Exogenous coordination in multi-scale systems: How information flows and timing affect system properties

Ada Diaconescu, Louisa Jane Di Felice, Patricia Mellodge

Version of Record: https://www.sciencedirect.com/science/article/pii/S0167739X20304726
Manuscript_dca652393814c9b1ab1465cc6c14b31a

Exogenous Coordination in Multi-Scale Systems: How Information Flows and Timing Affect System Properties

Ada Diaconescu[1], Louisa Jane Di Felice[2], and Patricia Mellodge[3]

[1] *Telecom Paris, LTCI, Institut Polytechnique de Paris*
[2] *Institut de Ciència i Tecnologia Ambientals (ICTA), Universitat Autonoma de Barcelona*
[3] *University of Hartford*

August 17, 2020

Abstract

The architecture of coordination mechanisms is central to the performance and behaviour of (self-)integrated systems across natural, socio-technical and cyber-physical domains. Multi-scale coordination schemes are prevalent in large-scale systems with bounded performance requirements and limited resource constraints. However, theories to formalise how coordination can be implemented across multi-scale systems are often domain-specific, lacking generic, reusable principles. In these systems, feedback among system entities is a key component to coordination. Building on theories of hierarchies and complexity, in previous work we formalised *Multi-Scale Abstraction Feedbacks* (MSAF) as a design pattern to describe the architecture of feedback across system scales, highlighting the role played by micro-entities and macro-entities, as well as their interconnections. Focusing on exogenous coordination, this paper refines the MSAF pattern, describing a feedback cycle across scales as one where information flows bottom-up and top-down through five actions: state information communication, state information abstraction, information processing, control information communication, and adaptation from control information. Abstracted state information at each scale is processed with control input from the scale above and provides control input to the scale below. Using the example of distributed task allocation through exogenous coordination, NetLogo simulations are implemented to analyse the impact that different exogenous coordination strategies, and their internal timing configurations, have on resource consumption and on convergence performance. The experimental insights and refinement of the MSAF pattern contribute to a general theory of multi-scale feedback and adaptation. This architectural pattern and associated analysis and evaluation tools are still developing, but offer a concrete basis for further expansion, improvement, and implementation, while addressing questions that are at the core of the behaviour of multi-scale systems.

Highlights

- Multi-scale self-* systems are described in terms of information flows that form multi-scale feedback cycles and comprise several entity types – generically identified via a Multi-Scale Abstraction Feedbacks (MSAF) design pattern;
- Feedback cycles are defined via five actions: state information collection, state information abstraction, information processing, control information communication, and adaptation from control information. They are interconnected via two further actions: inter-cycle information abstraction and inter-cycle communication of control information;
- Concrete implementations of multi-scale feedback strategies with exogenous designs are provided and evaluated, via a generic method, in terms of convergence behaviour, resource use, and timing;
- Results obtained from simulations run in NetLogo highlight the inherent trade-offs that these strategies feature between coordination convergence time and resource usage, for various system topology configurations, as well as different timing configurations;

- The MSAF pattern and associated evaluation approach contribute to a general theory of feedback in multi-scale systems, aimed to help understand existing systems of this kind, as well as to analyse, develop, and maintain new ones.

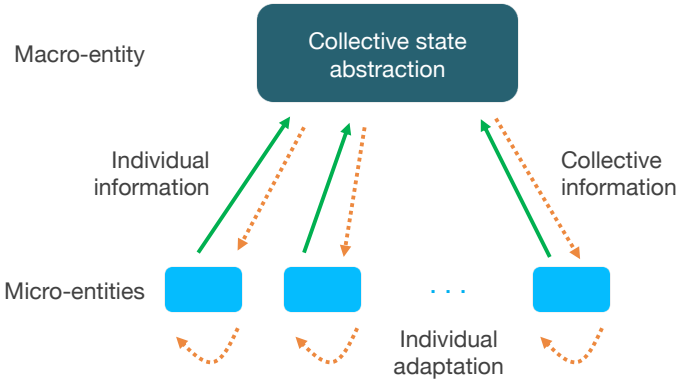
Keywords: multi-scale coordination; control feedback; state abstraction; hierarchy theory; information flows; resource analysis; complex system

1 Introduction

As socio-cyber-physical systems of increasing complexity become more and more prevalent, there is growing interest in understanding how system components integrate dynamically to achieve goals [1]. Self-integration refers to the ability of several entities to interact with each other at runtime, with the purpose of achieving individual or shared goals. Coordination regulates the distribution and timing of actions among system entities. It is a key challenge for (self-)integrating systems. In modern organisations, for example, the successful achievement of goals depends on the collective ability of human employees and automated systems to schedule and perform tasks. When multiple systems combine in unforeseen ways and in unpredictable environments, they have to master (self-)integration, and hence coordination, at increasingly large and heterogeneous scales [2]. This challenge affects both natural and artificial systems. As such, it does not only require expertise from computer science, engineering, and specific application areas, but also expertise from cross-disciplinary studies, generic systems, and complexity theories. We consider existing theories on hierarchical systems to be a promising approach for tackling complexity, by dividing systems into multiple loosely-interrelated processes that operate at different scales. Scales, throughout this paper, may refer to abstraction levels, spatial scales, or temporal scales.

A wide range of cross-disciplinary observations shows that systems composed of many parts, from molecules to animal societies and human organisations, tend to organise hierarchically [3, 4, 5, 6, 7, 8]. A multi-scale, hierarchical organisation does not necessarily entail "top-down" control, in a topological or authoritative sense [9] [10], although it might. Instead, it can be viewed more broadly as the existence of at least two scales occupied by system entities, affecting each other through feedback. In this sense, different processes in complex systems can be described as multi-scale feedback hierarchies, even if there is no centralized control. Feedback processes between scales give the scales a functional existence, through mutual impact phenomena. The absence of feedback between scales would turn scales into purely theoretical concepts, such as in an ontology, or in an evolutionary tree, and these types of hierarchies are outside the scope of this study.

Figure 1: Feedback in a two-scale system



In previous work [10], we introduced a feedback-centred definition of multi-scale systems through the *Multi-Scale Abstraction Feedbacks* (MSAF) design pattern. An overview of the main principles of MSAF is shown in Figure 1. Between subsequent abstraction levels, recursively, lower-level entities are referred to as *micro-entities* and higher-level entities as *macro-entities*. The definition of what constitutes a micro-entity and a macro-entity follows the distinction between the *observer* and the *observed* in Hierarchy Theory [11]. This implies dependence upon the observation scale, or 'focal level': a macro-entity at one scale (e.g., a tree composed of its cells and tissues) can become a micro-entity at another scale (e.g., a tree within a forest patch). For simplicity, Figure 1 only shows two system scales, with several micro-entities and one

macro-entity. Information about the state of the micro-entities, which we call *individual state information*, is collected and abstracted, for example through compression, coding, or filtering, into a macro-entity. This abstracted information about the global state of the system, referred to as *collective information*, is fed back to, or collected by, the micro-entities, who then adapt based on this information. These flows of information regulate the coordination of entities working towards a shared goal.

In the introduction of the MSAF pattern [10], three types of *macro-entities* were identified, depending on their relation to their micro-entities: *composed* (e.g., a forest patch composed of trees), *micro-distributed* (e.g., culture distributed across members of society), and *exogenous* (e.g., a manager coordinating workers in an organisation). The underlying similarity between different types of macro-entities is that abstracted collective state information at the macro-scale becomes available to the micro-entities, who adapt and coordinate accordingly. Consider, for instance, the case where ants coordinate to collect food sources through pheromone trails. Here, ants (the micro-entities) are responding to feedback generated by the pheromone trail (a macro-entity), which is in turn generated by the ants themselves. The pheromone trail provides collective state information about how many ants have traveled recently through that path. It influences each ant's decision on whether or not to follow that same path. As another example, managers (macro-entities) in an organisation collect individual state information about their employees' work (micro-entities) and feed control information back to them via adaptation control directives. Here, the collective state information is returned to workers indirectly, after being processed by managers at the macro-scale and transformed into control directives. This operation differs from the ants example, where collective state processing was carried out by individual ants at the micro-scale, before adaptation. Both cases fit our MSAF pattern in that collective state information, at the macro-scale, is employed in the feedback coordination cycle of individual entities, at the micro-scale.

The study of hierarchies and their properties has been a central approach to the analysis of complexity [4], under the name of Hierarchy Theory[1]. However, as noted by Wu [12], Hierarchy Theory "is not a formal theory, meaning that it lacks clearly defined terms, well developed methodologies, and unambiguous predictions" (p. 283). Many hierarchical designs that fit our description have been proposed for developing systems in specific application domains, for example through the study of smart grids, traffic control, or resource sharing [13, 14, 15]. While well-suited to their particular application, these ad-hoc designs cannot be easily reused, as such, for developing new systems, especially across domains. This is because the multi-scale feedback design is mixed with domain-specific coordination solutions (e.g., specific algorithms and their particular deployment onto the targeted platform). Moreover, each solution is presented through domain-specific terminology. These attributes make it difficult for non-domain experts to comprehend concrete multi-scale coordination solutions, to distill the essential design aspects from the domain-specific implementations, and to transfer these generic designs onto new application domains.

Our objective in introducing the MSAF design pattern is to avoid analysing hierarchies based on specific characteristics of their components (e.g., living vs. non-living, automated vs. intelligent, or master vs. slave), or of their interrelation topology (e.g., centralized, meshed, or tree-like). While important, these aspects are specific to each hierarchical instance, driven by its particular objectives and constraints. To remain generic, the MSAF design pattern characterises system entities from a relational stance, based on how they are connected to one another, on what role they play in the feedback process, on how they distribute resources amongst themselves, and on how information flows through them, bottom-up and top-down, forming feedback cycles. In this view, our definition of scales is also relational: a scale exists if it is occupied by entities (micro or macro) that are part of the feedback cycle.

The long-term aim of MSAF is to provide a formal framework and methodology for describing, analysing, and evaluating multi-scale feedback systems. This would offer the grounds for developing an applicable theory and engineering toolbox for scientists, system developers, and administrators, which is sufficiently generic for adoption and customisation across domains and sufficiently specific to provide concrete guidance beyond the theoretical status quo.

Expanding on our previous work [10], this paper further develops the generic description and analysis of feedback in multi-scale systems. Firstly, we view feedback cycles through the lens of information flows

[1] Capitalized hereon when referring to the specific field of Hierarchy Theory vs. a generic theory of hierarchies

travelling through different scales. The information flows are increasingly abstracted, losing information, as they propagate bottom-up (representing system state information at increasing scales); and progressively refined, with local information, as they return top-down (with more-and-more precise system adaptation controls). We formalise this feedback cycle via five actions, connecting micro- and macro-entities across two scales: *state information collection*, *state information abstraction*, *information processing*, *control information communication*, and *adaptation from control information*. Across more than two scales, feedback cycles are interconnected via two further actions: *state information abstraction* going upwards into the collection process of the cycle above, and *communication of control information* returning downwards into the information processing of the cycle below.

Secondly, we acknowledge the fact that various strategies for implementing these actions, or steps, lead to different performance characteristics in terms of convergence behaviour, bringing about different trade-offs and bottlenecks in terms of efficiency, stability, and flexibility. Focusing on systems with exogenous macro-entities, we exemplify four concrete coordination strategies for the specific application example of multi-scale task distribution. We analyse these coordination strategies in terms of their information flows and ensuing resource requirements. While the focus of our results is on exogenous macro-entities, the evaluation approach is generic and conceived to accommodate other designs (i.e., where macro-entities can be micro-distributed, composed, or mixed).

Thirdly, we acknowledge the key role that timing plays in determining the behaviour of coordination mechanisms. In Hierarchy Theory, it is generally understood that higher levels in both nested and non-nested hierarchies operate at slower timescales than lower ones. Allen and Starr, for example, include timing as one of the five general principles for ordering levels in ecological hierarchies, with higher levels operating more slowly and at lower frequencies [11]. However, it is unclear whether the difference in timing is a natural consequence of systems' hierarchical organisation, or whether there are specific design benefits to higher levels operating at slower timescales than lower ones. We explore this question by selecting one of the coordination strategies and analysing its convergence performance with respect to varying delays, both for inter-scale communication and for entity adaptation following incoming control information. This offers initial guiding considerations to take into account when timing multi-scale feedback systems.

This paper's contribution, thus, is three-fold: (i) it expands and refines the MSAF design pattern and associated analysis and evaluation approach; (ii) it exemplifies, through multi-agent simulations developed in NetLogo, different instances of the MSAF pattern via concrete implementations of multi-scale coordination strategies, focusing on exogenous designs; (iii) it analyses, evaluates and compares the concrete exogenous strategies in terms of resource use and convergence behaviour, and explores the impact of inter-level timing delays.

Importantly, our objective is *not* to identify or propose optimal algorithms for large-scale coordination, but to contribute to a generic theory, based on the MSAF design pattern, which can be implemented to analyse, develop, maintain, and/or utilise multi-scale systems for large-scale coordination. In this sense, our modelling is exploratory, with the purpose of recursively building a more robust theory, and of providing practical guidance for system design. The task distribution application included here is meant as an illustration of the theoretical concepts proposed. It is meant neither as a contribution to the task distribution problem, nor as a main means to validate the generic multi-scale design and evaluation framework.

The rest of the paper is structured as follows. The next section summarises the main streams of literature that we build upon. Section 3 discusses the key principles of the MSAF design pattern, expanding on previous work, as well as the associated evaluation approach. Section 4, then, describes the exogenous coordination strategies for the task distribution example. The strategies are implemented through an agent-based model and described following the Overview, Design Concepts & Details (ODD) protocol [16, 17]. Results are presented in Section 5 and discussed in Section 6, focusing on trade-offs between convergence time, resource use, and inter-level delays.

2 Background & Related Work

This paper builds on related work across two domains: (i) the theory of hierarchies and holarchies and (ii) research on coordination in multi-scale systems, including domain-specific studies in anthropology, computer engineering, and organisation theory.

We refer to Hierarchy Theory as the theory of hierarchies developed by Simon [3, 4], Allen and Starr [11], Pattee [6], and Salthe [18], among others (for an overview, see [12]). The theory's understanding of hierarchies strongly emphasises the role played by observers: according to Allen, for example, all systems are hierarchical, and if a system appears not to be so it is only because the observer is placed at the focal level of the system (without being able to "zoom out" and see the other levels). The focal level is the middle level of the hierarchy, which must be composed of at least two more levels, one above and one below it. Elements occupying higher hierarchical levels with respect to the ones below them follow one or more of five conditions: (i) they are the context of the lower-level elements; (ii) they constrain lower-level elements; (iii) they operate at a slower frequency; (iv) they have a higher bond strength and greater integrity; and (v) they contain or are made of lower levels. Thus, Hierarchy Theory appears to be mostly concerned with multi-scale systems where macro-entities are either composed (see point v above) or micro-distributed (see point i).

Simon [3, 4] also notes the prevalence of hierarchical systems, highlighting the *near-decomposability* property between hierarchical levels, which can be analysed quasi-independently from each other. This confers a substantial advantage to the evolution of complex systems, as each level provides a stable intermediate component for the level above. Koestler [5] follows a similar approach in his conceptualisation of *holarchies*, i.e., nested hierarchies composed of *holons*, where a holon is an entity simultaneously being a *whole* (composed of smaller parts) and a *part* of something bigger. Hence, holarchies refer to composed macro-entities. Operator hierarchy theory [8] also focuses on composed macro-entities, making the distinction between structural and functional closure across levels. The former refers to cases where lower levels are spatially enclosed by higher ones (e.g., the walls of a city fortress enclosing its inner components). The latter, also known as relational closure in the field of cybernetics [19], describes cases where higher levels are related to lower ones through mutually dependent transformation processes, with lower-level processes being necessary for the process of the whole (e.g., autocalytic sets of proteins [20]).

There are two key differences between the main principles of Hierarchy Theory and the MSAF design pattern: in terms of types of hierarchies, Hierarchy Theory makes the distinction between nested and non-nested hierarchies. However, most of the examples used to describe its principles tend to be ecologically nested hierarchies. MSAF expands and shifts this categorisation to types of macro-entities, rather than types of hierarchies, including *exogenous* and *micro-distributed* macro-entities, which are non-nested, in addition to *composed* ones, which are nested. This also allows for heterogeneous designs, where different hierarchical levels may feature different macro-entity types, such as an organism composed of cells and, at the same time, part of an exogenous social organisation. In terms of the scope of the theory itself, while Hierarchy Theory is mostly conceptual and descriptive, the aim of the MSAF pattern is to be able to model the structure and behaviour of multi-scale systems, focusing on cross-level feedback cycles and their resource distribution. This practical aim, however, does not disentangle us from theory: as mentioned in the Introduction, through practical applications we aim to progressively contribute to a more robust general theory of hierarchies.

Within this general theory, information plays a key role. Throughout this paper, we refer to information as the combination of (i) a resource flowing from one or more *output entities* to one or more *input entities* and (ii) the perception of that resource by one or more entities, leading to either short-term adaptation (the perceiving entity changes its behaviour in response to the resource) or long-term adaptation (the perceiving entity stores the resource, or a variant of it, and eventually adapts its behaviour). Entities that perceive a piece of information are a subset of the entities receiving it (i.e., the input entities). Following McKinney and Yoos' information taxonomy [21], the resource which is flowing is equivalent to the *information as a token* view, and its perception leading to adaptation is equivalent to the *information as adaptation* view. We use the term information to refer to both of these views, with the caveat that the resource flow is only relevant to us if it does, eventually, lead to some form of adaptation.

The ways in which entities process and store information, then, is crucial in determining coordination

mechanisms. Centralized coordination relies on one or more entities in the system playing a privileged role in the coordination of other entities, usually through a higher resource capacity (e.g., entities with higher information capacity and knowledge of the state and behaviour of the overall system). Decentralized coordination relies on entities sharing local resources (e.g., local information exchanges) to fulfil a task. It does not require that any single entity have knowledge of the system as a whole. The distinction between centralized and decentralized coordination, however, is often blurred, for example in ant colonies where certain ants are more central to information flow in the colony when compared to others [22]. In this case, coordination is based on local information exchange among all ants, but some ants still play a privileged role.

In anthropology, organisation types are often linked to the scale of communication [23, 24, 25, 26]. While a peer-to-peer, decentralized communication style is convenient for smaller groups [25], limiting communication overheads, larger groups tend to implement more centralized or tree-like structures [23], with entities at higher levels mediating information to avoid 'communication stress' [27]. Malone [28] compares different centralized and decentralized coordination structures in organisation markets, pointing to fundamental trade-offs between efficiency and flexibility: while centralized markets are coordinated more efficiently, they are also less flexible to changes. De Wolf and Holvoet [29] summarise the main characteristics of design patterns for decentralized coordination. Similarly to the efficiency vs. flexibility trade-off, they point to possible trade-offs across different design patterns in terms of optimality and flexibility.

From our feedback-oriented perspective, coordination among entities is not so much a matter of centralization or decentralization, but rather one of how information is communicated and transformed across different system scales, and how this impacts coordination behaviour and resource distribution. From this perspective, centralized, decentralized, and hybrid topological schemes are an orthogonal concern, representing the way in which a system's abstract information scales are actually implemented or mapped onto a resource platform. Moreover, as mentioned in the Introduction, the presence of multiple scales does not necessarily reflect a centralized organisation (in an authoritative sense). That is, the information abstractions about a system's state do not need to be co-located with the authority that enforces the ensuing system adaptations [30]. For instance, in stigmergy-based coordination approaches, abstracted information about the system state is encoded onto an external macro-entity (e.g., current frequency in smart grids [31] and pheromone trails in ant foraging [32]), yet the ensuing decisions are taken by the adapting micro-entities (smart devices in smart grids and ants in the foraging process, respectively).

Constructal Theory [33] highlights the important role that the topology of hierarchical systems plays in their evolution and viability. The Constructal Law stipulates that "for a finite-size flow system to persist in time (to live) it must evolve such that it provides greater and greater access to the currents that flow through it". This explains, for instance, the prevalence of tree-like shapes in river basins, plants, snowflakes, neural networks, traffic networks, or social organisations, due to the progressive optimisation of flows from a point to an area, or vice versa. Two main aspects differentiate our study from this theory. Firstly, while Constructal Theory mostly applies to matter and energy flows, where the flowing quantities conserve over time, we focus on information flows abstracted across system scales. Secondly, while Constructal Theory concentrates on uni-directional flows, we are only concerned with those flows that form feedback cycles leading to system adaptation.

Software Engineering also provides several hierarchical or multi-layer designs for system coordination, including [34] for Self-Aware Computing, [35] for Organic Computing, [36] for Autonomic Computing, [37] for Adaptive Systems, and [38] for Robotics. With respect to the MSAF pattern, these domain-specific designs correspond to the ones where macro-entities are exogenous. These are similar to nested or embedded controllers in Hierarchical Control Theory [39]. Composed macro-entities can be found in some component and service-oriented architectures (SOA) [40], Holonic Multi-Agent Systems (HMAS) [41], and recursive problem-decomposition designs (Wrappings) [42]. These differentiate from the MSAF composed type in that they are activated "on-call", as needed, rather than running continuously as in the case of feedback cycles. They are also typically activated top-down, from the hierarchy's root, whereas MSAF systems can be activated both top-down and bottom-up. While MSAF is compatible with the feedback designs proposed by the related domains above, it is more generic in that it only focuses on those design aspects that are

essential to *multi-scale* feedback cycles. It emphasises bottom-up information abstraction and top-down control refinement as generic flows occurring across system scales irrespective of their domain-specific details (e.g., concrete information acquisition, analysis, learning and decision making algorithms).

In previous work, we also proposed a generic goal-oriented holonic architecture for large-scale self-integrating systems [43]. This corresponds to encapsulated hierarchy systems with composed macro-entities, allowing for heterogeneous designs across levels. We then identified several design patterns for implementing each hierarchical level, such as tree-like, collaborative, and stigmergic, mainly exemplified for the smart micro-grid application domain [13]. This architectural view focused on specifying the goal-oriented top-down control flow as a means of (self-)integrating various system scales and on the design patterns available for implementing each scale. We further generalised this in [10], introducing the MSAF pattern and its three macro-entity implementation types, based on an extensive cross-domain literary study. The current paper expands beyond these initial proposals, studying information flows and feedback cycles in order to provide a generic framework for specifying information transmission and processing across the entire system, and for analysing the corresponding resource distribution consumption.

The next section introduces our approach to the description of feedback cycles and information flows in multi-scale systems.

3 Feedback Cycles and Resource Evaluation

3.1 Entities and Scales

Multi-scale systems are composed of N abstraction levels, indexed as $L_n = L_0..L_{N-1}$. Levels are populated by entities $e_{id}|id = 0..A-1$, with A being the total number of entities. We define *entities* loosely, as any system component that sends, receives, or transforms information within a feedback cycle. Entities across levels are connected through streams of information flows. As mentioned in Section 2, we refer to information as data that are passed between entities and that lead to adaptation. In this sense, material or energy flows that may be passing through entities are relevant to us in terms of their information content. Information flows are sent, received, or transformed through *actions*, with each action requiring one or more output entity e_{out}, one or more input entity e_{in}, and one or more acting entity e_{act}, i.e., the entity or entities that are doing work (spending energy) for the information flow to be sent, received, or transformed. The output and input entities can be the ones doing the work for the flow to be exchanged, or a third entity may play the role of e_{act}. Actions can also be internal to entities, for example when entities process an incoming flow, transforming it into something else.

When considering a feedback cycle between two abstraction levels, or scales, we define entities populating the lower hierarchical level as *micro-entities*, and entities at the higher level as *macro-entities*. In addition to information flows connecting micro and macro-entities at subsequent levels, these two types of entities are also connected through relations, which can be associations ($e_x \neq e_y$) or compositions ($e_x \in e_y$). When a micro-entity is related to a macro-entity above it, it is also referred to as a *child* with respect to the macro-entity, its *parent*. In previous work [10], we identified three types of relations between micro-entities and macro-entities, leading to three types of macro-entities:

1. *Exogenous macro-entities* exist separately from the micro-entities. They can be materially similar to the micro-entities (e.g., a manager as an exogenous macro-entity to workers) or different (e.g., a pheromone trail as an exogenous macro-entity to ants);

2. *Micro-distributed macro-entities* are macro-entities that are distributed across micro-entities (they are endogenous). Examples include culture or political identity distributed across a population;

3. *Composed macro-entities* are also endogenous to micro-entities. They are compositions of the micro-entities below them (e.g., forests made of trees or molecules made of atoms).

This characterisation of types of macro-entities further explains our broad definition of entities. People, insects, culture, a database, or an abstract idea of the world can all represent macro-entities if they are

associated with collective information which leads to the adaptation of micro-entities, which, in turn, may be different entities or hosted within the same entity. Different types of macro-entities can also co-exist within the same system. For example, we may consider a three-level system composed of individuals (micro-entities), their political identity (micro-distributed macro-entity), and the leader of each political group (exogenous macro-entity). In all cases, a macro-entity provides collective state information about the micro-entities it gets data from, and shares that collective state information with micro-entities, who adapt accordingly. We use terms such as "provide" and "share" loosely, as depending on the domain of the system, alternative terms may be better suited. We refer to the first set of micro-entities as *abstracted* micro-entities (from which information is collected and abstracted), and to the second set as *adapted* micro-entities (who adapt following control signals that result from the collective abstraction). The set of abstracted and adapted micro-entities may be the same, for example in the case of workers being monitored and directed by a manager. They may also be different, for example in the case of a set of trees generating a patch of light availability which impacts the growth of another set of trees.

We specify here a generic feedback cycle, identifying its main information flows and their interrelations, first in terms of a single cycle across two scales (3.2) and then in terms of multiple cycles across more than two scales (3.3). While the description remains generic, we acknowledge that systems may require assumptions and simplifications to fit within this framework, as scales are not always clear cut.

3.2 Single Feedback Cycle

In the MSAF pattern, we define a *single feedback cycle* as a set of interconnected actions where information flows link two system scales, with information being transferred bottom-up and top-down between the two scales, leading to the adaptation of micro-entities with respect to feedback received from macro-entities.

Focusing on the transfer of information across two scales, feedback cycles are characterized by five actions: *state information collection*, *state information abstraction*, *information processing*, *control information communication*, and *adaptation from control information*. To explain details of this generic definition in less abstract terms, we refer to two specific examples of exogenous coordination: food collection in ant colonies through pheromone trails, and worker-manager feedback in an organisation. In the first case, ants are the micro-entities while the pheromone trail is the exogenous macro-entity. In the second one, workers are micro-entities and their manager is the macro-entity. The five actions are characterised as follows:

1. *State information collection*: State information from the individual micro-entities is collected within macro-entities. For example, workers may send a log of what they have been doing to their managers, managers may collect this information directly from workers, or information could be collected by a third entity (e.g., a tracking system). In the ant example, ants deposit pheromones onto a trail;

2. *State information abstraction*: The collected information is abstracted, with an inevitable loss of information. For example, managers may compile a table summarising which actions have been carried out by how many workers, based on the information collected in the step above. Depending on the system, this action may be conflated with the one above. Such is the case for the ant example, where the trail receiving pheromones is simultaneously collecting information and abstracting it, by changing its pheromone concentration;

3. *Information processing*: The abstracted information is processed to determine what control signals to send to the micro level. Managers may work out which tasks still need to be completed, based on their compiled table, an external computer system may process the abstracted information and generate an output, or workers themselves may process the abstracted information stored in an external computer system. In the case of ants, the abstracted information from the pheromone trail is used as an input for information processing within the ants themselves, in deciding where to move next;

4. *Communication of control information*: Control information is communicated to the micro-entities. Managers may tell workers what to do next, or workers may fetch the feedback directly from managers. If workers are the ones processing the abstracted information, this communication action is null, or

internal, since the control information is already within the workers. This is the case for ants who are processing information from the pheromone trail. However, in these cases a communication step between the macro-level and the micro-level is still required to connect the cycle. Workers still fetch the abstracted information, and ants collect the information from the pheromone trail;

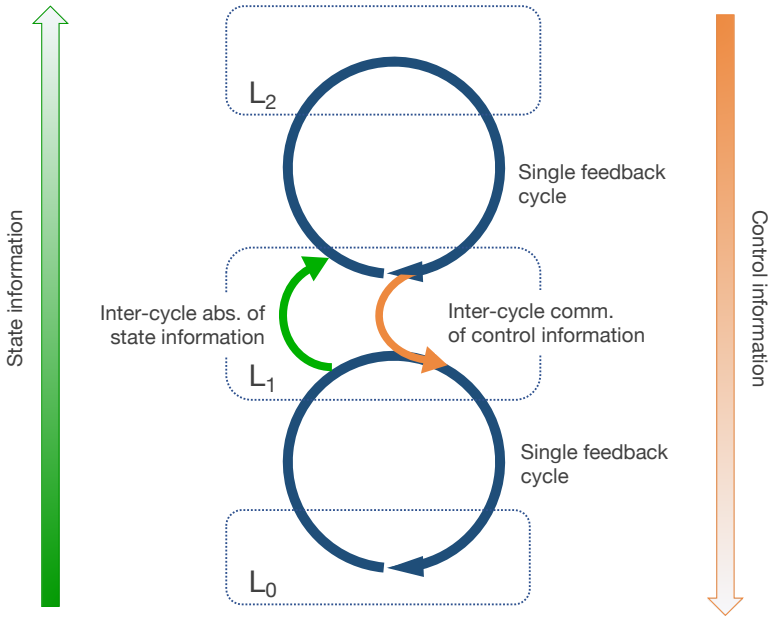
5. *Adaptation based on control information*: The micro-entities adapt their behaviour based on the received control information. For example, workers may pick which tasks to perform based on feedback from managers and other inputs, such as their current state or which other tasks are available. Similarly, ants decide where to move next based on a series of inputs which includes the information received from the pheromone trail.

While these five actions, as described above, can sometimes be conflated or happen in different ways, what remains the same is the continuity of the information flow passing through them. The information flow moves up and down the two scales, being abstracted first, and then transformed into control information used by the micro-entities to adapt. While, by definition, abstraction always happens at the macro-scale and adaptation always at the micro-scale, the other actions are not assigned to either scale in particular.

The overall feedback cycle is also not necessarily triggered by information collection. This is only the case when system coordination starts with actions performed by micro-entities at the bottom-most level, who then coordinate via the higher levels. Coordination may also be triggered by a macro-entity at the top-most level, by sending control information about a desired coordinated action to the level below (e.g., an organisation's manager sending objectives to the subordinates). A concrete example of such top-down coordination was developed in [13], in the context of goal-oriented hierarchical systems, applied to smart micro-grids.

3.3 Multiple Feedback Cycles

Figure 2: Multiple feedback cycles across three levels, extensible to N levels

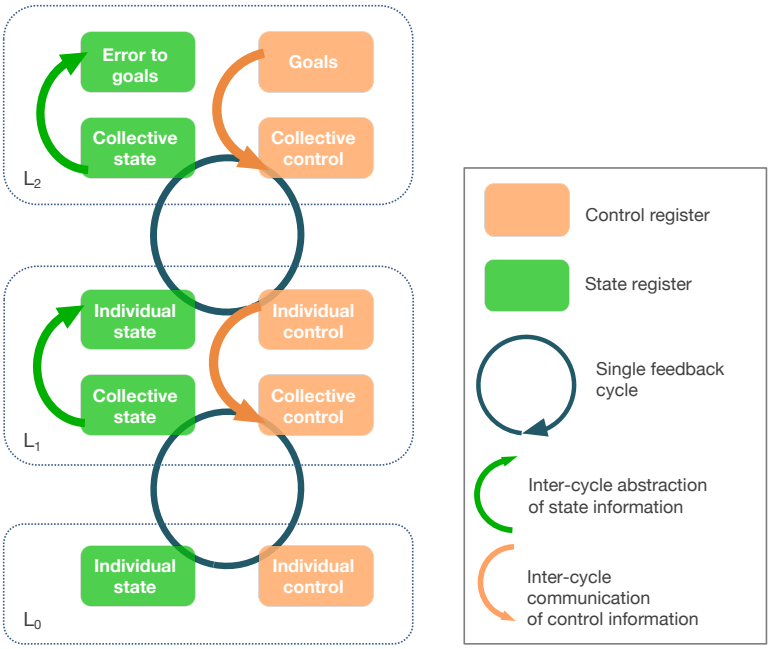


The feedback cycle introduced in (3.2) operates between two system scales, L_{n-1} and L_n, by collecting, transforming, and returning information. To connect this two-scale process to the scale above (L_{n+1}), we identify two further actions (Figure 2): *inter-cycle abstraction of state information* and *inter-cycle communication of control information*. As the ant-pheromone system is always two-scale, we describe these actions referring to the example of the worker-manager system, expanding it across three scales populated by workers, mid-managers, and a top-manager:

- *Inter-cycle abstraction of state information*: an abstraction of the state of an entity (at L_n) is sent to its parent (at L_{n+1}). The sent state is a further abstraction of the collective state that the entity has computed from the level below (L_{n-1}). For instance, mid-managers may send the statistics of actions performed by their workers to their top-manager;
- *Inter-cycle communication of control information*: an entity (at L_n) receives control information from its parent entity (at L_{n+1}) and employs it as input to its information processing. As this higher-level control input merges with the entity's processing flow, it is refined, or enriched with local information, which has not been sent to the higher level. For instance, a top-manager can tell mid-managers which performed actions were superfluous and which in higher demand. In turn, the mid-managers can use local information about the actions performed by each one of their workers in order to update their specific control directives to these, accordingly.

3.4 Information Storage Registers

Figure 3: Multiple feedback cycle across three levels, with storage registers



Having explained how information flows may connect multiple scales through a series of actions, we now introduce *information registers* as a form of intermediary storage for information flows between levels, schematized in Figure 3. This is useful for more easily transferring feedback cycles into real computing architectures and evaluating their resource requirements.

The *state information collection* and *state information abstraction* actions (condensed here into one) deposit information into a **collective-state** register. As the name suggests, this represents the collective state abstraction, stored within a parent entity at L_n, of the child entities at L_{n-1}. Then, the *information processing* action uses the collective state information in this register and deposits the transformed result into a **collective-control** register. This represents the control information sent to child entities (at L_{n-1}) via the *communication of control information* action. As before, child entities adapt accordingly.

The resulting feedback cycle (between L_{n-1} and L_n) connects to the level above (at L_{n+1}) via two further registers. Information from the **collective-state** register (at L_n) is further abstracted via the *inter-cycle abstraction of state information* and deposited into an **individual-state** register (at L_n). This new register represents the entity's individual state as viewed by the level above (L_{n+1}). It is transferred to the **collective-state** register of each parent entity at L_{n+1}, via the parent entity's *state information collection* and *state information abstraction* actions. Similarly, an **individual-control** register (at L_n) receives information about how to adapt itself (and implicitly its child entities) from parent entities (at L_{n+1}). Each parent entity's *communication of control information* (at L_{n+1}) transfers information from its **collective-control** register to the **individual-control** register of the child entity (at L_n). Then, the *inter-cycle communication of control information* forwards this information as input to the entity's *information processing* action. The *information processing* action merges the individual control information from above (L_{n+1}) with the collective state information from below (at L_n) to determine the collective control information for its child entities (at L_{n-1}). The parent's control information from above is refined, or enriched based on local information collected from below. For simplicity, we only consider here tree-like topologies (i.e., with a single parent per child), hence ignoring the necessary merging of input from several parents.

The entity at the top-most level (the top-manager, in our example) uses two additional registers: **goals** and **error-to-goals**. The **goals** register is where the the system objectives are stored (e.g., internal goals or external goals from a user) and is used as an input for the step of *information processing*, together with information from the **collective-state** register. The **error-to-goals** register stores the difference between the current system state and the objectives in the **goals** register. Depending on the system design, entities at lower levels may also be aware of the system goals, or hold local goals, in which cases they would also contain a **goals** register (not considered here).

For each entity at an intermediate scale (L_n), the two bottom registers (**collective-state** and **collective-control**) represent the entity's *macro-facet* with respect to the scale below (L_{n-1}), while the two top registers (**individual-state** and **individual-control**) represent the entity's *micro-facet* with respect to the scale above (L_{n+1}). This corresponds to Koestler's double-faced holon view (Janus) [44], showing their 'whole' face below (macro) and their 'part' face (micro) above.

3.5 Resource Requirements

We define E_n the set of entities at level L_n and $|E_n|$ as the number of members of this set. Hence, E_0 is the set of entities at the bottom-most level L_0 and E_{N-1} the set of entities at the top-most level L_{N-1}. We further define C as the number of children per parent, with each child identified via a unique $childId$ with respect to its parent ($childId = 0..(C-1)$). The maximum number of entities at level n is $C^{(N-n-1)}$. The number of levels for a multi-scale system with $|E_0|$ entities at the bottom level L_0 and C children per parent is:

$$N = \begin{cases} log_C|E_0| + 1, & \text{if } (|E_0| \bmod C) == 0 \\ log_C|E_0| + 2, & \text{otherwise} \end{cases} \quad (1)$$

To evaluate the resource requirements of feedback-driven multi-scale systems we aim to determine: (i) storage requirements, in terms of the sizes of each register type, at each abstraction level; (ii) the communication amount, in terms of number and size of inter-level messages engendered by the system's protocols; (iii) the processing requirements of the information transformation flows at each level.

To provide a general estimate of storage, or memory requirements, we consider that bottom-level entities (E_0) only feature two information registers, as they only display a micro-facet towards their parent: **individual-state**, to send their state to their parent and **individual-control** to receive control information

from their parent. Entities at intermediate levels (E_1 to E_{N-2}) have four registers each, as in Figure 3. In addition to the two registers representing their micro-facet (same as for E_0 entities), they feature two more registers for their macro-facet, displayed towards their children: **collective-state** to abstract the global state of their children and **collective-control** to send control information to their children. Finally, top-level entities (E_{N-1}) only feature a macro-facet towards their children - hence two registers (**collective-state** and **collective-control**). As explained in Section 3.4, the top entity features two extra registers: **goals** and **errors-to-goals**.

Based on these considerations, the total memory requirements of a multi-scale system, at any time t, is approximately:

$$Memory_{tot,t} = |E_0| * (|\textit{individual-state}_{E_0}| + |\textit{individual-control}_{E_0}|) + \sum_{n=1}^{N-2} [|E_n| * (|\textit{collective-state}_{E_n}| + |\textit{collective-control}_{E_n}| + |\textit{individual-state}_{E_n}| + \textit{individual-control}_{E_n}|)] + (|\textit{collective-state}_{E_{N-1}}| + |\textit{collective-control}_{E_{N-1}}| + |\textit{goals}| + |\textit{error-to-goals}|) \quad (2)$$

Here, $|<\textit{reg-name}>_{E_n}|$ is the size of a register $<\textit{reg-name}>$ of an entity in E_n. We assume that all entities at a certain level are homogeneous in terms of sizes of the same register type (though registers of different types may have different sizes). This is a simplifying assumption as we only focus on the heterogeneity of entities located at different levels.

Inter-level communication is the number of messages between each child entity at L_{n-1} and its parent entity at L_n, $Messages_{(E_{i,n-1} \to E_{j,n})}$, multiplied by the total number of child entities at L_{n-1} (assuming a single parent per child and homogeneous communication of all children with the parent):

$$Messages_{(n-1) \to n} = |E_{n-1}| * Messages_{(E_{i,n-1} \to E_{j,n})} \quad (3)$$

The total bandwidth can be obtained by multiplying each message exchanged by the size of the information register being transferred (e.g., $|\textbf{individual-state}_{E_n}|$ for bottom-up messages and $|\textbf{collective-control}_{E_{n+1}}|$ for top-down messages).

4 Exogenous Multi-Scale Task Distribution Strategies

To apply our general framework to a specific case of multi-scale feedback, we select the coordination problem of task distribution amongst multiple workers. As mentioned in the Introduction, the coordination problem is illustrative, as our aim is not to contribute to the task distribution literature, where the issue of centralized and decentralized task allocation is discussed from multiple perspectives (see, for example, [45] [46] [47]).

We test and compare four exogenous coordination strategies by developing an agent-based model (ABM), using NetLogo. To describe the ABM implementation, in the next sub-sections we follow the 'Overview, Design Concepts, and Details' (ODD) protocol provided by Grimm et al. [16] [17]. The ODD framework allows for each assumption of the model to be clearly stated, improving transparency not only within the algorithms implemented by ABMs but also regarding modelling choices and objectives, making it easier to understand and replicate the ABM and its simulation results. In the next sub-section (4.1), we specify the purpose of the simulations, and how the entities generalised in sub-section 3.1 are implemented in the strategies. Then, sub-section 4.2 specifies how the five actions of the feedback cycle are implemented across each strategy, including details on how the simulations are initialised. The ODD design concepts are briefly outlined in sub-section 4.3, before providing full details of the strategies we implemented in subsection 4.4, including the data model for each strategy, and details on what the feedback cycle looks like for each. Before moving to simulation results, we end this section with a resource analysis of each strategy (4.5).

4.1 Purpose and Entities

The purpose of the agent-based model is (i) to use the general design pattern described above to characterise strategies for exogenous coordination in multi-scale systems, (ii) to compare coordination strategies in terms of convergence performance (convergence time and convergence behaviour) and resource consumption, and (iii) to study the impact of execution time differences across levels on convergence performance.

The coordination problem consists of distributing a number of tasks of different types amongst a fixed population of workers, according to a predefined goal. The goal is provided as input and indicates how many worker entities should perform each type of task, at each simulation step. We limit the set of task types to two, assuming that the number of worker entities is equal to the sum of all tasks in the goal, meaning that no entity can be idle when meeting the goal. Starting from a random distribution of task types, the worker entities must coordinate and change the type of tasks they perform until reaching the goal. We explore and compare four coordination strategies for this task distribution problem, with different underlying algorithms of information abstraction, processing, and communication.

We call the task-performing entities situated at the bottom-most level (L_0) *workers*, the coordinating entity at the top-most level (at L_{N-1}) *top-manager*, and all coordinating entities at intermediate levels (from L_1 to L_{N-2}) *mid-managers*. We assume that each worker and mid-manager has one and only one parent, implying the presence of a single top-manager. We define W as the set of all workers (at L_0), with $|W|$ the number of workers in the set (equivalent to E_0 in the general model of subsection 3.5). Similarly, M is the set of all mid-managers, with $|M|$ their total number. $E = W \cup M \cup TM$ is the set of all entities, with TM designating the top-manager (E_{N-1} in in the general model of subsection 3.5). As in the general model, C is the number of children per manager, and $childId$ the id of a child with respect to its parent manager. The model does not have a spatial dimension to it, while time is measured in discrete time steps ($t = 0..$*max-simulation-steps*).

Each worker $w \in W$ is characterised by a task $selectedTask_{w,t}$, which it is carrying out at simulation time t. K is the number of task types to be distributed, with $K = 2$ in the presented simulations. Each task type has an index, denoted by $task_{id}$; for $K = 2$, $task_{id} \in \{0, 1\}$. Finally, the task distribution objectives are defined as *goals[]*. For $K = 2$, we have $goals = [goal_{task0}, goal_{task1}]$.

4.2 Process Overview, Scheduling, & Initialisation

The model follows the five actions of the feedback cycle outlined in subsection 3.2, interconnecting feedback cycles across scales via the two extra actions of subsection 3.3. Inter-cycle communication only occurs between subsequent scales (or abstraction levels). The four storage registers are instantiated into workers, mid-managers, and the top-manager as specified in subsection 3.4 for entities in E_0, $E_1..E_{N-2}$, and E_{N-1}, respectively. The size of the registers depends on the task distribution strategy used. The worker's **individual-state** register is an exception, always featuring the same size, as it is necessary to store its current state ($selectedTask$). The top-manager's **goals** register ($goals = [goal_{task0}, goal_{task1}]$) and an **error-to-goals** register also share the same size of $K = 2$. The sum of goals for all tasks is always equal to the number of workers (e.g., if the goal for $task_0$ is 2 workers, and the goal for $task_1$ is 10 workers, then there is a total of 12 workers).

Each simulation run is initialised by choosing: (i) the total number of workers $|W|$; (ii) the system goal of task allocation between $task_0$ and $task_1$, matching the total number of workers; (iii) the number of children per manager C, which together with $|W|$ determines the number of scales N of the system; and (iv) the strategy used to determine the behaviour of workers, mid-managers, and the top-manager.

At time step $t = 0$, workers are assigned a random task, after which the feedback cycle begins, based on the specific strategy selected, as follows:

1. *State information collection*: Workers send information of their current state ($selectedTask = 0$ or 1) to their manager, by placing it into their **individual-state** register – $e_{out} = worker$; e_{in} = worker's **individual-state** register; $e_{act} = worker$;

2. *State information abstraction*: Managers update their **collective-state** register with the status of each worker, along with the status of its other workers – e_{out} = child's **individual-state**; e_{in} = manager's **collective-state**; e_{act} = *manager*;

3. *Information processing*: Managers use a sub-algorithm (of the given strategy) to process what control information to send down to workers. If the manager is a mid-manager, control information is processed using, as inputs, the **collective-state** register that they have updated with information coming from below, and their **individual-control** register updated with information from the manager above (see next step). If the manager is the top-manager then **individual-control** is substituted by the **goals** register. We define the sub-algorithm used by mid-managers within a given strategy as *midmanager-control-subalgorithm* and the one used by the top-manager as *topmanager-control-subalgorithm*. Here, e_{out} = the manager's **individual-control** (or **goals**) and **collective-state**; e_{in} = the manager's **collective-control**; e_{act} = manager (mid-manager or top-manager);

4. *Communication of control information*: Children read the control information from their parent manager's **collective-control** register onto their own **individual-control** register. If the child is a mid-manager, it uses this information to compute which control signals to send further down. If the child is a worker, we move to the next step. Here, e_{out} = parent's **collective-control** register; e_{in} = child's **individual-control** register; e_{act} = *child*;

5. *Adaptation based on control information*: Using the specific sub-algorithm of the given strategy *worker-control-subalgorithm*, workers update their task. e_{out} = worker's **individual-control** register; e_{in} = *worker*; e_{act} = *worker*.

The two additional inter-cycle steps proceed as follows:

- *Inter-cycle abstraction of state information*: Mid-managers further abstract information in their **collective-state** register and place the result into their **individual-state** register. This means that mid-managers offer an abstracted view of their internal state to their parent manager. The parent manager acquires this information during step (1) of its feedback cycle (*State information collection*) and places it in its **collective-state** register. In the current implementation, inter-cycle abstraction occurs after information processing (i.e., between steps (3) and (4) of the feedback cycle).

- *Inter-cycle communication of control information*: Mid-managers acquire control information from their parent manager by fetching it from the parent's **collective-state** register and storing it into their **individual-control** register. This control information is used as input during the information processing step (3), to guide the control information sent to child managers, or workers. In the current implementation, this inter-cycle action occurs between steps (2) and (3) of the feedback cycle.

In terms of timing, coordination is synchronous: at each level entities execute one after the other, in random other. Levels also execute one after the other, starting from the lowest level and moving up, then starting over from the lowest level again. Pseudo-code for the generic feedback process is included in the Appendix (Listing 1).

4.3 Design Concepts

The ODD framework identifies eleven design concepts: basic principles, emergence, adaptation, objectives, learning, prediction, sensing, interaction, stochasticity, collectives, and observation. The *basic principles* underlying our model's design are the operation of inter-level feedback, its subsequent adaptation, and its role in the large-scale coordination of multi-scale systems. The model aims to provide insights into the behaviour of such systems under different assumptions (e.g., number of levels, number of entities, inter-level delays, task distribution goals), with the broader aim of contributing to a rigorous formalisation of a theory of feedback and adaptation in different multi-scale systems. In terms of *emergence*, system-level patterns arising from the entities' behaviour include convergence time, convergence behaviour, and resource use. The

adaptation design concept is central to our approach, as already explained in detail. In this sense, the *objective* is a system-level goal of task distribution. We do not include *learning* mechanisms for now, as the strategies determining the entities' behaviours do not change over time. As for *sensing*, the state variables included in the entities' decision-making process are the four storage registers which we discussed in Section 3.4. *Interactions* among agents are based on information communication, via information flows that pass through the storage registers. *Stochasticity* characterises system initialisation, with workers selecting tasks randomly until they receive control directives from higher levels. Some of the coordination strategies also use probabilistic processes, to avoid getting stuck or causing oscillations. *Collectives* of entities determine the entities' information abstraction and refinement across hierarchical levels. In this case, as we are only looking at exogenous macro-entities, collectives are formed via shared parent managers (groups of children linked to the same parent). *Observation*, finally, is based on the following output data: convergence time (in time steps), total error to the goals at each time step, number of entities, levels, and children per parent, type of strategy, and characterisation of its resource requirements (memory, processing, and bandwidth).

4.4 Task Distribution Strategies

We test and compare four task-distribution strategies: a *Random with Rewards Strategy* (shortened to 'Rewards'), a *Blackboard Strategy* (BB), a *Model Strategy*, and a *Basic Strategy*.

The *Random with Rewards Strategy* is a relatively simple coordination strategy. Workers carry out a task (either task 0 or task 1), and receive a positive reward if the task needs completing, thus staying in that task, or no reward if the task isn't needed, thus switching tasks. It uses a top-manager to collect the workers' selected tasks and to return rewards to a maximum of workers equal to the goal of each task type. Hence, the strategy only applies to two-level systems, with an exogenous macro-entity tracking which tasks need to be completed. The tracking does not necessarily require high processing capacities. For example, the exogenous macro-entity could simply be a pile of two resources - when a worker goes to pick a resource and that resource isn't there, it doesn't receive positive feedback (i.e., the resource itself), and switches tasks (to collecting the second type of resource).

The *Blackboard Strategy* is more sophisticated. It aggregates the amount of selected tasks for all children combined, separately for each task type. Propagated upwards, aggregates are summed up, recursively, for children at each level. The difference between this aggregate and the goals is provided as control feedback (a correction error for each task type). Propagated downwards, control feedback is split more or less equally amongst children, with a small random error. If the feedback for their selected task indicates that too many workers are already performing it, workers have a certain probability of changing tasks. The name *Blackboard* reflects the fact that this strategy utilises a common location (i.e., a blackboard) where entities access shared knowledge.

The *Model Strategy* keeps track of the amount of selected tasks of each type within each child branch. This differs from the Blackboard strategy, which only keeps the sum of selected tasks for all children combined. As the name suggests, managers here have a more complete model of the state of their children. The advantage of this strategy is that, in keeping aggregate information for each branch, a manager can know: (i) how many workers are available in each of its sub-trees, and (ii) what the selected task distribution is amongst its sub-trees. This extra information helps provide more accurate feedback to each sub-tree, in terms of necessary task switching, because each manager knows how many workers it has at its disposal and what exact changes they can perform. We include a variation of this strategy, *Model Optimisation with Initial Rewards* (ModelRew), combining the *Model* and *Random with Rewards* strategies, capitalising on their combined advantages. The strategy employs the Reward strategy during the initial steps, when feedback is not yet available from the higher levels (i.e., no adaptation model is provided from the top).

Finally, the *Basic Strategy* follows a simple mechanism whereby each level aggregates information using summation. Workers send their task (0 or 1) upward and the mid-manager sums the tasks over all its children. This procedure is completed for all levels until the top-manager has the total of all workers performing task 1. The top-manager then compares this value with the desired task distribution to generate an error. To propagate from the top downward, the error is divided equally amongst the children at each level. The feedback to the worker is a decimal number between 0 and 1 inclusive. The workers, then, switch tasks

with probability proportional to the manager feedback. Due to the simplicity of this strategy and to the large amount of task switching among workers, this algorithm was selected to perform timing analysis in the presence of inter-level delays.

For each strategy, the data model is summarized in Table 1. In addition to the four registers of Table 1, each strategy has an additional **goals[]** register, where the index is the $task_{id}$ and the value is the *goal* for that $task_{id}$. In addition, the *Random with Rewards* strategy features a **rewards[]** register, initialised with a copy of the **goals[]** register at the beginning of each step. Here, the index is the $task_{id}$ and the value is the number of rewards remaining for that $task_{id}$ in the current step ($rewards[task_{id}] = rewards_{task_{id}}$). The following sub-sections provide full details of the four strategies' feedback cycles, as well as describing the hybrid *Model Optimisation with Initial Rewards* strategy. Subsection 4.5 analyses these strategies in terms of their resource distribution requirements.

Register	Random w/ Rewards	Blackboard	Model	Basic
Individual State	**individual-state**: the $task_{id}$ of the selectedTask	**individual-state[]**: a table where index i is the $task_{id}$ and the value at i is the number of tasks performed with $task_{id}$ (by all the manager's children combined)	**individual-state[]**: a $taskModel[]$ table where index i is the $task_{id}$ and the value *individual-state[i]* is the number of workers that selected this task (in the sending entity's sub-tree)	**individual-state**: the number of workers performing $task_1$
Collective state	**collective-state**: the $task_{id}$ of the selectedTask	**collective-state[]**: a table where index i is the $task_{id}$ and the value at i is the number of tasks performed with $task_{id}$ (by all the manager's children combined)	**collective-state[][]**: a table of tables where index j is the $child_{id}$ of the child sending the information, and the value $collective\text{-}state[child_{id}]$ is a $taskModel[]$ (cf. above) for that $child_{id}$	**collective-state**: the number of workers performing $task_1$
Individual control	**individual-control**: the *reward* for the selectedTask (true or false)	**individual-control[]**: a table where index i is the $task_{id}$ and the value at i is the error correction amount for $task_{id}$ ($> / = / < 0$)	**individual-control[]**: a $goalModel[]$ table where index i is the $task_{id}$, and the value ($goalModel[i]$) is the number of tasks to be selected by the worker for that $task_{id}$	**individual-control**: the error between the current aggregate and the goal of $task_1$ divided equally amongst children
Collective control	**collective-control**: the *reward* for the selectedTask (true or false)	**collective-control[]**: a table where index i is the $task_{id}$ and the value at i is the error correction amount for $task_{id}$ ($> / = / < 0$)	**collectivecontrol[][]**: a table of tables where index j is the $child_{id}$ of the targeted child; and the value ($collective\text{-}control[child_{id}]$) is a $goalModel[]$ (cf. above) for that $child_{id}$	**collective-control**: the error between the current aggregate and the goal of $task_1$ divided equally amongst children

Table 1: Data model for each strategy

4.4.1 *Random with Rewards* Strategy

This approach assumes that an immediate and accurate reward process is available to all workers. It also assumes that workers remember their *reward* and $selectedTask$ from one step to the next. The feedback cycle between the workers and the top-manager proceeds as follows:

1. *State information collection*: Each worker sends the $task_{id}$ of its selected task to the top-manager,

requesting a reward;

2. *State information abstraction*: The top-manager processes each collected $task_{id}$ immediately (there is no $task_{id}$ storage). The abstraction is reflected in the remaining amount of rewards for each $task_{id}$ (*rewards[]* table, cf. Processing step, below).

3. *Information processing*: The top-manager processes requests sequentially, in the order of arrival. If $rewards_{task_{id}}$ is higher than zero, then the top-manager grants a reward (true) and decreases $rewards_{task_{id}}$ by one; otherwise it grants no reward (false);

4. *Communication of control communication*: Each worker receives its reward reply from the top-manager (true or false);

5. *Adaptation based on control communication*: Workers maintain their selected task if they received a reward in the previous step, and change their selected task randomly otherwise.

As this strategy only involves two levels, hence a single abstraction feedback cycle, it does not require any further inter-cycle steps. Listings 2 and 3, included in the Appendix, provide the pseudo-code for the worker and top-manager algorithms.

4.4.2 Blackboard (BB) Strategy

The Blackboard strategy can be configured with two parameters: the random error for the downward propagation of feedback (*random-param*) and the probability of worker task switching (P_{switch}).

The feedback cycle between workers and their parent managers proceeds as follows:

1. *State information collection*: Workers (at L_0) send the $task_{id}$ of their selected task to their parent manager (at L_1);

2. *State information abstraction*: Each parent manager calculates the sum of collected $task_{ids}$, for each task type (*collective-state[]*). The sum is updated upon reception of each $task_{id}$, so collected data is stored directly in abstracted form. This abstraction is also sent (as *individual-state[]*) to the higher-level manager (L_2).

3. *Information processing*: Each manager (L_1) uses the error correction received from its parent manager (at L_2) (via *individual-control[]*) to determine the error correction proportion to send to its workers (*collective-control[]*). All child managers at L_1 that share a parent at L_2 receive the same error correction amount, as they all see the same error displayed onto their shared blackboard (their parent). Knowing the number of children per manager C, each child manager estimates the fraction of the shared error from the parent that it must deal with. It thus divides the shared error by C, adjusts it by a random error of +/-1 (using the *random-param* configuration parameter) and forwards the result to its workers. This randomness avoids the system getting stuck in a cycle where managers send the same feedback to workers that cannot adapt to it (e.g., a worker already performing the task for which the manager is demanding more workers);

4. *Communication of control information*: Workers receive their error correction share (*individual-control[]*), indicating which tasks need to be selected and which ones dropped, for each $task_{id}$;

5. *Adaptation based on control information*: Workers check whether the error correction for their selected task is negative ($individual\text{-}control[task_{id}] < 0$), meaning that fewer tasks of that type are needed. If that is the case, the worker picks one of the other tasks which has a positive error correction ($individual\text{-}control[task_{id}] > 0$); meaning that more tasks of that type are needed. To avoid significant oscillations and divergence, workers only pick a new task with a certain probability (P_{switch}), which is inversely proportional to the number of children per manager C.

The feedback process between children and parent managers, at consecutive levels (L_{n-1} and L_n, $n = 2..N-1$), proceeds as follows:

1. *State information collection*: Child managers (L_{n-1}) send the sum of worker selected tasks, for each $task_{id}$ (*individual-state[]*), to their parent manager (L_n);

2. *State information abstraction*: Each parent manager (L_n) calculates the sum of collected $task_{ids}$, for each task type (*collective-state[]*). The sum is updated upon reception of each child's *individual-state[]* table, so collected data is stored directly in abstracted form. This abstraction is also sent (as *individual-state[]*) to the higher-level manager (L_{n+1}), except for top-managers ($n = (N-1)$);

3. *Information processing*: If the parent manager is the top-manager ($n = N-1$), then for each $task_{id}$ it calculates the error between the amount of selected tasks and the goal for that $task_{id}$ (*collective-control*$[task_{id}]$ = *goals*$[task_{id}]$ − *collective-state*$[task_{id}]$). Otherwise, for mid-managers ($n < (N-1)$), the control feedback from the higher level (L_{n+1}) is employed instead of the goals. The error correction (*collective-control[]*) is calculated in the same way as described above, for managers at L_1 – splitting it more or less equally amongst managers at L_n that share the same parent at L_{n+1}, with some randomness (*random-param*);

4. *Communication of control information*: Child managers receive the control feedback from their parent managers (*individual-control[]*);

5. *Adaptation based on control information*: Managers do not adapt their behaviour to control feedback, they merely process and transmit control information to lower managers and/or workers.

Mid-managers execute the following two extra inter-cycle steps:

- *Inter-cycle abstraction of state information* (between steps 3 and 4): Mid-manager copy their *collective-state[]* into their *individual-state[]* register, without further abstraction;

- *Inter-cycle communication of control information* (between steps 2 and 3): Mid-managers copy their parent's *collective-control[]* into their *individual-control[]* register and use it as input to their information processing action (step 3). It provides the error-to-goals for each $task_{id}$, indicating the number of workers that need to either adopt or drop each task.

4.4.3 Model Strategy

If information is perfect, the Model strategy converges as soon as abstract state information arrives at the top-manager and control feedback arrives back to the workers (i.e., in $N-1$ steps). This applies to any topology (e.g., large $|W|$, high N, and/or unequal numbers of children-per-manager) and any task-distribution goals. At the same time, this approach increases all resource usage proportionally to the number of *children-per-manager*. This increase is further augmented for coordination problems where the inter-children relations are also important (e.g., human organisations [48], featuring polynomial resource increase with C).

In this strategy's data model, for each child-parent pair, we have that the parent's *collective-state*$[child_{id}]$ is the child's *individual-state[]*, and the child's *individual-control* is the parent's *collective-control*$[child_{id}]$. The Model strategy implemented here assumes that information is accurate and that relations amongst workers' selected tasks are irrelevant.

The feedback cycle between workers (at L_0) and their parent managers (at L_1) proceeds as follows:

1. *State information collection*: Workers (L_0) send the $task_{id}$ of their selected task to their parent manager (at L_1) (the worker's *individual-state[]* is a $taskModel$ formatted to have *individual-state*$[task_{id}] = 1$, and the other values set to 0);

2. *State information abstraction*: Managers do not abstract collected information before processing. Each element in a manager's *collective-state[][]* is a $taskModel$ for one of its children (cf. above). After processing, a manager abstracts its *collective-state[][]* register, by summing the tasks of all children, for each $task_{id}$, and storing the resulting $taskModel$ into its *individual-state[]* register;

3. *Information processing*: Each manager at L_1 takes the goals received from its parent manager (L_2) and distributes them exactly to its workers. The sum of all tasks in the received goals equals the number of workers, as managers at L_2 are aware of the number of workers in each of their sub-trees;

4. *Communication of control information*: Workers receive the $newTask_{id}$ they must select next (the worker's *individual-control[]* is a $goalModel$, with *individual-control*$[newTask_{id}] = 1$ and all other values set to 0);

5. *Adaptation based on control information*: The worker selects the new task with $newTask_{id}$.

The feedback process between children and parent managers, at consecutive levels (L_{n-1} and L_n, $n = 2..N-1$), proceeds as follows:

1. *State information collection*: Child managers (L_{n-1}) send to their parent manager (L_n) the sum of worker selected tasks in all their sub-trees, for each $task_{id}$ (*individual-state[]*);

2. *State information abstraction*: Managers do not abstract collected information before processing. Each element in a manager's *collective-state[][]* is a $taskModel$ for one of its children. After processing, a manager abstracts its *collective-state[][]* register, by summing the tasks of all children, for each $task_{id}$, and storing the result as a $taskModel$ into its *individual-state[]* register;

3. *Information processing*: If the parent manager is the top-manager ($n = N-1$), then for each $task_{id}$ it splits the $goal[task_{id}]$ amongst its child managers, in exact proportion to the number of workers of each of its child managers, and places the result in the $goalModel$ of each child manager (*collective-control[][]*). Mid-managers (L_n) do the same, using the goals from their parent manager (L_{n+1});

4. *Communication of control information*: Child managers receive the control feedback from their parent managers, as a $goalModel$ (*individual-control[]*);

5. *Adaptation based on control information*: Managers do not adapt their behaviour to control feedback, they merely process and transmit control information to lower managers and ultimately to workers.

Mid-managers execute the following two extra inter-cycle steps:

- *Inter-cycle abstraction of state information* (between steps 3 and 4): Mid-managers compress their *collective-state[][]* into their *individual-state[]* register, by summing-up the number of workers that perform each task type ($task_{id}$);

- *Inter-cycle communication of control information* (between steps 2 and 3): Mid-managers copy from their parent's *collective-control[][]* register the part concerning them (based on their $child_{id}$), place this information into their *individual-control[]* register and use it as input to their information processing action (step 3). This control input indicates the absolute number of workers managed by this mid-manager that need to achieve each task type ($task_{id}$).

Listings 7 and 8 provide the pseudo-code for the worker and mid-manager algorithms, respectively. The top-manager procedure is the same as the mid-manager one, except for using the system $goals[]$ instead of the *individual-control[]* from a higher manager.

4.4.4 Model Optimisation with Initial Rewards

This strategy, *ModelRew*, combines the Reward and Model approaches. It uses the Reward strategy during the initialisation steps, while feedback from the top-manager is not yet available. After the initialisation steps, the combined strategy swaps to the Model approach, taking the adaptation model into account, and thus converging within the next step. In this way, it reduces errors considerably, while waiting for control feedback. The number of initialisation steps is the same as the number of management levels ($N-1$), unless we introduce further inter-level delays. This combined strategy guarantees that the worker coordination converges within a maximum of $N-1$ steps, while potentially converging earlier (especially for a small number of workers). It also reduces the error to the goal during the initialisation steps (for any topology).

4.4.5 Basic Strategy

This strategy is the simplest of all approaches presented here in terms of the amount of information communicated and stored and the amount of processing performed. It assumes that all entities know the number of children-per-manager, that workers can perform random operations, and that only the top-manager knows the goals. The Basic strategy sends upwards the aggregate number of workers executing $task_1$; and receives downwards the error between the goal and the aggregate of $task_1$. Workers start with a random $selectTask$. They then switch to the opposite task with a probability that is proportional to the overall system error divided by the number of workers (received as control feedback). Mid-managers aggregate their children's tasks (**collective-state**) and send it to their parent manager (**individual-state**). They divide the error received from their parent manager (**individual-control**) equally amongst their children (**collective-control**). The top-manager calculates the difference between the number of workers performing $task_1$ (**collective-state**) and the goal of $task_1$ ($\mathbf{goals}_{task_1}$), providing the result to its children (**collective-control**).

This strategy saves resources by only using one task ($task_1$) to communicate and evaluate their respective states. This works in our current setting as we only employed two task types ($K = 2$). To handle more task types, another aggregate should be found based on a subset of the task types.

The feedback cycle between the workers (at L_0) and their parent mid-managers (at L_1) proceeds as follows:

1. *State information collection*: Each worker sends the $task_{id}$ of its selected task (0 or 1) to its parent manager;

2. *State information Abstraction*: The parent manager processes each collected task id immediately (there is no $task_{id}$ storage) by summing the tasks of all children. The result is stored into its *collective-state[]* register;

3. *Information processing*: No processing is performed by the managers beyond the summation step and workers process control feedback received by computing *individual-control[]/C* which represents their share of the total error in the system;

4. *Communication of control information*: Workers receive the control feedback from their parent managers (*individual-control[]*);

5. *Adaptation based on control information*: Workers switch to the opposite task with a probability that is proportional to the value in *individual-control[]/C*.

The feedback process between children and parent managers, at consecutive levels (L_{n-1} and L_n, $n = 2..N-1$), proceeds as follows:

1. *State information collection*: Child managers (L_{n-1}) send to their parent manager (L_n) the sum of worker selected tasks in their sub-trees (*individual-state[]*);

2. *State information abstraction*: The parent manager processes the information in *collective-state[]* immediately by summing the values reported by each of its children. The result is stored into its *individual-state[]* register;

3. *Information processing*: For the top-manager, the value in *collective-state[]* represents the total number of workers performing task 1. It then computes the difference between the value in *collective-state[]* and the goal value for task 1. The result is stored in *collective-control[]*. For mid-managers, the value in *individual-control[]* is divided by the number of children C and stored in *collective-control[]*;

4. *Communication of control information*: Child managers receive the control feedback from their parent managers, as the error for their sub-trees (*individual-control[]*);

5. *Adaptation based on control information*: Managers do not adapt their behaviour to control feedback, they merely process and transmit control information to lower managers and ultimately to workers.

Mid-managers execute the following two extra inter-cycle steps:

- *Inter-cycle abstraction of state information* (between steps 3 and 4): Mid-managers copy their *collective-state[]* into their *individual-state[]* register, with no further abstraction;
- *Inter-cycle communication of control information* (between steps 2 and 3): Mid-managers copy their parent's *collective-control[]* into their *individual-control[]* register and use it as input to their information processing action (step 3). This control input provides an estimate of the error for the goal of $task_1$.

Listings 9 and 10 provide the pseudo-code for the worker and mid-manager algorithms, respectively. The top-manager procedure is the same as the mid-manager one, except for using the system **goals** (for task 1) instead of the **individual-control** from a higher manager.

4.5 Strategy Resource Analysis

Resource analysis is central to our approach as it allows us to compare coordination strategies and to evaluate the trade-offs between their convergence behaviours and the resources they consume. We describe resource analysis with respect to memory, abstraction complexity, process complexity, and communication. The analysis itself is approximate, as we focus on the order of magnitudes rather than on exact amounts. Memory and message sizes are estimated in bytes, considering that entities store and exchange information as numbers, and allocating one byte for each number. Other units may be used and more bytes per number may be allocated while preserving the relative differences amongst the strategies' resource consumption.

In terms of memory requirements for entities, workers store the $task_{id}$ of their $selectedTask$, and the two registers: **individual-state** and **individual-control**. Mid-managers store four registers: **collective/individual-state** and **collective/individual-control**. Finally, the top-manager stores two registers (**collective-state** and **collective-control**) and two registers for external communication (**goals** and **error-to-goal**). While the size of the registers (**collective/individual-state** and **collective/individual-control**) varies with the strategies, the $task_{id}$ always occupies 1 byte and the **goals** and **error-to-goals** always K bytes.

Hence, the total memory requirements for the multi-scale task-distribution system, for any strategy, is as follows:

$$\begin{aligned} Memory_{tot,t} = |W| * (|\textit{individual-state}_{wrk}| + |\textit{individual-control}_{wrk}| + 1) + \\ |M| * (|\textit{collective-state}_{mng}| + |\textit{individual-state}_{mng}| + |\textit{individual-control}_{mng}| + \textit{collective-control}_{mng}|) + \\ (|\textit{collective-state}_{mng}| + |\textit{collective-control}_{mng}| + 2 * |goals|) \end{aligned} \quad (4)$$

Where wrk stands for worker and mng for manager (both mid and top). In our simulations, each one of the four register types has the same size for all management levels ($L_1..L_{N-1}$).

In terms of communication messages, each entity (at L_{n-1}) initiates the communication towards its parent manager (at L_n), except for the top-manager (at L_{N-1}) who has no parent. Hence, at each step, two inter-level messages are exchanged between each child and its parent. First, the child sends its **individual-state** to its parent, which includes it into its own **collective-state**. Then, the child fetches its parent's **collective-control** and stores it into its own **individual-control**. This means that each child exchanges exactly two messages with its parent, at each time step t ($Messages_{(E_{i,n-1} \rightarrow E_{j,n})} = 2$, $n = 0..N-2$). This gives the total number of messages exchanged in the multi-scale system, at each time step t, as:

$$Messages_{tot,t} = \sum_{n=0}^{N-2} (2 * |E_n|) \quad (5)$$

The total bandwidth required for this communication is:

$$Bandwidth_{tot,t} = \sum_{n=0}^{N-2} [|E_n| * (\textit{individual-state}_n + \textit{collective-control}_{n+1})] \quad (6)$$

Strategy	Role	Memory (B)	Abstraction complexity	Process complexity	Parent comm.(B)	Abstract. Info Loss
Random w. Rewards	worker	≈ 2 (cnst.)	-	O(1)	2 (cnst.)	-
	top-manag.	$\approx 2(K{+}1)$	O(C)	O(C)	-	-
Basic	worker	2	-	O(1)	2	-
	mid-manag.	4	O(C)	O(1)	2	(C-1)
	top-manag.	4	O(C)	O(1)	-	-
Blackboard	worker	$2K$	-	O(K)	$2K$	-
	mid-manag.	$4K$	O(CK)	O(K)	$2K$	(C-1)K
	top-manag.	$4K$	O(CK)	O(K)	-	-
Model	worker	$2K$	-	O(K)	$2K$	-
	mid-manag.	$2(C{+}1)K$	O(CK)	O(CK)	$2CK$	(C-1)K
	top-manag.	$(2C{+}1)K$	O(CK)	O(CK)	-	-

Table 2: Strategy resource requirements (K: number of tasks; C: number of children)

Further considering that in our simulations all registers of managers at all levels have the same size (<*register-size*>), which depends on each strategy, and approximating the workers' registers to the same size, the above equation becomes:

$$Bandwidth_{tot,t} = \sum_{n=0}^{N-2} (2 * |E_n| * <\textit{register-size}>) \tag{7}$$

To evaluate the resource requirements of each coordination strategy, we determine the sizes they utilise for each register type, for both workers and managers. These values can then be inserted in the above equations, together with system-defining parameters ($|W|$, $|M|$, C, N), to determine the total resource overheads of each strategy, for each targeted system.

Table 2 summarizes the resources used by each strategy. The Memory column expresses how much information, in Bytes [B], is stored by each class of entities (workers, mid-managers, and top-managers). Abstraction Complexity, then, reflects the order of magnitude of how complex the abstraction algorithm is (moving information from lower to higher levels). Process complexity provides the order of magnitude of the complexity of the processing action. Parent Comm. provides, in turn, a measure (in Bytes [B]) of the amount of information that is communicated between one child-parent pair, in one simulation step. Abstract Info Loss, finally, measures (in Bytes [B]) how much information is lost in the abstraction from lower to higher levels. Each measure is provided in relation to the number of tasks K and to the number of children C.

To estimate the total resource requirements for each strategy, the values provided in Table 2 for each worker and manager, e.g., for Memory and Communication, can be used as inputs into the system-level equations provided in subsection 4.2 (e.g., Eq. 4 for total system memory and Eq. 5, 6 or 7 for total system communication).

In the Rewards strategy, memory-wise, workers consume one byte of data for their selected $task_{id}$ (*individual-state*) and one byte for the reward (*individual-control*). The top-manager requires one byte of data for incoming requests with a $task_{id}$ (*collective-state*), one byte for the reward (*collective-control*), a table of size K bytes for the goals (fixed) and a table of size K for the rewards (reinitialised at each step). Information abstraction is reflected in the top-manager as the sum of worker requests for each $task_{id}$, subtracted from $rewards[task_{id}]$ - hence O(C) complexity. Reward calculation in the top-manager also features O(C) complexity. Communication between each worker and the top-manager consists of two messages (of 1 byte each), leading to 2 bytes of worker-manager communication (hence a total of $2C$ bytes for the entire system) at each step.

In terms of resource consumption, the Blackboard strategy uses communication registers of size $| < register > | = K$. Hence, memory-wise, workers require $2K$ bytes[2] and (mid- and top-) managers $4K$ bytes.

[2]Worker's memory can be optimised to $K + 1$ bytes if only storing the $task_{id}$ of the *selectedTask* in *individual-state*.

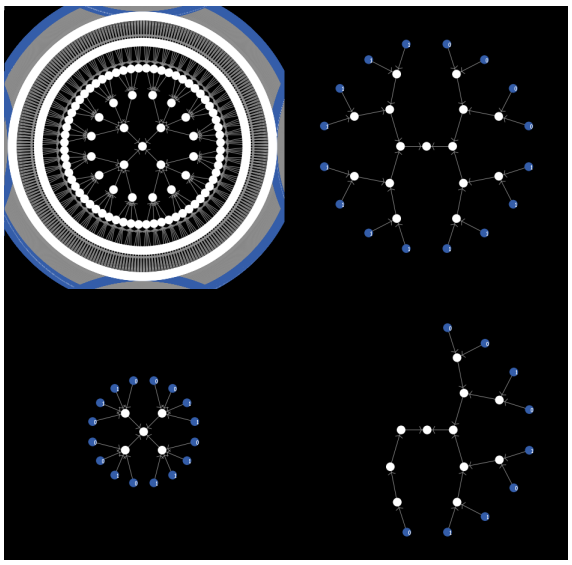
Information abstraction occurs in mid-managers by aggregating data collected from children (of total size $C*K$) into a single *individual-state[]* table (of size K) - hence $((C-1)*K$ less). The abstraction complexity is thus of the order $O(C*K)$ and the processing complexity of $O(K)$ (as iterating through tables of size K). Communication between levels, for each child-manager pair, is around $2*K$ bytes (for *individual-state[]* going upwards and *collective-control[]* coming downwards).

The Model strategy uses storage registers of $|<register>| = K$ bytes for *individual-state[]*, *individual-control[]*, and *goals[]*, and of $|<register>| = C*K$ bytes for *collective-state[]* and *collective-control[]*. Hence, workers require around $2*K$ bytes[3], mid-managers $2*(C+1)*K$ bytes and top-managers $(2C+1)*K$ bytes. Information abstraction occurs within mid-managers as they sum all their children's selected tasks (*collective-state[][]*) into a single *taskModel* (*individual-state[]*) - hence $(C-1)*K$ less. Abstraction and processing complexity is of the order of $O(CK)$ in both top- and mid-managers. Workers feature $O(CK)$ processing complexity. For child to parent communication, workers require $2K$ bytes (for each worker-manager pair) and managers $2CK$ bytes (for each inter-manager pair).

The Basic algorithm has a minimal resource impact (relative to the other strategies) as it only marginally depends on the number of tasks or the number of children-per-manager. Memory-wise, each worker keeps its *selectedTask* (which is also its *individual-state*) and the *individual-control* from its manager, hence 2 bytes. Managers use 4 bytes, one for each of their four input and output tables. Inter-level communication is only 2 bytes (for each child-manager pair) – one for state abstraction going up and one for control feedback coming down. The abstraction complexity is proportional to C, as managers sum-up their childrens' *selectedTask*. Hence, the information loss via abstraction is $C-1$. The processing complexity is constant $O(1)$, as mid-managers simply split their parent's feedback-in equally amongst their children, for $task_1$ only; and the top-manager simply extracts **collective-state** from the goal of $task_1$.

5 Experiments and Results

Figure 4: Four NetLogo configurations. From top left to right (clockwise): $|W| = 4096$ workers & $C = 4$ children per manager; $|W| = 16$ workers & $C = 2$ children per manager; $|W| = 9$ workers & $C = 2$ children per manager; $|W| = 16$ workers & $C = 4$ children per manager



[3]Worker's memory can be optimised to $K+1$ bytes if only storing the $task_{id}$ of the *selectedTask* in **individual-state**.

Experiments were run to compare the performance of the implemented strategies in terms of convergence time, behaviour, and guarantees. Convergence time is measured in number of simulation steps until the error between the goal and the workers' actual task distribution is zero (for a number of steps larger than the delay of control feedback from the top-manager). Convergence behaviour is assessed qualitatively, considering (i) the general shape of the error to goals series decreasing to zero (e.g., monotonically, asymptotically through oscillations) and (ii) how fast the error to goals diminishes before reaching zero. The guarantees are based on predictable, analytical evaluations that can be made on the above aspects. As previously mentioned, the purpose of these experiments is to show the relative qualitative differences amongst strategies with different information flows and resource requirements. Figure 4 shows examples of the NetLogo configurations.

Each experiment was repeated 50 times, at least. For the *Model* and *Rewards* strategies, the results variance between runs was relatively small ($< 10\%$ for *Rewards* and sometimes 0 for Models) so we considered 50 runs to suffice for accurate comparison of average convergence times. For *Blackboard* the variance was more significant (up to 100% for experiments with large $|W|$), yet the convergence time was many times higher than for the *Model* and *Rewards*, so an accurate comparison was unnecessary for the targeted qualitative evaluation. For evaluating the *Basic* strategy with various time delays, 100 runs were used for each experiment to account for result variations. We organise results into five sub-sections: overview of strategy convergence behaviours (5.1), goal variation in task distributions (5.2), topology height variation (5.3), scaling with number of workers (5.4), and timing and delay (5.5).

5.1 Overview of Strategy Convergence Behaviours

Rewards converges monotonically, since at each step more workers select a task that fits the goal, no longer changing it once they have a reward. This behaviour is efficient since we only used two task types ($K = 2$). This means that, at every step, workers have a 50% chance of picking a task type for which they receive a reward, and hence to keep that task. In this case, the error decreases monotonically by about 50% at every step. If the total initial error produced by workers as they select tasks randomly is maximum ($error_{total,0} = |W|$), which is rarely the case, then the *Rewards* strategy's convergence time is around $convergence_{steps} \approx (\log_2 |W| + 1)$ steps (for $K = 2$). This behaviour would be less efficient for larger numbers of task types ($K > 2$). In general, the upper bound of convergence is likely to be around $convergence_{steps} = \text{abs}(\log_{(1-1/K)} |W| + 1)$. For example, for $K = 64$, $convergence_{steps} \approx \text{abs}(\log_{0.985} |W|)$, so over 400 steps for $|W| = 500$ workers. This points to interesting trade-offs that call for further exploration, with the inclusion of more task types.

Model starts with a relatively constant random error in the initial steps, until information is aggregated to the top level and control feedback reaches the workers again ($N - 1$ steps). It then converges within one step. This is possible because we assumed the information model, the control plan, and the communication to be perfect. This means that *Model* is guaranteed to converge within exactly $N - 1$ steps (the number of management levels). *ModelRew* produces the minimum convergence time between *Model* and *Rewards*, because it uses *Rewards* in its initial steps ($N-1$) and *Model* once feedback from the top-manager becomes available. Its main advantage is to reduce the error during the initial steps (like *Rewards* does), while guaranteeing convergence when control feedback arrives (like *Model*).

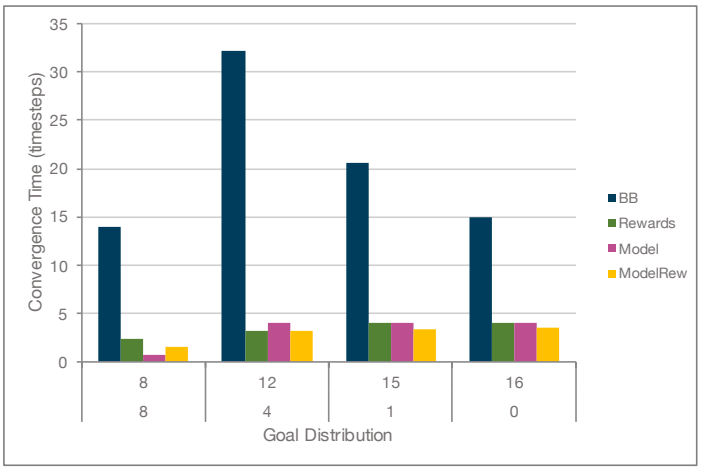
Blackboard, on the other hand, is sensitive to its two configuration parameters that control the level of randomness at the manager and worker levels (i.e., *random-param* and P_{switch}, respectively). Their values impact the strategy's convergence behaviour, leading to (asymptotic) oscillating convergence, monotonic convergence, or no convergence. We did not aim to search for optimal parameters, as the results already show the relative qualitative differences to the other strategies. Rather, we picked parameters that lead to convergence in each tested setting. We used *random-param* values between 0.5 and 0.9; and P_{switch} calculated as $1/(0.5 * |C| + coef)$, where $coef$ was either 1 or 5. When these parameters are set to enable convergence, *Blackboard* reduces the workers' initial error relatively fast (to error-to-goal $\approx$ 10% of goal), then features a longer and longer tail as the error decreases. Hence, aiming to reach a small error (e.g., of 1% or 0.1% of $|W|$), rather than zero error, significantly reduces convergence time, which may suit certain applications.

Finally, the *Basic* strategy features significant random task switching and only converges for small

numbers of workers $|W|$ and/or few management levels N. The convergence is not monotonic. Because of the large randomness in worker behaviour, convergence is highly facilitated when goals are distributed equally between task types.

5.2 Goal Variation in Task Distributions

Figure 5: Average convergence steps for 16 workers, $C = 2$, and different goal task distributions



The first set of experiments compared the strategies' sensitivity to variations in the goal task distribution (i.e., how many workers must perform $task_0$ and how many workers $task_1$). We selected a small scale of 16 workers with 2 children-per-manager ($|W| = 16$, $C = 2$, $N = 5$), allowing to rapidly explore several distributions. Figure 5 summarises the average convergence results for the Blackboard[4], Rewards, Model, and ModelRew strategies. The Basic strategy is not shown in the graph as its convergence for unequal task distributions was several orders of magnitude larger than for the other strategies. The convergence data for all strategies, including the Basic one, is included in the Appendix, under Table 4 (rounded to one decimal).

Generally, as strategies use random task selection – in the initialisation phase and for some strategies also when feedback is available – they all converge faster for equal task-distributions (e.g., $goal_{task0} = 8$ and $goal_{task1} = 8$) than unequal ones (e.g., $goal_{task0} = 16$ and $goal_{task1} = 0$). The convergence time for the *Basic* strategy increases dramatically as goals become more unequally distributed between the task types (i.e., more than 1000 times higher for $goals = [15, 1]$ than for $goals = [8, 8]$); for $goals = [16, 0]$ it does not converge).

Qualitatively, results show that the *Blackboard* strategy is less efficient (up to several times slower) than *Rewards* and *Model*, as well as more sensitive to goal variations in task distributions. As discussed above, these results could be improved for the *Blackboard* strategy by optimising the configuration parameters for each experimental scenario. *Model* always converges after 4 rounds ($N - 1$), except when $goals = [8, 8]$, as here the mid-managers solve the coordination problem without knowing the goals, hence before feedback from the top-manager arrives, by distributing tasks equally between children (by design). *Rewards* offers very similar performance to *Model* and, at times, can be faster. For 16 workers, convergence should occur in about 5 steps if the initial task selection produces the maximum error (16) to the goals, and less otherwise (which is the case in most experiments). *ModelRew* converges within a maximum of $N - 1 = 4$ steps

[4]BB config.: $coef = 1$; and $rand\text{-}param = 0.85$ for $goals = [8, 8]$ and $[12, 4]$; $rand\text{-}param = 0.5$ for $goals = [15, 1]$ and $[16, 0]$

(enabled by the use of $Model$), and sometimes faster (enabled by the use of $Rewards$ in the initial steps). The case where $goals = [8, 8]$ is an exception, as $Model$ is better here.

5.3 Topology Height Variation

Figure 6: Average convergence variation with the number of levels N (or children-per-manager C) - for $|W| = 5000$ workers and $goals = [4000, 1000]$ task distribution

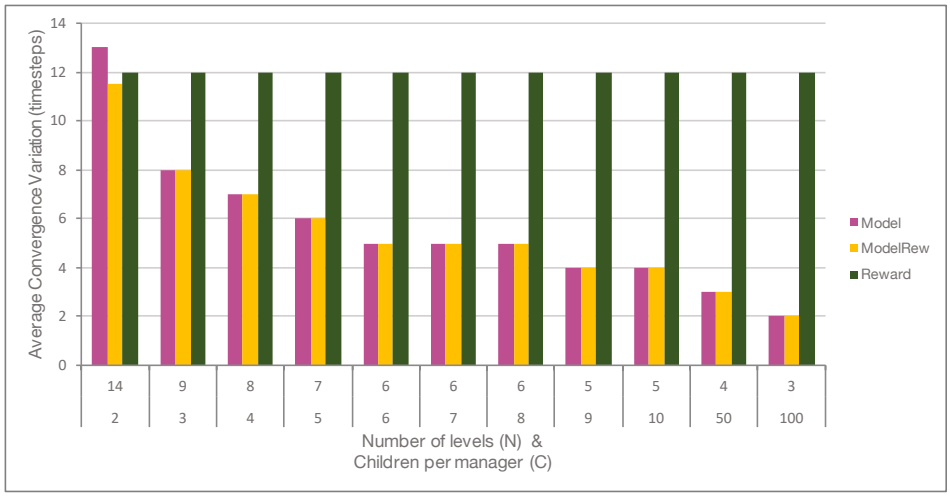
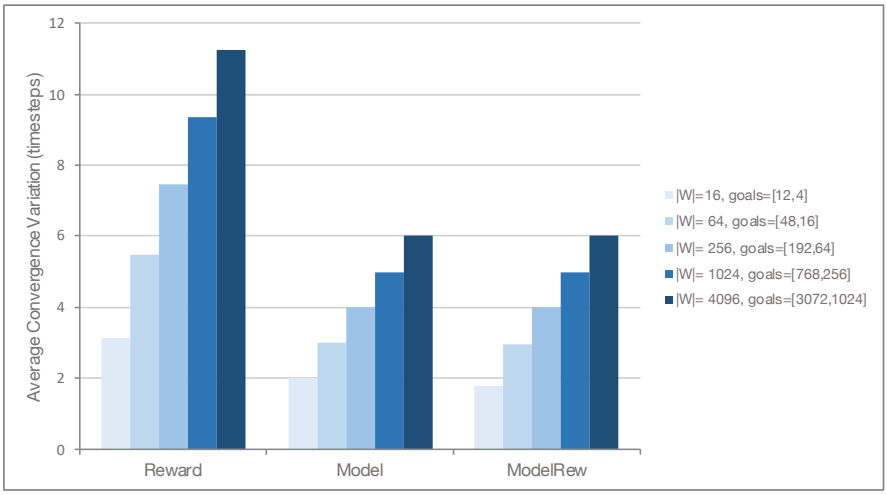


Figure 7: Average convergence variation with the number of workers, $C = 4$ (3 best strategies)



This set of experiments aims to further explore the differences between the $Model$ and $Rewards$ strategies, as in the small-scale experiments these strategies performed similarly. We increased the simulation scale to $|W| = 5000$ workers and picked an unequal task distribution with $goals = [4000, 1000]$. We performed

tests for various numbers of children per manager C, resulting in different numbers of hierarchical levels N. These configurations led to asymmetric topologies, meaning that some parent managers had fewer than the maximum number of children C. Hence, these tests also highlighted the strategies' ability to handle such asymmetries.

Results are shown in Figure 6. *Rewards* is not affected by the number of levels as it always uses a two-level topology. Therefore, its convergence time only depends on the number of workers $|W| = 5000$. In terms of convergence time, $Model$ has the same performance as $ModelRew$ at this scale, except for $|C| = 2$, as before. As expected, the convergence time of these model-based strategies is equal to the number of management levels $(N-1)$. This confers an increasing advantage to these strategies over *Rewards*, at larger scales $|W|$. *Blackboard* is not is included, as it is significantly slower than other strategies. Table 5, in the Appendix, collects average convergence data for all the strategies.

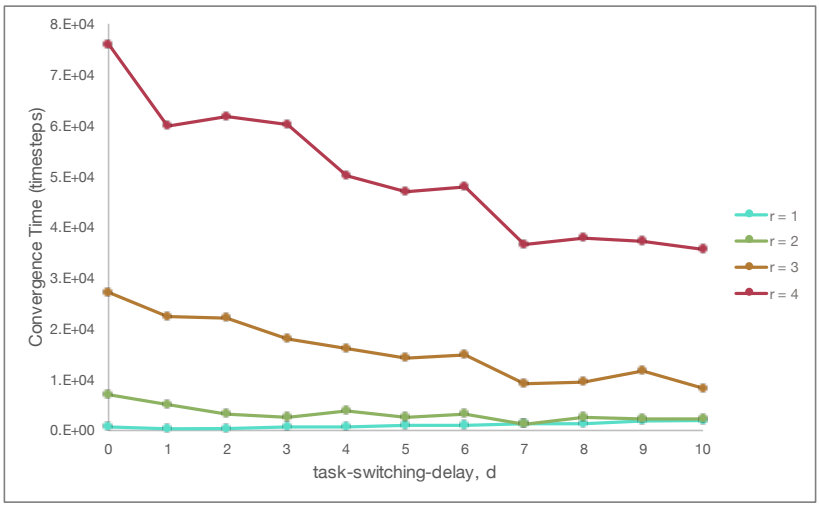
5.4 Scaling with Number of Workers

These experiments aimed to analyse how the strategies scaled with the number of workers coordinating to achieve the goal. We fixed the number of children-per-manager to $C = 4$ and varied the number of workers $|W|$, maintaining a symmetric topology with the same number of children for all managers. We also varied the goals, as to keep the same relative proportion between task types ($goals = [0.25 * |W|, 0.75 * |W|)$.

Figure 7 depicts the results for the best performing algorithms (with full results included in the Appendix, Table 6). As expected, convergence time for $Model$ and $ModelRew$ scales exactly with $\log_C |W| = (N-1)$, which is 4 in these experiments. For *Rewards*, it scales approximately approximately with $\log_2 |W|$. This provides an increasingly significant advantage to $Model$ over *Rewards* for increasing number of workers $|W|$, if the number of levels stays low (with high C values).

5.5 Timing and Delay

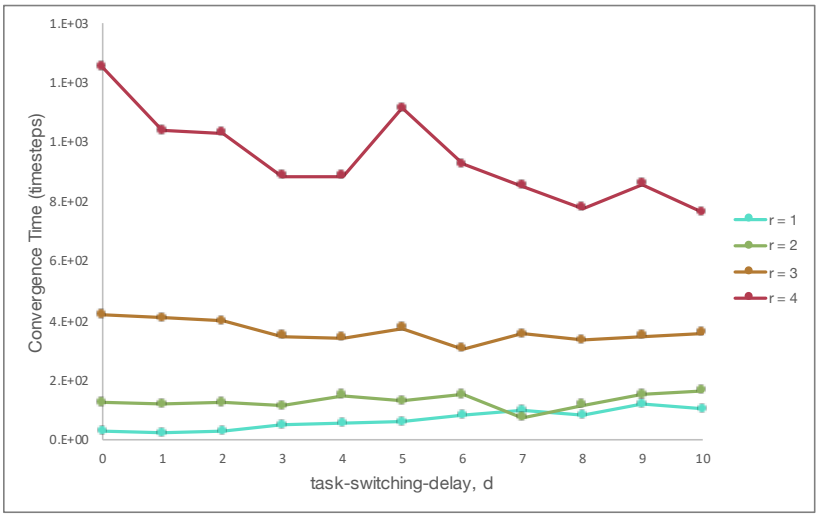
Figure 8: Convergence time versus *task-switching-delay* (d) for four different values of *inter-level-delay* (r) with $goals = [1, 7]$ task distribution, averaged over 100 trials.



The last set of experiments involves the investigation of delays between levels in the hierarchy and the effect of delays in task implementation by the workers. For these experiments, we selected the *Basic* strategy as it has the most worker task switching of all the algorithms implemented, providing a scenario in

Figure 9: Convergence time versus *task-switching-delay* (d) for four different values of *interlevel-delay* (r) with $goals = [4, 4]$ task distribution, averaged over 100 trials.

.



interlevel-delay	Upward Propagation Delay	Total Propagation Delay	Optimal *task-switching-delay*
1	1	4	1
2	8	13	7
3	27	34	20
4	64	73	30

Table 3: Optimal *task-switching-delay* for $goals = [1, 7]$ task distribution with 4 levels and 2 children per manager.

which delays would have a significant impact on convergence time. The main purpose of these experiments was to highlight the fact that interrelated timing parameters (inter-level delays and task-switching delays) are important to overall system behaviour and should be seriously considered during system design and configuration. It was not to evaluate and fine-tune these parameters for the *Basic* strategy in particular. We consider a scenario where the system is imposed with a certain delay between levels of the hierarchy, which may be due to transmission delays or an inherent/engineered slowness in the execution of the higher levels compared to the lower levels (e.g., more processing). The effect of task-switching delays is studied to determine whether a change in worker behaviour can offset delays between levels in terms of overall system convergence time to a desired task distribution.

We define two additional parameters in the system for these experiments:

- *interlevel-delay,* $r \in (1, 2, 3, \ldots)$: The ratio of execution intervals for levels L_n and L_{n-1}. For example, if $r = 1$, the level L_n executes each time level L_{n-1} executes. If $r = 2$, then level L_n only executes after level L_{n-1} executes twice. The top-manager therefore executes r^{N-1} times more slowly than the workers and the overall propagation delay is $r^{N-1} + r(N-1) - (r-1)$.

- *task-switching-delay,* $d \in (0, 1, 2, \ldots)$: The number of time-steps needed before workers take action based on the feedback received from their managers. For example, if $d = 0$, workers take action immediately to switch tasks. If $d = 5$, then workers receive feedback at time step 0, but do not modify

their tasks. They continue to perform their unmodified tasks for time-steps 1..4, and ignore feedback from the managers during these time-steps. Then on time-step 5, they take action to modify their tasks using the feedback received at time-step 0.

The number of children per manager ($C = 2$) and number of levels ($N = 4$) remained constant throughout all tests. The varied parameters were r, d, and task distribution objective (*goals*). For each condition, 100 trials were performed, where each trial sets the worker tasks to a (different) random initial state and executes until the system converges to the desired task distribution with zero error. Due to the randomness of the workers' task switching, it was necessary to set a threshold for the number of time-steps for which the error was zero. Convergence was not declared complete until the system had zero error for a sufficient number of time-steps, due to information propagation implementation delays in the system. The threshold was based on the values set for *inter-level-delay* and *task-switching-delay*.

Figure 8 shows the average convergence time for four different values of *inter-level-delay* ($r = 1, 2, 3, 4$) with $goals = [1, 7]$ task distribution. For $r = 2, 3, 4$ the system shows an overall decrease in convergence time as *task-switching-delay* is increased from 0 to 10. Relative improvement was 55% for $r = 4$ and 72% for $r = 3$ and 4. One case demonstrates worse performance ($r = 1$) with the system becoming slower as *task-switching-delay* is increased over the same interval. For this case, relative degradation was 432%.

Figure 9 shows the same data as Figure 8, except the task distribution was $goals = [4, 4]$. In contrast with the previous data, for $r = 3$ and 4 the system shows an overall decrease in convergence time as *task-switching-delay* is increased from 0 to 10, while the performance decreased for $r = 1$ and 2. Also, the magnitude of the improvement was much smaller. Relative improvement was 32% for $r = 4$ and 18% for $r = 3$. Relative degradation was 18% for $r = 2$ and 485% for $r = 1$.

The difference between the two sets of results above can be explained in the context of the *Basic* algorithm. The workers initialize to a random task with a 50% chance of being assigned $task_0$ or $task_1$. On average, initial task distribution is closer to [4,4] than [1,7] and the system has less switching to perform to achieve the [4,4] goal. As a result, the convergence times are much faster in Figure 9 compared to Figure 8.

The results also show that adding delay to the workers' implementation leads to improvement if the system experiences delays with information propagation up the hierarchy. One associated question is then, to consider how much *task-switching-delay* can be increased before the system's convergence time stops improving.

Figure 10: Convergence time versus *task-switching-delay* for *interlevel-delay* = 4 with $goals = [1, 7]$ task distribution, averaged over 100 trials.

.

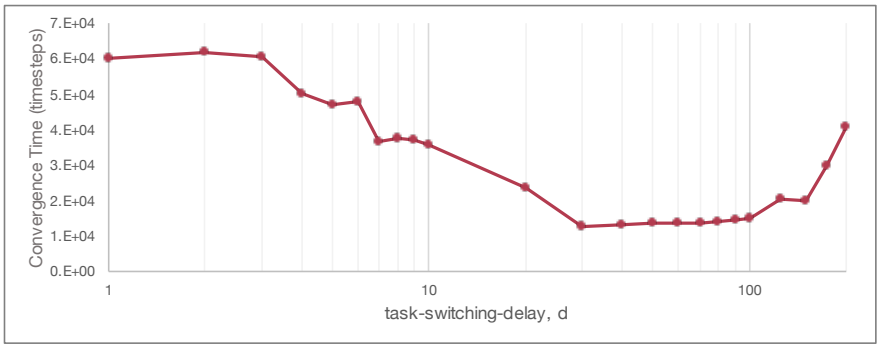


Figure 10 shows the convergence time for the case where $r = 4$ extended to larger task switching delay d values. The system continued to show improvement until $d = 30$, where it achieved a minimum value. This minimum is followed by a slight decrease in performance as d increased from 30 to 100, followed by a significant decrease for values of d greater than 100. These results indicate that there may be an optimum

amount of task switching delay that can offset the level-to-level delays present in a hierarchy. It is not beneficial for workers to act immediately in the presence of delays since they will be acting on manager feedback that does not accurately reflect the state of the overall system at that time. Similar results are seen for the other values of *inter-level-delay*. Estimates of optimal d values for $r = 1, 2, 3, 4$ are shown in Table 3.

6 Discussion

The comparison of different algorithms for large scale coordination allows us to tap into the question of underlying trade-offs in the behaviour of multi-scale systems. Overall, as expected, larger information flows allow for faster coordination convergence, with better guarantees (less variability), in a variety of cases. This comes at the cost of higher resource requirements (i.e., storage, processing, and communication of larger information flows). However, reduced information flows may be efficient, and even better than those using more information, in specific cases for which they can be optimised. This implies niche specialisation – optimising the resource versus efficiency trade-off for specific operational contexts – with the downside of being less adaptable to certain variations in their environment (e.g., different numbers of entities to coordinate or different goal configurations).

If statistical abstraction processes are used to reduce information as it travels from lower to higher scales, then this can help robustness and stability, as low-level information errors and small fluctuations can be leveled-out (on this, see [49]). Conversely, the ability to access and process more detailed information is advantageous for adaptability, if it allows to perceive relevant changes and to take them into account immediately. This comes at the cost of a higher resource utilisation, which may become prohibitive as the number of coordinated entities increases. When resources are limited, the availability of too much (fine-grained) information may be a serious disadvantage, clogging the system, slowing it down, or bringing it to halt. The effectiveness of information abstraction with respect to the targeted coordination problem is key to addressing this challenge.

Reliance upon accurate information and complicated processing increases sensitivity to error (e.g., in information communication, storage, and processing). Hence, information-intensive strategies must invest further resources and further complicate their procedures to ensure fault-tolerance and maintain accuracy and guarantees, or revert to less accurate procedures and more general guarantees. Dealing with new types of information remains an open issue for these strategies, requiring even more complicated processes (e.g., open-ended evolution and/or learning). Generally, the availability of more information does not, in itself, guarantee better coordination. In addition to the issue of resource overload, ineffective processing algorithms, feedback messages, and timing configurations may lead to large oscillations, instability, and divergence throughout the multi-scale system.

These general considerations are exemplified concretely by the strategies we tested. The *Model* strategy uses as much information as needed to guarantee rapid convergence. It is the most resource-intensive strategy, with resource requirements (for the managers) growing proportionally with $C * K$. These resource requirements would potentially grow much faster if inter-worker relations are also considered (e.g., polynomial scaling with C). This pays off in guaranteeing stability and a bounded convergence time, linearly linked to the number of scales N. For multi-scale *Model* systems with a given number of workers $|W|$, designers must consider the trade-off between the hierarchy's height (N) (which can be lowered by increasing the number of children per manager C) and the resource load of each manager (increasing linearly or polynomially with C). Finally, sensitivity to errors and overloading may become a significant issue.

Comparatively, the *Rewards* strategy is much less resource-intensive, with different resources scaling proportionally with either C or K. It is particularly well-suited for small numbers of task types, like in our experiments ($K = 2$). However, even for such a favourable application context, *Rewards* becomes increasingly slower (logarithmically) than *Models* as the number of workers increases and if *Model* uses higher C. *Rewards* also has the advantage of converging monotonically, which may be important for some systems. This strategy, however, does assume that an accurate reward is globally and immediately available to all workers. The *Blackboard* strategy differs in its dependency on system and problem parameters, in that its resource requirements mostly vary with the number of task types K. This is due to the fact

that it only processes and communicates aggregates from all branches, at each scale. Only its information abstraction process depends on the number of children per manager, scaling with $C * K$. As its convergence behaviour highly depends on configuration parameters optimised for each execution context (e.g., topology, goal distribution, number of workers), this strategy fits the general category that features efficient and robust information usage if optimised for its application niche. It is particularly well-suited for coordination problems not requiring absolute accuracy, as it can converge to solutions that are close to the targeted goals (i.e., 1% or 0.1% error) sooner than actually reaching them (i.e., 0% error).

Finally, the issue of timing was addressed by observing the effect of different time delays (inter-level communication delays r and worker task switching delays d) for the *Basic* strategy. As mentioned in the Introduction, timing plays a key role in hierarchical systems, with higher levels of the hierarchy typically operating at slower rates. This behaviour is seen in ecological [11] as well as engineered [39] systems. While it is generally recognised that systems must have slower timescales at higher levels of the hierarchy due to biological, physical, or organisational constraints, it is not clear if there are performance benefits to this type of timing.

The experiments run with the *Basic* algorithm were used to investigate the performance versus timing trade-off specifically for the two-task distribution coordination problem. The results show that in terms of convergence time, the more slowly higher levels of the hierarchy operated, the slower was the system's convergence time. Taken alone, this result indicates that this typical system characteristic does *not* benefit system performance. However, convergence time is not the only performance metric to consider and benefits such as improved transient response (e.g., oscillation or overshooting behaviour) and robustness to changing system goals, reconfiguration (e.g., addition/removal of workers or managers), or disruption of information flow should also be considered. The exploration of these metrics offers exciting avenues for future work.

Given that many types of systems exhibit slower timing at higher levels, and in several cases this timing scheme cannot be modified, the question of whether the system itself can compensate for decreased performance was investigated. In this task distribution problem, this compensation was implemented in the workers as task-switching delay d with the motivation being that workers may receive inappropriate feedback from managers if inter-level delay is present (i.e., managers may provide feedback to workers using outdated information).

In this implementation workers modified their behaviour to ignore manager feedback for a certain number of time-steps so as to not respond to fluctuating instructions and to be more certain of the feedback quality. The results indicate that this type of worker behaviour modification was effective in offsetting the inter-level delays r in that an increase in d resulted in faster convergence times for a given r in many cases. Furthermore, the slower the inter-level communication r and the further the system's initial conditions were from the task distribution goal, the more significant the benefit increasing d. Although not pursued rigorously here, the results seem to indicate that it is possible to determine an optimum value for d given the overall system characteristics and goals. Such a result would benefit system designers who may incorporate compensation techniques into worker algorithms to enable improved system performance.

7 Conclusions

Multi-scale systems are prevalent across observation scales and domains. Understanding their behaviour with respect to information flows and control feedback is central to many pressing applications, from smart grids to task planning and multi-scale governance structures. This study provided key general insights into their main architectural aspects, in terms of information flow cycles and their timing interrelations. As a refinement of the previously developed Multi-Scale Abstraction Feedbacks (MSAF) design pattern, we introduced a five-step feedback cycle to characterize information flows across scales, composed of collection of state information, state information abstraction, information processing, communication of control information, and adaptation based on control information. We added two further actions – inter-cycle abstraction of state information and inter-cycle communication of control information – to connect a two-scale feedback cycle to upper scales, leading to multi-scale feedbacks.

We further extended the MSAF pattern via a reusable approach for analysing and evaluating various

designs and algorithms for multi-scale systems, in terms of convergence performance and behaviour, as well as of associated resource consumption (i.e., information storage, processing, and communication). Results obtained through an agent-based simulation for four exogenous coordination strategies highlighted essential design questions about the effectiveness and efficiency of information abstraction and usage, at each scale, while considering associated resource costs. Experiments also highlighted the impact of different timing delays, and their interrelations, on convergence behaviour. These contributions offer an initial base to analysts and designers of multi-scale systems, for better understanding, evaluating, and controlling existing systems, and for better conceiving new ones.

While this paper focused on analysing coordination mechanisms with exogenous macro-entities, the MSAF design pattern also includes the description of multi-scale systems with composed and micro-distributed macro-entities, building towards a general theory of feedback in multi-scale systems. This study only scratched the surface of the types of questions that could be addressed by such a theory. It raises further significant research challenges to be tackled within each multi-scale problem domain, for example:

- What is the best way to abstract fine-grain information into coarse-grain information that is minimal and yet sufficient for the coordination process, at each scale? In other words, what does the coordination process at each scale need to know about the global state of lower scales?
- How can one determine the most effective and efficient use of available information, which comes at a resource cost (at each scale)?
- How can different types of information abstraction help to render coordination processes more robust and resilient to information errors, and/or more reactive and adaptable to relevant changes?
- How can coordination processes obtain new types of information and learn how to capitalise on them (at each scale)?

8 Acknowledgments

We are very grateful to two anonymous reviewers for their constructive insights. We would also like to thank all the participants of the Complex Systems Summer School (CSSS) of 2018, organised by the Santa Fe Institute (SFI), for all the insightful discussions and ideas related to the presented topic. Finally, we would like to thank Dr. Jean-Louis Dessalles, from Telecom Paris, for lucrative comments on information abstraction.

A Appendix

A.1 Algorithms

For these algorithms, the information registers were renamed into pseudo-code variables as follows:

- **individual-state** becomes *aggregOut*;
- **collective-state** becomes *aggregIn*;
- **individual-control** becomes *feedbackIn*;
- **collective-control** becomes *feedbackOut*;

Algorithm 1 Generic feedback process

User input: number of workers $|W|$; number of hierarchical levels N; worker goal for *task-0*; worker goal for *task-1*; number of children-per-parent C; type of *feedback-subalgorithm*

if *timestep* $== 1$ **then**
  each worker is assigned a random task between 0 and 1
end if

for each worker at level L_0, randomly picked **do**
  worker sends its current state to its parent manager's *aggreg-in*
  worker fetches feedback from *feedback-out* from its parent manager (if available)
  worker computes what do to using *worker-feedback-subalgorithm*,
  ...using as inputs its current state and *feedback-in* (if available)
  worker adapts and updates its current state
end for

for each mid-manager from level L_1 to level L_{N-2}, randomly picked at its level **do**
  mid-manager computes feedback using *midmanager-feedback-subalgorithm*,
  ...using as inputs *aggreg-in* and *feedback-in* (if available)
  mid-manager abstracts information from *aggreg-in* into *aggreg-out*
  mid-manager sends *aggreg-out* to its parent manager
  mid-manager sends feedback to children through *feedback-out*
end for

for the top manager at level $N - 1$ **do**
  if *aggreg-in* is empty **then**
    do nothing
  else
    top-manager computes feedback using *topmanager-feedback-subalgorithm*,
    ...using as input *aggreg-in* and *goals*
    top-manager sends feedback to children through *feedback-out*
  end if
end for

Algorithm 2 Worker algorithm for *Random with Reward strategy*

```
procedure workerRandomWithReward
    if timestep == 0 then                                  ▷ this worker has not executed before
        selectedTask_{t+1} ← random(K)                     ▷ pick a random task (here, either 0 or 1)
    else
        if rewarded_t == false then                        ▷ no reward granted in previous step
            selectedTask_{t+1} ← random(K)                 ▷ pick a random task
        else                                               ▷ reward granted in previous step
            selectedTask_{t+1} ← selectedTask_t            ▷ keep the currently selected task
        end if
    end if
    rewarded ← getReward(selectedTask_{t+1})               ▷ get reward for selected task from parent manager
end procedure
```

Algorithm 3 Top-manager algorithm for *Reward system*

```
procedure initialise Rewards
    for all i ∈ K do
        rewards_i ← goals_i
    end for
end procedure

procedure getReward(selectedTask)
    if rewards_{selectedTask} > 0 then
        rewards_{selectedTask} ← rewards_{selectedTask} − 1
        return true                                        ▷ grant reward
    else
        return false                                       ▷ no reward
    end if
end procedure
```

Algorithm 4 *Worker Blackboard Strategy*

procedure WORKERBB
  $feedbackIn[] \leftarrow feedbackOut[]$ of $myManager$
  if $feedbackIn[] ==$ null **then**
    $selectedTask_{t+1} \leftarrow$ random(K) ▷ pick random task
  else
    $delta_{selectedTask_t} \leftarrow feedbackIn[selectedTask_t]$
    if $delta_{selectedTask_t} < 0$ **then**
      $selectedTask_{t+1} \leftarrow$ getNewTask($feedbackIn[], selectedTask_t$)
    end if
  end if
  $aggregOut[] \leftarrow selectedTask_{t+1}$
  send $aggregOut[]$ to $myManager$
end procedure

procedure GETNEWTASK($feedbackIn[], selectedTask_t$)
  $positiveDeltas[] \leftarrow$ all positive deltas from $feedbackIn[]$
  if $positiveDeltas[] ==$ (null or empty) **then**
    return $selectedTask_t$
  else
    $randomIndex \leftarrow$ random($randMax$)
    if $randomIndex > 0$ **then**
      return $selectedTask_t$
    else
      $newTask \leftarrow task_p$ with a probability proportional to $\frac{postiveDeltas[task_p]}{\sum positiveDeltas[]}$
      return $newTask$
    end if
  end if
end procedure

Algorithm 5 *Mid-manager BB-Basic Strategy*

procedure MID-MANAGER PROCEDURE
  $feedbackIn[] \leftarrow feedbackOut[]$ of $myManager$
  for all $deltaIn_i \in feedbackIn[]$ **do**
    $deltaOut_i \leftarrow$ integer part of $(deltaIn_i / childrenPerManager)$
    $rest \leftarrow$ remainder of $(deltaIn_i / childrenPerManager)$
    $randomValue \leftarrow$ random(100)
    if $(childId < rest)$ and $(randomValue < randomParam)$ **then**
      $deltaOut_i \leftarrow deltaOut_i + 1$
    end if
    $feedbackOut[i] \leftarrow deltaOut_i$
  end for
  send $aggregOut[]$ to $myManager$
end procedure

Algorithm 6 *Top-manager BB-Basic Strategy*

procedure TOP-MANAGER PROCEDURE
  for all $aggreg_i \in aggregIn[]$ **do**
    $deltaOut_i \leftarrow (goals[i] - aggreg_i)$
    $feedbackOut[i] \leftarrow deltaOut_i$
  end for
end procedure

Algorithm 7 *Worker Model Strategy*

procedure WORKERMODEL
  $feedbackIn[][] \leftarrow feedbackOut[][]$ of $myManager$
  if $feedbackIn[][] ==$ null **then**
    $selectedTask_{t+1} \leftarrow$ random(K) ▷ pick random task
  else
    $selectedTask_{t+1} \leftarrow designatedIndex$, where $feedbackIn[childId[designatedIndex]]==1$
    ▷ a single one will be set to 1
  end if
  send $newTaskModel[]$ to $myManager$ ▷ via $aggregOut$
end procedure

Algorithm 8 *Mid-manager Model Strategy*

procedure MIDMANAGERMODEL
  $feedbackIn[][] \leftarrow feedbackOut[][]$ of $myManager$
  for each child c in children **do**
    $workersPerBranch[c] \leftarrow$ the number of workers for the sub-tree of child c
    ▷ based on the child's taskModel ($aggregIn[c]$)
  end for
  if $feedbackIn[][] ==$ null **then**
    set equal goals for each $task_{id}$, with $workersPerBranch[c]$ as the total tasks to perform
  else
    $goalModel \leftarrow feedbackIn[child - id]$
    for each $goal_i$ of $goalModel$ **do**
      for each $child_j$ of children **do**
        if $goal_i < workersPerBranch[child_j]$ **then**
          allocate all tasks in $goal_i$ to $child_j$
          decrement $workersPerBranch[child_j]$ by $goal_i$
          set $goal_i$ to 0 ▷ this goal is now covered
        else
          allocate as many tasks as $workersPerBranch[child_j]$ to $child_j$
          decrement $goal_i$ by $workersPerBranch[child_j]$
          ▷ this goal is only partially covered by this child,
          ▷ the rest will be assigned to the other children
          set $workersPerBranch[child_j]$ to 0 ▷ this child can be assigned no more tasks
        end if
      end for
    end for
  end if
  calculate and send $aggregOut[]$ to $myManager$ ▷ $taskModel$ sum-up of all children $taskModel$s
end procedure

Algorithm 9 *Worker Basic Strategy*

procedure WORKERBASIC
  $feedbackIn \leftarrow feedbackOut$ of $myManager$ ▷ the error for $task_1$
  $randomFloat \leftarrow$ random $\in [0, 1)$
  $equalError \leftarrow feedbackIn/childrenPerManager$
  if $randomFloat < equalError$ **then**
    $selectedTask \leftarrow$ the other task
  end if
  send $selectedTask$ to $myManager$
end procedure

Algorithm 10 *Mid-manager Basic Strategy*

procedure MIDMANAGERBASIC
  $feedbackIn \leftarrow feedbackOut$ of $myManager$
  $feedbackOut \leftarrow feedbackIn/childrenPerManager$
  $aggregIn \leftarrow aggregOut$
  send $aggregOut$ to $myManager$
end procedure

A.2 Result Tables

Strategy	8 $task_0$ 8 $task_1$	12 $task_0$ 4 $task_1$	15 $task_0$ 1 $task_1$	16 $task_0$ 0 $task_1$
Blackboard	14	32.2	20.6	15
Rewards	2.5	3.2	4.1	4
Model	0.7	4	4	4
ModelRew	1.5	3.1	3.4	3.6
Basic	44	352.4	44081	not converging

Table 4: Average convergence steps for $|W| = 16$ workers, $C = 2$, and different goal task distributions

Strategy	N=14 C=2	N=9 C=3	N=8 C=4	N=7 C=5	N=6 C=6	N=6 C=7	N=6 C=8	N=5 C=9	N=5 C=10	N=4 C=50	N=3 C=100
Blackboard	96	125	357	320	159	214	326	117	60	194	64
Rewards	12	12	12	12	12	12	12	12	12	12	12
Model	13	8	7	6	5	5	5	4	4	3	2
ModelRew	12	8	7	6	5	5	5	4	4	3	2

Table 5: Average convergence steps for $|W| = 5000$ workers, $goals = [4000, 1000]$, and various children-per-managers C, leading to different number of levels N

Strategy	$\|W\| = 16$ $goals = [12, 4]$	$\|W\| = 64$ $goals = [48, 16]$	$\|W\| = 256$ $goals = [192, 64]$	$\|W\| = 1024$ $goals = [768, 256]$	$\|W\| = 4096$ $goals = [3072, 1024]$
Blackboard	10.9	32.4	54.5	73.8	163.6
Rewards	3.1	5.5	7.5	9.3	11.2
Model	2	3	4	5	6
ModelRew	1.8	2.9	4	5	6

Table 6: Average convergence for increasing number of workers $|W|$, $C = 4$, and $goals = [0.25 * |W|, 0.75 * |W|]$

References

[1] Kirstie Bellman et al. "Self-improving system integration-status and challenges after five years of SISSY". In: *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS* W)*. IEEE. 2018, pp. 160–167.

[2] Gordon Blair. "Complex distributed systems: The need for fresh perspectives". In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2018, pp. 1410–1421.

[3] Herbert A Simon. "The architecture of complexity". In: *Facets of systems science*. Springer, 1991, pp. 457–476.

[4] Herbert A Simon. *The sciences of the artificial*. MIT press, 2019.

[5] Arthur Koestler. "The ghost in the machine." In: (1968).

[6] Howard Hunt Pattee. *Hierarchy theory*. Braziller., 1973.

[7] Valerie Ahl and TFH Allen. "Hierarchy theory: A vision, vocabulary and epistemology". In: *Journal of Ecological Anthropology* 3 (1999), pp. 85–86.

[8] GAJM Jagers op Akkerhuis. "The operator hierarchy: a chain of closures linking matter, life and artificial intelligence". In: (2010).

[9] Ada Diaconescu et al. "Hierarchical self-awareness and authority for scalable self-integrating systems". In: *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS* W)*. IEEE. 2018, pp. 168–175.

[10] Ada Diaconescu, Louisa Jane Di Felice, and Patricia Mellodge. "Multi-Scale Feedbacks for Large-Scale Coordination in Self-Systems". In: *2019 IEEE 13th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. IEEE. 2019, pp. 137–142.

[11] Timothy FH Allen and Thomas B Starr. *Hierarchy: perspectives for ecological complexity*. University of Chicago Press, 2017.

[12] Jianguo Wu. "Hierarchy theory: an overview". In: *Linking ecology and ethics for a changing world*. Springer, 2013, pp. 281–301.

[13] Sylvain Frey et al. "A generic holonic control architecture for heterogeneous multiscale and multiobjective smart microgrids". In: *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 10.2 (2015), pp. 1–21.

[14] Holger Prothmann et al. "Organic traffic control". In: *Organic Computing—A Paradigm Shift for Complex Systems*. Springer, 2011, pp. 431–446.

[15] Robert I Davis and Alan Burns. "Resource sharing in hierarchical fixed priority pre-emptive systems". In: *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*. IEEE. 2006, pp. 257–270.

[16] Volker Grimm et al. "A standard protocol for describing individual-based and agent-based models". In: *Ecological modelling* 198.1-2 (2006), pp. 115–126.

[17] Volker Grimm et al. "The ODD protocol: a review and first update". In: *Ecological modelling* 221.23 (2010), pp. 2760–2768.

[18] Stanley N Salthe. *Evolving hierarchical systems*. Columbia University Press, 2010.

[19] Francis Heylighen. "Relational Closure: a mathematical concept for distinction-making and complexity analysis". In: *Cybernetics and systems* 90 (1990), pp. 335–342.

[20] Stuart A Kauffman. "Autocatalytic sets of proteins". In: *Journal of theoretical biology* 119.1 (1986), pp. 1–24.

[21] Earl H McKinney Jr and Charles J Yoos. "Information about information: A taxonomy of views". In: *MIS quarterly* (2010), pp. 329–344.

[22] Benjamin Blonder and Anna Dornhaus. "Time-ordered networks reveal limitations to information flow in ant colonies". In: *PloS one* 6.5 (2011).

[23] Henry Tutwiler Wright. *The administration of rural production in an early Mesopotamian town*. 38. University of Michigan, 1969.

[24] Henry T Wright. "Recent research on the origin of the state". In: *Annual Review of Anthropology* 6.1 (1977), pp. 379–397.

[25] Gregory A Johnson. "Organizational structure and scalar stress". In: *Theory and explanation in archaeology* (1982), pp. 389–421.

[26] Herbert A Simon. "Decision-making and administrative organization". In: *Public Administration Review* 4.1 (1944), pp. 16–30.

[27] Richard L Meier. "Communications stress". In: *Annual Review of Ecology and Systematics* 3.1 (1972), pp. 289–314.

[28] Thomas W Malone. "Modeling coordination in organizations and markets". In: *Management science* 33.10 (1987), pp. 1317–1332.

[29] Tom De Wolf and Tom Holvoet. "Design patterns for decentralised coordination in self-organising emergent systems". In: *International Workshop on Engineering Self-Organising Applications*. Springer. 2006, pp. 28–49.

[30] Ada Diaconescu et al. "Hierarchical Self-Awareness and Authority for Scalable Self-Integrating Systems". In: *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS*W), Trento, Italy, September 3-7, 2018*. IEEE, 2018, pp. 168–175. DOI: 10.1109/FAS-W.2018.00043. URL: https://doi.org/10.1109/FAS-W.2018.00043.

[31] Jacob Beal, Jeffrey Berliner, and Kevin Hunter. "Fast Precise Distributed Control for Energy Demand Management". In: *Sixth IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2012, Lyon, France, September 10-14, 2012*. IEEE Computer Society, 2012, pp. 187–192. DOI: 10.1109/SASO.2012.12. URL: https://doi.org/10.1109/SASO.2012.12.

[32] Marco Dorigo, Eric Bonabeau, and Guy Theraulaz. "Ant algorithms and stigmergy". In: *Future Generation Computer Systems* 16.8 (2000), pp. 851–871.

[33] Adrian Bejan and Sylvie Lorente. "The constructal law of design and evolution in nature." In: *Philosophical transactions of the Royal Society of London. Series B, Biological sciences* 365.1545 (2010), pp. 1335–47. DOI: doi:10.1098/rstb.2009.0302.

[34] Ada Diaconescu et al. "Architectures for Collective Self-aware Computing Systems". In: *Self-Aware Computing Systems*. Ed. by Samuel Kounev et al. Springer International Publishing, 2017, pp. 191–235. DOI: 10.1007/978-3-319-47474-8_7. URL: https://doi.org/10.1007/978-3-319-47474-8%5C_7.

[35] Hartmut Schmeck et al. "Organic Computing - A Paradigm Shift for Complex Systems". In: ed. by Christian Müller-Schloer, Hartmut Schmeck, and Theo Ungerer. Springer, 2011, pp. 5–37. DOI: 10.1007/978-3-0348-0130-0_1. URL: https://doi.org/10.1007/978-3-0348-0130-0%5C_1.

[36] Jeff Kramer and Jeff Magee. "A Rigorous Architectural Approach to Adaptive Software Engineering". In: *J. Comput. Sci. Technol.* 24.2 (2009), pp. 183–188. DOI: 10.1007/s11390-009-9216-5. URL: https://doi.org/10.1007/s11390-009-9216-5.

[37] Rogério de Lemos et al. "Software Engineering for Self-Adaptive Systems: A Second Research Roadmap". In: *Software Engineering for Self-Adaptive Systems II - International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*. Ed. by Rogério de Lemos et al. Vol. 7475. Lecture Notes in Computer Science. Springer, 2010, pp. 1–32. DOI: 10.1007/978-3-642-35813-5_1. URL: https://doi.org/10.1007/978-3-642-35813-5%5C_1.

[38] Rodney A. Brooks. "A robust layered control system for a mobile robot". In: *IEEE J. Robotics and Automation* 2.1 (1986), pp. 14–23. DOI: 10.1109/JRA.1986.1087032. URL: https://doi.org/10.1109/JRA.1986.1087032.

[39] W. Findeisen. *Hierarchical Control Systems: An Introduction*. IIASA Professional Paper. IIASA, Laxenburg, Austria, Apr. 1978. URL: http://pure.iiasa.ac.at/id/eprint/923/.

[40] Gordon S. Blair, Thierry Coupaye, and Jean-Bernard Stefani. "Component-based architecture: the Fractal initiative". In: *Annales des Télécommunications* 64.1-2 (2009), pp. 1–4. DOI: 10.1007/s12243-009-0086-1. URL: https://doi.org/10.1007/s12243-009-0086-1.

[41] Sebastian Rodriguez et al. "Holonic Multi-Agent Systems". In: *Self-organising Software - From Natural to Artificial Adaptation*. Ed. by Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes, and Anthony Karageorgos. Natural Computing Series. Springer, 2011, pp. 251–279. DOI: 10.1007/978-3-642-17348-6_11. URL: https://doi.org/10.1007/978-3-642-17348-6%5C_11.

[42] Christopher Landauer and Kirstie L. Bellman. "New architectures for constructed complex systems". In: *Appl. Math. Comput.* 120.1-3 (2001), pp. 149–163. DOI: 10.1016/S0096-3003(99)00240-4. URL: https://doi.org/10.1016/S0096-3003(99)00240-4.

[43] Ada Diaconescu et al. "Goal-Oriented Holonics for Complex System (Self-)Integration: Concepts and Case Studies". In: *10th IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2016, Augsburg, Germany, September 12-16, 2016*. Ed. by Giacomo Cabri, Gauthier Picard, and Niranjan Suri. IEEE Computer Society, 2016, pp. 100–109. DOI: 10.1109/SASO.2016.16. URL: https://doi.org/10.1109/SASO.2016.16.

[44] Arthur Koestler. "Beyond atomism and holism—the concept of the holon". In: *Perspectives in Biology and Medicine* 13.2 (1970), pp. 131–154.

[45] Johannes van der Horst and Jason Noble. "Distributed and centralized task allocation: When and where to use them". In: *2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop*. IEEE. 2010, pp. 1–8.

[46] Ahmad Esmaeili et al. "A socially-based distributed self-organizing algorithm for holonic multi-agent systems: Case study in a task environment". In: *Cognitive Systems Research* 43 (2017), pp. 21–44.

[47] Alejandro Cornejo et al. "Task allocation in ant colonies". In: *International Symposium on Distributed Computing*. Springer. 2014, pp. 46–60.

[48] L. Urwick L. Gulick. "Papers on the Science of Administration". In: *Annual Review of Ecology and Systematics* (1937).

[49] Jessica Flack. "Coarse-graining as a downward causation mechanism". In: *Philosophical Transactions of The Royal Society A Mathematical Physical and Engineering Sciences* 375 (Dec. 2017), p. 20160338. DOI: 10.1098/rsta.2016.0338.