VoxelNET's Geo-Located Spatio Temporal Softbots

-including living, quiet and invisible data

Charlotte Sennersten, Craig Lindley and Ben Evans
CSIRO
Pullenvale, Australia
e-mail: charlotte.sennersten@csiro.au, craig.lindley@csiro.au, ben.evans@data61.csiro.au

Abstract— **Linnaeus and Darwin understood the need to classify 'living things' to determine the basis of their relationships and interrelationships. In an Internet-of-Things (IoT) world we need to do the same, to be able to identify and compute with objects by type. However, the IoT does not inherently deal with spatial or geometrical structure, and mass phenomena (e.g. air, water, rock) are not objects per se. This can exclude these 'non-object' things from the IoT, which can be a severe disadvantage in many application domains. The solution to this is voxelisation of mass phenomena in the world within an overall coherent three dimensional coordinate reference system. This allows 'non-things' to be coherently situated, classified and treated computationally in the same way as discrete things and individual objects. VoxelNET is a distributed digital architecture that supports this voxelisation model, providing a world of voxels containing various information at different geo locations that can be compared in terms of numerous and unlimited taxonomical categories, and over time. Performing computations across this highly distributed system of systems can greatly benefit from the use of distributed softbots or agents without the need for centralized computations or control. Hence the VoxelNET distributed architecture not only parses objects and materials into computable objects, but also includes spatially located and volumetric computational agents that can collectively achieve analytical outcomes in an inherently distributed way. Here this approach is exemplified by distributed VoxelNET agents collaborating to conduct 3D volumetric path finding through the VoxelNET space, using a distributed Dijkstra pathfinding algorithm. Stronger implementations of the agent concept can include supplementing the basic Dijkstra algorithm with more sophisticated competitive and/or collaborative behaviours on the agents/voxels involved.**

Keywords-Voxel Agents; Autonomy; Industry 4.0; Reasoning; Computation.

I. INTRODUCTION

Internet 4.0 is the currently emerging next stage in the evolution of the internet, expanding from client-server intercommunications to peer-to-peer systems, combining historical information based on where devices have been with information from diverse nearby sensors (eg. location sensors, smart home components) and using artificial intelligence analytics to create new knowledge and experiences [1]. In this evolution, the Internet will be available in all places and at all times in the background. Applied to industry, Internet 4.0 supports the concept of Industry 4.0, which has been defined as "the current trend of automation and data exchange in manufacturing technologies. It includes cyber-physical systems, the Internet of Things (IoT), cloud computing and cognitive computing. Industry 4.0 is commonly referred to as "The Fourth Industrial Revolution." [2]. The IoT extends internet connectivity to many kinds of devices and objects, especially in order to gather sensor data and/or parameterize and control the devices as part of larger scale systems and operations.

To digitise the world and comprehend it we need to classify knowledge in a digital system. The voxels in VoxelNET can be the basis for this systemization [3] and also support the use of various algorithms, such as Linear Classifiers, Decision Trees, and Nearest Neighbour [4]. Looking back at former scientists contributing, systematizing and taxonomising our (biological) world, Linnaeus published a system for classifying living things [5] and Darwin an evolutionary taxonomy (a branch of biological classification) with evolutionary change [6]. In a world of change we need to connect changes to understand cause and effect. The VoxelNET system is also at its base a taxonomical system, most fundamentally classifying things in the world by their location, but also supporting arbitrary further dimensions of classification. Also it is a distributed system, which makes collective computations highly amenable to agent-based computational processes, which are the focus of this paper.

Systems based upon Industry 4.0 technologies and methods constitute what Cardin [7] calls *Cyber-Physical Production Systems (CPPSs)*, defined as systems of systems of autonomous and cooperative elements connecting with each other in situation dependent ways, on and across all levels of production, from processes through machines up to production and logistics networks, enhancing decision-making processes in real-time, responding to unforeseen conditions and evolving over time. Cyber-physical systems in general monitor physical processes, using sensor data from the IoT layer to create and update a virtual copy of the physical world (a "digital twin"), and use the virtual twin together with analytical techniques to optimize operational decisions in real-time across the value chain. A digital twin can include site, object and agent spatial locations and

 ISBN: 978-1-61208-705-4

structure, operations and process models, assets, plant, equipment, vehicle and staff representations, times and locations, etc.. Since CPPSs are systems of autonomous and cooperative elements, it is natural to look to artificial computational agents as a basis for computing within and across these systems.

This paper describes the VoxelNET [3] Industry 4.0 and CPPS platform and the way in which computational agents can be created and function using this platform, especially agents that are founded upon its inherent 4D structure (three dimensions of space plus time). Demonstrated VoxelNET applications include mine operations and Unmanned aerial vehicle Traffic Management (UTM), both of which have strong requirements for computational modelling and analysis of 3D space. A distributed, agent-based approach to path planning is presented, which addresses requirements within both of these application domains.

First in Section II 'The VoxelNET System' is introduced and its 'Functionality' and 'Architecture' are described. Section III describes 'The VoxelNET Agent Model' and how agents use the platform, and in Section IV 'The Voxel Agent Example', a distributed agent-based planning system is described to exemplify the system.

II. THE VOXELNET SYSTEM

The 4D VoxelNET system is so-called because it deals with locations, objects, and materials within an integrated 4D spatiotemporal framework of voxels, where a voxel is a volume element. The voxelisation of locations is made in the form of a 3D geodetic spatial coordinate reference system having default unit voxels of approximately one cubic meter over the surface of the Earth to any depth or altitude. One cubic meter location voxels can be dynamically aggregated or decomposed to create a location voxel coordinate system having one or more location voxels of any required size and with variable geodetic dimensions along each coordinate axis. The system also supports the definition of Euclidean local coordinate systems that have an origin within the geodetic spatial reference systems and rotations around geodetic axes. The voxelisation of object representations can amount to tessellated 3D models of whole objects or object parts having naturalistic shapes, which allows arbitrary models to have voxel properties and functions. The voxelisation of materials that are not already divided into objects spatially quantises the spatial extension of material otherwise typically described in mass terms (such as air, water, rock) into collections of discrete 3D spatial units. In this way VoxelNET can represent all 'open space' entities such as air, soil, rock, space and water that otherwise would not conform to typical IoT concepts into units that allow them to be integrated into the IoT. Hence both extended natural phenomena, objects and artefacts and their associated sensor data can be integrated within one architecture.

A. VoxelNET Functionality

VoxelNET spatial structure provides several generic functions that specific applications can be built upon, many of which are derived from OGC spatial standards for simple geometry types (see OGC: 06-103r4 Part 1 and Part 2, [8]). These include:

- Search by spatial criteria, such as object enclosure, proximity, intersection
- Synthesis of new shapes using mathematical operations defined upon input shapes, e.g. intersections, differences, unions
- Derivation of spatial relationships between shapes, such as distance, overlap, encapsulation

VoxelNET extends these geometrical operators and functions with functions built upon an underlying conceptual model including:

- Complex spatial object definitions expressed in terms of a variety of structures and relationships among simple geometry types
- Associations of shapes with material types and properties
- Association of complex shapes and associated materials with models for operations definitions, capabilities, performance and schedules, personnel, assets, processes and process segments, resources and resource relationships, equipment, workflow specifications, work definitions, job lists and schedules, test specifications and results
- Classes, types, instances and both class and instance properties
- Material transforms
- Modalities potentially associated with any numeric values, such as probabilities and ontological/epistemic status
- Aggregation of these elements as types and instances with higher level (or meta-level) conceptual constructs, such as domain paradigms/interpretative frameworks, data/information/knowledge bases, ontologies/taxonomies, knowledge bases and/or expert systems, agents, agent societies/communities, and computational ecosystems

The latter higher level (or meta-level) conceptual constructs may include specialised computational models in addition to the data, information and knowledge included within the underlying conceptual models. VoxelNET computing is achieved via several programming paradigms, of which agents are a high-level example. These paradigms include:

- *Scripted programs that sit outside of the voxel structure* and traverse it to achieve outcomes such as: i) finding voxels that meet some criteria (i.e.

data base querying, data filtering), ii) analysing voxel collections, iii) parsing and editing subsets of voxel space.

- *Triggers* associated with specific voxels (that are logically or conceptually *within*) a voxel, such that if their associated data changes, one or more defined computations are carried out. An example is, if a location voxel is mined, this can trigger the spawning of a material voxel. Since the voxel structure is hierarchical, and voxels can be members of a larger scale voxel or association, triggers can be inherited (potentially upwards or downwards).

- *Processes* associated with specific voxels (that are logically or conceptually *within* a voxel/structure), may run continuously, carrying out one or more defined computations. An example is, a voxel process interrogates a defined neighbourhood of voxels to check if any have been mined, and to derive a cost/value hypothesis from the mined states of its neighbours, their rock harness, grades, etc.

- *Finite state machines* (FSMs) are state transition engines that move a sequence of voxels from one state to another in response to changes in one or more of the voxels (i.e. inputs, or internally driven state changes). They fall between triggers and processes, since they are complex sequences, but driven by trigger events, rather than running continuously.

- *Agents* are more complex computational processes based upon any of a range of computational cognitive models. The most critical features include declarative knowledge modelling and goal-directed decision processing. *Social* agents are a specialisation of computational agents that can engage in collaborative or competitive behaviour (for example). A voxel or voxel association can be an agent if some part of its behaviour is controlled by a cognitive model, e.g. [9].

B. VoxelNET Architecture

The total vision for the VoxelNET system described above presents a highly complex computational environment that unpacks the concept of Industry 4.0 and intelligent cyberphysical production systems into the full range of high level elements and components needed for their realisation in the case of systems of significant size and complexity (e.g. a flexible manufacturing factory). Smaller scale systems may require a much more limited subset of these features. This leads to the following broad and overlapping classes of system architecture:

- **Stand-alone systems** that encapsulate modelling, monitoring, analytics and decision processes without the need to interact with other systems. Examples of this that have been demonstrated so far in the VoxelNET case include: a system for real time control of a UAV operating in underground mines, based upon a digital twin and third person view of the vehicle and its environment (see [10]); and a system for the analysis, feature detection and visualisation of heterogeneity in mineral deposits.
- A **cloud-based client/server/repository architecture** that has been demonstrated for real-time multi-user interaction with mineral resource drill hole, blast hole and block model data. In this case, the client/server messaging interface and underlying repository schema provide foundations for interoperability among client systems that may include third party software using proprietary data formats, provided that those formats can be translated into canonical VoxelNET schema constructs.
- A **fully distributed architecture**, with many application clients interacting with distributed servers/repositories via a mediating VoxelNET distribution server network. While not yet implemented, the standardised message syntax, semantics and underlying conceptual models required extend those used for client/server interaction in the cloud-based architecture.

Challenges in meeting this Industry 4.0 vision at a large scale include:

- **Avoiding over-engineering**. The system should try to focus on representations and functions that provide the most critical inputs to system understanding and decision processes, or else it may be difficult to place bounds upon how much needs to be sensed and represented (e.g. the context of a system is essentially endless).
- **Consistent syntactic and semantic mapping of data, information and knowledge** from multiple diverse sources to a uniform underlying conceptual model.
- The need to accommodate **changes in the structure and functionality** of physical and operational systems over time.
- **Achievement of trust and reliability** in the system.
- The need for **standardized terminology, types and classes** that may be expressed in ontologies and/or taxonomies. For example, object types or classes should not be open, but documented, where new types/classes can extend the

 ISBN: 978-1-61208-705-4

documentation. A particular range of phenomena may have more than one ontology/taxonomy associated with it, reflecting different perspectives, uses, etc.

The following sections of this paper describe the agent-based functions of the VoxelNET system, especially those based upon the 3D voxelisation of spatial structure.

III. THE VOXELNET AGENT MODEL

A. Artificial Computational Agents

VoxelNET is a highly distributed system concept, including the virtual and physical distribution in 3D space of artificial intelligence (AI) and autonomous decision making functions. This means that autonomous decision making that uses data, information and/or knowledge, or has output implications, beyond the bounds of individual server/repository nodes will need to interact with AI functions resident on other nodes. This lends itself to an agent-based AI model, where a software agent can also be referred to as a *softbot*, meaning a software robot [11]. This does not exclude softbots that constitute elements of the control architecture of physical robots.

There are many definitions and variants of agents in the research literature that are distinguished along numerous dimensions of variation (e.g. see [12], [13]). Some of these variations include: reactive versus deliberative/goal-driven, stand-alone versus collaborative, distributed versus centralised, mobile versus immobile, situated versus disembodied, hardware versus software, adaptive/learning versus un-adaptive, etc. Many architectures are hybrids of the distinctions made along these various dimensions. VoxelNET is compatible with any of these models, since they can exist as computational components in the server ecosystem or interfacing with generic VoxelNET functions accessible via the VoxelNET messaging layer.

It is a requirement of VoxelNET agents that they can be intelligent, although what constitutes intelligence in synthetic or natural agents can be greatly debated. Teahan [14] characterises an agent as acting intelligently when "what it does is appropriate for its circumstances and its goals, taking into account the short-term and long-term consequences of its actions, is flexible to changing environments and changing goal, learns from experience and it makes appropriate choices given its perceptual and computational limitations".

A benefit of this definition is that it is expressed in terms of functionality rather than mechanisms. Definitions based upon mechanisms are in general undesirable, since specific mechanisms do not necessarily guarantee intelligent outcomes, while a given level of intelligent functionality might be implemented by a different set of mechanisms.

B. The VoxelNET Agent Model

The VoxelNET system constitutes a data, information and knowledge ecology that is compatible with numerous computational, agent and intelligence paradigms, interacting via a foundational conceptual model and standardised message formats and protocols. It nevertheless has a more inherent agent paradigm based upon voxel structures, which is described in more detail in this section.

A voxel itself is able to function in the system as a self-contained inter-netted agent that can virtually perceive, receive and issue messages, compute, change its state and generate outputs. As noted above, any specific voxel may have associated triggers, processes, finite state machines, or agent models non-exclusively associated with it (i.e. one voxel may be associated with several computational components). This means that all voxels in VoxelNET can function as agents within the interlinked voxel world.

VoxelNET users are presented with a default navigational voxel structure for the Earth of approximately 1 m^3 voxels that extends over the mean surface and to altitudes from 20 km to a depth of minus 10 km (this range is dynamic and can be extended as required). This results in 10^{19} voxels, any or all of which can be an agent. Voxels can be aggregated or decomposed, and the resulting larger and smaller scale voxels can also be agents. The question then is, what can an individual voxel agent do and process? In principle, this could include a broad range of intelligent behaviours, including:

- perceiving, including recognize patterns
- learning (form correlations)
- learning (form concepts, i.e. abstractions)
- understanding (form predictive models)
- apply logic and reason
- comprehend ideas (use abstractions)
- plan
- solve problems
- make decisions
- retain data, information and knowledge
- use language to communicate (using the syntax and semantics of the VoxelNET messaging system)

VoxelNET agents may belong to one or more classes, although this is not elaborated in this paper. Instead, some general agent functions are presented, as well as a specific example of how voxel agents can interact to create a collective function.

General voxel-agent-functions include the capacity to send and receive messages addressed by specific voxel ID, by geodetic or local volume, or broadcast across the VoxelNET, with each voxel selecting to respond to messages by type or according to one or more specific criteria specified in the message. Message inputs can be sent to voxel triggers, finite state machines, processes and agents, according to message metadata and rules for processing each computation type held within the voxel. Specialised messages may carry programs, including implementations of

 ISBN: 978-1-61208-705-4

agent decision processes, so that generic functions are primarily concerned with agent program input and management, rather than having extensive built-in behaviours.

It might be asked why a specific voxel should be an agent, rather than have external processes perform computations upon voxels? The reasons for this are:

i) That these two options are logically, and may be implementationally, equivalent. A computational agent in a closed computational environment (such as the single server/repository cloud-based version of the VoxelNET architecture) may be processed within a computational context that is external to the agent, but not external to the system. This may, for example, be an algorithmic loop that triggers or polls each agent to accept inputs, update its status, and generate outputs with each iteration of the loop. The more central question here is whether algorithms implementing agent state transitions are represented within the program structure of each agent, or exist as elements of an algorithm library that can be applied by the external process to a data structure that encapsulates the agent state. For a purely virtual agent system, the difference is one of packaging from the perspective of design and maintenance (analogous to the issue of procedural/structural versus object-oriented programming).

ii) An open computational environment (such as a fully distributed version of the VoxelNET architecture) is one in which agents may be created by different parties, using different languages, algorithms, etc. In this case, algorithms need to be implemented within agent components that can intercommunicate via standardised messaging formats, protocols and semantics, irrespectively of the languages and specific form of data structures and algorithms created by different parties that conform to standardised messages.

iii) Systems that include physical agents, such as robots, may include agents having widely varying internal implementation requirements and computational capacity and will generally require real time internal control, with potentially asynchronous and/or near-real time interaction with other agents as required within the ecosystem. This case is similar to case ii), but with additional requirements for encapsulating computational agents within physical bodies or structures arising from the needs of timeliness and the ability to operate in real time independently of external communication links.

Task types and performance details in different implementations may nevertheless be incorporated into task libraries, which can be accessed and downloaded into agents anywhere in VoxelNET, if a situation or message type arises for which a given agent does not already have an appropriate task definition.

IV. THE VOXEL AGENT EXAMPLE

An example of how voxels can function as agents is provided by considering an agent-based approach to path planning, based upon the Dijkstra algorithm [15]. VoxelNET can include physical robots and has been demonstrated as a platform for individual unmanned aerial vehicle (UAV) control in underground mines [10] and for control of systems of many UAVs in an urban environment. The voxel structure can be used to implement occupancy grids for autonomous vehicle navigation, and for the storage and representation of 3D mapping data [16], [17].

The path planning task has been demonstrated in two application scenarios for VoxelNET: i) UAV inspection and mapping in underground mines, and ii) UAV traffic management in urban environments. These were both single-point demonstrators of the path planning solution, while here we present a distributed approach to solve the same task. The high-level process is described by:

1. UAV sends StartPath message to starting voxel, using geo-located address and including flight start time, end time, priority, vehicle type, mission type, vehicle id, service class, etc.. The voxel state is changed to indicate that the voxel is part of this path planning process.
2. UAV sends End of Path message to geo-located address of end voxel, including flight start time, end time, priority, vehicle type, mission type, vehicle id, service class, etc. Voxel state is changed to indicate that the voxel is part of this path planning process.
3. Start and End voxel agents send the same message to each adjacent voxel not already involved in this path planning process, plus the path in progress, which path it is, the count of iterations and the count of path voxels.
4. If an adjacent voxel is available and is not already involved in this path planning process, steps 1 or 2 are repeated by this voxel, respectively, for the developing start and end path options from the perspective of the path developed so far. If an adjacent voxel is *not* available, its voxel state is changed to indicate that the voxel is part of this path planning process, but steps 1 and 2 are not iterated by this voxel for its adjacent voxels.
5. If a voxel in the starting path meets a voxel in an ending path, it will append the end path to the starting path and notify the originating voxel that a path has been found, with its parameters, including length.

 ISBN: 978-1-61208-705-4

6. The starting voxel agent can choose to accept the first path created, or wait to see if a shorter or more valued (optimal) path occurs.

This algorithm is potentially very inefficient, since, if not stopped at the first path found, it searches for all paths through a connected network. However, it lends itself to distributed computation and there is no need for any centralised representation of the total network – connected voxels are readily determined from the position of a given voxel and the structure of the coordinate system. This algorithm requires very simple agents, but more sophisticated agent capabilities could be built upon it, such as cooperative behaviours, in which voxels work together to negotiate which path a given request will have allocated to it, or competitive behaviours, such as pre-empting path calculations to effectively reserve paths or shorten pathfinding time.

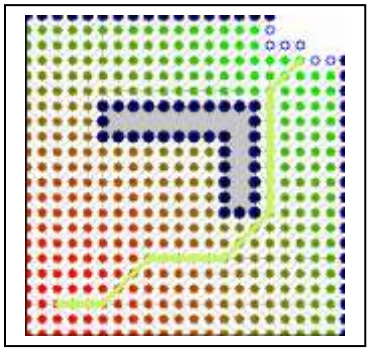

Figure 1. 2D Path planning problem.

A. *Implementation –Voxel Path Planning*

A path planning algorithm searching for the shortest path(s) [18] is often a top down 2 dimensional problem (see Figure 1), but here we are including the third dimension so the search space is volumetric.

Path length differences are illustrated in Figure 2; there can be many other paths adjacent to the shortest path, but they all involve at least 5 + 2 = 7 steps.

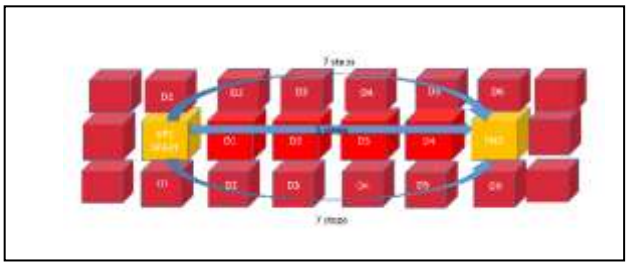

Figure 2. Example of path length calculation.

Here we adapt the well-known Dijkstra algorithm for finding paths between nodes in a graph, which may represent, for example, road or travel path networks. In this case the graph structure is the interconnected graph structure of a 3D, orthogonal voxel matrix. The algorithm exists in many variants. Dijkstra's original algorithm found the shortest path between two nodes. A more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

The path finding solution shown in Figures 3 and 4 was implemented in C#. For the path finder to operate, a start voxel and an end voxel must be selected, and the pathfinding process begins simultaneously at these two voxels. Each voxel has a flag to identify whether it was approached from the start or end (referred to as the direction flag), whether it has been processed, and whether it is available.

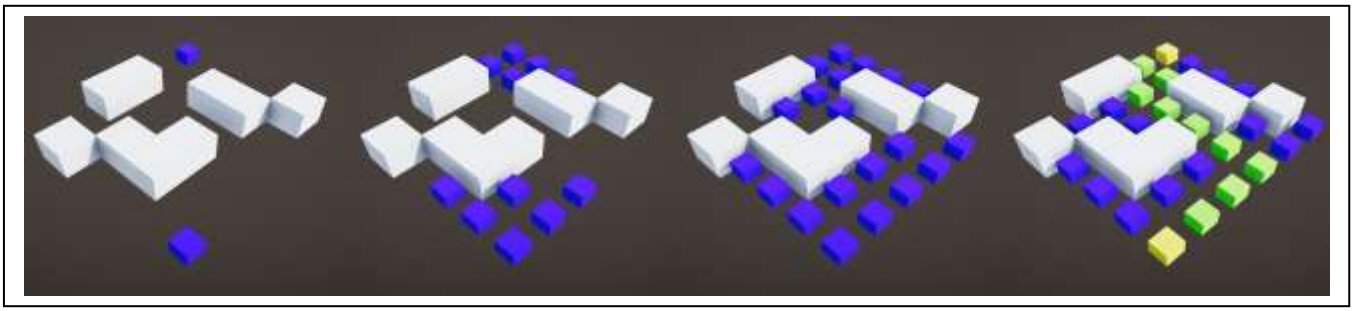

Figure 3. Voxel Path Planning implementation in 2D (white is an obstacle, blue is traversed and green is the final path).

 ISBN: 978-1-61208-705-4

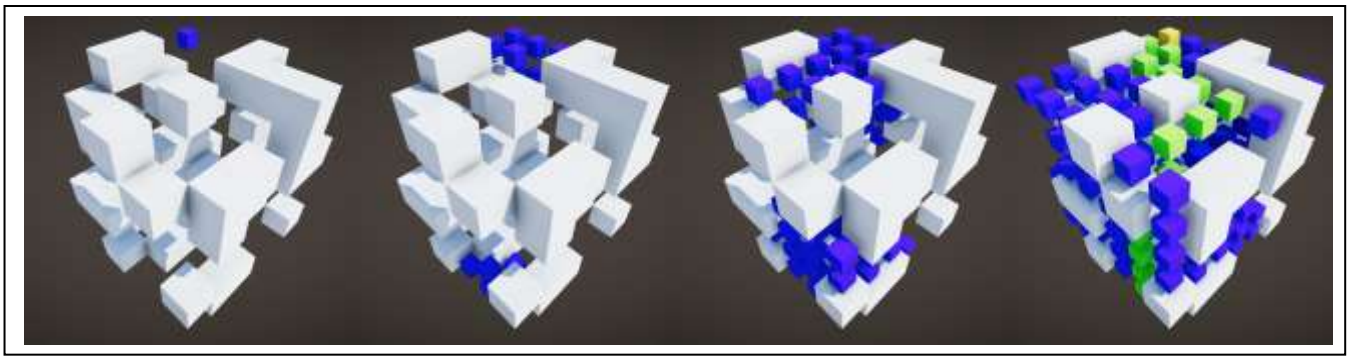

Figure 4. Voxel path planning implementation in 3D (white is an obstacle, blue is traversed and green is the final path).

Every adjacent voxel is evaluated and if it was not already evaluated and is available (meaning it is not obstructed), the voxel it was approached from is stored in its state and the evaluated flag is updated, as well as the direction flag, to identify whether it is in a path travelling from or to the end voxel. This voxel will propagate the evaluation to adjacent voxels and so on (implementing simple agent-agent interaction). If a voxel is already evaluated/processed and it has a different direction flag, it means that a path has been found connecting the start and end points. The voxel chain is traversed in a similar fashion to a linked list and each voxel is placed into a collection to make up the path, which is stored in an external object passed in at the start of the path finding process so that it can be retrieved externally or dealt with asynchronously (which was the case in the source code and should apply to a physically distributed system). The implementation finds the quickest path to compute (the path requiring the least number of iterations) but does not attempt to find the shortest or most optimal path. This functionality could be added with some small modifications.

V. CONCLUSION

This paper has described the VoxelNET distributed spatial data system and how the concept of agents can be applied within this system. A simple example of agent interaction has been implemented and described. Achieving path finding by the distributed agent approach allows voxels as agents to intercommunicate via voxel addresses as part of a logically connected voxel network, even though the information stored for specific voxels may be held on different computational and storage systems. Hence the voxel network is a single logical structure, implemented on potentially distributed processing architectures. The simple example presented does not do justice to the full, and even endless, range of sophistication possible in an agent-based system, but serves to illustrate the principals involved. The sophistication of agents within the overall VoxelNET can vary widely, by position, the types of data, information and knowledge that they can process, and in the complexity and adaptability of the agent cognitive architectures that they use. A more complex pathfinding example could involve agents that can compete to lure or drive away traffic, e.g. by including cost or payment data in addition to pure path data for traversable voxels. Pathfinding can then become an iterative Pareto optimisation process, where agents in potential paths can seek to increase their charges or payments to modify the optimal solution to meet their local goals of participation (e.g. to encourage more use of sparsely used path steps) or avoidance (e.g. to reduce demand and congestion on popular paths). Future work will include further investigation of these and related concepts.

ACKNOWLEDGMENT

We thank CSIRO Mineral Resources and Mining3 for supporting our VoxelNET research. VoxelNET™ is a patented technology.

REFERENCES

[1] http://www.tnl.net/blog/2017/02/11/internet-4-0/ [retrieved: 15 January 2019].

[2] https://en.wikipedia.org/wiki/Industry_4.0 [retrieved: 15 January 2019].

[3] C. Sennersten, A. Davie and C. Lindley, "Voxelnet - An Agent Based System for Spatial Data Analytics", short paper, *Eighth International Conference on Advanced Cognitive Technologies and Applications (COGNITIVE 2016)*, March 20 - 24, Rome, Italy, 2016.

[4] https://medium.com/@sifium/machine-learning-types-of-classification-9497bd4f2e14 [retrieved: 15 January 2019].

[5] https://www.sciencelearn.org.nz/resources/1438-classification-system [retrieved: 15 January 2019].

[6] K. Padian, “Charles Darwin’s Views of Classification in Theory and Practice”, Syst. Biol. 48(2):352-364, 1999.

[7] O. Cardin, “Classification of cyber-physical production systems applications: Proposition of an analysis framework”, Computers in Industry 104 (2019) 11–21, 2018.

[8] OGC: 06-103r4 Part 1 “Simple Feature Access - Part 1: Common Architecture” (https://www.iso.org/standard/40114.html, last accessed on 14 January 2019), 2004, and Part 2 “Simple Feature Access - Part 2: SQL Option” (https://www.iso.org/standard/40115.html, last accessed on 14 January 2019), 2004.

[9] D. Pentecost, C. Sennersten, C. A. Lindley, R. Ollington and B. Kang, "Predictive ACT-R (PACT-R): Using A Physics Engine and Simulation for Physical Prediction in a Cognitive Architecture", *Eighth International Conference on Advanced Cognitive Technologies and Applications (COGNITIVE 2016)*, March 20 - 24, Rome, Italy, 2016.

[10] C. Sennersten, C. Lindley, R. Lyu, A. Grace, B. Evans, D. L. Taylor, A. Davie, L. De Macedo Camargo, J. Craig, A. Hellicar, D. Biggins,

 ISBN: 978-1-61208-705-4

G. Timms, M. Coombe, S. Shariar Md, G. Smith, A. Morshed, A. Rahman, G. N. Salomão, 2015, "Unmanned Aerial Robots for Remotely Operated and Autonomous Surveying in Inaccessible Underground Mine Voids" *Third International Future Mining Conference*, Sydney, Australia, pp. 101-08, 4-6 November 2015.

[11] http://www.engyes.com/en/dic-content/softbot [retrieved: 15 January 2019].

[12] J. P. Müller. "Architectures and applications of intelligent agents: A survey." *Knowl. Eng. Rev.* 13, 4 (February 1999), 353-380. DOI=http://dx.doi.org/10.1017/S0269888998004020), 1999.

[13] D. L. Poole. and A. K. Machworth, "Artificial Intelligence: Foundations of Computational Agents", 2dn edn., CUP, isbn: 9781107195394, 2017.

[14] W. J. Teahan, "Artifical Intelligence –Agent Behaviour", bookboon, ISBN 978-87-7681-559-2, 2014.

[15] Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs", *Numerische Mathematik*. **1**: 269–271. doi:10.1007/BF01386390, 1959.

[16] D. Maturana and S. Scherer, "VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition", Conference Paper, IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 922-928, September, USA, 2015.

[17] Y. Zhou and O. Tuzel, "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection", USA, arXiv:1711.06396v1, 17 November 2017 (submitted).

[18] K. Mehlhorn and P. Sanders, "Chapter 10. Shortest Paths" (PDF). *Algorithms and Data Structures: The Basic Toolbox. Springer.* doi:10.1007/978-3-540-77978-0. ISBN 978-3-540-77977-3*, 2008.*