Self-Adaptive Systems Challenges in Delivering Object-Based Media

Barry Porter, Paul Dean, Nicholas Race
School of Computing and Communications
Lancaster University, UK
Email: {b.f.porter,p.dean1,n.race}@lancaster.ac.uk

Mark Lomas, Rajiv Ramdhany
BBC R&D
UK
Email: {Mark.Lomas01, Rajiv.Ramdhany}@bbc.co.uk

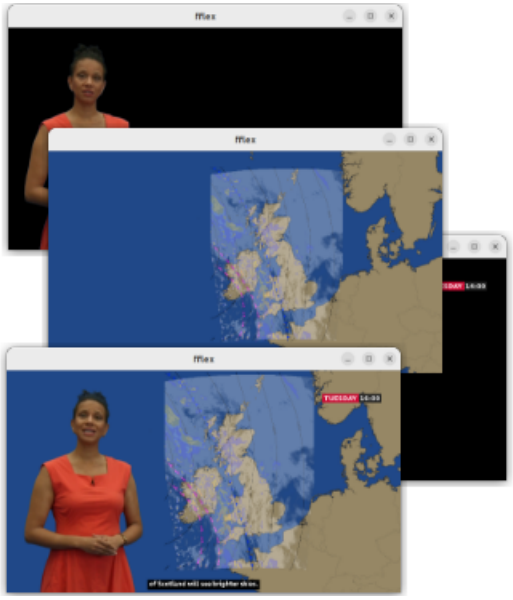
***Abstract*—Multi-agent, decentralised, and collective self-adaptive systems have been studied in a range of domains from smart-cities to the social dynamics of organisations. We present a novel application for research in this area, with the emerging distributed systems field of object-based media. Unlike traditional media, which is delivered over the Internet as pre-encoded compressed video segments, object-based media divides a media experience into its constituent parts, such as presenters, actors, backgrounds, and information overlays. These elements are then rendered and composited on-demand for each user, to support highly customisable experiences. We analyse the distributed systems challenges of delivering object-based media in terms of self-adaptive systems, and present work on the prototype technology we are using to explore potential solutions.**

I. Introduction

Multi-agent, decentralised, and collective self-adaptive systems have been studied in a range of domains, for example in autonomous drone coordination [1], resource sharing [2], vehicle traffic routing [3], or social dynamics of organisations [4]. We present a novel distributed systems challenge of **object-based media delivery** as an application domain for these research fields. This kind of system requires both *collective* and *multi-level decision making* at large scales.

Traditional media experiences are delivered as single-video streams via HTTP-carried segments (e.g. using MPEG DASH or HLS). Clients request a given video and it is delivered from server-side storage, either from disk or from a memory cache, in fixed-size segments. While DASH has a degree of adaptation (changing bitrate/quality depending on bandwidth and latency between the server and client), this works essentially in a point-to-point client/server model where decision-making is trivially negotiated between those two single points and is typically based on network metrics alone.

In contrast to the above, **object-based media** is a novel media production and delivery approach being explored across the media industry [5]. In this approach, each element of a media experience is captured and stored as an independent object. In a weather forecast, for example, the presenter will be captured and stored separately, as will the weather map, subtitles, overlays like meteorological data, and alternative presenters such as signers for hearing-impaired viewers. These different media objects are then rendered into frames in real-time for each user depending on how they configure the experience. An example object-based weather forecast is shown below, with some of its individual media objects and one of the composed render permutations. This example is formed from one video stream (presenter), two dynamically-drawn animation streams (map and time-of-day), and subtitles.



Producing, storing, and delivering media in this way supports an unprecedented level of user-customisation of a media experience. Extending our weather example, we might imagine small configuration changes like turning sign-language presenters or subtitles on and off, or more dramatic changes such as swapping the presenter to a children's TV character, changing the map and meteorological data to a more child-friendly representation, or using real-time rendered digital human presenters [6]. We could also consider blending a weather presentation with augmented reality, allowing a user to point their smartphone camera outside to see how it will look in the weather for the week ahead alongside the general weather presentation. Considering other kinds of media experience, we might imagine drama productions with branching narratives, virtual-reality blended experiences placing viewers and their friends into scenes, or highly configurable sports programming with custom overlays and interpolated camera positions.

While this approach to media production and delivery offers a broad range of new media experience possibilities, it also

requires a fundamentally different approach to the network and compute needed to realise it. Because each media frame in a highly-customisable experience requires real-time rendering, for example, in a simple implementation client devices must be powerful enough to render and present media frames from their constituent parts – potentially including complex 3D rendering. In cases where clients do not have this capability, compute must be offloaded to cloud or edge devices. Offloading is expensive for the media provider, however, and must be made cost-effective and sustainable by *sharing* rendered outputs across clients where possible. This must be achieved at scales of millions of users watching a wide range of media experiences (and their permutations) at the same time.

We are exploring this challenge in collaboration with the BBC, which operates the iPlayer platform providing between 100,000 and 800,000 simultaneous media streams over any given 5-minute window, depending on the time of day. These streams are delivered to a wide range of end-user devices with highly varying hardware capabilities, including mobile phones, traditional computers, games consoles, and smart TVs.

In this paper we introduce this domain as a novel challenge in self-adaptive systems, particularly for *collective decision making* and *multi-level decision making* at large scales and with potentially high system dynamics. The contributions are:

1) To provide a definition of the problem domain, and how this definition might suggest initial solution directions
2) To present our prototype platform, which supports continuous adaptation of the placement of compute, and multiple locales of decision-making control
3) To present a research roadmap towards real-time collaborative decision-making at scale in this domain

In the remainder of this paper we present a problem definition, our current prototype with its decision control points, and our future work plan.

II. Network Delivery of OBM

In this section we define the compute and network delivery challenge of object-based media (OBM), in the context of self-adaptive systems. We begin with a general introduction to OBM, then present our system model, and lastly define the problem space in the context of this model.

A. Object-Based Media and Customisation

We assume media is captured as a set of distinct media objects. These objects could, for example, be a presenter, subtitles, animated map, a camera angle, a 3D mesh with lighting information, or a narrative branch. These media objects may be stored across a range of different media servers, and may include meta-data about how to compose them with other kinds of media objects in a natural-looking way.

Drawing on this set of media objects, there are two broad dimensions of media customisation: temporal and spatial. Temporal customisation includes actions such as the dynamic editing of a programme to a given duration, such as reducing a documentary to half of its total length while retaining the narrative essence, or could include dynamic ordering of a show such as focusing on particular parts of a sports highlights presentation. Spatial customisation occurs when changing the composition of a scene, for example to change the presenter, enable information overlays, change the camera angle to either a physical or interpolated one, or otherwise change objects in the scene that are visible. Both kinds of customisation require a degree of *dynamic rendering* of each frame of a media experience to match the viewer's preferences.

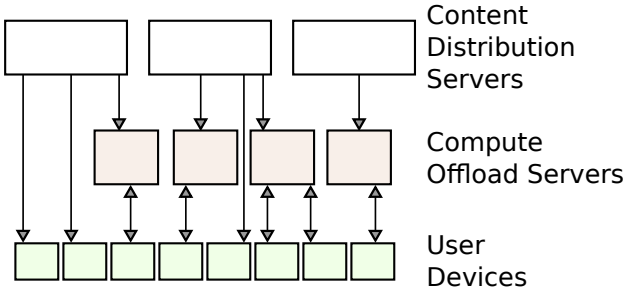


Fig. 1. Compute resources and their interactions. Content distribution servers stream individual media objects to where they are needed (either clients or resource-offload sites). Resource offload sites host rendering tasks for clients.

B. System model

We next present our system model, from the physical resource landscape to user behaviour and degrees of uncertainty.

1) Network and compute infrastructure: We assume we have a number of (i) media source servers, (ii) compute resource servers, and (iii) client devices. Media sources simply allow individual media objects to be streamed by time index, and would typically be realised as a *content distribution network*; compute resource servers allow rendering compute tasks to be migrated onto/off of them; and client devices are where a media experience is actually shown to a viewer.

The computation power of each resource server and client device may be highly variable, and may include CPU-only or CPU+GPU rendering support, varying amounts of memory, and varying levels of network bandwidth and latency. The geographical distribution of compute resource servers may not be harmonious with the set of client devices, such that network latencies between a subset of clients and available compute resources may be prohibitive to allow their use. The network characteristics of individual client devices may vary widely, from mobile devices on 4G connections to high-performance desktop computers with fibre-optic broadband.

We assume that users will only allow any client-device-resident computation to be used for their own media viewing, avoiding sharing their own device's compute with other viewers. This resource architecture is illustrated in Fig. 1.

2) User behaviour: Although OBM is not yet in wide distribution, we can approximate user behaviour models from how the traditional media-consuming population behaves.

The BBC's iPlayer streaming platform has an average of 3,200 programmes (36,000 episodes) available to choose from, with an average run-length of 81.96 minutes (ranging from 5 to 239 minutes). Overall, in any given 5-minute window,

between 2.5TB and 155TB of data is being streamed from BBC content distribution servers, at low and peak times respectively, with a total of between 152,000 and 840,000 streams being active. Most of the content is delivered via DASH-based adaptive bit rate (ABR) streaming, where each client climbs up/down the ABR ladder to suit their bandwidth constraints. On average, bitrates to individual clients vary between 1.3 Mbps and 5.7 Mbps. The infrastructure to support this delivery involves multiple different CDNs with a wide range of locations throughout the UK.

Traditional media delivery is almost entirely network-bound, rather than the additional compute-bound nature of OBM, but we can draw at least some inferences from this data around the volume of content access and potential compute load for OBM. Beyond the above statistics, the additional OBM factors are (1) the compute power of each user/client device is relevant to how much infrastructure support each user needs to access a given experience; and (2) users will have a set of different personalisation preferences for each media experience, which specialise each programme per-view in a large number of different ways.

3) Types of Compute: The types of compute task involved in rendering a media experience are highly variable, from lightweight tasks to far more resource-intensive ones.

Lightweight tasks include compositing partial frame elements into a full frame (where each element is an RGBA texture), or showing simple overlays such as sports statistics.

More intensive tasks include rendering a synthetic human presenter, dynamically relighting an object to appear natural in a scene, or depth-mapping a user's current physical environment for augmented reality experiences. These compute types may be able to render outputs at multiple different quality levels, where lower-quality renders are cheaper to produce and may also be faster to transport to a requesting user.

4) Degrees of Uncertainty: The use of a self-adaptive solution implies that an application domain has one or more degrees of uncertainty. Those present in media delivery are:

- The user set viewing the same experience (set UE).
- How many of those users have selected the same customisation options (set UEC)?
- How many of those users are watching the same time index of the media experience, within a threshold of +/- t seconds (set UEC_t)?
- How many of those users have insufficient local compute power to render some aspect of an experience ($UEC_{t.offload}$)?
- The current network latency & bandwidth between each user and each theoretically viable compute-offload site in the set OS_v of offload sites which *could* be latency-viable for that user in the absence of any network contention/
- What is the likely cache-hit-ratio for cached rendered outputs at each of OS_v relative to a given client within UEC (i.e., which sites have cache frames that are relevant to a given client's intended playback)?
- What is the current compute resource availability at each of OS_v to render the desired aspect of the experience?

The membership of set $UEC_{t.offload}$ is dynamic over the course of a day for each media experience, while the membership of set OSv and their likely cache hit ratios is dynamic per-experience and per-user as a factor of both a single user's behaviour and that of all other users of the same streaming system (where users with the same requirements may be able to re-use rendered outputs via caching).

C. Problem Definition

Drawing on the above system model elements, we can derive an abstract problem description as follows.

1) User Perspective: User devices, based on a user's specific customisation of a media experience, may decide whether or not to attempt to offload associated compute *per-experience-element* to available offload sites. The rendering of a given media experience is composed of various compute types, some of which may gain a higher offload benefit to others (reducing cost to the user's own device to a greater or lesser extent).

Some kinds of offload may present trade-offs where the pressure on one hardware aspect of a user's device is decreased at the cost of increasing pressure on another aspect of that device; for example rendering a 3D mesh on the user's device will use relatively little network to deliver the vertices of the mesh but will require significant compute to render and light the mesh; rendering a 3D mesh into on another device into a pixel stream, by comparison, uses very little local compute but requires far more network transport to encode and deliver the rendered artefact as a set of pixels. Balancing these resource consumption trade-offs is a key challenge.

2) Infrastructure Perspective: The infrastructure provider has a set of users who want to watch a set of experiences. The provider has a set of potential offload locations that they can pay for, but only a subset of these will be active at any time. Because network latency is a factor in quality of experience, it may be that only end-user devices are able to make quality measurements and so may need agency over which offload site(s) they actually use from a set of potential options.

For those users who have a preference to offload compute, the provider will wish to co-locate as many such users as possible on the same offload sites to reduce overall operating cost. To make this viable, the provider will want to be able to *share compute results* between those users as much as possible. For example, if 100 users are watching a weather presentation with the same synthetic presenter, each sequential rendered frame of that presenter may be the same for each user. It would then be beneficial to have the presenter-rendering task for all of those users placed on the same offload site; this gives a 1:99 render:re-use ratio and uses the smallest amount of compute. This only works, however, if that offload site is in the viable set for all users in terms of latency and bandwidth. It also only works if those users are watching a similar-enough time index of the stream (otherwise cached frames will likely have been evicted/replaced by other more-recent cached elements).

The delivery provider's task is therefore to matchmake users and offload sites, while network characteristics are changing, the user cardinality per media experience is changing, and the

local decisions made on each user device may be changing in terms of which render tasks are worth offloading.

3) Centralised to decentralised solutions: A wide range of solutions can be envisaged to tackle the compute-placement problem at scale, on a spectrum ranging from entirely centralised to hierarchical and entirely decentralised.

With perfect knowledge and infinite decision-making time, for example, a single centralised agent could solve the above matchmaking problem. Each time a new media request arrives from a user, the centralised agent would consider all in-progress streams, re-compute the ideal distribution of offloaded compute across available resources, considering latency between clients and each offload site, and apply the new ideal compute landscape across the set of available resources. Such a centralised agent has two problems in practice: knowledge of the system will be partial, and takes time to collect / refresh; and decision-making time must be low to keep users engaged (i.e., on the order of seconds rather than minutes).

A decentralised solution may involve the infrastructure provider offering a set of possible offload sites to each user, and allowing client devices to decide independently which offload site(s) to use at which times. This allows clients to use their own measurements of experience quality, and allows fast decision-making by each individual client considering its own requirements. Such an entirely decentralised solution invites hysteresis and oscillation, however, as each client tries offloading different aspects of rendering to different possible locations, without knowledge of how other clients' similar tests may be impacting the observed effects.

Hybrid solutions, meanwhile, such as hierarchical organisations, may offer a desirable mixture of both of these extremes.

III. Prototype

We have built a prototype OBM delivery platform using the Dana component-oriented language [7] which allows us to hot-swap components seamlessly at runtime. Many of the components of our delivery platform wrap low-level hardware-accelerated rendering libraries developed by BBC R&D.

In this section we describe the core architecture of our prototype, including how adaptation for compute offload is realised, and also discuss the points at which decision-controllers can be introduced, from centralised to decentralised.

A. Core architecture

Our system is divided into its application logic and distribution / adaptation logic. The application element is written as an entirely local system without any knowledge of networking or offload sites. Using component-based adaptation, a meta-level composer system then governs the wiring of this application-level system, dynamically injecting network proxies into the application to offload compute. The composer also interacts with wider distributed infrastructure to gain data on available resources and participate in offload decision-making.

A simplified view of our application architecture is shown in Fig. 2. Within the shaded box this contains a media player, which includes a render loop and a GUI, and a set

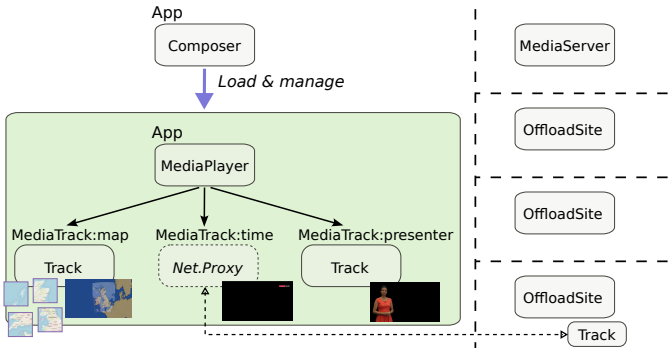


Fig. 2. Our system under the control of a composer, which communicates with remote MediaServer and OffloadSite instances, and determines when to inject network proxies to local media tracks to move their rendering to a remote server. Dashed lines delineate different host machines over a network, and dashed arrows indicate network communication.

of *media track* implementations such as *map* and *presenter*. Each track implementation uses a common `MediaTrack` interface which different track types can implement to render their associated element of a media experience. The presenter track is realised as a DASH video stream, while the map is composed in real-time from a set of individual map tiles which are stitched together and zoomed or panned throughout the presentation according to an animation instruction file, and are overlaid with weather icons at appropriate timings. The time-of-day image is also drawn dynamically as a timed animation using appropriate image assets.

Our media player instantiates a set of media tracks, the details of which are configured by an experience definition file, and uses a rendering loop which iterates through each track. The render loop runs at a fixed framerate (e.g. 60*fps*), and simply requests the current frame to render from each media track for the current playback time. A media track returns a *render command list*, which contains primitive render instructions, such as draw text at a given X/Y position, or draw an image at an X/Y position with suitable transforms. This approach is sufficiently flexible that we can seamlessly substitute a track that delivers mesh-style rendering commands with one that delivers pre-rendered frames of pixels.

Outside our core application business logic, our platform uses a set of management and distributed systems components to support compute offloading, shown outside the shaded area in Fig. 2. A *composer* program wraps the media player application, loading and wiring its components, and can dynamically inject network-proxy versions of each media track.

For network proxying we have an implementation of the `MediaTrack` interface which makes network calls to a remote server, acquiring rendered frames remotely rather than generating them on the local device. These remote frames can either be delivered as primitive render commands (so the offload site is only computing positioning information), or can be delivered as rendered pixels which are encapsulated as render commands. We use our component-based platform to then seamlessly swap a local-host media track implementation, which renders locally using the playback device's hardware,

for an offloaded remote instance of that media track.

We use these building blocks to layer an abstraction of switchable *software compositions*, similar to the emergent software systems concept [8], [9]. In this abstraction, the fully-local version of the media player is one possible software composition. If we have one available offload site, we then generate a proxy for each track with that site as its remote endpoint; each permutation of local and remote proxies then forms another distinct composition of the media player. We can then select one of these labelled options, and the client-side composer will perform the set of adaptations needed to reach that option (including communicating with offload sites to set up appropriate instances and data streams). Each time a new offload site becomes available, or becomes *un*available, this permutation set grows or shrinks accordingly.

When a user wants to watch a given program, the client program requests a new stream from the media server. The media server replies to the client with the experience definition file (indicating which track implementations to instantiate and layer, in which order), and content distribution locations for the case where the client is rendering locally. The composer program separately contacts the media server to acquire a set of offload sites that the client is permitted to use. The composer may occasionally poll the media server during the stream to update its list of available offload sites.

B. Decision control points

Our platform offers a range of decision control logic architectures to decide where to place render-compute logic for each track of each client, from centralised to decentralised.

To implement an entirely centralised approach, the media server would maintain a global view of all active clients and all available offload sites. For a newly-connecting client, the media server would perform a calculation to determine the ideal (or a *good enough*) placement of every track instance of every client including the new one, akin to a bin-packing algorithm. The media server would then inform both the new and existing clients of changes to available offload sites (where each client is given exactly one offload site for each track, or potentially no offload sites for some tracks). Clients may either still choose to use or not use the single available offload site for each track, or may be actively required to use the available offload site for each track, depending on the policy.

To implement an almost entirely *de*centralised approach, the media server would maintain only a list of resource offload sites, potentially with an estimate of the compute load of each one. When a new client connects, the media server would offer that client a list of (e.g.) 3-5 possible offload sites for each media track, potentially based on a simple calculation of geographic network proximity by IP address. The client would then be responsible for trialling the offload of each media track's rendering at each provided offload site to determine what works best for that client.

Hybrid approaches may use a mixture of the above decision control points, potentially sharing different granularities of monitoring data between clients, offload sites, and controllers.

In future work we intend to examine a wide range of different control alternatives, at various scales, to understand which approaches offer the best solutions towards objectives such as operational cost minimisation for infrastructure providers or user experience maximisation for audiences.

IV. Related Work

We have presented a novel application domain in which to explore adaptive decision-making, from centralised to decentralised, and have examined two initial points in this spectrum. To our knowledge there are three main sub-fields of the self-adaptive systems research community which may apply to this domain: multi-agent systems; collective adaptive systems; and decentralised or hierarchical control theory.

In multi-agent systems, such as the multi-agent reinforcement learning approach [10], researchers examine how co-operative behaviours can be realised via a set of individual algorithm instances operating on each agent. There are both formal methods branches of this research, in which aspects such as convergence time on an objective are proven under certain assumptions, and applied methods which may use fewer assumptions and show probabilistic or best-effort results. Examples in this space include P-MARL [11], which uses prediction of environment trajectories to limit the impact of non-stationary environments on learning rate; and the DPOP algorithm [12] which uses a dynamic programming approach to solve the distributed constraint optimisation problem [13].

In collective adaptive systems, or collective learning [14], decentralised decision logic is designed as an emergent property of interactions between humans and software agents. These approaches are designed to scale well to many participants, and to yield a global policy that is fair to its participants while presenting meaningful individual choices over time to each agent [15]. The objective of balancing a global trait of fairness while offering individual choices may be mappable to our application domain, for example where fairness may be the fair use of available resources, and individual per-agent choices may be potential offload locations or quality of a stream (e.g. measured in bitrate).

In decentralised or hierarchical self-adaptation theory, of which there is a recent survey by Quin et al. [16], agents attempt to optimise towards an objective function by sharing selected information using a range of communication overlays which align with the direction of information travel. This can involve decentralizing the monitoring, planning, and/or executing controllers of the system (common design patterns are captured in [17]), and includes a range of approaches to knowledge- and control-sharing such as multi-scale feedback systems [18] and collective management [19].

The above works suggest potential directions to explore, and while there is a good existing coverage of the higher-level design space, specific algorithms for our application domain appear more challenging to identify. In contrast to many of the above works, the particular features of our domain are that (i) the quality of an experience is ultimately measurable only at the client device, though some inference

of *likely* experience may be possible in the cloud; (ii) we have high scalability requirements, at around a million concurrent users; (iii) decision-time for initial placement decisions should be low (on the order of seconds). In addition to this, our application domain is required to operate over the real-world Internet in which delivering messages (such as measurements or control signals) takes unpredictable amounts of time, the network is lossy, and the overall deployment environment is non-stationary. In future work we aim to explore how some of the above approaches may fit into this problem domain, and how they compare with our initial results.

V. Summary and Future Work

We have introduced object-based media as a novel application domain and set of challenges for distributed self-adaptive systems. We have also presented our prototype technology to deliver object-based media, which supports dynamic offloading of pluggable track render implementations.

In future work we intend to examine the applicability of existing theory from the self-* community to solve some of the challenges we have identified, as well as developing novel theory for those areas which present new problems.

We will particularly focus on the distributed multi-agent adaptation problem, in which rapid 'good enough' solutions are found to compute placement, considering large numbers of geographically dispersed users who are watching related or partially-related media experiences lending themselves to compute re-use for resource reduction. This challenge is compounded by the fact that experience quality is measurable only at the viewing client device, and that we may have competing objective functions, from resource minimisation in cloud hosting to experience-quality maximisation for audiences.

The server-side approach to this problem faces issues of rapid yet high-quality decision-making at scale, and non-observability of actual client quality-of-experience. The client-side approach, meanwhile, is open to oscillation effects as many clients trial the same offload site, causing interference with each others' measurements of those sites' performance.

Placement decision-making is also affected by cache availability, particularly as producing cached frames comes at varying cost levels depending on the type of compute needed – from rendering simple 2D animation frames, to rendering suitable lit 3D synthetic presenters. This creates a further constraint set, such that placing client compute at offload sites that have usable cached frames may offer significant performance gains, even if those sites have less desirable network latency. We aim to examine cache-aware placement as a further step once the core placement challenge is addressed.

We will also examine how research in this domain may apply to related areas, particularly to other large-scale decision making challenges in distributed systems with uncertainty (especially those with temporally-connected streaming data).

Acknowledgments

This work was supported by UKRI EPSRC and BBC Prosperity Partnership *AI4ME: Future Personalised Object-Based Media Experiences Delivered at Scale Anywhere*, EP/V038087.

Our current implementation of OBM delivery relies on rendering libraries developed by BBC R&D, particularly in contributions by Mark Lomas, Rajiv Ramdhany, James Shephard, Sebastian Ward, Tim Pearce, and Michael Sparks.

References

[1] J. Boubin, C. Burley, P. Han, B. Li, B. Porter, and C. Stewart, "Marble: Multi-agent reinforcement learning at the edge for digital agriculture," in 2022 IEEE/ACM 7th Symposium on Edge Computing (SEC), 2022, pp. 68–81.

[2] E. Pournaras, S. Jung, S. Yadhunathan, H. Zhang, and X. Fang, "Socio-technical smart grid optimization via decentralized charge control of electric vehicles," Applied Soft Computing, vol. 82, p. 105573, 2019.

[3] I. Gerostathopoulos and E. Pournaras, "Trapped in traffic? a self-adaptive framework for decentralized traffic optimization," in 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2019, pp. 32–38.

[4] C. M. Barnes, A. Ekart, and P. R. Lewis, "Social action in socially situated agents," in 2019 IEEE 13th International Conference on Self-Adaptive and Self-Organizing Systems (SASO). Los Alamitos, CA, USA: IEEE Computer Society, jun 2019, pp. 97–106.

[5] J. Carter, R. Ramdhany, M. Lomas, T. Pearce, J. Shephard, and M. Sparks, "Universal access for object-based media experiences," in Proceedings of the 11th ACM Multimedia Systems Conference, ser. MMSys '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 382–385.

[6] H. O. Demirel and V. G. Duffy, "Applications of digital human modeling in industry," in Digital Human Modeling, V. G. Duffy, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 824–832.

[7] B. Porter and R. R. Filho, "A programming language for sound self-adaptive systems," in 2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS), 2021, pp. 145–150.

[8] R. Rodrigues Filho and B. Porter, "Defining emergent software using continuous self-assembly, perception, and learning," Transactions on Autonomous and Adaptive Systems, vol. 12, no. 3, pp. 1–25, September 2017.

[9] B. Porter and R. Rodrigues Filho, "Distributed emergent software: Assembling, perceiving and learning systems at scale," in 2019 IEEE 13th International Conference on Self-Adaptive and Self-Organizing Systems (SASO), 2019, pp. 127–136.

[10] R. S. Sutton and A. G. Barto, Introduction to Reinforcement Learning. MIT Press, 1998.

[11] A. Marinescu, I. Dusparic, and S. Clarke, "Prediction-based multi-agent reinforcement learning in inherently non-stationary environments," ACM Trans. Auton. Adapt. Syst., vol. 12, no. 2, may 2017.

[12] A. Petcu and B. Faltings, "A scalable method for multiagent constraint optimization," ser. IJCAI'05. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005, p. 266–271.

[13] F. Fioretto and W. Yeoh, "Distributed constraint optimization problems and applications: A survey," Journal of Artificial Intelligence Research, vol. 61, 02 2016.

[14] T. N. Garavan and R. Carbery, Collective Learning. Boston, MA: Springer US, 2012, pp. 646–649.

[15] E. Pournaras, P. Pilgerstorfer, and T. Asikis, "Decentralized collective learning for self-managed sharing economies," ACM Trans. Auton. Adapt. Syst., vol. 13, no. 2, nov 2018.

[16] F. Quin, D. Weyns, and O. Gheibi, "Decentralized self-adaptive systems: A mapping study," in 2021 IEEE International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2021, pp. 18–29.

[17] D. Weyns, B. Schmerl, V. Grassi, S. Malek, R. Mirandola, C. Prehofer, J. Wuttke, J. Andersson, H. Giese, and K. M. Göschka, On Patterns for Decentralized Control in Self-Adaptive Systems. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 76–107.

[18] P. Mellodge, A. Diaconescu, and L. J. Di Felice, "Timing configurations affect the macro-properties of multi-scale feedback systems," in 2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS), 2021, pp. 100–109.

[19] T. J. Glazier, D. Garlan, and B. Schmerl, "Case study of an automated approach to managing collections of autonomic systems," in Proceedings of the 2020 IEEE Conference on Autonomic Computing and Self-organizing Systems (ACSOS), Washington, D.C., 19-23 August 2020.