applicable to the event and selects the plan that will form its next intention. Plans have a body that defines the steps to be executed in response to the event. The agent can try alternative plans to achieve the same goal.</td>
</tr>
<tr>
<td>Beliefset</td>
<td>Beliefsets are used to represent the agent’s declarative beliefs – what it knows about itself and the world.</td>
</tr>
<tr>
<td>Agent</td>
<td>Agents are autonomous computational entities with their own external identity and private internal state.</td>
</tr>
<tr>
<td>Team</td>
<td>Teams are used to encapsulate the co-ordinated aspects of (multiple) agent behaviour.</td>
</tr>
</tbody>
</table>

Table 1 – Key JACK constructs
implementation and tracing of agent applications. Agents, team structures, and their components are represented by icons connected by lines that show their relationship to one another. This diagrammatic representation uses natural language to describe the goals, contexts, reasoning steps, and actions of agents/teams. The graphical and natural language descriptions can then be fleshed-out by programmers to produce executable behavior models whose computational structure maps closely to the SME’s (Subject Matter Expert’s) specifications. This facilitates the process of knowledge encoding/editing and ensures that SMEs can follow the application’s runtime behavior and so determine if and how it should be adjusted. In decision support and human behavior modeling domains, JACK’s graphical toolset is vital in providing the SME with control over the iterative process of encoding/evaluating/modifying agent/team behavior.

Figure 1 – JACK’s Graphical Plan Representation
JACK Attributes

JACK agents can be created as fully autonomous entities that operate without external supervision and continually strive to achieve their designers’ goals. They can also be designed for decision-support, where they act as an assistant to the human.

When time is available, JACK agents pursue their assigned goals, but will switch to a rapid response mode when the situation demands immediate action. Like humans, JACK agents are resilient in the face of unexpected events and will consider multiple courses of action if time allows.

Deployed JACK applications are lightweight, efficient and cross-platform. JACK has been deployed on PDAs, for example a Hewlett-Packard iPAQ PDA onboard a Codarra “Avatar” Unmanned Aerial Vehicle (UAV). For this application, JACK intelligently manages the mission of the UAV. JACK agents have a very small memory footprint, allowing thousands to run on a single piece of hardware.

JACK supports rapid specification and deployment of agent-based applications. The graphical development environment simplifies the process of specifying and coding agents. The runtime infrastructure supports transparent and high-performance inter-agent communication; messages are routed to the appropriate agent(s) regardless of the host they are running on. JACK provides graphical tracing and monitoring tools to aid in the runtime appreciation of what the agents are doing and why they are doing it. Agent interactions are displayed graphically, as are the reasoning steps followed while the agent steps through its plan of action.
JACK Teams™

JACK Teams is an extension of the BDI paradigm that facilitates the modeling of social structures and co-ordinated behavior. JACK represents teams as self-contained reasoning entities (separate from its team members). The behavior of a team, and in particular the co-ordinated activity of the team members, is defined within the team entity. Thus, in the software model, each team exists as an entity with separate beliefs from those of its members. This generic team-based capability provides a flexible basis upon which a wide variety of teaming algorithms can be designed, developed, and tested. JACK supports the programming of team-oriented solutions, with appropriate constructs for expressing social relationships between team members, as well as for expressing co-ordinated activity. JACK also includes the communication facilities needed for executing co-ordinated activity in a team-based application [9].

JACK Teams implements a belief propagation step that handles dissemination of information up and down the team hierarchy. A team can have access to synthesized beliefs that are derived from the beliefs of its sub-teams. JACK supports the definition of filters that determine if and when the propagation should occur, and what subset of beliefs should be propagated to the containing team. Similarly, sub-teams can inherit a synthesized subset of the beliefs of the containing team. Belief propagation is triggered by changes to a team or team member’s beliefs.

To illustrate, the teamed approach enables multiple UAVs to work together with piloted aircraft (e.g. the JSF) as a single team. As an example, consider identifying and locating an enemy SAM site. The manned aircraft gives the UAV team the goal: “Locate, confirm, and mark the SAM radar.” The UAV team flies to the target area, but then the JSF team leader notices the camera/marker UAV maneuvering in an unexpected way, and requests the vehicle’s status. Because the UAV team can reason about its goals, it responds: “Radar Warning Receiver indicates activity; current intention is to evade and jam the emitter; goal is to identify target and transmit photograph”. This provides the JSF team leader with an immediate picture of the situation and is infinitely preferable to a reply that says nothing about the team’s goals: “Currently at 200 feet, in a rate 2 turn at 250 knots”.
JACK in Simulation

One of the major applications of JACK is in Synthetic Environments (SEs). SEs simulate the physical aspects of the world, such as terrain, weather, vehicles and humans. Models of human behavior in SEs tend to be simplistic. This is where JACK can make a major contribution.

Simulation is now an indispensable part of the military environment, and is used in many areas including training, tactics development, mission rehearsal, course of action analysis and hardware acquisition. In contrast to hardware such as aircraft, tanks and weapons systems, even highly-trained humans can vary significantly in their response to a given situation. JACK, with the addition of its cognitive architecture CoJACK™, enables the creation of principled models of human behavior, i.e. models that predict how humans behave and vary in a given situation, based on CoJACK’s theory of cognition and emotion.

Human behavior can be modified (moderated) by a range of factors, including temporal, environmental, physiological and internal factors. Moderators can influence human behavior directly – for example, caffeine typically provides a 10% faster reaction time on a simple reaction time task. By simulating the effects of moderators on underlying mechanisms, it is possible to predict behavior variation that will occur for a task that has not yet been studied closely. For example, if the effects of caffeine on the low-level aspects of cognition and the body are well understood, but the effects of caffeine on, say, radar operators had not been specifically studied, it would be possible to provide initial predictions of caffeine’s effects on the behavior of radar operators based on knowing their cognitive mechanisms and the knowledge necessary to perform the task. In this way, CoJACK provides rich and generalized predictions of how a simulated human will behave in a given situation.
JACK in Aerospace

JACK is the culmination of a lineage of experimental BDI agent systems that began with PRS some 20 years ago, then led to dMARS in the 1990s. JACK, dMARS and PRS share a common approach to BDI. This is because there is considerable overlap between the members of the teams that designed PRS, dMARS and JACK. Members of the JACK team worked on a real-time fault diagnosis system for the Space Shuttle as well as an air traffic flow management system that optimized runway utilization, responding immediately to unexpected changes in the situation aloft. Experience gained building these two systems led directly to the improvements we see in JACK today.

AOS has sold JACK to major defense organizations in the USA, UK, Canada and Australia. JACK has been successfully tested in flight, providing autonomous vehicle management of a BAC1-11 airliner in 2007 (owned by QinetiQ Ltd and operated under contract to the UK Ministry of Defence as a surrogate UAV test aircraft). The success of the BAC1-11 flight tests has allowed the next step in the UK industry/Ministry of Defence program for autonomous UAVs to proceed. The "Taranis" stealth UCAV (Unmanned Combat Air Vehicle) demonstrator will fly in 2010 and JACK is on board to provide the decision-making capability.

Future military fighter, strike and surveillance aircraft will need to continuously obtain updated external sensor data from the Global Information Grid (GIG) network. To meet this requirement, the reactive ISR Broker™ is being developed as a prototype by AOS for the Joint Strike Fighter, in conjunction with the Australian Defence Department’s New Air Combat Capability (NACC) Office. The core of the ISR Broker is an autonomous decision support capability provided by AOS’s JACK product.
References


