Rational Physical Agent Reasoning Beyond Logic

Sandor M Veres and Nick K Lincoln
University of Southampton, Highfield, SO17 1BJ, UK, Email. s.m.veres@soton.ac.uk

Abstract

The paper addresses the problem of defining a theoretical physical agent framework that satisfies practical requirements of programmability by non-programmer engineers and at the same time permitting fast realtime operation of agents on digital computer networks. The objective of the new framework is to enable the satisfaction of performance requirements on autonomous vehicles and robots in space exploration, deep underwater exploration, defense reconnaissance, automated manufacturing and household automation.

1. Introduction

Agent theory has a substantial history within the field of computer science and has been theorized over greatly, stemming from the widely held view that an agent is a system most appropriately described by the intentional stance, wherein the agent is an entity subject to anthropomorphism [1]. Whilst, to date, there is no panacea for agent theory, significant contributions have been made relating to what properties an agent should have and how these properties should be formally represented and reasoned about [1,2,3,4,5]. Application areas for agent systems have been presented for the realms of software, industry and autonomous control systems [1,5,6].

Agent architectures are divisible into three groups: reactive, deliberative and hybrid. Reactive architectures operate through a mapping of data to action, in a stimulus-response fashion, as exhibited by Brook's subsumption architecture. Subsumption architectures present intelligence as an emergent property resultant from a hierarchy of finite state machines such that there is a behavioral response resultant from goal oriented desires; there is no form of planning. Such simplistic architectures are advantageous in dynamic environments, though in addition to the difficulty of engineering a set of behaviors to achieve a particular task, undesired behaviors may also evolve. Moreover, omitting the ability for such agents to reason eliminates the possibilities of planning and learning: both highly desirable proactive traits for agents [7,8].

Deliberative architectures can be separated into symbolic and intentional systems; these latter architectures are based upon abstracted notions written within the intentional stance. Intentional systems, as philosophized over by Dennett, encompass system descriptions abstracted out using human notions such as belief and intent [9]. Such abstractions allow irrelevant details to be omitted and hence save in computational complexity, permitting subsequent reasoning to be based upon high level notions and not modeling physics as would be resultant from modeling within the physical stance. Both deliberative theories are strongly bound to logical systems and it is these engines which are used for agent reasoning. The earliest cited use of such logical formalism within agent reasoning is that of the possible world semantics, as presented by Hintikka, now commonly formulated in a Kripke structure using normal modal logic [1]. Unlike reactive systems, deliberative architectures do not imply any form of output, merely theorizing over system knowledge. By definition an agent is proactive, thus requiring some form of output: early work involving the integration of action and knowledge is from Moore, who presents a formalism wherein knowledge is used as a pre-condition for action. This is in agreement with Hintikka, who postulated that not only is knowledge is the goal of inquiry, but what

'decisions and actions are based upon'. This work was closely followed by a theory of intention by Cohen and Levesque [10].

Symbolic architectures are logical systems, wherein the environment is represented symbolically and manipulated through reasoning. It is common practice now that the symbolic representations are logical formulae and it is these which are manipulated: this syntactic manipulation is theorem proving. Such manipulation permits an agent not only to reason about the consequence of current percepts, but states of the world which it may bring about. Although advantageous in both mathematical rigor and concept, since human knowledge can be considered as symbolic, it is difficult to translate the world effectively into symbolic representation and the computation times required for formal proofs generally precludes implementation on real time systems, even if using a concurrent theorem proving mechanism [10] .

Intentional systems are rooted in philosophy and based upon logical reasoning of the intentional stances used to represent the agent. Epistemic and doxastic logic, the logic of knowledge and belief respectively, are two forms of logic which fit within the anthropomorphist nature of an agent. A logical framework wherein beliefs, desires and intentions are primitive attitudes, as formulated by Rao and Georgeff, has proven to be the most popular: the three BDI primitives are highly cited within literature and used within numerous agent programming languages including the logic based PRS, Jason and 3APL, as well as the Java based implementations of JADE, Jadex and JACK [2,3,4].

Systems aiming to combine the timely nature of reactive architectures and the mathematical rigor of deliberative architectures are termed as hybrid, or layered architectures. Layered systems, as their name would suggest, involve a hierarchy of interacting subsystem layers; these layers may be horizontal or vertical. Complexities of information bottlenecks and the need for internal mediation within horizontal architectures are partially alleviated within vertical architectures, though these structures do not provide for fault tolerance. Both horizontal and vertical architectures have been implemented within TouringMachines and InterRRaP respectively [12].

Possibly the most widely known BDI implementation is PRS [13, 14]. PRS is a situated real time reasoning system that has been applied to the handling of space shuttle malfunctions, threat assessment and the control of an autonomous robot. Within the PRS framework a knowledge database, detailing how to achieve particular goals or react to certain situations, is interfaced by an interpreter. Conceptually this is similar to a horizontally layered TouringMachine, though notions if intention are contained and consequently placing PRS as a deliberative BDI model. Whilst certainly a notable and proven method, the knowledge database is fixed and invoked upon perceived condition triggers for the particular knowledge item of relevance. The set of possible agent behaviors is restricted by the number of knowledge items programmed and held within the library: an agent cannot add knowledge items during runtime, self generated or otherwise.

Here we propose interleaving of physical stances within the intentional stances used to abstract and reason about the system in order to enrich the reasoning process. Some might question the validity of such an approach as beneficial to aiding means–end reasoning, since conceptually we are providing an elaborate plan mechanism, which could be instigated within the PRS architecture. However, PRS interprets percepts and in conjunction with goals, will select a particular pre-formed plan to run: we are proposing the possibility for physical stances to act internal to deliberation as well as being the result of deliberation; not instigated as a result of deliberation until a preferential alternative is selected.

2. An agent framework for unstructured physical environments

A physical agent system (PAS) is a 3-tuple consisting of the agent, the environment in which the agent exists and a coupling between the agent and environment.

Definition A *physical agent system,* α, is defined by

$$\alpha = \langle A|E|C\rangle$$

where A is a set of agents, $A = \{A^i \big| i \in S\}$, E is a set of environment objects, $E = \{E^i \big| i \in S\}$, and C is a set of couplings between $\Gamma \subseteq \{\gamma\langle a, b\rangle | a, b \in A \cup E\}$ and any members of the joint set $A \cup E$. Couplings between members of A are called communications, $\boldsymbol{C}$, and couplings between members of E are called physical interactions, $\boldsymbol{P}$. Couplings between members of A and E are called action and sensing, Æ, in the environment.

A coupling is an abstract object at this stage of our discussion that will be specified later for its types, attribute and models (see Section 4). In this paper we are interested in physical agents that are not pure software agents but ones that have a set of environmental objects associated with them and are called the "body" of the agents. Strictly speaking a body is a set of environmental objects associated with a set of agents.

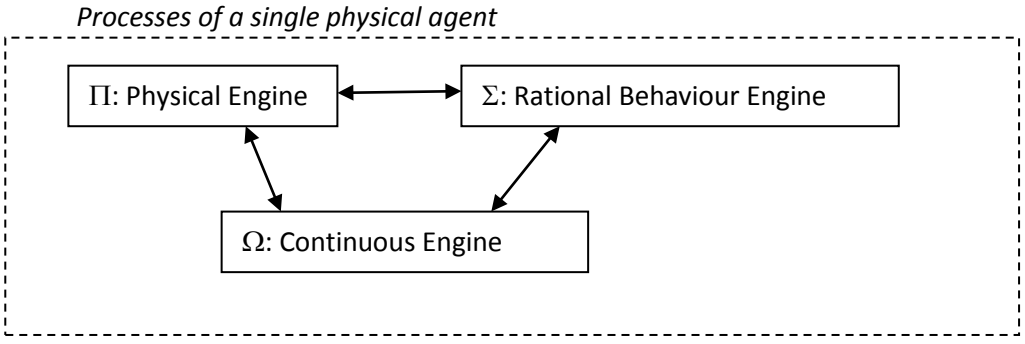
Definition The *body association* of agents is a map, *β,* defined by splitting A and E into a disjoint set of sets of agents and sets of environmental objects, respectively:

$A = \bigcup A_k$, $E = \bigcup E_k$,

and allocating to each A_k a subset of E : $A_k \overset{\beta}{\rightarrow} E_k$ where $E_k \subseteq E$.

In exceptional circumstances the $E = \bigcup E_k$ may be allowed not to be disjoint, meaning that agent teams can "share some body parts", but this is not the norm. For logical clarity one should organize teams of agents, such that $E = \bigcup E_k$ is a disjoint union.

Definition. A single *physical agent* is a tuple $A = \langle \Pi|\Sigma|\Omega\rangle$ consisting of the three main constituents of any agent: its physical engine, its rational behavior engine and its continuous engine.



This formal representation of a physical agent has been created to serve three practical purposes in engineering autonomous systems that interact with the physical environment:

1. *Easy programming* by engineers through defining sensing and actuation for the physical engine, definition of prediction in the continuous world (using the continuous engine) and by defining goals and behaviour constraints for the rational behavior engine.

2. *Formal verifiability* of the complete hybrid system of the agent system and its environment class, based on assumptions made on probabilities of physical engine malfunctions, numerical precision of the continuous engine and a hybrid system model-class of the environment.
3. *Fast realtime operation* on digital computers with asynchronous processes running in parallel. Any physical agent must have at least one process for each of Π, Σ, Ω . Realtime functionality may well require that some or all of Π, Σ, Ω are split into several, or many, sub processes.

If any of these practical requirements are not met then the uptake of the agent system by industry can be seriously limited.

The physical engine can be a complex process or set of processes. It can contain a multitude of devices, including communication, sensor and actuator devices, as well as sensor data abstractors (SDAs), control data developers (CDDs) and feedback processes (FBPs) for realtime processing of perception and actions of the agent. The SDAs are to symbolize events for Σ, the CDDs are to develop symbolic action into control signals for the actuators and the FBPs are needed for linking CDD and SDA pairs into realtime feedback loop executors.

Definition. A *physical engine* Π is a quad tuple, $\Pi = \langle \gamma | \sigma | \alpha | \Delta \rangle$, where γ is a set of communication devices, σ is a set of sensor data abstractors, α is a set of control data developers and Δ is a set of realtime feedback loop executors. For practical reasons any of σ, α, Δ may be empty, but this is not true for γ, which represents communication devices and consequently may never be empty.

Definition. A *software agent* is a physical agent that has only communication devices present within its physical engine, i.e. all of σ, α, Δ are empty.

The rational behavior engine, Σ, is formally defined so as to permit the accommodation of simple reactive subsumption based architectures as well as deliberative belief-desire-intention, i.e. BDI agent architectures. Whatever agent architecture is implemented, they must handle sensed event abstractions, agent action abstractions and decision making processes (DMPs).

From a theoretical point of view, it is the DMPs where agent architectures mostly differ in complexity. Given a set of $\gamma, \sigma, \alpha, \Delta$ abstractions, one can define simple behavior rules for a reactive subsumption architecture, or one can define a deliberative agent decision making process. The latter can use a belief set that can influence goal list prioritization and commitments to intentions that are executed, while monitoring beliefs and changing action if necessary. Of course a BDI architecture may involve more complex $\gamma, \sigma, \alpha, \Delta$ abstractions, however the overall architecture described above remains valid. As the next section will show, humans formulate their behavior rules in abstract temporal logic statements, so to satisfy the requirement of "ease of programming", it would be inappropriate to ignore temporal logic as a fundamental component of the DMP. As described later, and as one of the main contributions of this paper, there will be another equally important component of any capable advanced agent, that is via Ω, the continuous engine. The Ω moves beyond logic and facilitates the handling of unknown unstructured environments by the agent.

A question remains relating to the complexity that the DMP going to take on. It is inevitable that the DMP will be implemented using the abstractions of $\gamma, \sigma, \alpha, \Delta$ plus some memory and suitable data structures. These data structures may possibly be different resolution maps of the world in which the agent resides that may be uncertain in their fine detail. Similar to the way we (as humans) use maps to learn the abstract layout of streets or connections of metro lines, robots may only have a rough and highly abstract guide to the world. In the case of the metro line maps even the rough geometry of where the lines go in the city is neglected and the map is only useful to know how the stations are connected. These examples from human context point to the importance of a robot needing maps with different resolutions to make goal achievement and planning fast. In an unknown environment, for which there are no detailed maps available, the robot must implement SLAM (self localization and mapping) techniques.

Apart from maps there may also be a need for a *memory for abstract skills* (SAM) execution. Although the Physical Engine has $\gamma, \sigma, \alpha, \Delta$ for the actual execution of physical contact with the world, physical skills are to be controlled at an abstract level in feedback/feedforward. Speed of execution in some skills may require sending out a sequence of abstract commands in α for execution, without concurrent feedback but rather details relating to the end result of the abstract command sequence. For humans this happens in driving a car where we learn what action to take in a particular situation, such a requirement to swerve or perform an emergency stop, and feedback can only be registered with some delay; speed of feedback is limited. Abstracted physical skills command sequences, and their feedback mechanisms, are learnt and improved upon by actually practicing the activity, a testament to the phrase "practice makes perfect". The same should be true for robots as it is too laborious for programmers to develop physical skills of robots and the DMP system. Additionally, self-learning skills also infers learning how skills fit into the DMP system: integrity of the DMP operation is preserved. This integrity may become fragmented if programmers develop code for physical skills of agents and this highlights the importance of *learning skills execution (LSE) and skills abstract memory (SAM)* at an abstract level using discrete symbols $\gamma, \sigma, \alpha, \Delta$.

Most physical agents developed so far do not have a DMP with LSE and SAM. Existing robots tend to link abstractions directly with a limited set of physical actions. Using LSE and SAM the set of performable actions can be several magnitude greater than the set of abstractions for sensing and actions in $\gamma, \sigma, \alpha, \Delta$, which may be called primitives of sensing and actions from which more complex ones can be built. Good choice of primitives in $\gamma, \sigma, \alpha, \Delta$ will fundamentally influence the sophistication of physical abilities that the robot will be able to learn via practice.

Other components of DMP can be:

- Planning of movements in the physical world
- High level planning of goal achievement using various resolutions of world maps (self-learned or acquired)
- Ultimate goals of operation and behaviour constraints at higher levels of abstraction

The following definition grasps the minimum of what the DMP must contain for a capable physical agent, without specifying the actual mechanisms of operation, i.e. no "agent architecture" is specified.

Definition. A rational behavior engine Σ is a tuple, $\Sigma = \langle W | M_p | M_g | C | G \rangle$, that contains a granulated multi-resolution and physical multi-domain world models *W,* abstract physical skills memory, M_p, goal achievement memory (problem solving memory), M_g , abstract formulation of behavior constraints, *C*, abstract formulation of short and long term goals, *G*.

Although no specific agent architecture is required, we will briefly outline how some of the best known architectures fit into this. In all cases the $\gamma, \sigma, \alpha, \Delta$ are assumed to be common.

Subsumption architectures:

The Σ of a physical agent with "subsumption architecture" can for instance be filled in by the following:

W, M_p, M_g, G – Empty sets.

C – A set of "if A then B" rules where the premises A are propositional logic formulae in terms of abstraction symbols in γ or in σ. The conclusions are propositional logic formulae of symbols in .

BDI architectures in Jason or similar agent oriented languages:

W – Modeling structures in various physical domains and varying resolution

M_p, M_g – Plans and logic rules programmed in the agent programming language

G – Goal symbols that appear at the head of some plans

METATEM deliberative agents using temporal logic :

W – Modeling structures in various physical domains and varying resolution

M_p, M_g – Databases of symbol vectors in terms of $\gamma, \sigma, \alpha, \Delta$ and temporal propositional logic connections

G – Temporal logic statements in terms of higher level abstractions as explained by temporal logic formulas using abstraction from: $\gamma, \sigma, \alpha, \Delta$.

The latter realization is the most natural for humans in terms of goal definitions and executions of tasks in a logical manner. The question of modeling structures relating to the continuous world, and their role in planning and decision making, has so far been neglected. It is a fact however, that decisions can be influenced by hypothetical planning and not decision to be made first and then planning put into motion. This latter can lead to the agent making bad decisions. This fact and that initial abstractions in σ, as prescribed by the designer of the agent, may not be sufficient to capture the essence of the changing world correctly, leads us to the next section defining the "Continuous engine" of the physical agent.

Definition. The continuous engine, Ω, is a tuple $\Omega = \langle M|S|O|B|L \rangle$ where *M* is a set of approximate continuous models of the world, S is a continuous time simulator that uses analytical and empirical data based dynamic models to predict future state of the world, *O* is an optimizer that can optimize continuous time planning of actions, *B* is a Boolean evaluator of propositions in terms of σ statements and *L* is a library of useful numerical computations in terms of continuous variables.

As W is the primary, symbolic model of the world with relationships stored on symbols, the M is related to W. The M is a collection of models from various physical domains (geometric, mechanical, electrical, gravitational, heat, fluid flow, etc…) that make symbolic descriptions in W either more precise in numerical or in qualitative, time evolution sense. Even without precise numerics, the agent may perform a simulation using the simulation tools in S to see possible qualitative outcomes in terms of σ, α, Δ abstractions. These qualitative outcomes of predictions can be used to make the right decisions and planning by Σ.

The O is a set of optimization tools of continuous problems. The agent can perform planning in continuous time and can set up a problem formulation. Then it searches in O for a suitable optimization tool. The results are formulated and used at the symbolic level of σ, α, Δ. Optimization for path planning, robust feedback control or a low complexity numerical dynamical model may all be tools that are available in O.

B is important since Boolean values of primitives in σ in prediction (simulation) outcomes must be inferred by some continuous computations that cannot be performed elsewhere except in the continuous engine Ω. B contains a Boolean valued functional, by example "will the agent body be within the allowed boundary if it carries on moving the same way for the next 10s" is an evaluation that is not done in symbolic computation but using M, S and B: the statement is converted into a symbolic temporal logic statement for Σ.

Finally L is a library of auxiliary computations with continuous quantities: for instance equation solvers for linear systems of equations, nonlinear equation solvers and generic nonlinear optimizers. Use of L by the agent assumes that the agent is capable of setting up problems in Σ by first abstracting them and picking a solution tool from L.

It is evident that Σ and Ω run hand-in-hand, i.e. the Σ frequently delegates computations to Ω, and then uses the output from Ω to continue deliberation of prescribe action.

3. Programming of Physical Agents by Non-programmers

A human being is represented here as a "super agent" that has considerably more in its DMP then the formal agents to be engineered. The extra features of the human DMP can be briefly summarized as follows:

- Conceptual structures formed from $\gamma, \sigma, \alpha, \Delta$ abstractions.
- Seamless manipulation of conceptual structures to achieve lowest complexity models of the environment for goal achievement.
- Rich set of skills to cope with in various situations in a physical environment. This includes manual skills as well as skills of analytical modelling performed in the head or on a computer.
- Near "completeness" of abstract knowledge in an artificially created physical environment that permits reliable operation of human agents by design of the environment.

The latter is an interesting point: our educational system complements the human built environment to be safe for humans by providing skills to cope and introducing laws and unwritten conventions to render the human – physical world interactions relatively safe for mature adults.

Unfortunately, physical agents that we discuss in this paper do not enjoy the kind of advantages as listed. Instead the problem is to find a mapping of these human capabilities into formal agent properties that make them safe and achieving goals despite the very limited resources they have relative to humans. To do that we will formalize the human capabilities a bit further.

4. An Example Software Implementation Using J2M

There are many ways to implement the agent architecture described above. Ideally C, C++ could be used for all the components to permit operation on all platforms (Windows, Linux, Unix, MacOS and embedded systems) but there are both practical and legacy issues that make realization biased towards software systems that have ready tools available and which would be far too expensive to reproduce. For instance the MATLAB family of toolboxes provides a rich set of numerical processing, optimization and simulation algorithms that can be exploited. Similarly the Java environment has numerous software components in the area of agent reasoning that may be implemented directly. The following table provides some possible options for the software platform to be used for the implementation of the agent architecture outlined in Sections 2-4.

Π platform	**Σ platform**	**Ω platform**
C	C	C
C++	C++	C++
Java	Java	Java
MATLAB	Java	MATLAB
SIMPOL	SIMPOL	SIMPOL

In the following we describe a possible software implementation in terms of MATLAB+Java+MATLAB (J2M) that has the advantage of minimal programming effort due to the rich set of programming tools presently available. Within the J2M implementation, the Π and Ω platforms are developed within MATLAB, whilst the Σ

platform is exclusively developed using Java. Although it is theoretically possible to form a rational engine within a MATLAB framework, since MATLAB does not allow for multi-threaded code execution, this would be a restricting factor. Also, to do so would neglect the existing frameworks which have been developed in more suitable languages. It should be noted that the high level prototyping in J2M may be converted into concise machine code for embedded applications. A description of the J2M will follow, within which we are assuming the implementation of a physical agent, that is one for which σ, α, Δ are not empty.

The physical engine, Π, is an element within an environment; this environment may be purely numerical, instantiated within a virtual world or a true dynamic real world environment. Regardless of construct, the physical engine is capable of extracting relevant abstract data from the inhabited world to be encapsulated within σ and communicated to the rational engine, Σ, via γ. This is the only interaction mechanism Π has with Σ.

Within the developed implementation, the physical engine continually passes sensory information to the rational engine, which is used to update W, though the rational engine may chose to ignore information it receives.

Sensor data, σ, may relate to position and velocity information given in some coordinate frame, as would be required for roving vehicles such as UAVs or AUVs. σ may also relate to other functional data types such as temperature and pressure information, extracted from relevant sensors. The way in which σ is communicated to Σ is critical, since Σ must be in the position to update W correctly based upon information received via γ. FIPA provides a comprehensive treatment of software standards and specifications for interacting agents and agent based systems, to enable intelligible communication between agents and agent processes. For the instance of sensor data flow from the physical engine to the rational engine, communication is unidirectional and so the communication considerations are reduced to the rational agent requiring knowledge only of the communicating entity (in this case Π) and the subject matter (σ). W is updated as a consequence of σ, from which Σ will evaluate the appropriate actions and this may instantiate invocations of Ω.

During the rational agent deliberation cycle, a point may be reached wherein calls to the continuous engine are required. Instances of these calls have already been entered upon; here we shall concentrate on the data flow. The continuous engine is constructed within MATLAB and consequently all functionality of this system is achieved through use of the appropriate m-files. Σ requests execution of a component from Ω, either S, O, B or L, dependent upon the solution sought by the deliberation cycle occurring within Σ, and also requests a response in the form of a pointer, character array, numerical or Boolean value.

Communication between Σ and Ω is more complex than that of the data pushing performed by Π, since actions executed by Ω are conditional upon input from Σ, and in turn Σ expects some form of result. Consequently we are presented with the need for more elaborate communications protocols to ensure efficient interaction between Σ and Ω. Messaging from Σ is of the form $<m_c,c_c,r_c>$, whereby m_c is a character string specifying the particular m-file to be executed, based upon the data encapsulated within c_c and expecting the number of returned evaluations to be the number specified within r_c. Since Σ is aware of the continuous engine invocation instance, the data type being returned by Ω need not be specified. Return dialog from Ω to Σ is in a similar format, of the form $<m_r,r_r>$, wherein m_r signifies the m-file which was executed and r_r the result of the routine. This format could be interpreted within the FIPA framework as <in-reply-to, data>.

MATLAB is unable to function in a multi-threaded nature and so Ω is restricted to parallel processes only; this is not true for Σ, which is intrinsically multi-threaded: whilst waiting for a response from Ω, Σ is able to continue other forms of deliberation which are unrelated to problems associated with the current Ω task. An example of such an instance is that of vehicle path optimization: Σ requests execution of O from Ω based upon current σ and M. Whilst O is executing within Ω, Σ is still capable of monitoring σ from Π and dealing with any instances which may arise, such as the need for control reconfiguration. Upon completion of O within Ω, the

relevant response (in this instance a pointer to the file detailing the optimized path) is sent to Σ, from which the invoking thread may be resumed. Ultimately all threads within Σ result in some form of action to be completed by Π and this requires some form of assertion by Σ; note that the prescription of 'no-action' represents the simplest assertion which may be prescribed by Σ. In the same way that Π acts to 'push' data to Σ, Σ prescribes action for Π: here Σ prescribes the initiation of Δ which acts directly upon α. Returning to the example of path following, Σ initiates Δ within Π relating to the task of path following using the plan formed by Ω and accessed by the file pointer provided to Σ by Ω, which is consequently available to Δ. Δ executes without additional action from Σ, using internal feedback devices, though Σ is capable of over-riding the execution of a Δ.

All engines run concurrently: Π is continuously feeding data to Σ, which prescribes action to Π via γ and invokes intermittent communication with Ω. Whilst Ω and Π do not communicate directly, shared memory between these engines allows access of the results from Ω by Π: such an instance is invoked when Ω is tasked with path optimization. Here Ω is invoked by Σ to optimize a path, or thruster sequence, and this plan is stored within a file. A pointer to this file passed to Σ, which communicates this pointer to Π. If instructed to follow the plan, Π reads the file directly and acts accordingly. A schematic of the J2M construct, indicating interaction of the three agent engines, is given within Figure 1.

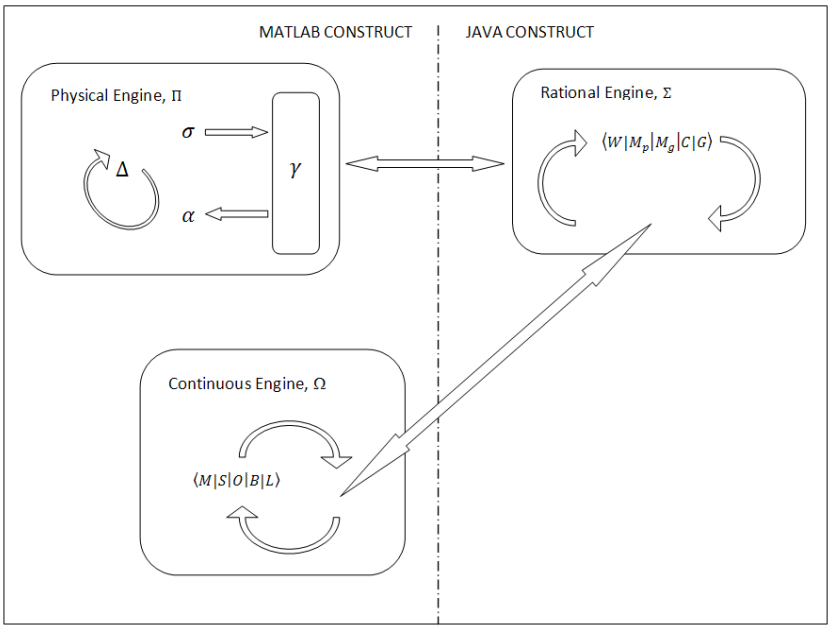


Figure 1: J2M Framework Construct

An example implementation of the presented agent framework has been developed based upon a spacecraft agent scenario. Within the scenario a spacecraft agent is tasked with acquiring and tracking a geostationary orbital location upon a failed orbital insertion. During the simulation the agent is also presented with instances of actuator failure, requiring control reconfiguration in order to achieve the mission objective.

The agent world is developed within Simulink (MATLAB), and is inclusive of disturbances resulting from Earth oblateness (triaxiality), solar radiation pressure and Luni-Solar third-body perturbations [15]. The agent is not aware of these impacting factors: within the continuous engine, Ω, the relevant physics engine for internal simulation which may be utilized are based upon the Hill equations, a simplification of the orbital dynamics with respect to the desired orbital location [16]. The spacecraft agent is availed with numerous possible control methodologies, afforded by the considerable amount of literature available on the subject of

spacecraft control, inclusive of planning, adaptive and reactive control systems [17-23]. It is the task of the reasoning engine to select an appropriate control methodology to deal with the presented scenario, perform the appropriate control and analyse the resultant output in order to improve performance. Concurrent to the control requirements, the spacecraft agent is tasked with monitoring internal systems and reacting accordingly to any diagnostics made. Within simulation, the actuators present the agent with situations of gain reduction and propellant supply issues resulting from supply valves being stuck open/closed and a fuel-line breach.

The upper level Simulink block is shown within Figure 2, with the VRML sink linking to a virtual world shown in Figure 3. Whilst not essential, it was chosen to include a VRML world to aid visualization of the dynamical processes occurring as a result of agent action or inaction. The world is initialized with the agent in an undesired location and with a state representative of a failed orbit insertion. A sphere indicates the desired geostationary location bounds: when within these bounds the sphere surface is green; when in violation of the bounds the surface is red. Note that for visualization purposes, scales have been altered greatly.

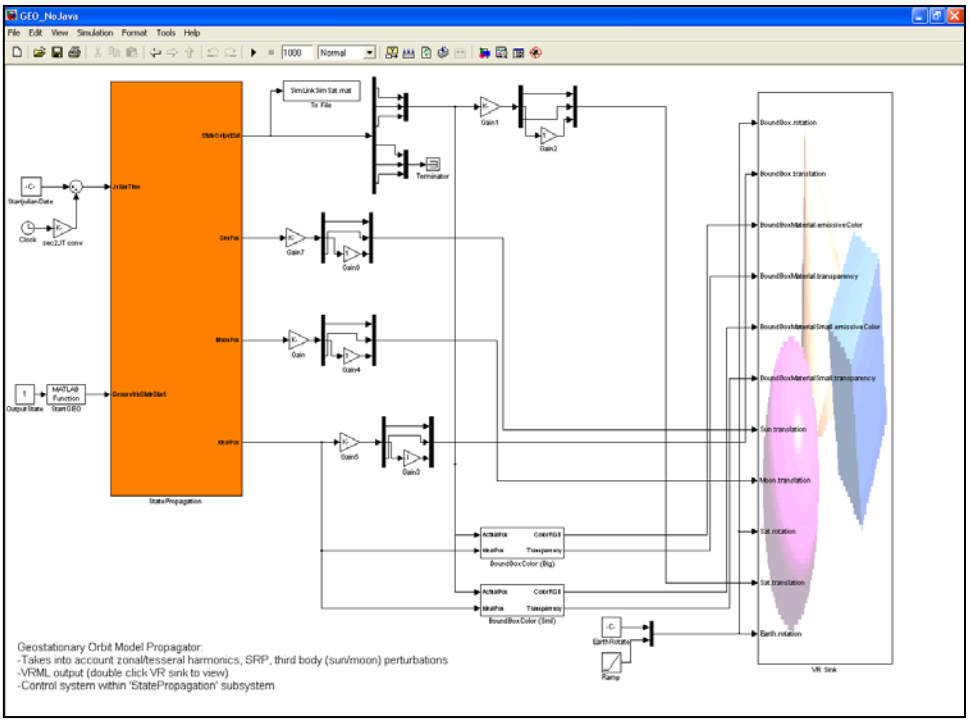


Figure 2: Agent Simulink Environment

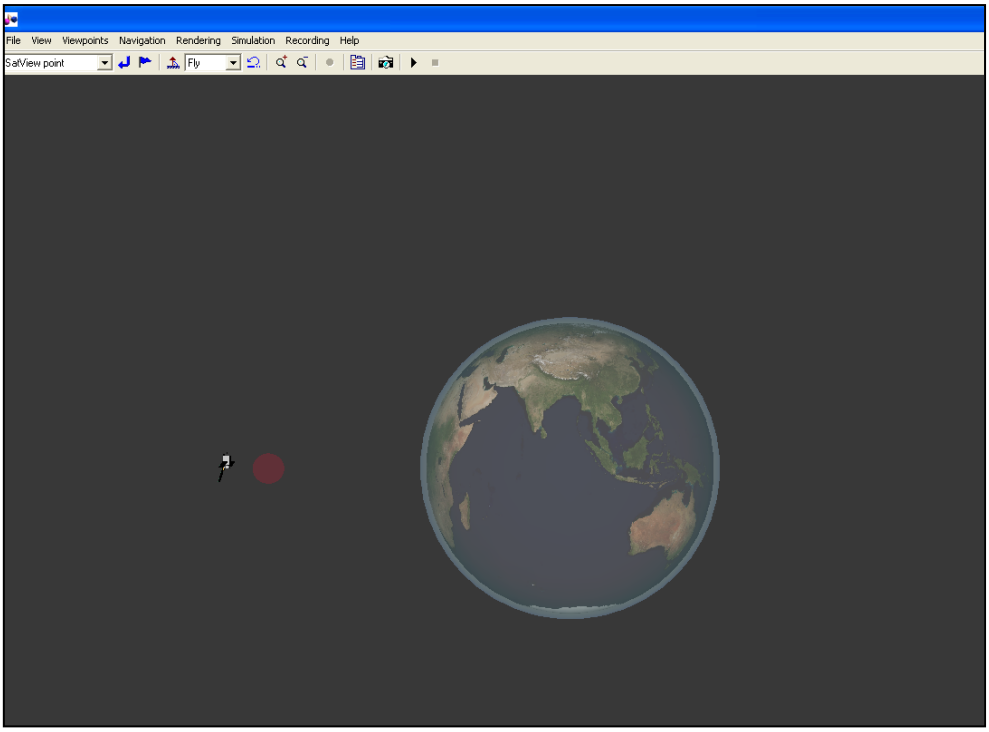


Figure 3: Agent VRML World Initialized State

Agent percepts are taken from the simulated world, from which abstractions are made and passed to the rational engine developed within Java. As presented within the proposed agent framework, the rational engine may task the continuous engine and these results may be used within deliberation to result in a rationalized prescription of action: agent action is currently limited to thruster actuation and internal reconfiguration to adapt for faulty thrusters.

Using the presented framework within the simulation, successful geostationary orbit attainment and regulation has been achieved in the presence of unknown disturbances and actuator failure, necessitating online control reconfiguration.

Summary

This paper has discussed and presented a theoretical physical agent framework that satisfies the practical requirements of programmability by non-programmer engineers and realtime operation of deployed agents. The presented framework has been developed and implemented within simulation for autonomous spacecraft control tasked with position tracking, subject to orbit insertion error, unmodeled orbital perturbations and control reconfiguration requirements under the presence of actuator performance degradation and failure. Further development of this model, and the extension of the existing framework to additional agent scenarios, is the subject of continued research.

References

[1] Michael Wooldridge and Nicholas R. Jennings, *Intelligent Agents: Theory and Practice*. Knowledge Engineering Review Journal, 1995, volume 10, pp 115-152.

[2] Anand S. Rao and Micheal P. Georgeff, *Formal Models and Decision Procedures for Multi-Agent Systems*. Technical note 61: Australian Artificial Intelligence Institute, June 1995.

[3] Rafael H. Bordini, Jomi F. Hubner and Micheal Wooldridge, *Programming Multi-Agent Systems in AgentSpeak Using Jason*. Published by John Wiley and Sons, 2007

[4] Fabio Bellifemine, Giovanni Caire and Domonic Greenwood, *Developing Multi-Agent Systems With Jade*. Published by John Wiley and Sons, 2007.

[5] H. Van Dyke Parunak, *Practical And Industrial Applications of Agent-Based Systems,* 1998.

[6] Sandor M. Veres, *Autonomous Control Systems Using Agents: An Introduction*. In, Proceedings of IEE Workshop on Agent Based Control Systems. IET (IEE), 1-10 (2005).

[7] Sandor M. Veres and Aron G Veres, *Learning and Adaptation in Physical Agents.* In, *9th IFAC Workshop "Adaptation and Learning in Control and Signal Processing" (ALCOSP'07), St Petersburg, Russia, 29-31 Aug 2007.* , 6pp.

[8] Sandor M. Veres and Aron G. Veres, *Learning and Adaptation of Skills in Autonomous Physical Agents.* In, *17th World Congress of International Federation of Automatic Control (IFAC), Seoul, Korea, 6-10 Jul 2008*. Seoul, Korea, Elsevier - Pergamon Press, 6pp, 2671-2676.

[9] Daniel C. Dennett, *The Intentional Stance*. Published by MIT Press, 1989.

[10] Cohen, P. R. and Levesque, H. J. (1990). *Intention is choice with commitment*. *Artificial Intelligence*, 42:213-26

[11] Michael Fisher, *An Open Approach to Concurrent Theorem-Proving*. Technical report, Department of Computing, Manchester Metropolitan University.

[12] Michael Wooldridge, *An Introduction to MultiAgent Systems*. Published by John Wiley and Sons, 2002.

[13] Georgeff, M. P. and Lansky, A. L. (1987). Reactive Reasoning and Planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 677-682, Seattle, WA.

[14] K. L. Myers, *Procedural Reasoning System User's Guide: A Manual for Version 2.0*. Technical report, Artificial Intelligence Center, SRI International, Menlo Park, CA, 2001.

[15] David A. Vallado and Wagne D. McClain, *Fundamentals of Astrodynamics and Applications (Space Technology Library)*. Published by Springer Link, 2001.

[16] Marcel J. Sidi, *Spacecraft Dynamics and Control*, Published by Cambridge University Press,1997.

[17] S. M. Veres, S. B Gabriel, D. Q Mayne and E. Rogers, *Analysis of Formation Flying Control of a Pair of Nano-satellites*. AIAA Journal of Guidance, Control, and Dynamics, 25 (5). pp. 971-974.

[18] R. Pongvthithum, S. M. Veres, S. B. Gabriel and E. Rogers, *Universal Adaptive Control of Satellite Formation Flying*, International Journal of Control, Volume 78(1), January 2005

[19] D. Ya. Rokityanski and S.M. Veres, Application of Ellipsoidal Estimation to Satellite Control. *Mathematical and Computer Modelling of Dynamical Systems*, 11, (2), pages 239-249 (2005).

[20] N.K. Lincoln and S.M. Veres. *Components of a Vision Assisted Constrained Autonomous Satellite Formation Flying Control System*. International Journal of Adaptive Control and Signal Processing, 21(2-3):237–264, October 2006.

[21] S.M. Veres, (2006) Autonomous formation flying of satellite robots: the mechanical control layer. In Proceedings, *TAROS 2006: Towards Autonomous Robotic Systems, Guildford, UK, 4-6 Sep 2006*. .

[22] S.M. Veres, Thanapalan, K., Gabriel, S. and Rogers, E. (2006) Reconfigurable Controller Design For Operational Safety in Satellite Formation Flying. In Proceedings, *International Control Conference 2006, Glasgow, Scotland, UK, 30 Aug - 1 Sept, 2006*.

[23] Sandor M. Veres and Nicholas K. Lincoln, Sliding Mode control for Agents and Humans. In Proceedings, *TAROS'08, Towards Autonomous Robotic Systems 2008, Edinburgh, UK, 1-3 Sept 2008*. Edinburgh, UK, IC London, 13pp, 61-73.