OCbotics: An Organic Computing Approach to Collaborative Robotic Swarms

Sebastian von Mammen, Sven Tomforde, Jörg Hähner,
Patrick Lehner, Lukas Förschner, Andreas Hiemer, Mirela Nicola, Patrick Blickling
Organic Computing
University of Augsburg
Eichleitner Str. 30
86159 Augsburg, Germany
{sebastian.von.mammen, sven.tomforde, joerg.haehner}@informatik.uni-augsburg.de
{patrick.lehner, lf_foerschner, andreas.hiemer}@gmx.de
mirela_nikola@yahoo.com, patrick.blickling@hotmail.de

***Abstract*—In this paper we present an approach to designing swarms of autonomous, adaptive robots. An observer/controller framework that has been developed as part of the Organic Computing initiative provides the architectural foundation for the individuals' adaptivity. Relying on an extended Learning Classifier System (XCS) in combination with adequate simulation techniques, it empowers the individuals to improve their collaborative performance and to adapt to changing goals and changing conditions. We elaborate on the conceptual details, and we provide first results addressing different aspects of our multi-layered approach. Not only for the sake of generalisability, but also because of its enormous transformative potential, we stage our research design in the domain of quad-copter swarms that organise to collaboratively fulfil spatial tasks such as maintenance of building facades. Our elaborations detail the architectural concept, provide examples of individual self-optimisation as well as of the optimisation of collaborative efforts, and we show how the user can control the swarm at multiple levels of abstraction. We conclude with a summary of our approach and an outlook on possible future steps.**

I. Introduction

In order to benefit from an ever more complex technical environment, its behavioural autonomy needs to increase appropriately as well. Only then, it may serve its users without requiring overwhelming amounts of attention. At the same time, a technical system is expected to offer appropriate access for controlling its individual components as well as its global goals. The control of robotic swarms lends itself well to elucidate this challenge: Ideally, the user would communicate his goals to the swarm as a whole, without the need of micromanaging the individuals' every parameter and interaction. For instance, the user might navigate a flock of flight-enabled robotic units towards a building and make them work on facade maintenance, e.g. scrapping off paint, cleaning windows, or trimming greenery. For this to work, a line of command has to be established that links several levels of the system's architecture—the user needs to communicate target and task to the swarm and the swarm individuals communicate to coordinate their efforts. In addition, each swarm individual needs to learn how it can contribute to the newly posed, global goals, and how it can maximise its contribution.

The field of Organic Computing (OC) aims at translating well-evolved principles of biological systems to engineering complex system design [1]. It provides the theoretical underpinnings to quantitatively capture system attributes such as their autonomy and robustness, or processes of emergence based on measures of entropy. It also promotes complex system design by means of a universal, observer/controller-based architecture for adaptive, self-organising behaviour. With respect to robotics, OC research initially focussed on failure tolerant and robust hardware architectures, mainly applied to multi-legged walking machines. The most prominent example is the *Organic Robot Control Architecture* (ORCA) [2], [3]. In ORCA, two kinds of behavioural modules are discerned. *Basic Control Units* (BCUs) implement the core behaviour of the robot, rendering it fully functional with respect to the range of possible tasks. In addition, *Organic Control Units* (OCUs) observe and modify the BCUs' configuration during runtime. The separation between a system's basic and its extended functionality has proven itself numerous times—the sympathetic and the parasympathetic division of the human autonomous nervous system may serve as a famous biological example.

Similarly to ORCA, we follow an OC approach to self-organising robotic systems. In our approach, each agent in a robotic swarm implements a multi-layered observer/controller (O/C) architecture that allows for local, and in unison, also in global optimisation of the swarm agents' behaviours. The user interface is explicitly included as one layer which accepts modifications of the swarm's and the individual agents' goals. We present the details of the multi-layered O/C architecture of a single swarm robot individual and we explain how it works in organising ensembles (Section II). In Section III, we give an example of the reactive, self-regulatory capacity of the architecture. Section IV highlights the longer-term evolution of collaborative behaviour, and Section V demonstrates the workings of the user interfacing layer of the architecture. We provide links to related works in the respective sections, and we conclude with a brief summary and an outlook on future work.

II. The OCbotics Approach

As mentioned in the introduction, our approach relies on an architectural setup similar to ORCA. Therefore, we first reinforce the link between our approach and ORCA and related

works. Next, we build on these analogues to preceding works to detail our approach—from the perspective of a generic architecture as well as of its concrete implementation.

A. From Single Adaptive Units to Teams

In ORCA, the Organic Control Units change the system under observation and control (SuOC) based on periodically issued *health signals*, i.e. messages from the Basic Control Units indicating their functional working state. In contrast, our approach observes all kinds of available data about the SuOC. An according *observation model* specifies exactly, which input data, configuration parameters, or internally computed results of the SuOC are passed on to the observer/controller layer. ORCA's restrictive policy of data retrieval matches its fairly conservative array of options for changing the system. Few choices, however, drastically limit the system's configuration space and thus promote ORCA's primary design goals of (a) unearthing an optimal learning guidelines for adaptation ("the law of adaptation"), and of (b) protecting acquired knowledge against corruption and maintaining its validity and consistency [4].

The ORCA approach is further limited to single, isolated robots—information exchange with other robots or collaborative efforts among robotic teams were not envisaged in the original architecture. Yet, it has been shown that Observer/Controller-driven robots can increase their learning speed imitating each other [5]. Local communication between robots allows for establishing real teams that collaboratively perform tasks such as the exploration of unknown terrain, and that assign each other subtasks in a fair manner—decentralised, without the need for global control [6], [7].

B. O/C Architecture

As suggested above, the OCbotics approach is founded in a multi-level observer/controller architecture. An according diagram is presented in Figure 1. It shows four interwoven architectural levels. Level 0 denotes the system under observation and control, the base of the architecture located at the bottom of the figure. Immediately above, level 1 retrieves and evaluates data about the SuOC's performance. Based on this data, it changes the SuOC's configuration in order to optimise its performance, to adapt it to varying conditions and needs. In particular, the SuOC's parameters/behaviours are optimised that may result in both good and bad performance values with respect to a predefined goal (introduced by level 3). As a consequence, the best possible configuration set, or behaviour, known to level 1 is exhibited by level 0 at any given situation. True innovation is realised by level 2, one step above in the multi-level architecture. Here, completely new behavioural options are generated, simulated and optimised in a sand-boxed simulation environment. Only if the new model specifications satisfy all safety constraints considered as part of the simulation process, they are eventually fed into level 1.

C. Modified XCS

Several studies in Organic Computing have emphasised the adequacy of Learning Classifier Systems (LCS) as a comprehensive framework to support adaptive observer/controller architectures, see for example [8], [9]. Already the first LCS

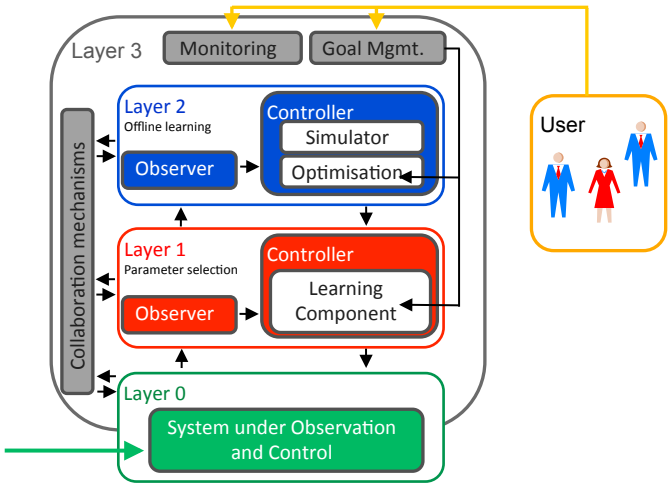

Fig. 1. Multi-level Observer/Controller Architecture. The System under Observation and Control (SuOC) at the bottom is observed and adjusted by the O/C layers above. Their responsibilities are (in this order): reinforcement of existing behaviour, innovating new behaviour, and interfacing local behaviour with (a) user-set targets and (b) cooperative units' goals.

presented in 1975 combined basic rule-based reactive behaviour with an evolutionary component to evolve and improve the rule base [10]. With the introduction of accuracy-based reinforcement of classifiers, LCS research reached an important milestone in 1995 [11] (for an overview of LCS research and ongoing research topics see, for instance, [12]). In the context of safety-critical Organic Computing applications, the latter extension to LCS, also referred to as XCS, was further modified to suit the multi-layer O/C architecture outlined above. In particular, three modifications were implemented: (1) The use of continuous value ranges as promoted in [13], (2) the generalisation of the closest fitting existing rule in layer 1 instead of the generation of a new rule, in case that a given situation is not covered by the existing rule set ("widening" covering mechanism), and (3) "sandboxed" offline learning in layer 2 to ensure safety and maturity of new rules/behaviours. In addition, one can track the impact of those triggered rules effecting changes identical to the newly generated rule and, thereby, building up trust in new rules before they are considered by themselves.

In the remainder of this paper, we show preliminary examples of the OCbotics approach each of which works at a different level of the presented architecture. In particular, we show an example of reactive behaviour of a particular system under observation and control (layer 0) and we elaborate on its integration with layer 1 (Section III). An instance of evolved behaviour (layer 2) in a collaborative swarm robotics setting is presented in Section IV. Its communication across a swarm of agents as well as the interface mechanism with the user of the system, i.e. layer 3, is explained in Section V.

III. Self-organised Aerial Robotic Construction

Tensile structures play an important role in post-modern architecture [14] and they promise to become increasingly important still considering their unique versatility and flexibility in combination with advances in technologies in built material and construction methods [15]. They have also been subject

to Aerial Robotic Construction (ARC) research [16], [17] due to their light mass, load-carrying ability, and their ability of connecting large distances. Quadrotors have been identified as vehicles apt for aerial manipulation mainly due to their robust flight behaviour and their hovering capability [18]. In [17], prototypic building primitives such as single and multi-round turn hitches, knob and elbow nots as well the trajectories resulting from their concatenation have been discussed. Different from pre-calculating trajectories, we have been working on a self-organising approach to building tensile structures. We detail our approach below, followed by elaborations on its OCbotic-specific features.

A. Stigmergic Web-weaving

Typically, a spider weaves its web by itself [19], [20]. Complex web constructions, however, may require collaborative entanglement and tightening of ropes. This can, for instance, be achieved by synchronised flight through pre-calculated control points to cross the ARC quad-rotors' trajectories. Alternatively, the swarm individuals may coordinate themselves relying on local stimuli, like social insects do [21], [22]. In this section, we present a first such locally motivated ARC experiment[1]. Currently, it involves only one quad-rotor that tightens a rope around a tent pole's four suspension lines, see Figure 2.

1) Technical Setup: For our lab-experiments, we currently employ the AR.Drone Parrot 2.0 quad-rotor system. It is connected to a standard PC via WLAN. The PC retrieves the sensory data of the quad-rotor and issues the according navigational instructions. We make use of the quad-rotor's VGA camera that has a 90° field of vision, built-in image-processing capabilities such as marker recognition, and the estimates of its ultrasonic distance sensor. As this sensor and a downward directed camera are used by the quad-rotor to stabilise its flight, we attached a coil at the top of the vehicle and unwind the cord through an eye at its back. We interface with the quad-rotor relying on Nodecoper.js and the node-ar-drone module [23].

2) Behavioural Definition: The quad-rotor behaves only based on locally available sensory information. In particular, it implements the reactive behaviour schematically summarised in Figure 3: After taking off, it searches for a orange-green-orange marker, which is one of the designs that the vehicle is programmed to recognise automatically. It keeps spinning right until it eventually finds one. If the distance to the marker is less than a certain threshold (1m worked quite well), it drifts left. As a consequence, the detected marker moves outside of its field of view. At this point, the quad-rotor has surpassed the previous marker and looks for the next one, which is attached to the next suspension line (also consider Figure 2). The distance to the next marker along the circumference of the pole is greater than the given threshold. The quad-rotor can go straight ahead, if the tag is within the right-hand side of its view (this condition is labelled 'tag in area' in Figure 3). Otherwise, it needs to shift a bit to the left.

B. Adapting Reactive Behaviour

Programmatically, the quad-rotor's behaviour (Figure 3) is represented as a set of simple *if-then* rules. As such, they can be easily subjected to standard LCS implementations and its extensions such as XCS (Section II-C). Hereby, those rules with the best prediction accuracy are reinforced to gain the

[1]Please find an accompanying video at http://youtu.be/tZNeL-n1dDE.

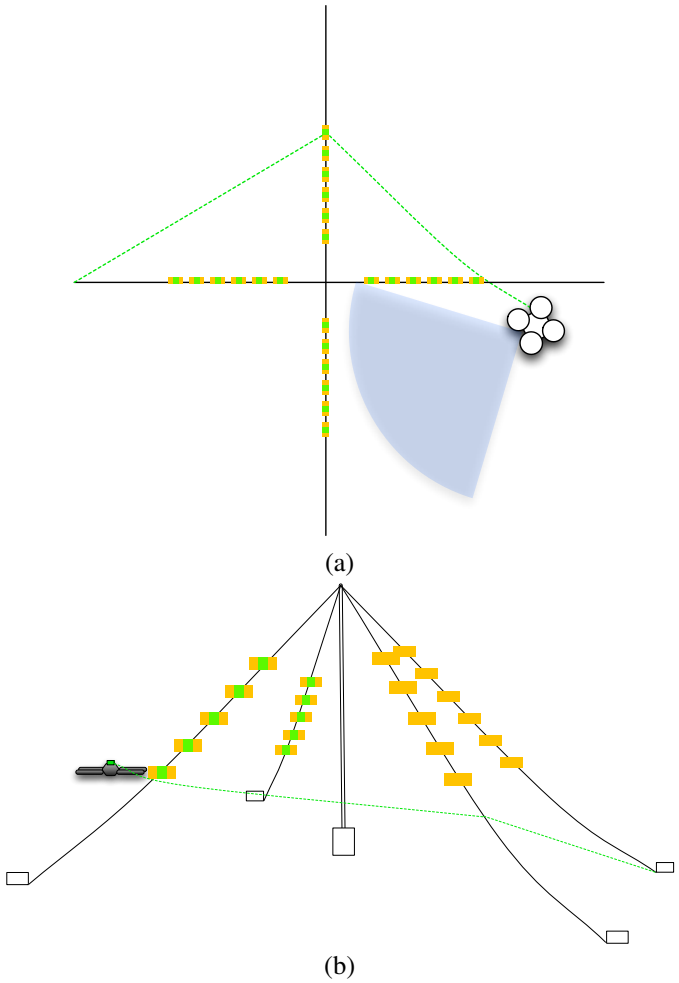

Fig. 2. (a) The quad-rotor hovers clock-wise around a pole that is suspended by four lines. It tightens a rope (green, dashed) along the suspension lines. (b) A schematic side-view extracted from a photograph, highlighting the orientation of the markers pinned to the suspension lines.

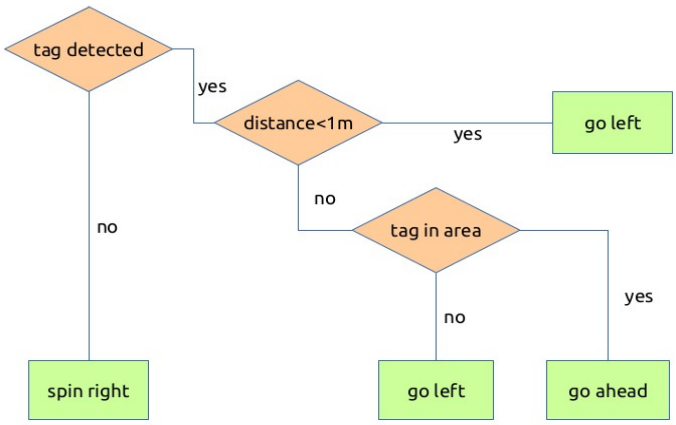

Fig. 3. Weaving Behaviour of a Self-organising ARC Quad-rotor. It circles clock-wise around a pole, tightening its thread around suspension lines tagged with directionally oriented markers.

greatest fitness over time, yielding the best possible behaviour. In this way, the quad-rotor of the ARC example would learn to query the proper sensors at the right times to react in the best possible way, if the behavioural rule set was enriched with according alternatives. At the interface of level 0 (the SuOC) and level 1 (the reinforcement learner), the measure of success can typically be calculated based on locally available information such as the distance flown or the number of recognised tags. For good learning results, the parametrisation of the behaviour should be realised at a rather high level, focusing on the selection of queries and operations and only cover small ranges of variability. Potential benefits of level 1 learning would not only be optimisation of one particular learning pattern but also behavioural rules that adapt to hardware particularities such as deviating sensory intake or imbalanced motor control.

IV. Collaborative Aerial Robotic Maintenance

At level 2 of the multi-layer O/C architecture, behaviours can be created by means of generative model building approaches such as evolutionary algorithms and be optimised for deployment by means of simulations. As a first OCbotics prototype of offline level 2 generation and optimisation, we have evolved quad-rotor behaviour for collaborative surface maintenance. In this section, we introduce the challenge of optimising collaborative surface maintenance. We detail the technical setup we relied on for both simulation and optimisation and we describe the behavioural options of each swarm individual. Afterwards, we draw a very rough picture of the evolutionary experiments that we have run, and we discuss the interactions between layer 2 and 3 for propagating successfully bred behaviours that require synchronisation between the individuals in an OCbotics swarm.

A. Collaborative Facade Maintenance

Consider the facades of large office buildings as examples of vertical surfaces: They are subject to cleaning [24], [25], trimming greenery [26], and other maintenance tasks. As in the previous example, these tasks might benefit more from collaborative efforts than only in terms of efficiency. For instance, fast growing greenery might require one machine to bend, the other one to cut a branch. Equally, during cleaning, several hovering robots might have to join to build up sufficient pressure to remove persistent dirt. Of course, the respective operations might also be split into several procedures performed by individually optimised machines. In this example, however, we only consider the most modest objective, namely collaborative efficiency.

1) Technical Setup: The technical setup of our level 2 experiment comprises (a) a simulation environment to calculate aviation and robotic mechanics, and (b) a machine learning environment with a generative model component and an optimisation component. Figure 4 depicts the software modules that we have used in order to simulate collaborative quadcopter swarms. The Robot Operating System (ROS) acts as a hub for these modules. It provides a high-level software interface for programming and communicating with different kinds of robots [27]. Gazebo is a simulation engine that natively integrates with ROS, offering 3D rendering, robot-specific functionality and physics calculations [28]. Thanks to a ROS driver for the AR.Drone Parrot quad-rotor [29],

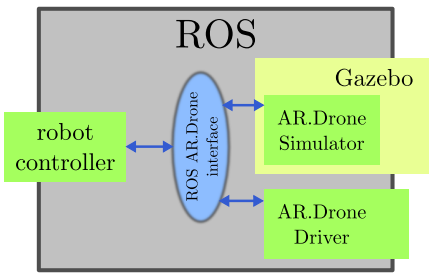

Fig. 4. Interwoven Simulation Modules. The Robot Operating System integrates various software components to simulate quad-rotor swarms. In particular, we control the robot using a common ROS-based instruction set that is understood by both the Gazebo simulation software and its quad-rotor simulation plugin as well as a ROS driver that steers the actual quad-rotor hardware.

and thanks to a Gazebo plugin that simulates the quad-rotor's behaviour based on the very same ROS-based instruction set [30], any of the generated behaviours immediately work in-silico and in-vivo. For the generation of novel behaviours as well as for their evolution, we decided to use the Evolving Objects framework (EO) [31]. EO is an open framework for evolutionary computation featuring an extensible, object-oriented architecture, and turnkey implementations of genetic algorithms, particle swarm optimisation, and genetic programming.

2) Behavioural Definition: Our approach to collaborative facade maintenance is inspired by nest construction of social insects [21]. Each individual works on a small part of the construction proportional in size to the insects' physique. Accordingly, each simulated quad-rotor divides the target surface in a grid, each cell measuring 2 by 2 metres, its field of view covering six cells, two rows of three (Figure 5). This partitioning scheme is a result of the size of the quad-rotor itself and its perceived area from a vantage point close to the surface. Without loss of generality, a dirtiness value is assigned to each cell that indicates whether it needs to be worked on or not. The quad-rotor's internal state, i.e. its remaining battery life, as well as the configuration of dirty and clean cells that reveals itself in front of it trigger specific actions. The quad-rotor may return to the base station to recharge. It may fly to one of the cells in its field of view and clean it. Alternatively, it may move to one of the four neighbouring vantage points to inspect the respective neighbouring batches of cells.

B. Evolving Collaborative Behaviour

Figure 6 captures the behavioural options of a quad-rotor in the context of facade maintenance. Any activity is initiated by the decision-making component, subsequent events guide the quad-rotor back into the decision-making process. Again, the behaviours can easily be written as *if-then* rules which ensures the coherence and simplicity of interfacing across the layers of the O/C architecture. Notice that in this model, quad-rotors cannot stop working. Instead, the whole simulation is terminated after a given amount of time. During this period of time, the decision-making component determines the success of the simulated swarm. We generate an according program tree using Genetic Programming [32]. In this paper, we refrain from presenting the evolutionary approach in all detail, but

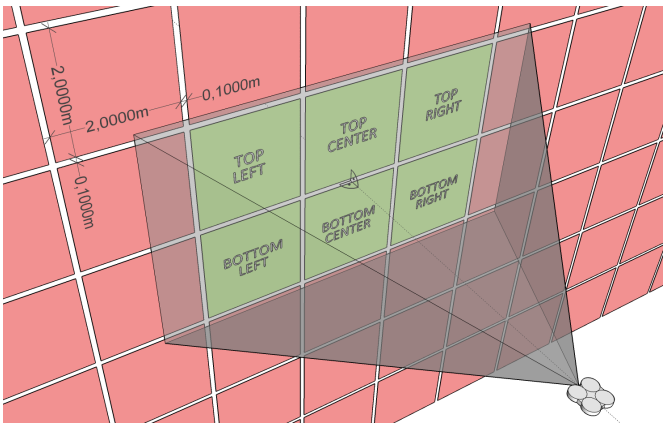

Fig. 5. Cell Grid for Surface Maintenance. The quad-rotor divides the building facade in a grid of cells. The individual cells represent the immediate target areas to work on. Their states of cleanliness also provide the local cues for decision-making, i.e. for approaching an individual cell or to moving to another vantage point.

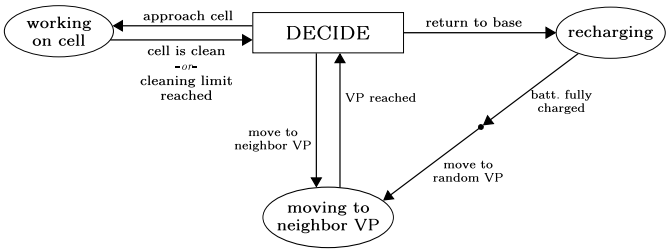

Fig. 6. Options of Activity of a Facade Maintaining Quad-rotor Agent. Any activity—working on one of the cells ahead, flying to the base station to recharge, or moving to a neighbouring vantage point—is initiated by the decision component which considers the agent's battery state and the surface configuration in its field of view.

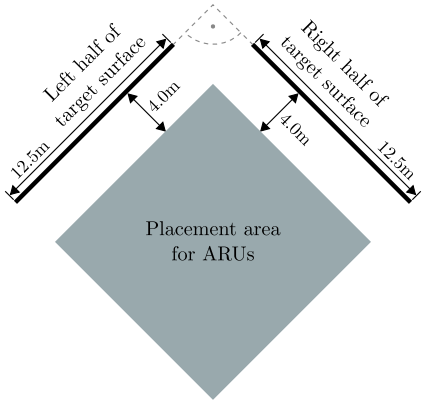

Fig. 7. Simulated Environment for Collaborative Surface Maintenance Evaluation. Two flat surfaces are presented to the quad-rotor swarms as target area which needs to be cleaned. The base stations of the swarm individuals are randomly placed in the rectangular area between the surfaces.

we want to convey its basic mechanisms and how it ties into the OCbotics approach. The generated decision program may conditionally deploy the operations outlined in Figure 6, and introduce references or primitive values as their parameters. The resultant behaviour trees are considered the individuals in an evolutionary optimisation cycle, and thus their fitnesses (penalty, a.o., for remaining amount of dirt) are calculated in according simulation runs, and they serve as an important criterion for selecting ancestors for subsequent generations of individuals (deterministic tournaments).

We ran experiments featuring two or four quad-rotors, or "aerial robotic units" (ARUs), working in parallel for 900 to 1500 simulated seconds. Their individual base stations were distributed randomly in rectangular area sharing two sides with the target surfaces as seen in Figure 7. Population sizes varied from 30 to 100 individuals, the generational cycle was repeated between 10 to 50 times—depending on the work load of an individual simulation which was mainly determined by the number of interacting agents and the simulated time. One of the best individuals in an experiment that started from a set of previously evolved specimen worked as follows: Having arrived at a random cell of the target surface, work through single rows of vantage points from right to left. If the border of the target surface is reached, return to the base station and approach the target area again. It turns out that this behaviour proved significantly faster than two decision programs we manually designed before running the evolutionary process: One of them stochastically selecting dirty cells and considering the remainder of the battery before taking action (low batteries are also penalised by the fitness calculation), the other one letting the quad-rotor follow the dirt gradient exhibited in the perceived 3 by 2 cell matrix.

C. Sandboxed by Layer 2, Letterboxed by Level 3

Level 2 is capable of generating and evolving collaborative behaviour such as the one described above. Initially, the novel behaviour does not have any impact on the system under observation and control. One may say the innovation process is encapsulated in a sandbox and runs completely separated process, offline. At the same time, collaborative behaviour needs to be communicated, if it is required to be performed by all individuals of a swarm in order to function in a coordinated way. The observation of Layer 2 by Layer 3 has to detect such impending necessary changes and broadcast it to all the other members of the swarm. Similar to an auction in multi-agent systems [33], the best broadcast solution, i.e. decision program and fitness value, would be implemented. As an extension, any population-based simulation and optimisation approaches could be distributed among the OCbotics individuals and their evolution be concerted across the whole swarm (for distributed population-based optimisation see, for instance, [34] and [35]). Especially in situations with imbalanced computational loads across the swarm, following a smart distributed optimisation strategy could yield an important advantage.

V. User-guided System Behaviour

In our last example, we demonstrate an early prototype of the user interfacing component of layer 3 of the multi-layered O/C architecture that drives the OCbotics approach. As hinted at in Figure 1, layer 3 mediates the user's goals vertically to all system layers below and horizontally to all OCbotics individuals of the system.

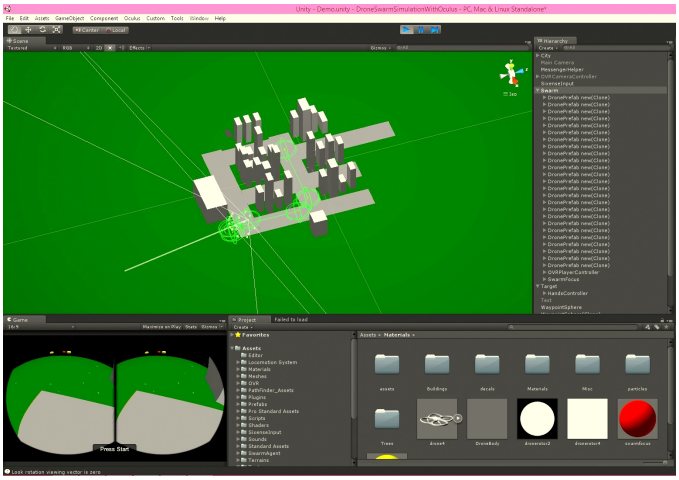

Fig. 8. OCbotics Swarm Modelled in Unity3D. The Unity3D environment allows us to integrate complex simulation models and immersive user interaction hardware such as motion-based input controllers and head-mounted displays.

A. Immersive Swarm Control

The preceding examples of web-weaving quad-rotors in Section III and collaborative surface maintenance swarms in Section IV implement spatial operations. To some extent, both culminate in the tandem of local cues and resultant trajectories. As a consequence, defining spatial targets for arbitrary subsets of a swarm deems to be an adequate task generalisation for a first prototype of a level 3 user interface. A "human-in-the-loop" system design forces one to clearly define the level of influence a user may exercise versus the level of autonomy the system may keep [36]. Therefore, we elaborate on the different levels of access implemented by our prototype right after we outline its technical foundation.

1) Technical Setup: Focussing on interactivity, we decided to utilise the turnkey infrastructure of one of the comprehensive game and simulation engines. In particular, we decided to use Unity [37] as it provides a very shallow learning curve (compared to its competitors) while still providing a powerful coding infrastructure that allows to write custom plugins in C# and which offers a wide range of third-party plugins in a dedicated asset store. Aiming at the implementation of a high-level interface, we tapped into these resources as much as possible and bought, for instance, commercial code bases for simulating flocking behaviours [38] and automated path finding in three-dimensional environments [39]. We further built on Unity demos and plugins that support current hardware solutions such as the Oculus Rift head-mounted display [40] and the Razer Hydra motion controller [41]. In combination, these hardware solutions allow us to emulate an augmented reality scenario for controlling an OCbotics swarm. Figure 8 shows the model of an OCbotics swarm being setup in Unity3D. The light green circles depict waypoints computed by the path finding algorithm, the dual-view perspective at the bottom-left corner of the screen indicates the current view of the attached head-mounted display. The bottom-right window displays the library of components used for modelling the scene, the list at the right-hand side of the screen shows the components that already constitute the scene.

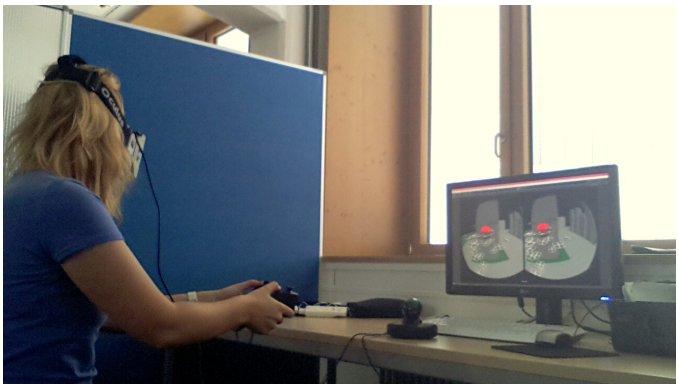

Fig. 9. Simulated Immersive Swarm Control. One of the authors is immersed into an OCbotics simulation. She navigates through movement of her head and using a continuous joystick of the two motion-controllers. The pair of controllers empowers her to (literally) draw new spatial relations between the simulated objects, e.g. to set new targets for subsets of the swarm.

2) Behavioural Control: Our user interface prototype immerses the user into a virtual reality shared with the OCbotics swarm. In the long run, the simulated swarm is meant to make way for a real one, and the virtual reality for an augmented reality. Already, the user can observe the whole swarm or a subset tracking it with a virtual camera that follows in a distance and which aims at the centre of the set of selected individuals. The user can exercise control on any subset of the swarm, hence he may direct flocks of individuals or single individuals at a time. The interface provides all kinds of state information about the selected individuals, such as (averaged and variance of) remaining battery life, current target, current trajectory, and currently perceived neighbours. The user may switch between individuals and greater subsets of the swarm by simply selecting them. Next, he may change the target of flight or even individual control points along the way. Of course, he may also change the parameters of the selected individuals such as their urge for alignment. In our prototype, the user is immersed into the scene of the simulated swarm (see Figure 9) so he can easily trace its activity, understand its relationship to the current target and to obstacles, and to rectify it, whenever necessary.

B. Semi-automated Control between Exploration and Exploitation

The presented simulated prototype for immersive swarm control shows how high-level goals such as setting a new target of the swarm can be communicated in an intuitive way. Differentiated selection of swarm individuals as well as setting local attributes, such as local targets or local waypoints, are simple yet clear examples of moving from abstract, high-level goal descriptions (target/swarm) to specific low-level commands (trajectory waypoints/individual). For a swarm and and individual to reach the specified targets or waypoints, complex calculations have to be performed. In the given example, the need to avoid obstacles and to find optimal paths as well as the coordination among swarm individuals on their way are outsourced to third-party plugins [39], [38]. In the general case, also considering other tasks communicated on layer 3, the necessary behaviours could evolve in sandboxed simulations (layer 2) and be optimised based on local perfor-

mance feedback (layer 1).

VI. Conclusion

In this paper we have introduced OCbotics as a comprehensive approach to designing swarm-based, self-organising robotic systems. OCbotics is driven by a multi-layered observer/controller architecture that allows to optimise and adapt an adaptable system. Adaptation is required in order to maintain or increase the performance exhibited by the system under observation and control—either by optimising or extending existing behaviours, or by innovating, i.e. generating, simulating, and optimising novel behaviours. The performance, in turn, is measured in terms of user-defined goals which may also change over time. We have presented three different projects that operate at different levels of the discussed architecture: Web-weaving quad-rotors with an emphasis on optimised local reactive behaviour, evolution of collaborative behaviour to efficiently work on surfaces, and an immersive user-interface for setting and changing user-defined goals. While the three examples slightly vary regarding their applications, they are connected through the common themes of self-organisation, rule-based behaviour, and adaptation, and of course, the O/C architecture to host them all. With the pieces of the puzzle at hand, the next obvious step is to put them into place, to forge the software components into one (if heterogeneous) code base, to connect the layers of the architecture, to develop a repertoire of recombinable goal definitions, and to transfer the partially still virtual implementations of all levels onto an actual OCbotics infrastructure.

References

[1] C. Müller-Schloer, H. Schmeck, and T. Ungerer, Eds., *Organic Computing - A Paradigm Shift for Complex Systems*, ser. Autonomic Systems. Birkhäuser Verlag, 2011.

[2] W. Brockmann, E. Maehle, and F. Mösch, "Organic fault-tolerant control architecture for robotic applications," in *IARP/IEEE-RAS/EURON Workshop on Dependable Robots in Human Environments*, 2005.

[3] F. Mösch, M. Litza, A. E. S. Auf, E. Maehle, K.-E. Großpietsch, and W. Brockmann, "Orca - towards an organic robotic control architecture," in *Proc. of First International Workshop on Self-Organizing Systems, IWSOS/EuroNGI*, 2006, pp. 251–253.

[4] W. Brockmann, N. Rosemann, and E. Maehle, "A Framework for Controlled Self-optimisation in Modular System Architectures," in *Organic Computing - A Paradigm Shift for Complex Systems*. Birkhäuser Verlag, 2011, pp. 281 – 294.

[5] A. Jungmann, B. Kleinjohann, and W. Richert, "Increasing learning speed by imitation in multi-robot societies," in *Organic Computing - A Paradigm Shift for Complex Systems*. Birkhäuser Verlag, 2011, pp. 295–307.

[6] B. Kempkes and F. Meyer auf der Heide, "Local, self-organizing strategies for robotic formation problems," in *ALGOSENSORS*, ser. Lecture Notes in Computer Science, vol. 7111. Springer, 2011, pp. 4–12.

[7] P. Brandes, B. Degener, B. Kempkes, and F. Meyer auf der Heide, "Energy-efficient strategies for building short chains of mobile robots locally," in *SIROCCO '11: Proc. of the 18th International Colloquium on Structural Information and Communication Complexity*, 2011, pp. 138–149.

[8] U. M. Richter, "Controlled self-organisation using learning classifier systems," Ph.D. dissertation, Universität Karlsruhe (TH), Fakultät für Wirtschaftswissenschaften, Karlsruhe, DE, July 2009.

[9] S. Tomforde, H. Prothmann, J. Branke, J. Hähner, M. Mnif, C. Müller-Schloer, U. Richter, and H. Schmeck, "Observation and Control of Organic Systems," in *Organic Computing - A Paradigm Shift for Complex Systems*. Birkhäuser Verlag, 2011, pp. 325 – 338.

[10] J. H. Holland, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[11] S. W. Wilson, "Classifier fitness based on accuracy," *Evolutionary Computation*, vol. 3, no. 2, pp. 149–175, 1995.

[12] R. J. Urbanowicz and J. H. Moore, "Learning classifier systems: a complete introduction, review, and roadmap," *Journal of Artificial Evolution and Applications*, vol. 2009, p. 1, 2009.

[13] S. Wilson, "Get Real! XCS with Continuous-Valued Inputs," in *Learning Classifier Systems*, ser. Lecture Notes in Computer Science, P. Lanzi, W. Stolzmann, and S. Wilson, Eds. Springer Berlin / Heidelberg, 2000, vol. 1813, pp. 209–219.

[14] W. Lewis, *Tension Structures: Form and Behaviour*. Thomas Telford, 2003.

[15] H.-J. Schock, *Soft shells: design and technology of tensile architecture*. Birkhäuser, 1997.

[16] J. Willmann, F. Augugliaro, T. Cadalbert, R. D'Andrea, F. Gramazio, and M. Kohler, "Aerial robotic construction towards a new field of architectural research," *International journal of architectural computing*, vol. 10, no. 3, pp. 439–460, 2012.

[17] F. Augugliaro, A. Mirjan, F. Gramazio, M. Kohler, and R. D'Andrea, "Building tensile structures with flying machines," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 3487–3492.

[18] R. Mahony, V. Kumar, and P. Corke, "Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor," *Robotics & Automation Magazine, IEEE*, vol. 19, no. 3, pp. 20–32, 2012.

[19] K. von Frisch, *Animal Architecture*. Harcout Brace Jovanovich, New York, 1974.

[20] M. Hansell, *Animal architecture*. Oxford University Press, 2005.

[21] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*, ser. Santa Fe Institute Studies in the Sciences of Complexity. New York: Oxford University Press, 1999.

[22] S. Camazine, J.-L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau, *Self-Organization in Biological Systems*, ser. Princeton Studies in Complexity. Princeton: Princeton University Press, 2003.

[23] B. Childers, "Hacking the parrot ar drone," *Linux Journal*, vol. 2014, no. 241, p. 1, 2014.

[24] T. Bohme, U. Schmucker, N. Elkmann, and M. Sack, "Service robots for facade cleaning," in *Industrial Electronics Society, 1998. IECON'98. Proceedings of the 24th Annual Conference of the IEEE*, vol. 2. IEEE, 1998, pp. 1204–1207.

[25] N. Elkmann, T. Felsch, M. Sack, J. Saenz, and J. Hortig, "Innovative service robot systems for facade cleaning of difficult-to-access areas," in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 1. IEEE, 2002, pp. 756–762.

[26] G. Pérez, L. Rincón, A. Vila, J. M. González, and L. F. Cabeza, "Green vertical systems for buildings as passive systems for energy savings," *Applied energy*, vol. 88, no. 12, pp. 4854–4859, 2011.

[27] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5.

[28] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2004, pp. 2149–2154.

[29] M. Hamer, J. Engel, S. Parekh, R. Brindle, and K. Bogert, "ardrone autonomy: a ros driver for parrot ar-drone quadrocopter," Published online https://github.com/AutonomyLab/ardrone_autonomy, July 2014.

[30] H. Huang and J. Sturm, "tum simulator," Published online http://wiki.ros.org/tum_simulator, July 2014.

[31] M. Keijzer, J. J. Merelo, G. Romero, and M. Schoenauer, "Evolving objects: A general purpose evolutionary computation library," in *Artificial Evolution*. Springer, 2002, pp. 231–242.

[32] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza, *A field guide to genetic programming*. Lulu. com, 2008.

[33] M. J. Wooldridge, *An Introduction to MultiAgent Systems*, 2nd ed. West Sussex, UK: John Wiley and Sons Ltd, 2009.

[34] V. Sarpe, A. Esmaeili, I. Yazdanbod, T. Kubik, M. Richter, and C. Jacob, "Parametric evolution of a bacterial signalling system formalized by membrane computing," in *CEC 2010, IEEE Congress on Evolutionary Computation*. Barcelona, Spain: IEEE Press, 2010, pp. 1–8.

[35] C. Jacob, V. Sarpe, C. Gingras, and R. Feyt, "Swarm-based simulations for immunobiology," *Information Processing and Biological Systems*, pp. 29–64, 2011.

[36] S. Narayanan and L. Rothrock, *Human-in-the-loop Simulations: Methods and Practice*. Springer, 2011.

[37] Unity Technologies, "Unity - game engine," Published online http://unity3d.com/, July 2014.

[38] Different Methods, "Swarm agent," Published online http://u3d.as/content/different-methods/swarm-agent/608, July 2014.

[39] Allebi Games, "Easy path finding system," Published online http://u3d.as/content/allebi/easy-path-finding-system/4a7, July 2014.

[40] Oculus VR, Inc., "Oculus rift: Next gen virtual reality," Published online http://www.oculusvr.com/rift/, July 2014.

[41] Razer Inc., "Razer hydra portal 2 bundle," Published online http://www.razerzone.com/de-de/gaming-controllers/razer-hydra-portal-2-bundle/, July 2014.