Journal of Integrated Design and Process Science 26 (2022) 369–391
DOI 10.3233/JID-220009
IOS Press

Perspective

The convergence of computational thinking, computational intelligence, and multi-agency

Duncan Anthony Coulter*
Academy of Computer Science and Software Engineering, University of Johannesburg, Johannesburg, South Africa

Abstract. The design of real-world industrial systems is subject to a natural tendency towards modularization in order to manage complexity. In addition, this article considers that patterns of self-similarity in many problem domains have made many such solutions naturally representable as holarchies. Likewise, the increasing need for autonomous local decision making as well as the demand to produce solutions at scale has increased the relevance of the multi-agent paradigm to the creation of modern software systems. A variety of software development patterns are explored for their compatibility with holonic multi-agency. The current skill sets required by software development workers and concomitant training activities focus on instilling computational thinking abilities, a set of related cognitive competencies useful in the development of such systems. Intelligent systems play an increasingly important role in modern development and often benefit from computational intelligence techniques for the purpose of parameter tuning. This position paper explores the intersections between holonic multi-agency, modern information systems development, the computational intelligence which train them and the computational thinking skills those developers should be trained in.

Keywords: Rational agency, multi-agent systems, holonicity, computational intelligence, computational thinking

1. Introduction

This position paper considers the inherent tendency of complex systems to approximate multi-agent systems over time and the consequential need to augment the conventionally understood set of computational thinking competencies to include reasoning about multi-agent designs. In addition, the growing use of computational intelligence-based approaches to machine learning and data science related problem domains, coupled with the recent growth in those domains, necessitates a reasoned consideration of the intersection between those undoubtably complex systems. Many such problems have a nested recurrence relationship embedded into their structure and while recursive reasoning is an already well recognized computational thinking skill its intersection with multi-agent-based modelling results in the concept of holonic multi-agency a powerful tool for the creation of robust designs for complex systems.

*Corresponding author: Duncan Anthony Coulter, Academy of Computer Science and Software Engineering, University of Johannesburg, Johannesburg, South Africa. Tel.: +27 11 559-2842; E-mail: dcoulter@uj.ac.za.

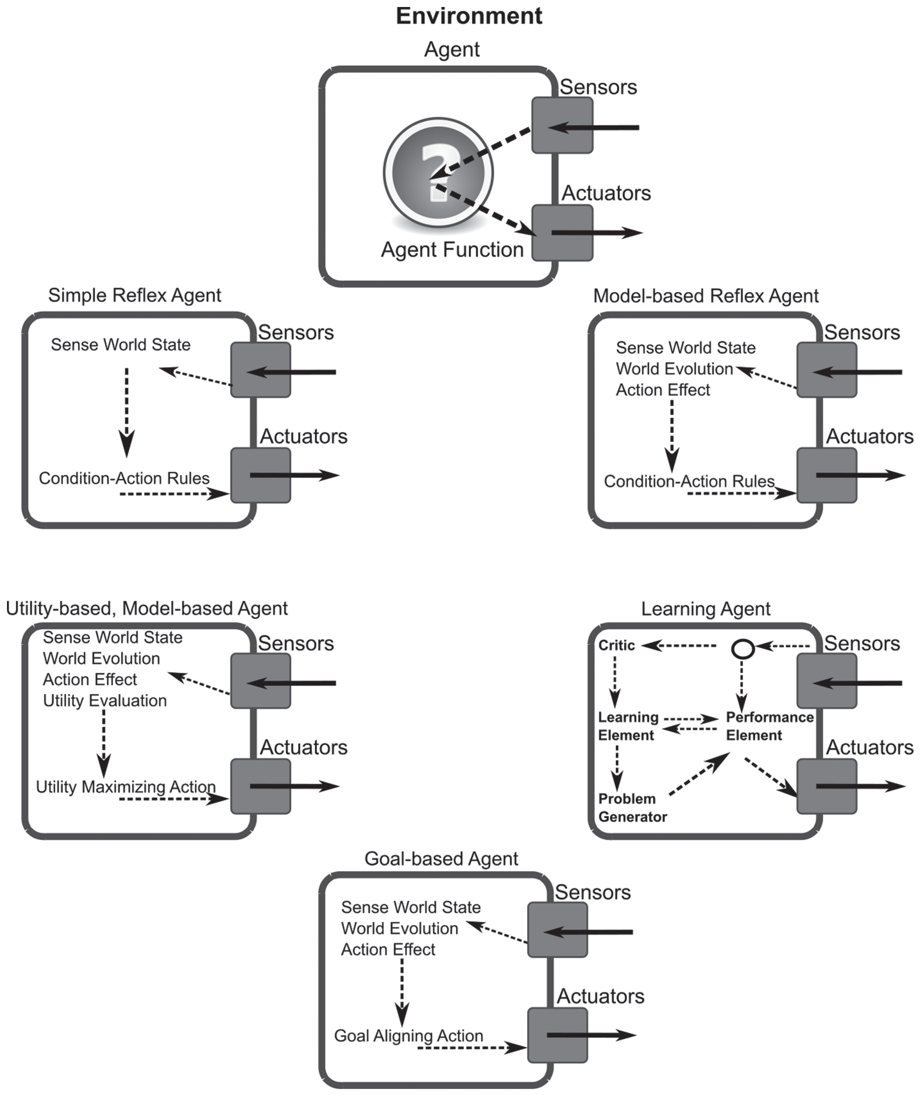


Fig. 1. Agent architectures (Coulter, 2014) / (Russell & Norvig, 2020).

This paper argues that multi-agent systems are a natural end point (or at least a strong attractor) within the space of distributed systems development patterns. By this it is meant that complex system design approached acquire the attributes of multi-agency over time. The multi-agent paradigm, a design philosophy in which systems are composed of interacting sets of loosely coupled autonomous entities, is first explained then cases argued for the aspects of multi-agency present within several classes of real-world systems. This key concept is illustrated in Fig. 1. The independence of the agents increases the modularisation of complex system development while the autonomy of each agent reduces the vulnerabilities brought on my central command and control architectures. This is then continued to consider computational intelligence systems which have become increasingly important over the last few years. The emergence of multi-agency across a variety of related subdomains is discussed in Section 3 while the key requirement of autonomy is increasingly implanted by way of the computational intelligence techniques described in Section 5. Emergence in this context refers to the convergence of design principles to this common end state across a variety of sub-disciplines.

While the development of such systems will require increasing use of computational intelligence techniques the developers of such systems will increasingly need to exhibit computational thinking skills themselves.

2. Rational multi-agency

2.1. Bounded rational agency

Intelligent systems do not have infinite temporal or computational resources to devote to decision making. Bounded rational agency deals with the decision making activities of autonomous entities under such constraints.

The development of intelligent systems has had a troubled history moving through a very pronounced series of booms and busts as the limits of their theoretical underpinnings, computational hardware, and the availability of sufficient quantities of machine-tractable data were encountered placing development at odds with the expectations and aspirations of both practitioners and the public at large. As a result, long periods of so-called AI-winters of reduced funding and development followed periods of overoptimistic hype. Consequently, the techniques for developing such systems have spanned a considerable portion of the development of the discipline of software development itself and as such a need for a unified approach to framing the development of such systems was needed.

The divergent approaches to the development of such systems were not only manifest at an architectural level but were also present in terms of the fundamental approaches used to represent both knowledge and reasoning processes ranging from purely symbolic reasoning, logical inference, to todays dominant data-driven approaches. As such a way to abstract over these differences was needed in order to create an effective container for intelligent processes. The concept of rational agency (Russell & Norvig, 2020) has emerged a useful tool for the containerization of intelligence.

The lack of a formal definition of intelligence has necessitated the adoption of more pragmatic working definitions for the functional and non-functional requirements and other externally observable properties of intelligent systems. These definitions are often very problem specific (for example maximising accuracy, recall, and precision in classification systems or finding an acceptable set of local minima / maxima in a constraint satisfaction problem). The lack of precise definition for intelligence, while troubling on an existential level, is not unique to artificial intelligence research and is reflected in the study of human psychology where the debate regarding the nature of multiple, task-specific, intelligences versus a generalized intelligence (known as *g*) is ongoing (Visser, Ashton, & Vernon, 2006).

As such research tends to focus on developing systems which exhibit rationality in their decision-making processes rather than true intelligence. Rationality being taken to mean making the best decision (or at least an acceptable one) given the information available to at the time. This allows for task-specific quality measurements to be defined for each problem domain.

As implied these systems are often modelled as decision making processes where the intelligence of the system is captured in a so-called agent function which maps a sequence of observable precepts together with the state of the environment and any knowledge the agent has embedded within it on an a priori basis onto the selection of an action from a finite set of possibilities. Agent oriented systems are considered to be embedded within an environment as opposed to being non-localized inference engines common in the design of older expert-systems. This environmental embedding is a natural extension of the need for modularity and complexity management as discussed later.

The exact mechanism for mapping a percept sequence onto action selection varies in complexity from simple reflexive mappings up to and including complex goal-oriented behaviours incorporating

Table 1
Agent architectures in a computational intelligence context (Coulter, 2014) / (Russell & Norvig, 2020)

Architecture	Cognitive Components	Description
Simple Reflex	Condition Action Rules	Direct deterministic mapping between the percept sequence and selected action. The sohpistication of implementation is not an issue in that the condition action rules need not take the form of an explicit sequence of branching code. A feedforward neural network which does not undergo anyfurther training may also consitute a simple relfex agent. Consider that reflexive responses to stimula in humans are implemented neurologically.
Model-based Reflex	Model of World Evolution Condition Action Rules Model of Action Effect	Such agents consider the fact that the environment within which they are embedded may not be static over time and that actions have consequent impacts on the state of the environment. The model of the enironment is not adaptive however. A pre-trained feedforward neural network may capture both aspects of the architecture in terms of world evolution and condition action mapping.
Utility-based, Model-based	Model of World Evolution Condition Action Rules Model of Action Effect Utility Function Utility Optimization	By augmenting the world evolution functionality with a metric that ennumerates the desirability of a given world state the agent is able to select from a set of optimal (or pareto optimal) actions i.e. those that lead to a more desirable world-state. While finding a locally optimal or globally optimal world-state could be considered as a goal the lack of explicit representations of such end states and intermediate states diffeerentiates these agents from true goal-based agents
Goal-based Agents	Model of World Evolution Condition Action Rules Model of Action Effect Goal-aligning Action	Goal-based agents are often implicitly an extention of utility-agents which have been augmented with the ability to label certain possibile world states as desiable end-states as well as to identify the required intermediate states (sub-goals) needed to reach those end-states along with the sequence of action selections required to reach those states (planning).
Learning Agents	Critic Learning Element Problem Generator Performance Element	Learning agents are made up of a set of highly coupled sub-components which each play a role in increasing the adaptability of the agents action selection behaviour in the face of dynamic or initially unknown environments. The performance element encapsulates the agent function but takes input on the efficacy of its selections from the critic according which are adjusted according to the learning element. Balance between exploration and exploitation is managed by the problem generator.

an understanding of the environment within which they are embedded and its evolution over time and in response to the selected actions themselves. Table 1 summarised the main agent architectures while the accompanying Fig. 1 shows more details regarding those structures.

The exact nature of agent environments also varies considerably depending on the problem domain which has profound impact on the design decisions involved in agent architecture selection, percept

Table 2
Agent environments extended from both (Coulter, 2014) and (Russell & Norvig, 2020)

Environment Type	Symbol	Representation	Description
Observability	O	$\{O \in \mathbb{R} \vert 0 < x \leq 1.0\}$	This is a measure of the degree to which each percept includes the totatilty of the environment. Traditionally defined as only partially or fully observable in terms of whether some information is hidden from the agent. It makes more sense to denote this using a continous value indicating possible values from unobservable to fully observable. Rational behaviour is not possible in a completely unobservable environment
Agent Multiplicity	M	$\{M \in \mathbb{R} \vert 1 \leq x < \infty\}$	This is a measure of the number of agents within the environment. The interval is closed on the left hand side with a value of 1 (conventional single-agent system) and open and unbound to the right. Practically there can only be a finite number of agents.
Predictability	P	$\{P \in \mathbb{R} \vert 0 < x \leq 1.0\}$	This is a measure of the degree to which transitions from one state to another in the world model can be followed in a deterministic manner. A value of 1 indicates a truly deterministic environment with lower values indicating increasing amounts of uncertainty in the outcomes of actions.
Dynamism	D	$\mu_D^{(A_{en})} \rightarrow [0, 1]$	This is a measure of the whether the environment changes state independantly of the actions of the agent. This is represented as a fuzzy set whose set membership function ranges from static to dynamic. The reason for the use of partial set membership across the domain of discourse of the agent environment is the existence of semi-dynamic environments which include some aspects of each.
Episodic	E	$E \in \{e, s\}$	A measure of whether each interaction is self contained (episodic) or if environmental changes propogate forward in time (sequential).
Continuous	C	$C \in \{c, d\}$	A measure of the granularity of the environment as being either continuous or discrete.

representation, knowledge representation, and the details pertaining to the agent function implementation. The following describes the most common taxonomy of agent environments which may be represent as environmental N-tuples according to Equation (1). Table 2 shows the composition of such an N-tuple in detail.

$$A_{en} = (O, M, P, \tilde{D}, E, C) \tag{1}$$

Regardless of the nature of environment or the selected agent architecture decision making is subject to constraints both in terms of the time available for computation of the agent function as well as access to required data and communication resources. The end result of decision making under such constraints is the concept of bounded relationality which accounts for the cognitive limits of the agents as well as the consequent opportunity costs of deliberation in a real-time environment (Byrant, 2021). Such opportunity costs arise as a result of actions not being taken due to incomplete / ongoing evaluation of possible action selections.

2.2. The principal-agent problem

Real world systems are both complex and multi-faceted and often require the interaction between human-driven and software-oriented processes. As automation becomes increasingly prevalent and the amount of time available for deliberation is reduced human beings will less able to meaningfully participate in the modern economy unless assisted to do so by artificially intelligent systems (Tegmark, 2017). This gives rise to the principle-agent problem wherein the agent must represent the interests of its principal (the human) while maintaining some degree of autonomy in the execution of its duties (Alon, Dütting, & Talgam-Cohen, 2021) as the mismatch in terms of temporal scales operated within by both agent and principal does not allow for real-time human participation.

Unfortunately, the nature of bounded rationality and the inability to completely specify human interests in a formal manner leads inevitably to the emergence of the alignment problem (Hadfield-Menell & Hadfield, 2019). The alignment problem is considered a wicked problem in terms of its multi-faceted nature. Weaker versions of the problems are often addressed for example by focussing on confirming that an agent has been authorized to act on behalf its principle rather than the degree to which those actions will necessarily benefit the principle.

3. The Emergence of Multi-agency

E pluribus unum -tr Out of many, one"
Traditional motto of the United States of America

The following section considers the emergence of multi-agency in a variety of complex systems with a particular emphasis on software engineering considerations. The recursive self-similarity (i.e. nested structural patterns repeated at different levels) exhibited in many such problems is used as a motivation for a holonic approach to considering such systems.

Environments with an agent multiplicity greater than one are considered to be multi-agent in nature. The addition of multiple agents dramatically increases the complexity of developing such systems as any internal model must account for the actions of other agents and their impact on the environment. Such systems are typically classified as either cooperative or competitive in nature based on the degree of goal alignment shared between agents. As with other such classifications it is not truly a binary one due to the emergent nature of such systems. Cooperation may emerge in ostensibly competitive systems if such cooperation yields a greater utility even if only in the short term. Likewise, selfishness may emerge in otherwise cooperative multi-agent systems due to the restrictions of bounded rationality preventing comprehensive evaluations of all possible agent plans within the collective (Coulter, 2014).

3.1. Biological emergence

The definitions of life are numerous and often imprecise ranging from broadly information theoretical approaches centring around replicative patterns through to thermodynamical based definitions focused on the local minimization of entropy. Such definitions are often framed as the importation of negative entropy, through to more conventional biological definitions based (Macklem & Seely, 2010).

Early proto-biological systems took the form self-replicative chains of nucleic acids in the form of either Ribonucleic acid (RNA) or Deoxyribonucleic acid (DNA) in either shallow bodies of water or deep oceanic vents. These replicators were subject to damage due to the varying conditions within these bodies of water. A common theory regarding the transition from prebiotic to cellular life centres around phospholipids suspended within these primordial bodies of water have a marginally hydrophilic head and marginally hydrophobic tail i.e. they exhibit an attractive force to water at one end and a repulsive force at another. As a result, membranes which are formed out of these will find a spherical least energy configuration trapping a small amount of water in their centre. Replicators which through chance and replicative drift happened to code for the production of these compounds therefore enjoyed a survival advantage due to the protection offered by these membranes and become the progenitors of cellular life. Over time these membranes developed to become more selectively permeable with chemically regulated gated channels allowing for control over the passage of chemicals to and from the external environment.

Cooperation between these early cells, mediated by way of chemical signalling, leads to a distinct survival advantage for participating cell lines in terms of risk mitigation and function specialisation. Multi-cellular life is thus an instance of the emergence of cooperative multi-agency.

It is important to note that similarities and differences do exist between natural properties and similar properties in engineered systems. Such distinctions have been considered in the context of natural adaptability in contrast with adaptability in systems (Horváth, Rivero, & Castellano, 2019).

3.2. Early software modularization

Software systems were originally written using very low-level flow control primitives in order to create branching and iterative logic. These goto statements allowed the programmer to directly modify the program counter register used by the CPU to determine which byte in memory would be subject to the fetch-decode-execute cycle. While this directly matched the operation of the underlying hardware and provided the programmers with maximal freedom regarding how to sequence the logical steps in their algorithms the lack of defined boundaries between each logical component in the system meant that the logical intent was exposed to a high degree of entropy both during the initial development and subsequent maintenance.

Such spaghetti-code systems were replaced by more structured flow control and procedural constructs after urging from well known computer scientists such as Edsger Dijkstra (Dijkstra, 1968). Eventually these components were grouped into modules such as software libraries with well defined interfaces of programmer exposed functionality. Since each of these components now had distinct boundaries between themselves and their environments, they were able to lower their exposure to high degrees of entropy as were systems constructed therefrom. By considering each of these as units of functionality the overall cyclomatic complexity of systems developed from these building blocks was significantly lowered (Sommerville, 2018). While not yet able to be considered multi-agent systems the initial stages of the separation between the system and their external environment had begun.

3.3. Object modelling

The process started by the early modularisation of software systems continued with the emergence of object-orientation in the late 1990s and its widespread adoption in the development of mainstream systems in the 2000s. Object orientation (i.e. the encapsulation of related data and the operations legal upon those data into a single entity together with other related techniques) was built upon further enhancing the boundary between system components and the internal environment of the system itself in addition to the external environment the system as a whole operates in.

While there are several principals underpinning the object-oriented design of systems the fundamental one is the concept of encapsulation which unifies the data relating to a single problem domain entity along with the operations legal on those data into a single unit. Additional relationships are possible

Table 3

A selection of UML relationship types (The Object Modelling Group, 2017)

Relationship	Symbol	Description
Association	→	A set of messages passed with a common purpose may be summarized as an association between the classes
has-a (composition)	—◆	A containment relationship representing a whole-part relationship with the implication that the container and contained do not have a meaningful independant existence.
has-a (aggregation)	—◇	A containment relationship where container and contained may have meaningful existence independantly of each other.
is-a Realisation	- - - ▷	One object implements the interface defined by another, more abstract class.
is-a Inheritance	—▷	A more general base class is specialized into a more concrete derived class.

between objects and between classes (the design-time analogue of an object prior to its instantiation). These well-defined relationships are standardised by the Object Modelling Group in the Unified Modelling Language (The Object Modelling Group, 2017), key examples of which are shown in Table 3, and further enforce the boundary between objects and the internal system environment as object-oriented systems accomplish their goals by way of message passing between objects in the system. Direct access to the internal state of an object is not possible and such values may only be accessed or modified by way of such message passing.

While object orientation represented a firm step in the direction of realising multi-agent system design the key contribution the methodology is the encapsulation of entity attributes and related operations. In order for true agent-orientation to be possible autonomy regarding the selection of those operations (i.e., the implementation of an agent-function) needs to be included.

It is worth noting that there is evidence that imperative approaches to programming such as exist in conventional object oriented programming may not be ideal for comprehension and skill acquisition and that those oriented around pure immutable mathematical functions may be preferable (Mirolo & Izu, 2019). Approaches and dedicated development languages exist which attempt to unify both paradigms (Richard-Foy & Doeraene, 2021).

3.4. Microkernels

The trend of defining boundaries between systems and the environment within which they are embedded coupled with imbuing them with a certain degree of autonomy has manifested across many categories of software system. This section and the subsequent one will continue two such important cases where this has emerged in the form of operating system design and web stack backend frameworks.

Operating Systems serve to fulfil a variety of complex needs, but their primary purpose is to serve as unified abstraction layer separating the application developers from the complexities and differences found in the underlying hardware as well as to provide fair access to those application programs to the limited resources available.

As such a distinction is drawn between code running in the core kernel of the operating system, which is able to execute a selection of privileged instructions unavailable to conventional application software, and user-mode code. This distinction, while necessary to allow the operating system to play its management role, does come at a cost in terms of context switching as values are transferred between user and kernel space and the traps necessary to switch into a mode necessary for the execution of privileged instructions. In a sense this can be seen as the boundary between the operating system kernel and the environment within which it is embedded. Operating systems differ in terms of the modularity of their internal architectures and the degree to which those modules may operate autonomously of each other. These approaches range from the monolithic architecture followed by the Linux kernel, microservice-oriented approaches followed by kernels such as the Mach kernel used within OSX and the hybrid approaches used by the Windows NT family of operating systems (Tanenbaum, 2016).

Monolithic kernels may be internally modular as is the case with the Linux kernel but still be considered monolithic due to the lack of internal autonomy of those components as well as their existing in a shared address space (Novikov & Zakharov, 2018). Truly microkernel-based approaches have independent modules for each functional area within the system while hybrid systems exist a compromise between the two. It is worth noting that the microkernels are themselves often connected via message passing systems in a manner analogous to the object-oriented approach discussed previously, the microservice approach discussed subsequently, and the actor and agent-frameworks discussed thereafter.

Although there are numerous advantages to the microkernel-based approach in terms of increased fault tolerance and scalability operating systems are very performance critical pieces of software and as such the advantages of a monolithic approach in terms of minimizing transitions to and from kernel space outweigh the advantages of loose coupling.

3.5. Microservices

The development of web-based systems has echoed similar trends towards adopting aspects of agent orientation as shown previously. Initially web-based systems were largely monolithic and although being externally facing systems that are therefore de facto embedded within an environment their high degree of internal coupling coupled with their lack of autonomy left them far from the agent-oriented ideal. Recent trends in the development of such systems have favoured the creation of systems as bundles of independently invokable web-services echoing the development of library-based modularity in conventional application development (The Open Group, 2014).

The latter part of the last decade has seen an increase in the implementation of service-oriented software systems by way of microservices architectures (Bravetti, Giallorenzo, Mauro, Talevi, & Zavattaro, 2020). In such approaches each service in the bundle is implemented by way of a microservice. Microservices themselves are implemented as cohesive independent processes which communicate with each other by way of message passing and have their own distinct data storage. There are numerous advantages to implementing service-oriented architectures in this manner.

Each microservice is its own unique process or sufficiently isolated thread of control which means they may be implemented in different programming languages and distributed across multiple compute nodes in cloud environments. Such isolation has security advantages in that their address spaces are disjoint as well as advantages in terms of parallel development in addition to parallel deployment. Scalability is also more naturally archived as more instances may be spun up and down as needed. Microservice based systems are often deployed in cloud contexts using containerisation technologies adding a further layer of modular abstraction. With a sufficient degree of abstraction, and the sequence of messages taking the place of a percept sequence microservices may be considered as multi-agent systems.

Figure 2 shows an example of a microservice oriented architecture with distinct data storage on a per problem domain entity basis (Song, Chauvel, & Nguyen, 2020).

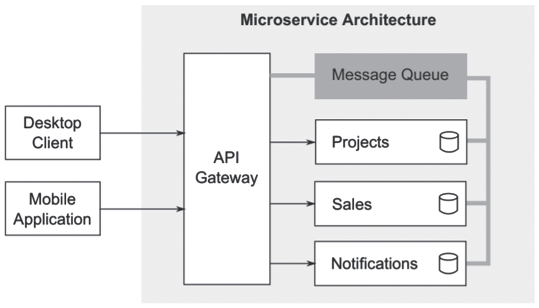


Fig. 2. A microservice architecture (Song, Chauvel, & Nguyen, 2020).

3.6. Agent development platforms

Due to their nature as critical information infrastructure (Presente, Rolim, & Moreto, 2015), telecommunication systems have many of the same needs in terms of robustness, distribution, fault tolerance as are offered by multi-agent architectures (Dorri, Kanhere, & Jurdak, 2018). This is evidenced by the contributions to multi-agent technologies made by service provides in the telecommunication space as is discussed in the section and the following one.

Explicitly agent-oriented standards and software development frameworks have been developed in the past two decades and successfully deployed. The most important standard for the development of software-based agents is ironically the communication standard defined by the Institute for Electrical and Electronic Engineering Computer Society's (IEEE-CS) Foundation for Intelligent Physical Agents (FIPA). While the degree to which this standard is used in the development of physical agents may not align with the name of the foundation however the standards they defined have formed the backbone for the development of software-based agents where the need for interoperation is a concern as they have defined an implementation-neutral standard in the form of the Agent Communication Language (ACL) specification (Foundation for Intelligent Physical Agents, 2002).

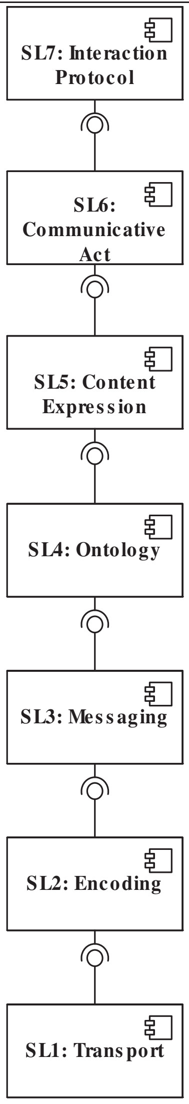
The ACL is based around Searle's Speech Act Theory (Searle, Kiefer, & Bierwisch, 1980) in which each message is strongly typed with a performative meaning allowing for both information to be conveyed as well as instructions to be issued within the same construct. A layer-based decomposition of the agent communication problem is made in a manner analogously to the decomposition used by the International Standards Organisation (ISO) in their model of internet communication (Coulter, 2014) such a decomposition is shown in Table 4.

The Java Agent Development Environment (JADE) is the most mature and well-established implementation of the FIPA standard and was developed by Telecom Italia under an open-source licence (Telecom Italia, 2021). The portability of the system due to its implementation on the Java Virtual Machine (JVM) has made it easy to deploy across heterogenous hardware configurations common in real world infrastructure environments. Development of agent-oriented systems using the framework is accomplished by specializing the abstract base classes provided by the framework and specifying the logic of each of a set of behaviours of which an agent is considered to always be running one. Agent behaviour logic consists of the sending and receiving of ACL messages and occasional switching between running behaviours selected from a pool of possible behaviours.

Figure 3 shows the static structure of such a specialisation for the purpose of creating a collaborative multi-agent system while Fig. 4 illustrates the dynamic behaviour of such a running system.

Table 4

FIPA ACL Modular Layered Decomposition (Coulter, 2014)



● Messages typically exhibit locality of reference in that the messages exchanged are often part of overarching conversations. This layer establishes the rules followed in such conversations, for example the *contract-net* protocol.
● This layer classifies the message into one of the predefined performative classes used by speech act theory.
● While the message contents may be of any form, FIPA does provide guidelines regarding the format of logical formulae and predicates. An example of a language conforming to these guidelines is the FIPA semantic language or FIPA-SL.
● This layer allows for the message contents to be mapped against a domain-specific ontology/model. FIPA-ACL does not mandate the use of any particular ontology representation.
● This layer exists in order to maintain the structure of an ACL message in a manner which is independent from the markup used in the layer below.
● This layer exists in order to avoid the problems inherent with simple byte-encoded streams. ACL messages are intended to be represented in high level markup such as XML.
● The lowest layer in the ACL is the transport layer, which represents the protocol over which ACL messages will actually be delivered such as HTTP, IIOP and WAP.

3.7. The actor model

The Actor Model is a system for the modelling of concurrent systems which has converged on a set of features making it well suited to the creation of multi-agent systems. The Actor Model was defined in 1973 by Hewitt et al to express a modular, concurrent, message-oriented model of distributed compotation (Hewitt, Bishop, & Steiger, 1973). The Actor Model has been applied successfully in the telecommunication industry for similar reasons that lead to the applicability of standard multi-agency discussed previously. As an example of this consider that the Actor Model is deeply integrated into the Erlang programming language developed by Ericsson (Chechina, et al., 2017)

Actors, together agents within multi-agent systems, can be viewed as a realization of the object-oriented paradigm in that each actor encapsulates a single processing unit together with a set of data and

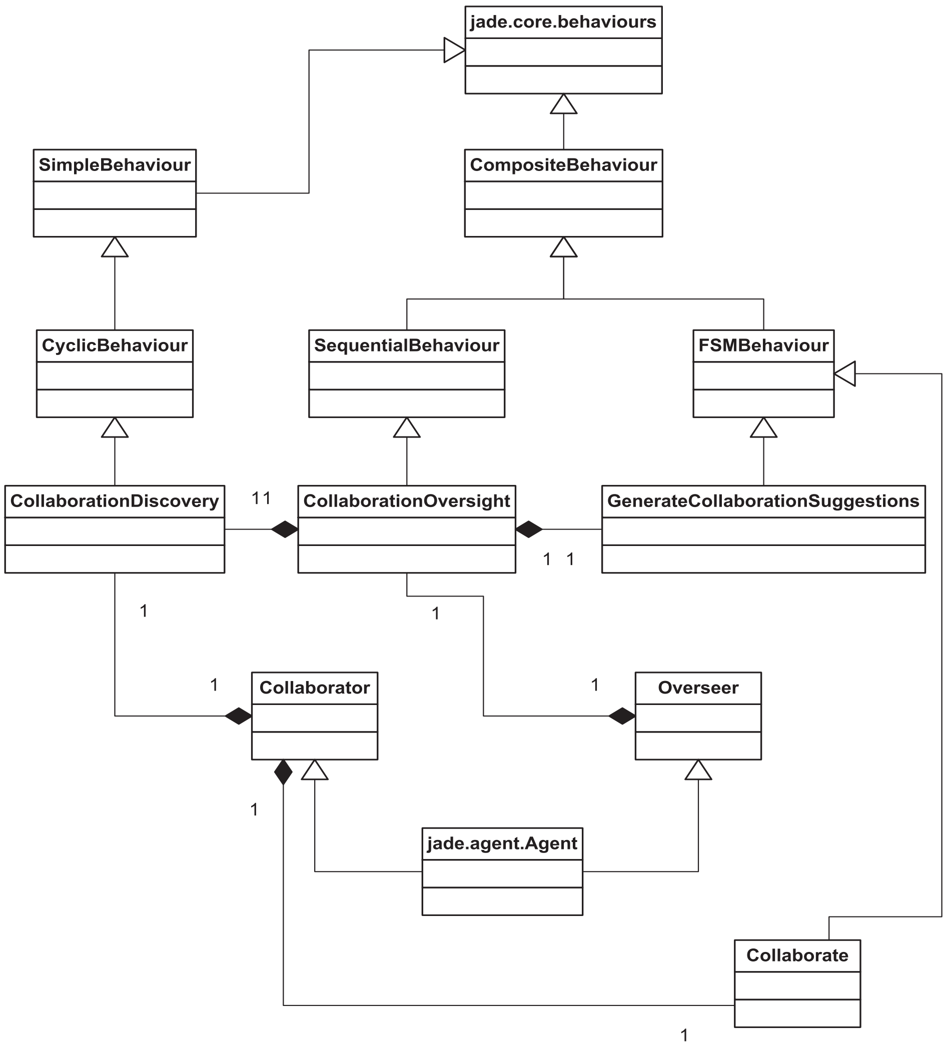


Fig. 3. Static structure of class specialisation in JADE (Coulter, 2014).

communicate by way of message passing. Interestingly enough, the Actor Model has been successfully implemented using non imperative languages, as was the case with Erlang, hybrid object-oriented / functional languages such as Scala along with more conventional object-oriented languages such as Java. The canonical implementation of the Actor Model on the Java Virtual machine is the Akka library (Typesafe Inc, 2021) which is officially supported by both the Scala and Java languages (in fact it supplanted Scala's original actor implementation of the model) although the framework should be usable by any JVM language that supports Java interoperability.

Actors, like objects and agents, communicate through a set of messages and do not have direct access to each other's internal states. Within the context of the Akka framework actors are grouped within actor systems and form nested hierarchies of actors Each actor is represented by an actor reference which encapsulates the actor's state and behaviour, its mailbox, references to each of its immediate child actors.

When an actor receives a message, it may react by either sending further messages to other actors to which it has access and/or by changing its internal behaviour. This behaviour switching approach is

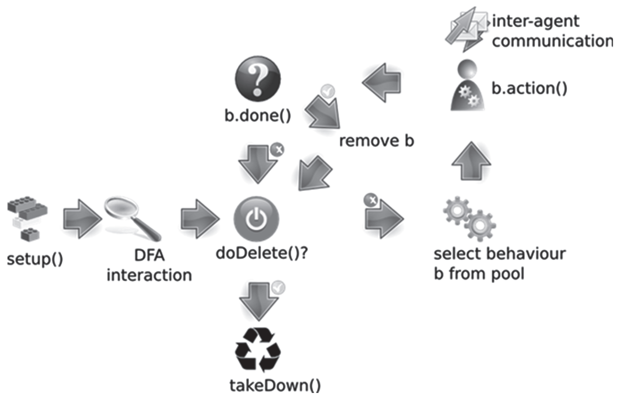


Fig. 4. Dynamic behaviour selection in JADE (Coulter, 2014).

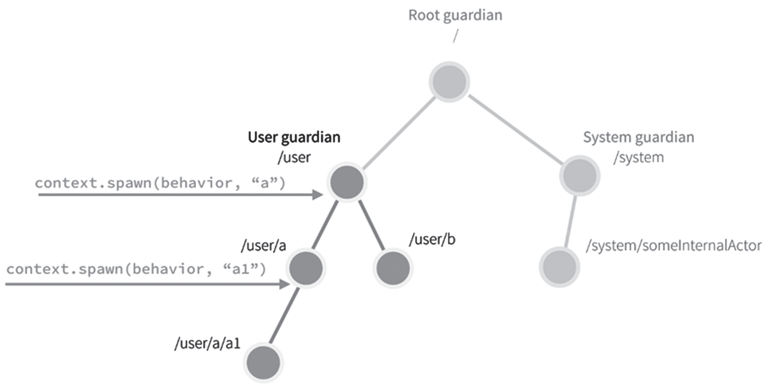

Root guardian
/
User guardian
/user
System guardian
/system
context.spawn(behavior, "a")
/user/a
/user/b
/system/someInternalActor
context.spawn(behavior, "a1")
/user/a/a1

Fig. 5. Akka actor system hierarchies (Typesafe Inc, 2021).

analogous to how agent behaviours are modified in the JADE framework supporting multi-agency as a natural endpoint for the development of distributed complex systems. An actor may also create a set of child actors over which it assumes an oversight role with errors being propagated up to parent actors in order for them to handle by way of their supervision strategy. These actors and their children form a hierarchy of nested actors akin to a process hierarchy or file system in operating systems. Actors may additionally be located by way of their ActorPath which represents their location within the actor system hierarchy. Figure 5 shows the actor hierarchy within an actor system.

3.8. *Holonic multi-agency and holarchies*

As the previous section shows actor-oriented systems where individual actors are invested with a degree of autonomy are equivalent to software based multi-agent systems whose percept sequences are implemented as a series of messages.

Many problem domains have self-similar properties reflecting the fact that examples of these problems may be recursively defined as smaller instances of the same problem in their own right. The philosophical concept of a Holon represents an entity which is simultaneously a part of something and a whole in and of itself. A nested hierarchy of holons is referred to as a holarchy. Ongoing work involving the implementation of holonic multi-agent systems which build upon these ideas using the Akka

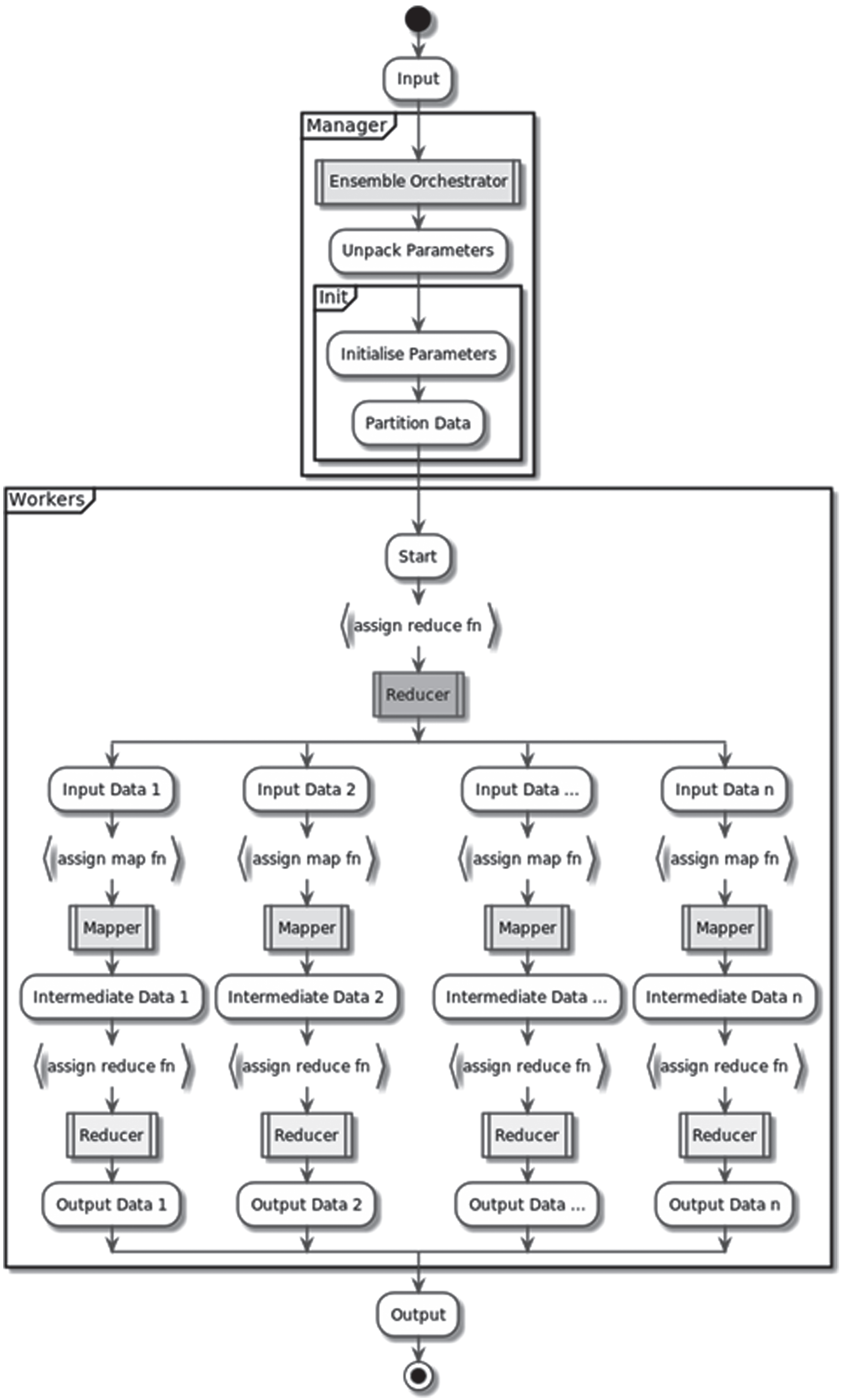


Fig. 6. A holonic multi-agent implementation of Map Reduce (Cullinan & Coulter, 2021).

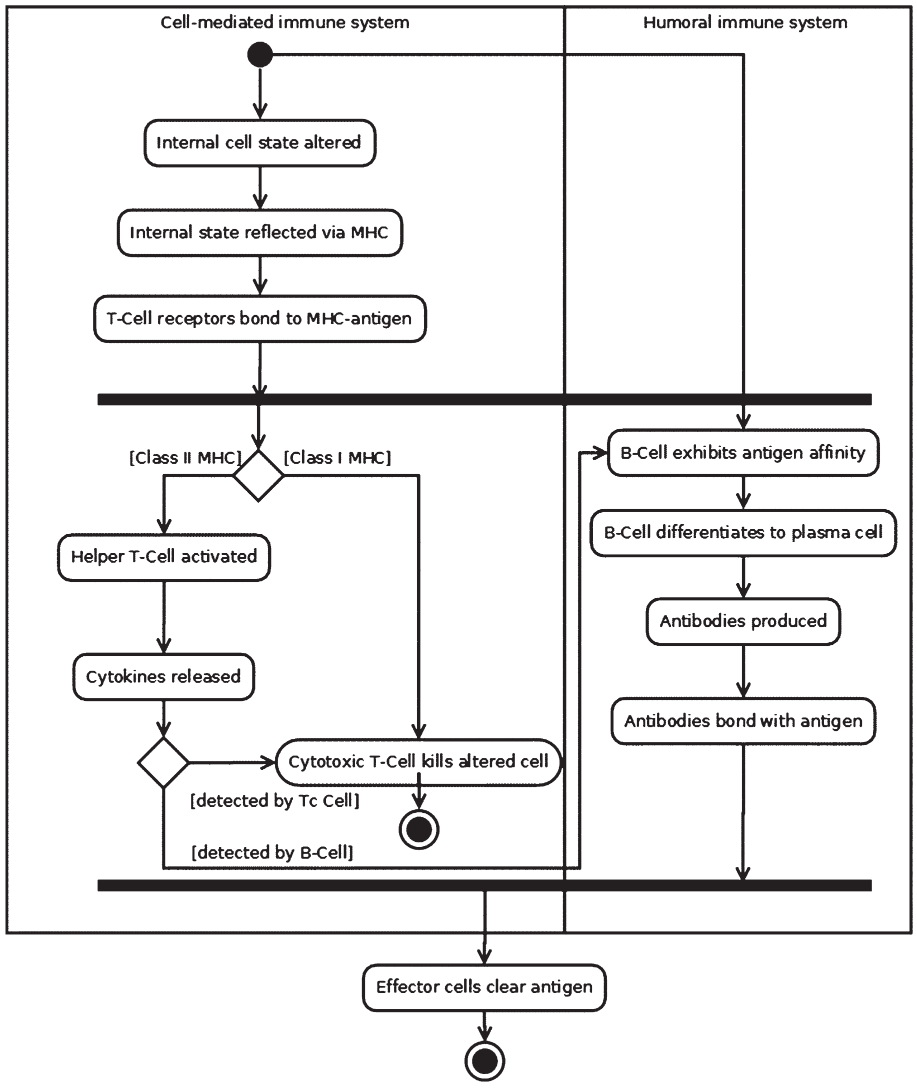

Cell-mediated immune system
Humoral immune system
Internal cell state altered
Internal state reflected via MHC
T-Cell receptors bond to MHC-antigen
[Class II MHC]
[Class I MHC]
Helper T-Cell activated
Cytokines released
Cytotoxic T-Cell kills altered cell
[detected by Tc Cell]
[detected by B-Cell]
B-Cell exhibits antigen affinity
B-Cell differentiates to plasma cell
Antibodies produced
Antibodies bond with antigen
Effector cells clear antigen

Fig. 7. Multi-agent artificial immune algorithms (Coulter & Ehlers, 2012).

framework's actor systems to realize holarchies of multi-agent systems has been explored (Cullinan & Coulter, 2021). Although complete details of the work can only be made fully known following the publication of the work the basic concept is that the self-similar nature of the widely used Map Reduce approach to structuring Big Data solutions is exploited to create a holonic multi-agent system on the Akka platform. Figure 7 illustrates this concept in abstract.

4. Computational thinking

4.1. Core competencies

The cognitive competencies required for effective software development have been discussed under many names since the inception of the discipline itself. The modern cycle of the required competencies debate was rekindled by an article in the Communications of the Association for Computing Machinery (Wing, 2006), furthered across a number of important and influential venues such as the Royal Society (Wing, 2008), and continues to this day (Bradshaw & Milne, 2021).

Computational competencies in this discussion have focussed on algorithmic thinking, abstraction, and automation. These competencies are often kept at a fairly high level as many mechanisms exist for realizing abstraction and over specifying these details is dangerous in a field as rapidly evolving as information technology. The position adopted here is that previously important aspects of developing complex systems should have been explicitly included as an aspect of computational thinking and that its omission has proven harmful. Likewise thinking in terms of multi-agent systems has become increasingly important and as discussed previously is a natural progression across multiple disparate branches of software development. The nested natures of holonic multi-agent systems necessitates the development of referential thinking capabilities as discussed in the following subsection.

For the purposes of this discussion the core competencies considered are referential thinking, parallel problem solving, and multi-agent problem decomposition.

4.2. Referential thinking, parallel problem solving, and multi-agency

A particular competency required in the development of complex information systems is the ability to reason about systems with high degrees of indirection i.e., those whose architecture makes use of multiple layers of references. Such thinking is particularly important in lower-level programming languages which directly expose the underlying memory model of the underlying hardware in the form of explicit pointer variables to store and modify memory addresses. This holds true in principle even if the actual physical addresses are abstracted behind virtual memory addresses. Errors in reasoning about such matters has resulted in a significant number of logical errors and security vulnerabilities across the years.

For this reason many attempts to solve the issue have focussed on removing direct access to pointers in favour of system managed references and while this is commendable in that it does reduce the total number of errors produced it does not negate the need for developing competency in this manner of thinking. Many problems require referential thinking (the ability to follow and correctly reason about layers of indirection) that do not involve pointers such as reasoning about deeply abstracted generic type systems (such as those involving templated code) and highly relational databases. Previous attempts to offload aspects of computational thinking onto the programming languages themselves in the form of garbage collected memory management has had similar issues in that they only considered the management of one type of resource which follows the acquire-use-release pattern namely dynamically allocated memory. There are many resources which follow such usage patterns such as file handles, both database and network connections, mutexes and semaphores all of which need well-developed computational thinking skills to be managed correctly. For this reason resource exhaustion remains an issue in long running systems even in the face of garbage collected memory.

Parallel problem solving is qualitatively different from conventional sequential algorithmic thinking. A large number of error classes such as deadlocks and livelocks are only possible in environments which consist of concurrently executing components. Due to the physical limits of expected future improvements regarding computational power as well as the growing volumes of data being processed means that most systems going forward will tend to be distributed in some form or another. The methods for reasoning about concurrently executing processes and the management of shared resources between those processes is well studied but often presented in a manner focussed on the development of multi-threaded applications in a shared memory environment. As argued throughout this paper multi-agency is a commonly reoccurring pattern in the development of distributed systems indeed if not the natural

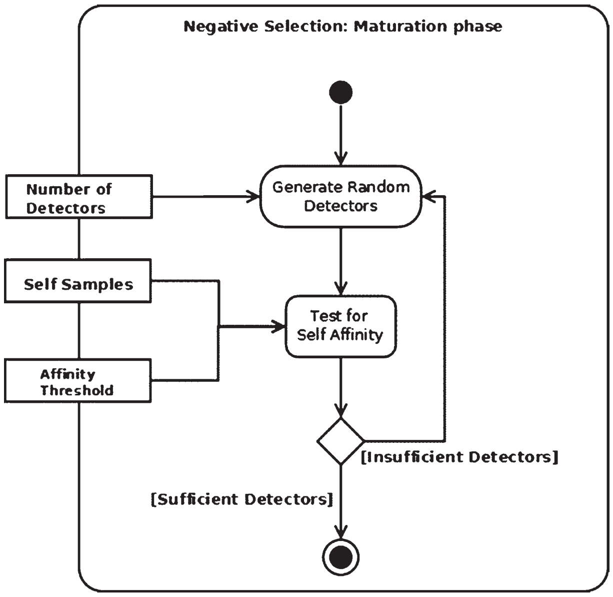


Fig. 8. A T-Cell based agent function (Coulter & Ehlers, 2012).

end point for such development. Therefore, we should consider including multi-agent reasoning as one of the core competencies of computational thinking.

5. Computational Intelligence

5.1. Immunological computation

Complex natural and biological systems exhibit properties of recursive self-similarity and multi-agency and therefore require the computational thinking core competencies as described in order to be meaningfully reasoned about.

Due to the global COVID-19 pandemic appreciation has grown regarding the ability of the vertebrate adaptive immune system to respond to previously unknown pathogens and, in general, for the quality of its response to improve over time. This learning process has served as the inspiration for numerous learning algorithms over time however, in practice, most do not capture the truly multi-agent nature of the underlying process. The immune system comprises of several layers which have evolved over millions of years. The system is incredibly complex and, as is common in biological systems, both highly coupled and highly redundant with functionality duplicated as well as the same system elements playing very different context specific roles depending in circumstances.

It is therefore natural that any tractable algorithms derived from the immune system must be highly abstracted and simplified, seeking to capture the essence of the learning process without the unnecessary complexity.

Unfortunately, the vast majority of implementations ignore the inherently multi-agent nature of the immune system. The immune system can be viewed as consisting of actors in the form of lymphocytes (most algorithms incorrectly focus on the antibody as the fundamental representation of the algorithms when they are more correctly viewed as effectors), whose percept sequences are made up of the antigens presented to them coupled with the degree of affinity exhibited to those antigens by receptors on their cell membranes (the boundary between the agent and its environment is easily defined by way of the cell membranes of the lymphocytes), and as stated previously the effectors are the various immune responses such as the antibodies produced.

In addition to not using a multi-agent approach most algorithms derived from the immune system are not implemented in a parallel manner (i.e. do make use of concurrent threads of execution) which loses the major reason for the efficacy of the biological systems upon which they are ostensibly based. The chemical and replicative processes which govern the adaptive response are in fact terribly slow it is only the fact that they are carried out in a massively parallel manner which allows the system to function reasonably at all.

The exact implementation of the agent function depends on the aspect of the immune system under consideration. Figure 7 shows both B-Cell and T-Cell mediated immunity implemented in a parallel manner while Fig. 8 shows the agent function architecture of a single T-Cell based agent.

5.2. Swarm intelligence

ALGORITHM: Multi-Agent Particle Swarm Optimization

```
particle behavior : waiting
  if(receives message of type receiving) : return behavior receiving
particle behavior : receiving
  if(receives message of type particle listing)
    onListing : tell all particles found by receptionist of my properties
    return same behavior
  if(receives message of type request to calculate fitness)
    calculate fitness :
        calculate force of attraction :
        if(fitness < currentFitness || fitness !=0)
          pbest = compared particle
          fitness = force of attraction
  if(total comparisons==threshold) : return behavior startAdjusting;
Particle behavior : startAdjusting
  if(this particle and pBest are on the same point in dimensional space) : set velocity = 0
  calculate x direction & calculate y direction &calculate euclidean distance
  if(euclidean distance < velocity) : set velocity = euclidean distance
  update particle values
  tell the system you are complete with iteration
  return behavior waiting
```

A similar issue occurs in particle swarm optimisation in which each potential solution to a problem is mapped onto a position in a high-dimensional problem space. Effectively each candidate solution can be viewed as an agent encapsulating a utility function and a set of vectors representing its position and velocity within that space. Agents then communicate the values of that utility function, using their current location as input, to a limited subset of the population of agents. Their velocity and hence position is updated by calculating a resultant vector of their local neighbourhoods' best positions as well as the best position encountered by the agent itself. Because the neighbourhood definitions for each agent overlap incompletely information is transmitted between different subsections of the population reducing the likelihood of prematurely converging on a local, rather than global, optimum.

While the narrative descriptions of such algorithms use the language of multi-agency the actual algorithms themselves do not exhibit multi-agency or indeed even concurrency in most cases. The following listing presents a truly multi-agent implementation of a particle swarm system (Thomas & Coulter, 2021).

5.3. Reinforcement learning

As a counterpoint let us consider reinforcement learning which is an explicitly agent-oriented learning paradigm in which an agent embedded within an environment selects actions from a set of possible actions given their environment state. They then receive feedback in the form of an immediate reward percept (which may be either positive or negative) and must then develop a policy mapping future percept sequences onto action selections which are expected to maximize its total cumulative reward over time. Once such family of algorithms is known as Q-Learning (Clifton & Laber, 2020).

Unlike other branches of computational intelligence reinforcement learning adheres to the agent-oriented paradigm in both the terminology used as well as the fundamental approach to realising the paradigm. Multi-agent reinforcement learning incorporate game theoretic underpinnings in addition to the Markov chains of the single agent version of the approach.

5.4. Connectionist learning

The one subdomain of artificial intelligence which has experienced the greatest surge in both interest, funding, and expectations are those based around connectionist learning approaches such as neural networks. The improvement in utilisation and applicability of these techniques is a result of simultaneous cost-effective access to computational processing power (both locally in the form of cheaper microprocessors and remotely in the form of cloud computing) as well as access to large amounts of data with which to train such models.

The most natural intersection between artificial neural networks is in the implementation of the agent function. This would typically take the form of percepts being mapped onto the first layer of such a network and each node in the output layer representing a different possible action. The most strongly activated node after a feed-forward pass through a trained network would represent the action to be selected. Training of the network would be conducted using either conventional backpropagation based supervised approaches or a hybrid computational training approach using evolutionary or particle swarm algorithms (assuming a dataset of correct action selection training examples were available).

Such approaches are of course limited in that they assume that people's interests lie in the creation of multi-agent systems in and of themselves rather than simply their being the most appropriate tool for implementing distributed intelligent systems. For this reason, research has been undertaken in implementing neural network architectures themselves in a multi-agent manner akin to how intelligence is an emergent property arising from the interaction of inherently non-intelligent components (Cullinan & Coulter, 2018). Figure 9 demonstrates such a multi-agent-oriented implementation of a convolutional neural network.

6. Conclusion

In conclusion this paper presents the position that multi-agency is an increasingly important tool within the development of complex systems. Computational intelligence techniques are increasingly used to embed such systems with the autonomy required by the paradigm. In addition, the development of such systems increasingly requires the skills of computational thinking.

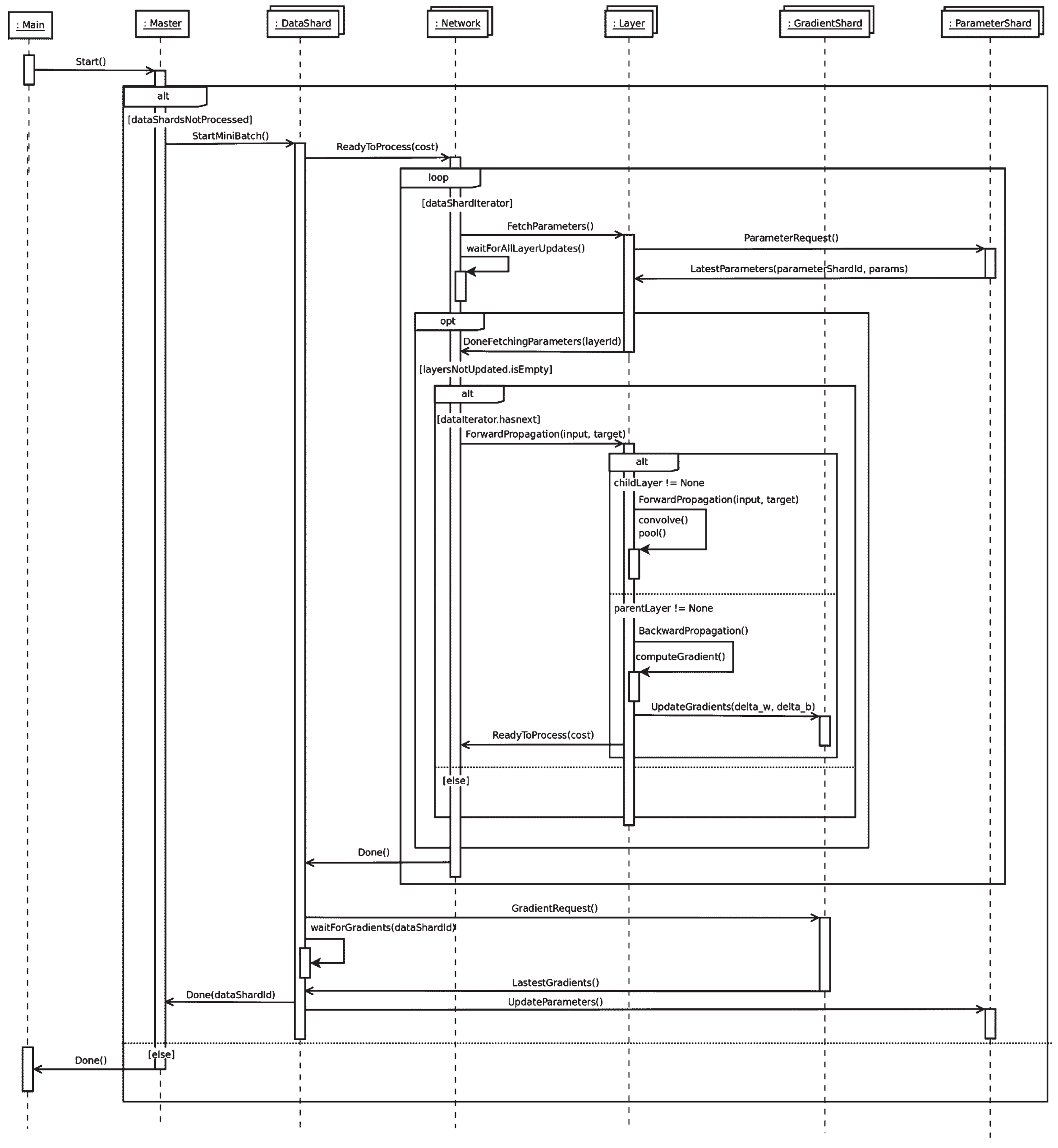


Fig. 9. An actor/agent-oriented implementation of a convolutional neural network (Cullinan & Coulter, 2018).

The tendency for complex distributed systems to acquire attributes of multi-agency has been explored and occurs with such regularity that it may be possible to view multi-agent systems as the natural end point of distributed complex system evolution.

The need for multi-agent-oriented thinking to be considered as part of the core competencies of computational thinking was elaborated upon. This point was reinforced by considering how an emerging class of economically important complex systems (namely computational intelligence systems) may be better framed in terms of multi-agency and discusses some of the work done to that effect over recent years.

Acknowledgments

The exploration of the intersection between multi-agency, computational thinking, and computational intelligence would not have been possible without the support of the Faculty of Science at the University of Johannesburg and the Head of Department at the Academy of Computer Science and Software Engineering, Professor Elizabeth Ehlers.

References

Alon, T., Dütting, P., & Talgam-Cohen, I. (2021). Contracts with Private Cost per Unit-of-Effort. *Proceedings of the 22nd ACM Conference on Economics and Computation*, pp. 52-69. Association of Computing Machinery. doi: https://doi.org/10.1145/3465456.3467651

Bradshaw, K., & Milne, S. (2021). Mapping Computational Thinking Skills to the South African High School Mathematics Curriculum. *South African Computer Lecturers Association*. South African Institute for Computer Scientists and Information Technologists.

Bravetti, M., Giallorenzo, S., Mauro, J., Talevi, I., & Zavattaro, G. (2020). A formal approach to microservice architecture deployment. *Microservices Science and Engineering*, 183-208. doi: 10.1007/978-3-030-31646-4

Byrant, P.T. (2021). *Augmented Humanity: Being and Remaining Agentic in a Digitalized World*. Springer-Nature. doi: https://doi.org/10.1007/978-3-030-76445-6

Chechina, N., MacKenzie, K., Thompson, S., Trinder, P., Boudeville, O., Fördős, V.,. . . Hernandez, M.M. (2017). Evaluating scalable distributed erlang for scalability and reliability. *IEEE Transactions on Parallel and Distributed Systems*, *28*(8), 2244-2257. doi: 10.1109/TPDS.2017.2654246

Clifton, J., & Laber, E. (2020). Q-learning: Theory and applications. *Annual Review of Statistics and Its Application*, *7*, 279-301. doi: 10.1146/annurev-statistics-031219-041220

Coulter, D.A. (2014). *Immunologically Amplified Knoweldge and Intentions Dimensionality Reduction in Cooperative Multi-Agent Systems*. Johannesburg, Gauteng, Johannesburg: University of Johannesburg. Retrieved from http://hdl.handle.net/10210/12341

Coulter, D.A., & Ehlers, E.M. (2012). NSFlock: Design Evaluation via Immunological Agents. *Tools and Methods of Competitive Engineering*, Karlsruhe, Germany: TU Delft, pp. 187-199.

Cullinan, M., & Coulter, D.A. (2018). Revisting the Society of the Mind: Convolutional Neural Networks via Multi-Agent Systems. *Tools and Methods of Competitive Engineering*, Las Palmas de Gran Caneria: TU Delft, pp. 121 - 132.

Cullinan, M., & Coulter, D.A. (2021). Random forest classification with mapreduce in holonic multiagent systems (accepted for publication). *Advances in Intelligent Systems*.

Dijkstra, E. (1968). Go To Statement Considered Harmful. *Communications of the ACM*, *11*(3), 147-148. doi: 10.1145/362929.362947

Dorri, A., Kanhere, S.S., & Jurdak, R. (2018). Multi-Agent Systems: A Survey. *IEEE Access*, *6*, 28573-28574. doi: 10.1109/ACCESS.2018.2831228

Foundation for Intelligent Physical Agents. (2002). *FIPA ACL Message Structure Specification*. Document ID SC00061.

Hadfield-Menell, D., & Hadfield, G.K. (2019). Incomplete Contracting and AI Alignment. *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, Association for Computing Machinery, pp. 417-422. doi: 10.1145/3306618.3314250

Hewitt, C., Bishop, P., & Steiger, R. (1973). A Universal Modular Actor Formalism for Artificial Intelligence. *International Joint Conferences on Artificial Intelligence*, pp. 235-245.

Horváth, I., Rivero, J.P., & Castellano, P.M. (2019). Past, present and future of behaviourally adaptive. *Journal of Integrated Design and Process Science*, 1-15. doi: 10.3233/JID190006

Macklem, P.T., & Seely, A. (2010). Towards a definition of life. *Perspectives in Biology and Medicine*, 330-340. doi: 10.1353/pbm.0.0167

Mirolo, C., & Izu, C. (2019). An Exploration of Novice Programmers' Comprehension of Conditionals in Imperative and Functional Programming. *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, pp. 436-442. doi: 10.1145/3304221.3319746

Novikov, E., & Zakharov, I. (2018). Verification of operating system monolithic kernels without extensions. *Lecture Notes in Computer Science*, *11247*, 230-248. doi: 10.1007/978-3-030-03427-6_19

Presente, J.R., Rolim, J.G., & Moreto, M. (2015). MultiAgent systems in power system protection: Review, classification and perspectives. *IEEE Latin America Transactions 14*(7), 3285-3290. doi: 10.1109/TLA.2016.7587632

Richard-Foy, J., & Doeraene, S. (Eds.). (2021). Proceedings of the 12th ACM SIGPLAN International Symposium on Scala.

Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach*, (4th ed.). London: Pearson.

Searle, J.R., Kiefer, F., & Bierwisch, M. (1980). *Speech Act Theory and Pragmatics*. Londom: D. Reidel Publishing Company. doi: 10.1007/978-94-009-8964-1

Sommerville, I. (2018). *Software Engineering*. Pearson India.

Song, H., Chauvel, F., & Nguyen, P.H. (2020). Using microservices to customize multi-tenant software-as-a-service. *Microservices Science and Engineering*, 291-331. doi: 10.1007/978-3-030-31646-4

Tanenbaum, A.S. (2016). *Modern Operating Systems* (4th ed.). Pearson India.

Tegmark, M. (2017). *Life 3.0: Being Human in the Age of Artificial Intelligence* (1st ed.). New York: Knopf Doubleday.

Telecom Italia. (2021). *Jade Documentation*. Retrieved from Java Agent Development Environment: https://jade.tilab.com/documentation/tutorials-guides/

The Object Modelling Group. (2017). *Unified Modeling Language.* The Object Modelling Group. Retrieved from https://www.omg.org/spec/UML/

The Open Group. (2014). Standard for Service-Oriented Architecture Ontology, Version 2.0.

Thomas, P.C., & Coulter, D.A. (2021). Coordination Lattice Definition via Swarm-based Neighboring Actor Phase-transitions. *The 9th International Conference on Computer and Communications Management*. Singapore: Association for Computing Machinery.

Typesafe Inc. (2021). *Actor Architecture*. Retrieved from Akka Documentation 2.6.15: https://doc.akka.io/docs/akka/current/general/actors.html

Visser, B.A., Ashton, M.C., & Vernon, P.A. (2006). G and the measurement of Multiple Intelligences: A response to Gardner. *Intelligence*, *34*(5), 507-510. doi: 10.1016/j.intell.2006.04.006

Wing, J.M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35. doi: 10.1145/1118178.1118215

Wing, J.M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *366*(1881), 3717-3725. doi: 10.1098/rsta.2008.0118

Author Biographies

Duncan Anthony Coulter is an Associate Professor and Head of Department at the Academy of Computer Science and Software Engineering in the Faculty of Science in the University of Johannesburg. His research interests primarily concern the application of biologically inspired artificial intelligence techniques in the context of multi-agent systems development with a special focus on cloud deployed multi-agent ensemble learning.