straining the process. Agents directly or indirectly commit to certain actions using a predefined protocol. Agents may join or form an organisation with additional rules.
The norm change definition describes attributes, which are required for Normative Multi-Agent Systems [BPV09]. Ten guidelines for implementation of norms in MAS are given. When norms are involved, agents need to make decisions based on these norms. Rosaria et al. [CCD98] argue that agents have to be able to violate norms to maintain autonomy. However, the utility of certain actions may be lower due to sanctions.
According to [SC11], Normative Multi-Agent Systems can be divided into five categories: norm creation, norm identification, norm spreading, norm enforcement, and network topology. Related approaches use Normative Multi-Agent Systems for governance or task delegation in distributed systems [Sin+13].
To detect the system state, we use social network analysis. All algorithms used for this purpose can be found in [WF94]. A survey of different analysed social networks was done by [New03].

5.3 Goal-oriented Holonic Systems

Ada Diaconescu

About this Section

Why do we need this section?
When the complexity of OC systems rises, special-purpose infrastructure and mechanisms must be provided to deal with the ever-increasing system scale, heterogeneity and frequency of internal and external changes. Engineering solutions that function correctly for relatively small system scales, executing within rather simple environments, may break as soon as the system's internal and external complexity crosses a certain threshold. This section explores the common design properties that can be observed in complex natural systems, justifies their criticality for system viability and robustness, and shows how they can be adopted and applied for building complex artificial systems. These properties are studied from both structural and procedural perspectives.

The content in a nutshell

- **Complexity@Runtime**: Complex OC systems must deal with large numbers of resources, increased diversity, frequent changes, multiple goals and conflicts.
- **Integration@Runtime**: Complex OC systems must be able to (self-) integrate resources discovered opportunistically, within their execution context, in order to achieve stakeholder goals.
- **Engineering support**: Special-purpose engineering artefacts are needed to support the development and maintenance of complex OC systems, ranging from conceptual models and architectures (the focus of this section), to concrete frameworks, platforms and tools (which require further research).
- **Goals@Runtime**: *Goals* provide critical reference points for engineered OC systems, which may undergo a wide range of changes, both internally and externally. A system's goals may also change, hence updating the reference points.
- **Holonic systems**: Complex natural systems feature recursively-encapsulated hierarchical structures. Holons are composed of sub-holons and included into supra-holons, recursively. Holons posses several key properties, such as *semi-isolation* among holons, *abstraction* of a holon's internals with respect to external observers and users, and, suitable *time-tuning* among self-* processes.
- **Goal-oriented holonics**: To manage complexity, OC systems must be endowed with *goal-oriented (self-)integration processes*, organised within a *holonic structure* that preserves the holonic properties of semi-isolation, abstraction and progressive reactivity of self-* functions.

Who should read it?
This section is intended for readers interested in better understanding, developing and/or administering complex OC systems, including sub-activities such as system modelling, analysis, design, evaluation and adaptation. Before reading this section it is preferable to acquire some familiarity with the development of single and collective OC systems, which are context-aware, self-adaptive and self-organising, as discussed in the previous Sections 5.1 and 5.2, respectively.

5.3.1 Introduction and Motivation

Previous sections of this chapter focused on building single context-aware adaptive systems (Section 5.1) and collective socially-aware systems (Section 5.2), respectively. As such systems grow larger, more distributed, highly heterogeneous and dynamic, system engineers and administrators must also deal with non-functional properties such as *scalability*, support for *diversity*, *detection and resolution of conflicting goals*, and the ability to cope with *openness* and with a certain degree of *unpredictable change*.

To achieve this, designers and managers must be able to *(re)integrate* simpler and smaller-scale self-* systems, subsystems and components, into larger-scale, increasingly complex systems-of-systems; rather than (re)building them from scratch each time. Also, ideally, the system (re)integration process would be carried-out dynamically and by the system itself; rather than performed offline and manually. This kind of OC system would aim to *self-(re)integrate* available resources, discovered opportunistically, for achieving its goals, whenever it detects a change in its internal resources, external context, or goals pursued.

Similar challenges occur when different OC systems, potentially belonging to different authorities, are designed separately, in order to achieve different purposes, and subsequently end-up executing in shared environments, where they interfere with each other in unforeseen ways [Tom+16]. Here, dynamic system integration is no longer a design choice for dealing with system complexity, but rather a de facto situation that raises complex challenges to be dealt with, such as contention for resources or conflicting actions on shared resources. From a design perspective, the difference between the two situations – purposeful and accidental system integration – is mostly related to an external actor's 'intention'. Importantly, both cases raise the same challenges mentioned above, which can be dealt with via the same infrastructure and mechanisms.

To facilitate, or even to make possible, the *engineering of complex self-integrating OC systems*, it is necessary to first provide *conceptual models* and *generic architectures* that can support the design processes of such systems. Such models represent a stepping stone towards more concrete and domain-specific engineering artefacts, such as *development methodologies*, *frameworks* and *tools*, which embed essential expertise into reusable platforms, and thus enable less experienced professionals to develop complex OC systems.

This section focuses on presenting such a *conceptual architectural model* for enabling the self-integration of complex OC systems from simpler self-* subsystems and components. The conceptual model relies on a core paradigm – *goal-oriented holonics* – in order to address two important issues. Firstly, it deals with the system's necessity to achieve measurable objectives – by defining system *goals* explicitly (Subsection 5.3.3). Secondly, it deals with the system's necessity to achieve its goals while ensuring scalability, capitalising on diversity, and coping with high-frequency changes – by structuring the system according to *holonic* principles (Subsection 5.3.4).

The section is organised as follows. Subsection 5.3.2 identifies the main challenges to be addressed and the key requirements to be met when modelling, developing and managing complex OC systems. Subsection 5.3.3 discusses the essential role that *goals* play when modelling and (self-)integrating self-* systems. It then proposes a conceptual architecture for goal-oriented (self-)integrating systems, provides a generic formalism for goal definitions, and highlights the main inner workings of a goal-resolution process, including conflict management. To help system (self-)integration as complexity increases, Subsection 5.3.4 identifies some of the key structural and behavioural properties of natural holonic systems (see Subsection 3.3.3) that seem to ensure their viability and survivability within complex competitive environments. The aim here is to then propose these properties as reusable design principles for complex OC systems. Subsection 5.3.5 merges the goal-oriented system models with the holonic design principles, and proposes a novel design paradigm – *goal-oriented holonics* – for multi-level self-integrating systems. A series of examples from the smart micro-grid domain illustrate the use of goal-oriented holonics for designing OC systems at increasing complexity levels. Finally, Subsection 5.3.6 summarises the main concepts introduced and concludes the section.

5.3.2 Challenges and Requirements for Complex Organic Systems

This subsection identifies the key challenges to be addressed when building and maintaining complex OC systems, as well as the essential characteristics of the system (self-)integration processes that can help to deal with such challenges.

It is worth noting here that the purpose of any such engineered systems is to achieve *stakeholder goals*. Here, *stakeholder* is a broad term used to include human developers, owners, users, administrators and other systems – in other words, any entity with some interest and authority over the system in question. Please note that for autonomous systems the stakeholder can also be the system itself, in which case we consider that the system itself sets some *self-goals*. The notion of goal is further developed in the next subsection (Subsection 5.3.3).

In short, engineering solutions for (self-)integrating OC systems that feature increasing complexity must meet additional key requirements, including: scalability; suitable balance between diversity and non-interference of subsystems; abstract description of relevant state-related and behavioural features of subsystems; detection and resolution of multi-goal conflicts; and, careful time-tuning of integrated self-* processes. We further discuss each of these challenges next. Subsection 5.3.5 shows how the generic design paradigm of goal-oriented holonics can contribute to addressing these challenges.

Scalability – manageable complexity from reusable simplicity. As the complexity of OC systems increases, the system development processes are, on the one hand, increasingly based on the (re)integration of existing subsystems, and, on

the other hand, increasingly pushed into the runtime, when adaptation to frequent changes is needed. For simplicity, in the following, we will use the term *integration* broadly, to include both *offline* and *online* integration, as well as *manual* and *automatic* integration; specific usages will become clear from the context or by explicit specification, as needed.

One important question when integrating complex systems is how to find the right combination of primitive system components that will achieve stakeholder goals, in a given context? One option is to follow a trial-and-error technique and to explore all possible combinations of compatible components until finding the optimal one that fulfils the goals. While feasible in principle, such an approach quickly leads to a combinatorial explosion of possibilities when integrating large systems. *Scalability* becomes a critical issue in such cases.

The ability to learn from past experiences and to reuse pre-integrated components that have already proven successful becomes essential in addressing this issue. Hence, rather than only integrating basic components, the system must also be able to reuse pre-integrated subsystems, which can achieve relevant sub-goals, and to integrate these in order to achieve global goals. This implies, among others, the ability to integrate existing self-* systems (like the ones discussed in sections 5.1 and 5.2, respectively). This approach may not find the optimal system integration configuration, yet it is able to find *'satisficing'* solutions – meaning, good-enough solutions found in limited time, even at large scales. Importantly, this helps to address the *limited rationality* problem, which basically stems from the combinatorial explosion of alternatives, as identified by H. Simon in his book on "The Science of the Artificial" [Sim96] (discussed in Section 3.3).

For example, when building an OC system that must achieve two goals simultaneously, it may be easier and faster to integrate two separate OC subsystems, each one able to achieve one of the goals, and, if necessary, to add special-purpose conflict-resolution components; rather than to start from scratch and to assemble basic components until finding the exact combination that meets the two goals, without conflict. Furthermore, the first option also provides more flexibility with respect to future adaptations, where the two subsystems might have to be reused separately, or to be re-integrated differently, in order to consider different goal priorities or to meet an additional goal.

Diversity for reusability, adaptability and robustness. If systems are built by integrating existing subsystems, their overall architecture and control processes cannot be designed starting from a clean slate. Instead, integrated systems must be able to support the interconnection and co-existence of *diverse* subsystems, with specific architectures, control policies, self-* processes, configurations and technologies. Moreover, such diversity of subsystemtems may actually become necessary and beneficial to the overall system. This is both in terms of *optimisation* – since each subsystemtem executes in a different local context, with different environmental constraints, historical preferences and limitations; and of *robustness* – since the system may evolve in unpredictable environments, where a broader range of alternative approaches, as implemented by diverse subsystems, offers a better chance

of dealing with unforeseen conditions, like internal and external disturbances (see Section 4.4 on robustness).

Minimising interference and resolving conflicts. Integrating diverse subsystems risks to create *interference*, or *conflict*, among them. This means that subsystems that can achieve their objectives in isolation, may disturb each-other when coupled too tightly, and so no longer fulfil their objectives in the integrated system [Kep+17].

One category of solutions possible here are based on various types of coordination among the subsystems concerned – e.g. higher-level centralised control, peer-to-peer coordination, or hybrid solutions.

When the number of interfering systems increases dramatically, this type of solutions may reach a scalability limit. This is also exacerbated if systems join or leave the shared environment dynamically, and each subsystem self-adapts over time. To enable further scalability, collectives of well-integrated OC systems may have to be partially isolated from each-other, so as to allow necessary communication while avoiding detrimental interference. This also minimises one subsystem's dependence on the others (i.e. loose coupling), which is essential for maintaining system flexibility, robustness and adaptability. Minimising interference among (collectives of) OC subsystems facilitates their reuse within different higher-level integration configurations and hence helps scalability, as discussed above.

Abstracting system descriptions. As soon as subsystems are successfully integrated into a coherent system, which can achieve well-defined goals, the new system can, in turn, also be integrated with other systems, into an even larger-scale system-of-systems, which can achieve higher-level goals. When this process extends to several integration and abstraction levels, recursively, it can become difficult to achieve coherence and meet goals, while keeping track of the details of design, implementation and configuration of all integrated systems, subsystems and components.

To keep complexity under control – with respect to an external observer's ability to model, analyse, understand and control the system – each system level should be describable via a specific set of *abstractions*. Such external observer may be a human developer, administrator or user; another organic system; or a higher level of the same organic system. In other words, each subsystem should be able to provide a useful abstraction, modelling its state and behaviour, and providing sufficient information to allow the subsystem to be integrated into a higher-level system. Such modelling abstractions include the system's goals, both in terms of goals pursued by the system and goals required from other systems. They also include the evaluation of the system's state and behaviour with respect to the achievement of such goals. Other abstractions can describe the system's negotiation capabilities, self-awareness levels [Lew+17], self-* functions [LMD13], and so on.

Time-tuning self-* processes. When integrating several self-* subsystems, special attention must be given to the *overall dynamics* of the integrated system, in terms of the synchronisation (or timing) of its own self-* (sub-)processes, in order to avoid unwanted oscillations and divergent phenomena, which may occur across several integrated abstraction levels and/or subsystems. More generally, the way in

which lower-level self-* processes influence and interfere with higher-level self-* processes; and, conversely, the way in which self-* processes at higher levels feedback into lower-level self-* processes, must be carefully studied and tuned.

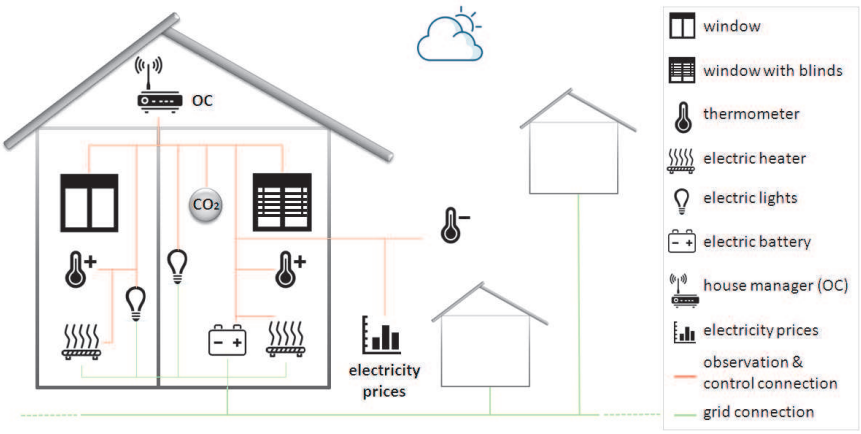


Fig. 5.17: Generic view of the smart home and grid example. Note that the OC House Manager needs not be centralised.

Example: Smart Home and Smart Microgrid

We illustrate the conceptual model introduced here via a case study from the *smart home and smart micro-grid* domain (Fig. 5.17).
In short, smart houses within a neighbourhood are equipped with energy producing and consuming devices, or *prosumers* – e.g. thermostats, lamps, solar panels and batteries. Devices are endowed with self-* processes that enable them to reach simple goals – e.g. temperature, luminosity, or power prosumption. Smart houses are equipped with controllers that coordinate their devices, in order to achieve higher-level goals, such as comfort and power prosumption targets. Smart houses are connected to a smart micro-grid, which must balance power prosumption, avoid peaks, and compensate for any imbalances by prosuming from the main grid.
We use this case study as a running example to illustrate concepts introduced throughout this section (Subsection 5.3.3, Subsection 5.3.4, Subsection 5.3.5). We then provide several concrete use cases that show how these concepts come together into integrated engineering solutions for OC systems with increasing degrees of complexity (Subsection 5.3.5).

For now, let us just briefly illustrate the requirements above via a simple example. Within a smart house, we have a smart window that opens and closes automatically to maintain air freshness within a room. Additionally, we have a thermostat that switches an electric heater on and off, in order to maintain room temperature, while minimising power consumption. While the two devices work well independently, when placed within the same room they may *interfere* with each other, hence causing a *conflict* – e.g. on a cold day, the window may open, in order to air the room, causing the thermostat to struggle to keep-up indoor temperature, while increasing power consumption.
One way to resolve this conflict is to somehow coordinate the actions of the thermostat and the smart window. This may be achieved by introducing a centralised controller; or, by giving the thermostat control over the window; or, via direct negotiations between the thermostat and the window; or, by giving the window access to temperature sensors and rendering it sensitive to the temperature objective.
Additionally, on a warm day, the thermostat may use the window as a way to heat the room without consuming electricity. Here, *diversity* of solutions for reaching the temperature goal help optimise the power consumption goal. In case of a blackout, it would also help pursue the temperature goal, by controlling the windowing based on the weather.
At a higher *scale*, like the entire house, the solution above is *reused*, by having well-coordinated thermostats and smart windows placed in different rooms. Conveniently enough, this provides a straightforward way to *limit interference* among such well-coordinated device collectives, since heat transfer between rooms is slower and smoother (than within a single room). This situation has two important consequences. Firstly, in most cases, the smart devices in one room only perceive the aggregated effects on the local temperature resulting from the actions of devices in other rooms – i.e. *abstraction*. Secondly, the smart devices in one room impact and hence react faster to temperature changes in that room (leading to faster control cycles within each room) than to impacts that devices in other rooms have on the same temperature (leading to slower control cycles across rooms) – i.e. *time-tuning of self-* processes*.
Achieving the above requirements – replication, diversity, abstraction, limited interference, conflict resolution and time-tuning among self-* processes – means that the well-coordinated collective of thermostats and windows can be instantiated, in principle, across any number of semi-isolated enclosures, hence scaling to large surfaces and various dynamic changes.

5.3.3 Goals as Reference Points in Organic Systems

This section discusses the key role of explicit *goal* definitions for the (self-)integrating processes of complex OC systems and proposes a generic design for goal-oriented OC entities. It then introduces high-level goal specifications, transformations and resolution within OC systems, which can be applied and refined across a wide variety of OC systems. It also discusses issues related to interference and conflicts in multi-goal systems. Later on, Subsection 5.3.5 shows the importance of goals in the modelling of interactions among highly diverse subsystems that are (self-)integrated opportunistically.

GOALS AS REFERENCE POINTS IN A WORLD OF CHANGE

When everything within and without an OC system may change, from its internal resources and integration architecture to its external environment, we need to set a *reference point* for the system. Indeed, if a system can self-adapt or self-organise, then what does it do this for? when does it start and stop such processes? and, how does one know whether or not it was successful? In engineered systems, we argue that this reference point is the stakeholder's *goals* for that system.

Therefore, *goals are first class elements in the conceptual model* defined here. This simply means that goals are key notions at the model's abstraction level. For this, firstly, goals need to be defined more clearly (goal specification); and secondly, a method must be provided for achieving them (goal resolution).

Goal specifications have been studied extensively in Software Engineering (SE) to define system requirements [Lam01], [Yu+11], provided and required services, or component interfaces; in Multi-Agent Systems (MAS) to define agent objectives [JDT11]; or, in Artificial Intelligence (AI) to define problems to solve [LB01]. Here, we converge the goal concepts from these areas and propose the following informal high-level definition (reproduced from [Dia+16]).

A goal is an evaluable property that should be achieved, or a verifiable statement that can be deemed true (or not), of a state or behaviour of a system under consideration.

Goals can be defined at different abstraction levels, from high-level statements (e.g. functional or qualitative services, economic targets, social values, constraints, policies, rules and norms) to low-level specifications and ultimately actions (e.g. technical requirements, plans, architectural styles, commands and method calls). Hence, the term goal is here an umbrella label to include a variety of concerns, signifying anything that the system's stakeholder cares about - i.e. the system's 'raison d'être'. Since goals can be made more or less obligatory (e.g. via sanctions or incentives), they can be used to model all forms of action guidance (Section 5.2).

To achieve a high-level goal, an OC system must transform it, progressively, into lower-level goals, and finally into actions performed onto actual resources. This *goal resolution* process is further discussed later in this section.

Example: Smart Home and Smart Microgrid (continued)

Referring to the smart micro-grid case study (introduced in Subsection 5.3.2), goal examples may include: the grid owner's profit targets; the grid manager's safe grid operation; the home owners' comfort, safety, low electricity bill and fairness requirements in terms of prosumption distribution; the smart home controllers' provided services, security, and prosumption constraints; the thermostats' temperature; or the lamps' luminosity targets.
We will later see how we can more formally define these goals and the interrelations among them.

In contrast to engineered systems, natural systems appear to have as their default goal that of their very existence, survival, and derived sub-goals. This default motivation can also occur in some emergent socio-technical systems, which self-organise progressively, over time, by adding individually-motivated links between pre-existing systems, with no prior global purpose. This may be the case, for instance, in systems like global markets or social networks, which behave as systems and have consequent global effects, yet without having been constructed by a particular stakeholder aiming to obtain these effects.

Conceptual Architecture of Goal-oriented Entities

Fig. 5.18 offers a generic view of a *goal-oriented entity* – which, for simplicity, we also refer to as a *Goal-oriented Thing (GoT*[6]*)*. Fig. 5.19 indicates that a GoT may be able to provide and require multiple goals, yet, at any one time, it may only activate a subset of these goals (details below). Fig. 5.20 then shows how such goal-oriented entities (GoTs) can be integrated with one another to form systems, which can in turn be integrated into systems-of-systems, recursively – which we also refer to as *Goal-oriented Things-of-Things (GoTT)*. Each goal and its evaluation is depicted explicitly, via a dedicated input and output port – this is an abstract notation and implies nothing about the goal's actual definition.

We refer here to an *entity*, or *thing*, very generally, to include a broad range of types, with very diverse capabilities and complexity levels: an entire complex OC system, with many stakeholders, goals, self-* functions and components; or, a single system part, implementing a single control loop; or, a 'traditional' component with no self-* capabilities; or, an intelligent self-aware agent; or, a human actor; an entire

[6] GoT: term and acronym inspired from the Internet of Things (IoT), to emphasise the fact that any engineered artefact should have a well-defined goal, and that its known or predicted side-effects should also be identified and documented as unintended goals.

Fig. 5.18: Generic architecture of a goal-oriented entity – or Goal-oriented Thing (GoT)

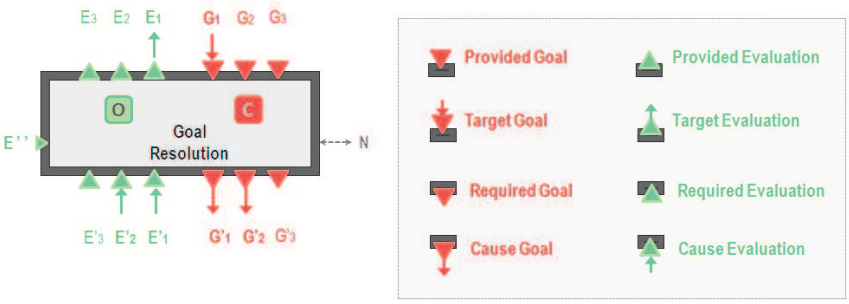


Fig. 5.19: Generic architecture of a multi-goal entity

human organisation; or, any combinations of the above. This architectural view is sufficiently generic to represent the entire range of such entity types. Based on this basic design, an engineered goal-oriented system is a composition of (sub-)systems that fit this design.

Let us now take a closer look at the various types of goal and evaluation ports, their significance and their interrelations, both within one GoT and among several GoTs. Each GoT has a set of *provided goals*, which it can pursue, and a set of *required goals*, which it needs from other GoTs, in order to reach its provided goals. When a GoT *activates* one of its provided goals, as required by a stakeholder, this goal becomes a *target goal* for that GoT. The GoT must then *resolve* the target goal, by finding the actions to perform on internal resources and the required goals to

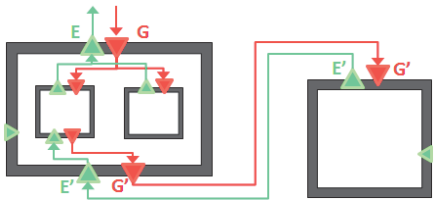


Fig. 5.20: Generic architecture of a goal-oriented system-of-systems – or Goal-oriented Things-of-Things (GoTT)

activate so as to demand goals from external GoTs. Activated required goals become *cause goals* since they can cause the achievement of the target goals. For each of its cause goals, a GoT must find a match in one of the provided goals of another GoT, which, if it accepts the request, activates its provided goal and makes it a target goal. GoT integration is performed in this manner, progressively. Fig. 5.19 shows how each GoT can provide and require multiple goals, from which only a subset are activated and hence become target goals and cause goals, respectively.

This approach is similar to the manner in which integration is performed among objects (in Object Oriented Programming (OOP), such as Java or C++), components (in Component-Oriented Software (COS) [Szy97], such as CORBA Components Model[7], Fractal[8] or Java EE EJBs[9]) and services (in Service-Oriented Architectures (SOA) such as Web Services[10], Spring[11], or iPOJO[12]), possibly at runtime, based on their provided and required *interfaces*. A typical solution to support dynamic binding among such entities is to publish provider entities in well-known repositories where they can be searched for, found, and bound to by client entities, at runtime. These cases are totally compatible with our model, where such interfaces represent a type of low-level goal to be achieved or required by the GoTs in question, which may, amongst others, be implemented via objects, components or services. Finally, while in most of the above technologies *all* required interfaces are either compulsory or optional, GoTs can be more flexible, in that they can rely on different *subsets* of their required sub-goals for achieving their target goals.

Also similar, in Multi-Agent Systems (MAS), agents can find and coordinate with each other, at runtime, based on the tasks they require and provide. The Contract Net Protocol (CNP) [Smi80] is a seminal example for implementing this approach. Here, agents can break composite tasks into sub-tasks, advertise these to the other agents, collect bids from the 'interested' agents and decide on sub-task assignations. More generally, many problem-solving approaches rely on back-tracking

[7] CORBA Component Model (CCM): http://www.omg.org/spec/CCM/

[8] Fractal Project: http://fractal.ow2.org/

[9] http://www.oracle.com/technetwork/java/javaee/ejb/index.html

[10] https://www.w3.org/TR/ws-arch/

[11] https://spring.io

[12] https://felix.apache.org/documentation/subprojects/apache-felix-ipojo.html

strategies that split higher-level problems (or queries) into sub-problems (or sub-queries) recursively, until solutions (or facts) are found to the leaf problems (e.g. Prolog logic programming language, or Problem Posing Interpretation for programming languages [LB99]). Finally, also from the MAS domain, the BDI model [RG91] (Section 9.7) is similar to the proposed approach, with beliefs mapping to knowledge, desires to target goals and intentions to internal actions or cause goals.

For each provided goal, a GoT also provides a means of evaluation for that goal – i.e. *provided evaluation*. Hence, stakeholders can determine, via such provided evaluation, the extent to which the target goal that they required from a GoT has been attained. Similarly, there is a means of evaluation for each required goal – i.e. *required evaluation*. Hence, a GoT can estimate via a *required evaluation* the degree to which the cause goals that it required from other GoTs have been attained.

This generic design focuses on the goal-related port types and the corresponding evaluation-related port types of system entities – hence the Goal-oriented Thing (GoT) labelling. This is because goals have been identified as a key element for enabling system (self-)integration, by matching the provided and required goals of various GoTs, as discussed above. However, this does not limit the port types of a GoT to goal-related ones. Additional port types may include ports for context monitoring, for inter-GoT negotiations, or for the acceptance/entrance and exclusion/departure of composing sub-GoTs (when forming a Goal-oriented Thing of Things, or GoTT).

This generic architecture represents an extension of the observer/controller (O/C) architecture (Section 5.1). This extension aims to support the explicit definition of goal- and evaluation-oriented interfaces[13]. The aim is to go beyond the ability to self-manage a production system, via observation and control, and to capitalise on these functionalities to also enable the (self-)integration of complex OC systems-of-systems. For simplicity, in the following, we sometimes refer to the main composing entity as a system, possibly made of subsystems, and part of larger systems-of-systems.

The process of achieving a target goal – referred to as *goal resolution* – has to be determined at runtime. Goal resolution can take two forms (or a combination thereof). Firstly, a GoT can achieve its target goal by performing actions on its internal resources – i.e. internal action. This includes requiring sub-goals from its composing sub-GoTs, if available. Secondly, a GoT can achieve its target goal by requiring sub-goals from external GoTs – i.e. external action. These, in turn, may require further sub-goals from yet other GoTs, and so on. In such cases, the resolution of a stakeholder goal relies on the goal-oriented integration process among GoTs, based on the matching of their respective required and provided goals. This goal-oriented integration process, both internal and external, leads to the formation of a system-of-systems, or GoTT, that achieves the stakeholder goal. Subsequent evaluation results may re-trigger the same integration process, in order to change the

[13] It is important to understand that the single context-aware system as introduced in section 5.1 necessarily has goals as well; but those are not made explicit and, hence, cannot be changed at runtime by an external entity. Those goals are implicit and distributed, e.g. in the form of the reward assignment on level 1, or the optimisation objective function on level 2.

GoTT composition, e.g., to replace sub-GoTs that fail to achieve targeted sub-goals. When GoTs are autonomous, or self-aware, they may refuse to accept requests for their provided goals, or only accept them partially (negotiation needed) – e.g. due to lack of sufficient resources, or of satisfactory remuneration.

Example: Smart Home and Smart Microgrid (continued)

A smart house controller achieves the home owner's comfort goal by requiring sub-goals (or services) from smart devices, such as thermostats (temperature range), lamps (light ambiance) and multimedia devices (visual and sound ambiance). In turn, a thermostat may achieve its provided temperature goal by requiring further sub-goals (services) from an electric heater (heat production) and a thermometer (temperature measurement). Hence, the overall smart home is a system-of-systems (i.e., GoTT) composed of the main controller and all smart devices, which discover each other dynamically, then provide and require goals from each other.

The (self-)process that integrates (and disintegrates) these subsystems into a system composition that fulfils the stakeholders' target goals is composed of the subprocesses for achieving the target sub-goals of the subsystems involved. These subprocesses typically run in parallel. Each resolution process aims to minimise the difference between its evaluation and its target goals, by requiring and evaluating cause goals from other subsystems, selected at runtime. This, of course, assumes that (sub-)systems are trustworthy, meaning that they do their best to achieve the goals. When dealing with intelligent or self-aware systems [Kou+17b], or in open environments with unknown resources, this assumption may no longer hold (e.g. [Kan+16]). This possibility further complicates goal resolution processes.

Depending on the type and complexity of the system, or entity, involved, such internal resolution processes can range from 'traditional' computations with no self-* capabilities, through basic reflexes (with/out self-learning), and to intelligent and self-aware agent processes (based on knowledge-acquisition, reasoning and/or planning), or humans in the loop. Also, subsystems can be interconnected via various organisation patterns – e.g. hierarchy (Section 3.3), peer-to-peer, or stigmergy) – and/or various types of relations – e.g. cooperation, competition, parasitism, symbiosis or ignorance (please refer to [Dia+17] for a more detailed discussion of such collective options). This variety does not change the nature of the goal resolution process, only its efficiency, and hence can be modelled via a single abstraction.

DEFINING GOALS IN A WORLD OF CHANGE

At the level of abstraction of interest here, goals should be well-defined yet minimal. This means that goal definitions should specify precisely what the stakeholder wants to achieve; and not more. Such goal definition focuses on *what* to achieve rather than *how*, and leaves maximal flexibility to the (self-*) subsystems and their (self-)integration processes.

To make an analogy with Object Oriented (OO) programming, the *goal* concept discussed here is similar to that of an object *interface*. In OO, an interface defines *what* methods are provided by the objects that implement it, and does *not* define *how* these methods are provided. This abstraction serves, among others, to help specify object functionalities and to find compatible objects that can be integrated, via interface matching. When an object uses another object via its interface, then this interface becomes a sort of a contract between the two objects, specifying: for the requesting object, what methods can be called and how to access them; and, for the providing object, what methods it must provide and how they can be called. The actual method implementations are separated from their definitions in the interfaces. They are specified within the classes of objects that implement these interfaces, allowing for different method implementations to be defined in different classes. Beyond this abstract concept of an interface, the exact formalism via which interfaces are defined depends on the programming language, such as C++ or Java. Also, the actual manner in which methods are defined depends on the developed application, and will be different for e-banking and online chatting systems for instance.

Similarly to the interface abstraction, the goal concept as employed here serves to define, at a high abstraction level, system capabilities and requirements in a way that can be used to assess the system's integration compatibility with other systems and to evaluate the efficiency of such integrations. Just as with object interfaces, the exact manner in which such goals are specified will highly depend on the specific application domain and formal language used.

Beyond low-level interface definitions, many formalisms exist for higher-level goal specifications, including modelling languages for agent intentions, abilities, commitments, or desires, such as i* [Yu+11]; requirement engineering models, such as Kaos/Objectiver from Respect-IT[14]; objective specification standards, such as the IEEE-Std-830/1993 standards; rules, policies, constraint-oriented languages; or domain-specific formalisms, such as [DDL12]. All such formalisms can be seen as refinements with respect to the abstract goal model discussed here.

Let us now identify those generic elements that should be present in all goal definitions, irrespectively of their particular formalism and application-dependent specificities. Generally, a goal definition should comprise at least three elements ([Fre+15] for details): $G = (V_f; S_R; S_T)$, defined as follows:

- V_f is a *viability function*, which encapsulates both *what to achieve* and its *evaluation criteria*;

[14] Respect-IT: Requirements Engineering & Specification Techniques for IT

- S_R is a *resource scope*, indicating *where to achieve* it, and hence over which set of resources is V_f defined and evaluated;
- S_T is a *time scope*, indicating *when to achieve* it, and hence over which periods is V_f evaluated.

Example: Smart Home and Smart Microgrid (continued)

A thermostat's goal can be defined as:
$G_T = (20°C - 23°C; home_H; 7pm - 9pm\ daily)$. This means that the thermostat's viability temperature V_f is defined between $20°C$ and $23°C$ and its evaluation function determines the distance between temperature measures and this targeted interval. It also means that the thermostat's V_f is assessed within a home ($S_R = H$) at specified intervals ($S_T =$ every day between 7pm and 9pm).
Note that a viability function V_f could return a binary value, a utility measure, or other semantics, such as 'too hot' or 'too cold' in this thermostat example.

Additional attributes may be included in the goal definition, such as goal *priorities*, in order to help resolve conflicting goals; or *rewards* and *sanctions*, in order to help deal with goal-aware systems that can choose to pursue a goal or not. Many goal formalisms also include goal dependencies, or *links*, in order to define, for instance, the positive or negative contribution of goals upon other goals; logical operations, such as AND, OR, or XOR, among goal links; or other goal-correlation link types, in order to express that goals can satisfy, deny, be dependent upon, or in conflict with, other goals [Lam01]. Moreover, dedicated formalisms are available to indicate the relations between goals and the entities that fulfil them, such as a goal's fulfilment by a particular agent, or agent collective.

Such goal-correlation specifications are compatible with the general model introduced here, yet should be separated from the stakeholder's goal specifications and determined at runtime instead, via the goal resolution process. Hence, target goals are defined by human stakeholders, while cause goals are derived from these – either automatically or via human intervention; directly or via trial-and-error; top-down or bottom-up, or both.

Resolving Goals in a World of Change

For each system (GoT), resolving its target goal(s) G_T implies, in short:

1. Identifying the cause goals G_C that lead to the target goal(s) G_T. As discussed before, these can be internal or external goals, or actions. G_C may be achieved

in whatever order in time, or may be part of a sequenced plan for achieving G_T. If G_C cannot be found, then return to the higher-level stakeholder that issued G_T and indicate the incapability to fulfil G_T; optionally explaining the associated reasons, suggesting necessary actions, or re-negotiating G_T. Otherwise, continue with the following steps for the identified G_C;
2. Finding systems (internal or external GoTs) that provide these cause goals G_C, and sending them goal requests, until finding systems that accept to fulfil G_C;
3. Negotiating or estimating some time for the providing systems to fulfil G_C, then waiting for that time;
4. Evaluating G_C as provided by the systems that accepted to pursue G_C (and going to step 2 if they fail to provide); and
5. Evaluating the extent to which the achievement of the cause goals G_C actually leads to the achievement of the target goals G_T; and going back to Step 1 if failure.

The above processes are propagated recursively, each step identifying further cause goals and the systems that can resolve them. Each system resolves its target goals either by its own means (internal actions on internal resources, or cause goals for sub-GoTs) or by transforming them into further cause goals for other systems (external actions or cause goals for external GoTs); or both (partial internal and partial external actions or cause goals). Hence, within a system-of-systems, or GoTT, the goal resolution process can propagate both horizontally, among GoTs at the same composition level; and vertically, among GoTs at subsequent composition levels (where a GoT maps its target goals to cause goals that are provided by its composing sub-GoTs, which may do the same with respect to their sub-sub-GoTs, recursively; until actual actions are executed on basic resources).

This process may be implemented via various techniques, yet at the level of abstraction of interest here we merely focus on the *goal transformation types* that occur as systems achieve target goals collectively via mutual cause goal requests.

We identify two basic goal transformations, and respective reverse-transformations:

- *translation*, and *inverse-translation*: changing the *type* of one or more of a goal's defining elements – V_f , S_R or S_T;
- *splitting* and *composition*: changing the *value* of one or more of a goal's defining elements – V_f , S_R or S_T.

Often, both translation and splitting operations occur simultaneously during a goal's resolution process. They basically map, or transform, target goal(s) into cause goal(s), by changing their semantics and/or values. The resulting cause goals may or may not be inter-related with respect to the points in time when they must be achieved. For instance, when the translation process of a target goal relies on *planning*, the resulting cause goals are typically inter-related in time – e.g. one cause goal must be achieved before another one can be pursued (planning is a broad subject in itself, and out of the scope of this section; for an example of distributed hierarchical planning please refer to [NWH02]).

Example: Smart Home and Smart Microgrid (continued)

Goal translation: A thermostat's temperature target can be *translated* into an electric heater's power configuration – hence translating the thermostat's viability function (V_f). A generic resource scope (S_R) specified as 'all heat-producing devices within a house H' can be *translated* into 'the set of electric heaters and air conditioning devices available in H'. A broad time scope (S_T) of 'forever' can be *translated* into 'daily' or 'hourly'.
Goal splitting: A thermostat's temperature goal can have its resource scope (S_R) defined initially over an entire house H, and *split* subsequently into several temperature goals, with the same viability temperature V_f, and with reduced resource scopes ($S_{R'}$) representing different parts of the house. Similarly, the time scope (S_T) can be defined initially from 7pm to 9pm daily and split subsequently into two daily intervals, such as S_{T1} between 7pm and 8pm, and S_{T2} between 8pm and 9pm.
Simultaneous goal translation and splitting: A thermostat's temperature goal can have its resource scope (S_R) defined initially as 'all heat-regulating devices' and translated and split simultaneously into two resource scopes S_{R1} = 'all heaters' and S_{R2} = 'all air conditioning devices'.

Goal Resolution within an OC System

Let us now take a more detailed look into how an observer/controller (O/C) system, such as the one depicted in Fig. 5.19, may implement the goal resolution process, based on the concepts introduced so far in this section.

When an external stakeholder R sends a request to an OC system S for one of its provided goals G_P, the system S proceeds as follows, based on its Observer (S_O) and Controller (S_C) functions:

1. The Controller S_C decides whether or not to *accept* R's request for pursuing G_P. This decision is based on the OC system's (S) current state, including current target goals and available resources; and perhaps on non-technical considerations, such as remuneration or reputation gains. If S_C accepts the request, then it activates the provided goal G_P which then becomes a target goal G_T.
2. The Controller S_C *resolves* G_T in the sense that it maps it to the set of cause goals G_C – via various translation and/or splitting operations (Subsection 5.3.3);
3. The Controller S_C *finds* a set of systems P_{GC} that provide the cause goals G_C and requests them to provide G_C; if they accept, then the system S binds to these systems P_{GC} (i.e. dynamic self-integration). Note that the providing systems P_{GC} can be resources of a production system, in which case G_C are actually actions to be performed on those resources; this does not change the general process.

For simplicity, we assume a single cause goal in G_C and a single providing system in P_{GC}.

4. The Observer S_O *evaluates* G_C as provided by P_{GC} and forwards the result to the Controller S_C. If the evaluation is positive, then S_C sends positive feedback, potentially including a reward, to P_{GC}. If the evaluation is negative, then S_C sends negative feedback, including a penalty, to P_{GC}; or, renegotiates G_C with P_{GC}; or, unbinds from P_{GC} and finds alternative providers P'_{GC} for G_C (redoing step 3).
5. The Observer S_O *evaluates* the extent to which the target goal G_T is achieved and forwards the result to S_C; S_O also provides an aggregate evaluation of G_T for R. If the evaluation of G_T is negative despite G_C being provided successfully, then S_C remaps G_T to G'_C (going back to step 2).
6. The stakeholder R may *update* its request for G_P, depending on the evaluation it received (in step 5); the process starts over from step 1.

CONFLICTS OF GOAL COMPOSITION

Goal composition raises the important and challenging issue of goal *conflicts*, and the necessity for *conflict detection* and *resolution* processes, with respect to composed goals. The generic goal definition discussed above can help identify *where* and *when* such conflicts may occur in a multi-goal system. Namely, a goal conflict may arise if composed goals (G_1 and G_2) have incompatible viability functions (V_{f1} *incompatible with* V_{f2}) and overlapping resource scopes ($S_{R1} \cap S_{R2} \neq 0$) and time scopes ($S_{T1} \cap S_{T2} \neq 0$). Such conflicts can arise over those system resources that belong to the resource scopes of both composed goals (where), and within those periods that belong to the time scopes of both composed goals (when). The incompatibility between goal viability functions typically comes down to the necessity of setting different values for the same system parameters. This provides some insight into the form of conflict that can arise (what).

The ability to (pre)identify potential conflicts within a multi-goal system – in terms of *what* the conflicts may be and *where* and *when* they may occur – provides a valuable opportunity to introduce conflict-resolution mechanisms, as suitable for each system. From a purely architectural perspective, such application-specific goal detection and resolution processes may be implemented in a centralised, decentralised or hierarchical manner. Many design patterns are possible for placing the conflict-related logic within a multi-goal system, such as those presented in [FDD12].

Example: Smart Home and Smart Microgrid (continued)

The resource scopes and time scopes of a thermostat's temperature goal (G_{Temp}) and of a power manager's cost-saving goal ($G_{MinCost}$) intersect over

an electric heater. This may lead to incoherent power configurations on that heater, since its power configuration may need to be increased to reach G_{Temp} and to be decreased to reach $G_{MinCost}$.

When a system's own target goals are in conflict with the system's external target goals, the system must either: choose between egoistic and pro-social behaviours, so as to prioritise self goals over collective goals (or vice versa); or, find a 'compromise' goal that meets both self and collective requirements. Finding the appropriate balance between internal and external goals is a challenging and subjective issue, with multiple technical and social implications. For instance, solutions typically take the form of an online multi-objective optimisation, i.e. finding a Pareto-optimum. A key consideration here can be the period of time over which the optimisation should be considered – e.g. short, medium or long term. Indeed, a system may adopt a selfish behaviour to optimise local revenue over the short term, yet this may lead to sub-optimal results on the long term – e.g. tragedy of the commons. Other variations may take into account, or not, collective objectives such as fairness or justice among the systems. Discussing such aspects in detail is outside the scope of this section (see e.g. [JPD15] for a detailed discussion).

5.3.4 Holonic System Structuring with Specific Properties for Complexity Management

Holonic systems were introduced in Section 3.3, based on observations of recurrent structural characteristics in complex natural systems. Such systems feature self-encapsulated hierarchical structures. The main structural entities are represented by holons, which, as wholes, are composed of sub-holons, and also, as parts, belong to supra-holons, recursively.

This subsection aims to identify some of the key properties of holonic structures, as observed in complex natural systems, that seem to be critical to their occurrence, or creation, and to their viability and survival in complex environments [Sim96], [Koe67].

We intend to then find ways to transfer these properties into the engineering process of artificial systems, in order to cope with similar complexities [VBH08]. Hence, in Subsection 5.3.5, we will merge the holonic structuring concepts discussed here with the goal-oriented design principles introduced in Subsection 5.3.3, in order to define a novel paradigm – *goal-oriented holonics*, [Dia+16]–intended to help model, engineer and administer complex systems.

The following structural properties seem to play a critical role in successful system development and adaptation when complexity raises.

VIABLE HOLONIC COMPLEXITY FROM SIMPLICITY

Holonic systems are more likely to achieve *viable* structural and functional complexity, meaning that they can survive and achieve their objectives, than other organisations, since their complexity can be built progressively based on combinations of simpler, viable structures and functions; and rebuilt from intermediate composites, when current formations fail, rather than restarting from scratch (see the *watchmaker's analogy* in [Sim62] or [Sim96]).

In natural systems, evolution only has to 'come up with' stable compositions based on simpler ones, and then find ways to combine these into new composites, recursively. This is much faster than evolving complex systems directly from basic elements. When this leads to dead-ends, existing composites can be dismantled and new alternative composites tried-out at each level; via a partial, progressive roll-back process rather than by restarting from scratch.

In engineered systems, this allows designers to concentrate on one component at the time and to reuse basic components and intermediate composites across systems (e.g. in Object-, Component- and Service-oriented software systems; or in electrical and mechanical systems). Complexity is built by integrating more complicated composites. In OC systems, this can enable the progressive (self-)integration of self-* processes into coherent stable composites.

Example: Smart Home and Smart Microgrid (continued)

The developer of a Smart Home system will design Information and Communication Technology (ICT) applications for controlling electrical devices available in the home. These devices are assumed to be pre-existing and connected into the home's pre-existing ICT and electricity grids. Similarly, control applications are deployed onto pre-existing gateways, or set-top boxes. Moreover, controlled devices might already be equipped with their own controllers – i.e. "smart" devices, like thermostats, light-sensitive window blinds, or electricity price-sensitive washing machines.
In other words, developers do not build Smart Homes from transistors and cables, but rather from pre-existing, stable, well-tested controllable devices, ICT platforms and infrastructure. Similarly, the designers of a "smart" device may not develop the integrated processor themselves, but rather select an available one from the market. Hence, an essential part of Smart Home development is about integrating pre-existing components, some of them already endowed with self-* controls. The integration process involves designing the overall system architecture, selecting compatible devices, and developing control and middleware code for gluing everything together.
Finally, any device that is part of a Smart Home can also operate independently from it. At the same time, a Smart Home may lose its "smartness" if

all controllable devices are removed from it; yet it should still prevail as a Home.

Example: Natural Systems

Let us now also consider an evolutionary example from natural systems. Multi-cellular organisms could evolve from pre-existing uni-cellular organisms; which could form based on pre-existing chemical compounds; in turn based on atoms, such as Carbon (*C*), Oxygen (*O*), Hydrogen (*H*) or Nitrogen (*N*); in turn based on fundamental particles such as quarks. When an organism grows, it does so by self-organising pre-existing chemical compounds found in its immediate environment; rather than basic particles. When it dies, it disintegrates into chemical compounds which can be reused by other growing organisms.

HOLONIC ENCAPSULATION AND SEMI-ISOLATION

A holon's internal sub-holons can be *partially isolated* from the external environment, including other holons or supra-holons. This is achieved via a well-defined border or interface with the environment, creating an identifiable closure around the holon. For instance, the Operator Theory[15] [Akk10] introduces the generic concepts of structural and functional closures, as the defining elements for various kinds of entities, which can self-organise, in a bottom-up fashion, to form hierarchical structures (i.e. successions of pairs of closures of different kinds).

The interactions of such encapsulated sub-holons with the external environment are limited to a well-defined range of exchanges, or in/out-put types. This creates a semi-controlled environment within each holon, diminishing environmental unpredictability for internal entities and hence facilitating successful adaptation to a limited internal environment (with a limited state space) [Akk10]. Of course, this limits the holon's adaptability; and if the isolation is breached, the holon can be corrupted or destroyed. Still, this is the case in most systems, irrespectively of their internal structure.

When holons can function autonomously, in complete isolation, if needed, their robustness and resilience can further improve. Here, a holon can be integrated within

[15] Operator Theory: http://the-operator-theory.wikispaces.com (accessed in November 2016)

a supra-holon (in order to benefit from it, possibly at a cost) when possible, but can also survive as a standalone system when needed (when the supra-holon fails). This allows lower level holons to survive and self-organise into a more suitable supra-holon, rather than restarting from scratch. Finally, isolation also stops cascading failures from propagating through the entire system.

Example: Smart Home and Smart Microgrid (continued)

Smart devices, such as thermostats or light-sensitive window blinds, can operate semi-autonomously and independently of whether they are integrated within a Smart Home or not; as long as their electricity supply is provisioned for. When integrated within a Smart Home, they only interact with their environment via clearly-defined observation probes and control inputs.
Similarly, Smart Homes that are equipped with local electricity generators may operate semi-autonomously within a Smart Micro-grid, interacting with it via power prosumption, and pre-defined observation and control ports. Furthermore, such Smart Home may decide to disconnect itself from the grid altogether in case of major grid disturbances, to avoid damaging internal devices.

Example: Natural Systems (continued)

Organisms are semi-closed chemical systems, considered as individual units of life (notion subjective to an external observer), yet not completely isolated from their environments, with which they exchange energy and various chemical compounds. An organism's interface with its environment, including its skin and external orifices, filters and controls such exchanges, protecting its internals from damaging factors.
Similarly, within a multi-cellular organism, one cell represents an individual autonomous holon, encapsulated within a membrane, which filters the types of chemical compounds that can transit in and out of the cell. This limits the contextual space to which the cell's internal metabolic processes must react to in order to maintain homoeostasis.
In some cases, collective autonomous holons, such as human societies, also create semi-closed borders around themselves; which can be both conventional, like national borders, or physical, like actual gated walls. The aim is to control the influx and/or departure of both human members and physical products, hence limiting disturbances to internal self-organising processes.

HOLONIC ABSTRACTION

Each holon is influenced by other holons only in a coarse manner, via an *aggregate* of their states and behaviours. While encapsulation limits external influence on a holon's internals, abstraction protects external components from the holon's details, which are only exposed in aggregate form (e.g. [MF05]). This means that a holon can use or rely on another holon's aggregate effects, or functions, irrespectively of how these are obtained (from its internals). This can make holons less sensitive to changes in other holons' internals, and render their integration more stable. It also facilitates the development and co-existence of holons with diverse structures and implementations, as only their aggregate effects matter. This helps system robustness, as diversity increases chances of survival in unpredictable environments. It also allows for local optimisations to specific contexts. Finally, it helps external observers to represent and reason about the system, as each holonic level can be specified separately, via the abstractions of its contained holons and their interrelations.

Example: Smart Home and Smart Microgrid (continued)

In a Smart Home, devices are only accessible via special-purpose interfaces for observation and control; their internal designs and operations remain opaque to external entities, and only reflected via their provided functions. The same applies to Smart Homes integrated into Smart Micro-grids, where the grid is only aware of a home's aggregate prosumptions that were not provisioned for internally (e.g. via local generation, consumption or storage). This greatly simplifies the Smart Micro-grid's control operation.

Example: Natural Systems (continued)

We can consider that a multi-cellular organism merely uses its cells via the overall functions they perform, like nerve cells' communication, muscle cells' contractions, or adipose cells' storage functions; and only indirectly depending on the cells' internal structures and processes that support those functions. This means that the organism is less exposed and reactive to all the intricacies and micro-fluctuations happening within its cells, hence simplifying its own self-* processes.

Progressive Reactivity across Holonic Levels

In natural holonic systems, sub-holons typically are more tightly coupled among themselves within a holon – meaning there are more links and/or stronger influences among the sub-holons – than with sub-holons in other holons. The same applies to the holons within a supra-holon, with respect to external holons. Consequently, changes and reactions within a sub-holon propagate faster within the containing holon than between holons. The same applies between holonic levels, with lower levels featuring higher change rates than higher levels.

This property can help limit chain reactions and oscillations through the entire holonic system, since each holon may stabilise after an internal change faster than this change can propagate to other holons. If the holon's stable aggregate state does not change, then no impact is felt on the other holons.

If the holon's aggregate state does change, then the other holons must adapt to it, but only after this new state is stable. Similarly, higher levels only adapt to aggregate changes in the lower levels, once they have stabilised. In some cases, lower levels subsequently detect and adapt to changes in the higher levels, which they have caused in the first place – i.e. causing a *yoyo effect*. Yet, when these dynamics hold, such oscillations occur over longer periods and may not cause major instability.

Example: Smart Home and Smart Microgrid (continued)

Smart devices within a Smart Home should be better interconnected and react faster to changes amongst themselves than with devices from other Smart Homes. Hence, even if the operation and state of one Smart Home may impact the self-* processes of another, this interference should propagate slower than the time it takes for the internal devices to react and re-stabilise.

Example: Natural Systems (continued)

Let us also take a more wide-spread example from the human organisations realm. Such organisations feature hierarchical authority structures. Most enterprises and corporations, members of different groups, or departments, have tightly-coupled relations among themselves, yet are only supposed to communicate with members of other groups via a small number of superiors. Hence, ideas and influences spread faster within each group than among groups. In reality, many human systems are actually organised via multiple overlaid hierarchies, where communication can pass via alternative superiors, or even across members directly, skipping the higher hierarchical levels. This renders the spread of information more efficient across the entire or-

ganisation. The more isolated groups are from each other, the slower the information spreads between them (relatively to internal dissemination).

5.3.5 Goal-oriented Holonics for Complex Organic Systems

This subsection aims to discuss the benefits, implications and recommendations for designing OC systems that can achieve stakeholder goals (Subsection 5.3.3) and rely on holonic properties to manage complexity (Subsection 5.3.4).

Since complex OC systems are engineered for a purpose, merging the goal-oriented paradigm with the holonic structuring principles – hence giving rise to *goal-oriented holonic systems* – can capitalise on the benefits of both approaches. Each (self-)integrated holon has at least one provided goal (plus mere existence by default), which should be achieved via a composition (linear or non-linear) of required goals, in turn provided by its sub-holons (internal) or peer-holons (external) – e.g. in Fig. 5.20, the goal G is provided via two required sub-goals, provided in turn by two sub-holons (internal), and by one required goal G' provided by a peer-holon (external). When holons belong to several supra-holons they may receive conflicting goal requests.

ENGINEERING FOR HOLONIC BENEFITS

Let us now look at how engineers can aim to achieve the holonic properties identified in natural systems in order to build and maintain goal-oriented OC systems.

Complexity from simplicity via explicit goals – This property can be achieved by structuring the OC system as a holarchy, where more complex holons targeting stakeholder goals are composed of simpler holons targeting sub-goals, recursively. This way of compartmenting system functionality into interrelated self-encapsulated components is a well-known ‘divide and conquer’ engineering technique. It is also the source of severe system integration issues, especially when integration is performed during runtime. Making holons goal-oriented, with well-defined provided and required goals, facilitates this task by enabling dynamic goal-based system interconnections (goal matching).

The uncertainties, and their consequences, inherent in the runtime integration of unknown entities can also be minimised, over the medium and long term, via two main functions. Firstly, a system’s goal-evaluation capabilities enable it to determine the suitability of its integration with other systems that provide the goals

they require. Secondly, a system's ability to find and demand required goals from alternative systems enables it to change the providers of its required goals dynamically, when the current ones do not meet the expectations (as indicated by the goal-evaluation functions).

As previously indicated, these mechanisms are similar to those employed in service-oriented architectures (SOA); with two main differences. Firstly, goal-orientation enables, in principle, much richer functions for determining the compatibility between entities, based on their provided and required goals; than service-orientation where compatibility is merely based on interface type matching (sometimes with additional quality and cost-related requirements, via a Service Level Agreement, or SLA). Also, (re)integration mechanisms based on the results of goal evaluations can go beyond mere software exception handling or SLA-breaking penalties, as is often the case in SOA. Secondly, holonic system structuring may serve for reducing the search space of compatible holons, i.e. holons that provide required goals, first to the local scope of the encapsulating supra-holon; and only if no match is found for progressively extending this search space to the scope of other supra-holons (see encapsulation and semi-isolation property below).

Of course, this approach should not be applied to critical systems, such as nuclear plant controllers or autonomous vehicles, where a trial-and-error search for successful goal-oriented system integrations would be mostly unsuitable. We only target systems where dynamic integration was already foreseeable and/or required, in order to propose more generic and richer goal-oriented compatibility matching; and in order to provide more support for self-integration based on goal-evaluations.

Goal-orientation allows for both: i) top-down goal translation and splitting into finer-grain goals, plans and actions; and ii) bottom-up goal composition and (re)definition. Both processes may occur simultaneously, within each holon, and between holons. Bottom-up processes may lead to the formation of smaller-scale subsystems, that provide intermediate goals, which can then shorten top-down resolution processes that require these as their cause goals (the details of goal publication, finding and dynamic matching are out of this section's scope). Each holon evaluates and adapts its internal sub-holon organisation and implementation so as to meet its target goals (e.g. replace failing sub-holons or reorganise). Advanced holons – e.g. intelligent, self-aware holons – may also justify the reasons behind goal failures to their requesters and suggest alternative goals or external help.

Encapsulation and semi-isolation via border control – In engineered systems semi-isolation can be achieved by encapsulating holons into a special-purpose container, or *membrane*, which defines a clear boundary between the holon's internals and its external media [Akk10]. Border control mediates and regulates the holon's inputs and outputs. In the presented conceptual model, these inputs and outputs mainly consist of acceptable incoming and outgoing goal requests and evaluations. Additionally, border control can also regulate the acceptance of sub-holons in and out of the system. The membrane that implements border control can perform additional functions, such as: transforming the holon's target goals into formats suitable for the internal sub-holons; detecting and resolving conflicts among multiple target

goals; and, distributing the resulting non-conflicting target goals to internal sub-holons (further research studies are necessary to establish such membrane-specific processes).

The concept of membrane, or container, is already implemented in many component-oriented technologies, such as CORBA Component Model (CCM), Java EE, .NET, Fractal or iPOJO. Here, the container serves as a protective and functionally-enhanced proxy between the external environment and the internal business logic. External access to the internal component business logic is only enabled via well-defined interfaces. The container also offers non-functional services such as security, transactions and dynamic binding. Hence, the traditional component container concept is compatible with the holonic membrane introduced here, yet lower-level and more limited.

Semi-isolation allows the holon's internals to fine-tune and stabilise their self-integration processes, for target goal resolution, within the space delimited by well-defined provided and required goals, and influences from the execution context. This enables holons to develop diverse internal configurations that best meet the target goals with local resources and within local environments (also taking into account their self-goals).

In open environments, system semi-isolation can prevent the integration of unknown types of entities, which would be unable to self-organise with existing ones; or of unidentified entities that may be malevolent, hence keeping high internal trust levels and a more efficient interaction between accepted members [Ede+16]. Self-organisation techniques such as Trust Communities (see Subsection 8.5.1) can be used to assemble groups of efficient and benevolent trustworthy holons. Surely, if the membrane is breached, these guarantees no longer hold. The holon should take appropriate action, such as reinforcing the membrane; triggering an immune response; signalling to external entities; self-disassembling; and preventing failure propagation.

Abstraction via goals – Goal-oriented holons can be abstracted as entities that reach well-defined goals, in certain contexts, without worrying about how they achieve this. The holon's success or failure, and its usefulness within a supra-holon's organisation, is determined merely by its provided goals and their evaluation. These represent aggregates, or abstract models, of the holon's capabilities, state and behaviour. This goal-oriented abstraction helps human administrators, designers or higher-level observer/controllers to model, analyse and communicate about complex holonic systems, by focusing on one holonic level, in one local context, at the time. It also helps to engineer holonic systems since each holon can be developed and maintained quasi-independently from the others, only taking into account their aggregate goal-based influences. It also facilitates the analysis of inter-holon goal dependencies, to assess the satisfiability of target goals.

The difficult system integration process can, in principle, be automated and moved into the runtime, since the system itself may be able to search for successful holonic compositions via trial and evaluation processes, at increasing holonic levels. While the search process is a complex topic in itself (outside this section's scope)

we re-emphasise that it is much facilitated by holonic structuring, since intermediate composites attaining intermediate goals can be reused as intermediate search results. Combined with the previous feature (semi-isolation via border control), the ability to represent and interact with holons via goal-oriented abstractions is key to supporting interrelations between heterogeneous holons, via standardised goal models. It hence facilitates their integration into coherent supra-holons. This allows dealing with local issues locally, and global issues globally, while (dynamically) finding a balance between the two. In authority-based hierarchies, this helps to balance the power between a holon's supra- and sub-holons [DP14].

Progressive reactivity via tuning of self-* processes – In engineered holonic systems, the self-* processes within holons must be designed and tuned with respect to each other so as to ensure convergence and (at least relative) stability; or at least to avoid major instabilities and divergence.

One way to achieve this is to ensure progressive reactivity among holons and across holonic levels [Sim62]. For instance, goal-requesting holons may obtain goal evaluations (and hence react to these) less frequently than the rate at which goal-providing holons change internally. This can be achieved as evaluations rely on aggregate measures that take longer to collect and compute than the measure-producing processes. This means that the frequency of reactions increases as holons are farther from the roots of the goal dependency graph (i.e. from the initial stakeholder goal). Across holarchic levels, this type of progressive reactivity tuning may imply that supra-holons will react with lower frequencies than lower-level holons. This, assuming that the initial stakeholder goal was given to a supra-holon. Among peer holons (i.e. at the same holarchic level), suitable reactivity tuning may be more difficult to establish; future research is necessary, case by case.

Among reflex systems, progressive reactivity may be achieved by interconnecting a holon's sub-holons via topologies that favour communication and change propagation; and sub-holons in different holons and levels via more seldom links (e.g. community network). This can be facilitated by border control, which limits link formation across the holonic membrane; and hence encourages internal connections. Such tuning should allow self-* processes to stabilise within each holon, before triggering self-* processes in peer-holons and supra-holons. Achieving such dynamic properties is a rich research topic in itself (further research is required); here we merely identify it as a key engineering objective.

Meeting Complex OC System Requirements via Goal-oriented Holonics

Let us now see how the above properties of a goal-oriented holonic system can help address the challenges confronting the development of complex OC systems (Subsection 5.3.2). Table 5.1 summarises the main considerations in this respect.

Holonic properties / OC System requirements	Complexity from simplicity	Encapsulation & semi-isolation	Abstraction	Progressive reactivity
Scalability	Holons represent reusable, pre-integrated subsystems, providing (pre-tested) sub-goals	A holon's internals are only exposed to a fraction of the overall system, via provided and required goals. Sub-holons entering or leaving the holon are filtered. Holon internals do not impact other holons, and their contents, directly	A holon only exposes an abstracted view of its state and behaviour to external entities. At each holonic lecel some information is lost from the underlying level	Tuning the reactivity to change of each holonic level, with respect to the reactivity of upper and lower levels, helps avoid rapid oscillations across the entire holarchy
Diversity	Different compositions of sub-holons may lead to a wide diversity of holons, supra-holons, etc.	Each holon may feature specific internal design, implementation, configuration, policies, ect.	Each holon is viewed externally via an abstract description that hides its internal details; this helps integration of diverse hollons	Holons with diverse implementations can be integrated as long as their dynamics are tuned to avoid instabilities

Minimal interference	More complex holons should only be built from simpler sub-holons if there is limited interference among these, or if their interference results in a desired global effect for the new holon.	A holon' internal changes may be invisible to peer-and supra-holons	Only the aggregate effects of a holon's internal changes are visible to peer and supra-holons	Interference among sub-holons of different holons occurs at a slower pace than interference among sub-holons of the same holon
Abstract system descriptions	A complex system can be described via the abstract description of its root holon(s), with limited or no details about its internals.	Hide holon internal details from external holons	Holons are described via their provided and required goals (and evaluations)	Knowing the location of a holon within a holarchy may provide a hint about its relative reactivity
Time-tuning of self-* processes	Once convergence is ensured within lower-level holons, these can be reused as stable components in higher-level self-* processes (hence to use with progressive reactivity)	Holons do not react to other holons' interal changes that are not observable externally	Having supra-holons only react to aggregates of the state and behaviour of their sub-holons can provide a way to tune self-* processes via progressive reactivity, in cases where aggregate measures take time to compute	In the goal-dependency graph, holons closer to the roots (closer to the stakeholder) should react slower than holons farther away from the roots (closer to resources); additional tuning must be done among peer holons at the same level

Table 5.1: How properties of a goal-oriented holonic design help address the challenges of complex OC systems

In brief, **complexity from simplicity** allows the progressive assembly of reusable entities – supra-holons, made of holons, made of sub-holons, and so on (*scalability*); enabling a wide variety of assemblies (*diversity*); only allows complexity to be incremented at each level if the interference among holons at that level is under control (*minimal interference*); provides progressively abstract descriptions at increasing complexity levels (*abstract system descriptions*); and, only builds more complex holons from (relatively) stable sub-holons (*time-tuning of self-* processes*).

Encapsulation and semi-isolation enables each holon's internals to function without being aware of the entire system complexity (*scalability*); enables each holon to feature different designs and implementations (*diversity*); limits interferences between each holon's internals and externals (*minimal interference*); hides holon internals from external entities (*abstract system descriptions*); and, hide most internal changes from external entities, which thus cannot react to them (*time-tuning of self-* processes*).

Abstraction loses information among levels, hence decreasing observable complexity (*scalability*); helps the integration of heterogeneous holons via standard descriptions or interfaces (*diversity*); only presents aggregate changes of a holon's internals to external entities (*minimal interference*); describes holons via provided and required goals (*abstract system descriptions*); and, can help time tuning among holons at different levels, since aggregated changes may propagate increasingly slowly from lower levels to higher levels of a holarchy (*time-tuning of self-* processes*).

Finally, **progressive reactivity** helps avoid oscillations across holarchic levels (*scalability*); allows the integration of heterogeneous holons, as long as their mutual reactivity does not cause instabilities (*diversity*); decreases the mutual reactivity between holons that belong to different supra-holons (*minimal interference*); may contribute to the description of a holon via its positioning in the reactivity chain *abstract system descriptions*); and, implements time-tuning among holons by having supra-holons react slower than holons, which in turn react slower than sub-holons, and so on (*time-tuning of self-* processes*).

Application Scenarios

We aim to illustrate the goal-oriented holonic concepts introduced here via four example scenarios from the smart micro-grid case study (Subsection 5.3.2). Each example highlights a typical issue that occurs in OC systems and illustrates a conceptual solution based on the proposed model (Subsection 5.3.5, and [Dia+16]). While both the problem addressed and the solution are customised for the concrete example given, they can be generalised and reused across similar cases. We start with a simple example involving a single smart device, and then progressively add complexity by introducing new challenges and corresponding solutions. The generic problems introduced are as follows:

1. Multilayer translation from goals, to rules, to rule-enforcement, to device control;
2. Goal conflict resolution;
3. Top-down facilitation of bottom-up coordination;
4. Bottom-up goal definition and top-down goal enforcement.

The generic solutions provided ground the recommendations we made above to concrete realistic cases. For each example, we 1) offer a brief description and 2) highlight the goal-oriented holonic properties. Goal definitions and transformations are conceptual; the exact formalisms have to be specified case by case. The aim is to illustrate the benefits of the presented modelling method, at a high abstraction level.

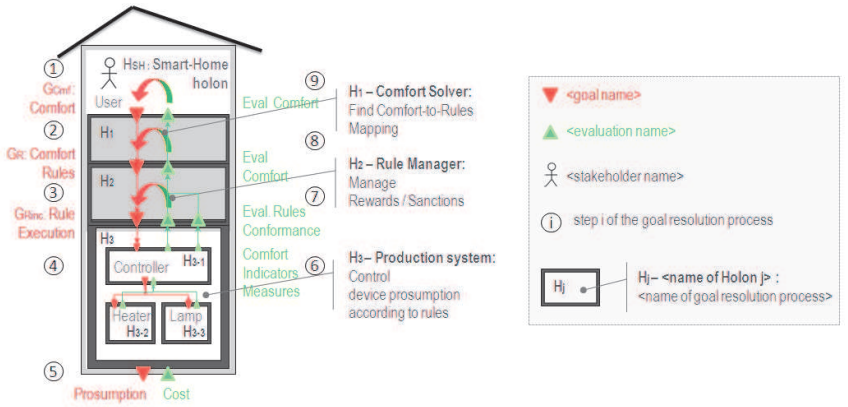
APPLICATION SCENARIO 1. MULTI-LEVEL TRANSLATION OF A SINGLE GOAL WITHIN A SMART HOME

(1) Overview and selected scenario: This first example focuses on a single-goal smart home. It depicts a typical case of progressive multi-level goal transformation – from a high-level stakeholder goal on an entire holonic system, to low-level technical commands and configurations on system resources. Each level is implemented via a distinct self-* process (holon H_i), transforming target goal(s) into cause goal(s), and adapting this transformation dynamically based on feedback from higher and lower levels; and from observations of the environment. The cause goals that a higher level requires become target goals that the lower level provides, recursively; until, at the lowest level, cause goals represent actions on resources. This type of multi-level transformation is likely to occur across a wide range of systems. In short, in a typical case, a high-level stakeholder goal is translated into a set of behavioural rules, which are then enforced into a production system, in order to control resources and achieve the high-level goal.

Let us now see in a little more detail how these different levels of transformation may operate in a typical case (Fig. 5.21). In the first level (H_1), a stakeholder *goal* G (target goal of H_1) is translated into a *set of behavioural rules* G_R (cause goals of H_1). These rules specify how to control an OC production system in order to achieve the target goal G. For instance, if the target goal is *comfort* G_{Cmf} (definition detailed below), then one rule ($rule1 \in G_R$) for achieving G_{Cmf} can be, specified informally, e.g., *rule1: when a person enters a room, if the room is dark, then switch on one light*.

The mapping of the target goal G_{Cmf} into rules G_R can be adapted dynamically, based on feedback. Such feedback can be received dynamically via the two kinds of evaluations that level H_1 has access to.

Firstly, H_1 evaluates the ability of the underlying level H_2 to enforce the rules G_R. For instance, H_2 may indicate that $rule1$ cannot be enforced since no light is available. Here, H_1 can self-adapt and specify a new rule ($rule2 \in G_{R'}$), e.g. *rule2: when a person enters a room, if the room is dark and if there is daylight outside, then open the blinds on one of the room's windows*. The question of *trust* also plays



Step N°	Step Name
①	User sets goal G_{Cmf}
②	H_1 translates goal G_{Cmf} into rules G_R
③	H_2 adds incentives to G_R => G_{Rinc}
④	H_3 manages smart devices using G_{Rinc}
⑤	H_{SH} demands aggregate prosumption from the external grid
⑥	H_2 monitors H_3' conformance to G_{Rinc} and H_3' comfort indicators
⑦	H_2 rewards/sanctions H_3 (which may adapt its control strategy); and/or adapts G_{Rinc}
⑧	H_1 monitors comfort and may adapt G_R if deemed inefficient to reach G_{Cmf}
⑨	User monitors G_{Cmf} and may adapts G_{Cmf} if feedback suggests it is unattainable

Fig. 5.21: Multi-level transformation of goals to rules and actions

an important role here, since H_1 must be able to trust that feedback from H_2 is true. In cases where H_1 suspects that H_2 is trying to mislead it, or is simply incapable of enforcing the rules G_R, H_1 can try to find and bind to an alternative holon $H_{2'}$ that also provides the ability to enforce G_R.

Secondly, H_1 is evaluated and receives dynamic feedback from the stakeholder. For instance, the user may indicate that luminosity is insufficient and hence the comfort goal G_{Cmf} is insufficiently achieved. Provided that G_R is achieved, such negative evaluation of G_{Cmf} indicates that the rules G_R are inefficient for achieving G_{Cmf}. Here, H_1 can again self-adapt to update *rule*2, e.g., *rule2': when a person enters a room, if the room is dark and if there is daylight outside, then open the blinds on all of the room's windows and the glass ceiling*. A critical question here is how to obtain the rules G_R, and their updates G'_R. Options include dialogue with the stakeholder (e.g. [Des16]); swapping between predefined rule sets; inference of new rules from knowledge and predefined meta-rules (e.g. *meta-rule: to increase luminosity, activate more light-increasing devices*; then knowledge on the functions of each device available); combinations of the above; or other. Surely, the risk taken in trying-out untested rules must be evaluated against the estimated criticality of the

service provided and its potential consequences; this is no easy matter indeed (e.g. sometimes it is wiser to allow for suboptimal indoor temperature rather than risk to burn the house down).

The second level (H_2) manages the actual *enforcement* of the behavioural rule set G_R (target goals for H_2). The transformation process of H_2 consists in extending the rule set G_R so as to also specify the *incentives*, such as rewards and penalties, for the OC production system to follow these rules. Hence, the cause goals at this level H_2 are the extended set of rules $G_{R_{inc}}$ that include these incentives. Based on feedback on the effectiveness of the current incentives, these extended rules can be adapted at runtime, by changing the incentive values or by adopting a different rule-enforcement strategy. For instance, if the main Controller $Ctrl_1$ of the OC production system H_3 is not autonomous or self-aware – meaning that it cannot decide on its own whether or not to abide the rules G_R – yet it cannot follow these rules – for example because it does not understand the rule specification syntax – then there is little point in increasing the penalties towards H_3. The better strategy here, would be for H_2 to disconnect from H_3 and find an alternative production system $H_{3'}$. In this case, the penalty for non-conformance can be seen as an elimination from the system; and the reward as a maintenance in the system.

Finally, the third level H_3 controls smart devices by following the rules $G_{R_{inc}}$. In the example in Fig. 5.21 H_3 is in turn composed of a hierarchy of lower-level holons, with a root Controller (H_{3_1}) and several leaf devices (H_{3_2} – Heater, H_{3_3} – Lamp, and so on). H_3 can also self-adapt in order to maximise rewards and minimise penalties from H_2. For instance, to avoid exclusion from the system as in the example above, H_3 can replace the non-compliant Controller $Ctrl_1$ by an alternative one $Ctrl_2$. In case no alternative controller is available, H_3 may attempt to create a peer-to-peer network among its smart devices, provided that these can interpret and follow the rules collectively. H_3 is evaluated by H_2 based on its ability to follow the rules $G_{R_{inc}}$. H_3 also provides additional measurements, like room temperature or luminosity, that can be used as comfort indicators at the higher level H_1.

Fig. 5.21 identifies the main steps of a multi-level goal-translation process. While some of these steps may execute in parallel, we list them here sequentially, by first going top-down through the holonic levels (from H_1 to H_3) – from the User's goal all the way to device controls (i.e. progressive goal translation and splitting) – and then coming back-up (from H_3 to H_1) – from device evaluations all the way to assessing the User goal. Whenever a holonic level evaluates the underlying level, it may accordingly adapt its internal goal-resolution process. This includes the User, who may adapt its initial goal, based on evaluation feedback. The main goal-resolution steps are as follows:

1. A User sets a *comfort goal* for the smart home: $G_{Cmf} = (comfort; home; forever)$. Here, $comfort$ represents the viability function V_f of G_{Cmf} – *what* should be achieved, and *how* to evaluate it. Also, $comfort$ is to be achieved within the *home* – which represents the resource scope S_R of G_{Cmf}. Finally, $comfort$ is to be achieved within the home $forever$ – which represents the time scope S_T of G_{Cmf}. Hence, G_{Cmf} is a target goal for the smart home, which requests it from

its internal holon H_1. Hence G_{Cmf} becomes a target goal for H_1, that is, iff H_1 accepts this requested goal as its target goal;

2. A Comfort Solver (within H_1) translates and splits the comfort goal G_{Cmf} into a set of rules for the home's smart devices: $G_R = (rules; devices; intervals)$. Here, the G_R's viability are the actual rules, the resource scope S_R are the devices to which these rules apply, and the time scope S_T is a set of subsequent intervals over which the rules apply and are evaluated. In order to obtain these rules G_R, the Comfort Solver first maps the comfort goal G_{Cmf} to intermediate goals, such as temperature $G_{Tmp} = (T; rooms; intervals)$, or luminosity $G_L = (L; rooms; intervals)$; and sets the priorities of these intermediate goals (to simplify, these operations are all shown as a single translation from comfort to rules in Fig. 5.21);
3. A Rule Manager (within H_2) is requested to enforce the rules G_R. As detailed above, it extends the rules to $G_{R_{inc}}$ that include specific incentives and sends these to the OC Production System (H_3) for execution;
4. A centralised home Controller within the OC Production System (H_3) manages the prosumption of smart devices to meet the rules $G_{R_{inc}}$ (e.g. [Fre+15]); this controller could also be decentralised, hierarchical, and so on.
5. Device usage (within H_3) results in an aggregated prosumption for the entire house (further discussed in examples 3 and 4);
6. The Controller (H_3) provides comfort indicators, and is also monitored for rule conformance (provided evaluation for H_2);
7. The Rule Manager (H_2) returns rewards and sanctions to the Production System (H_3), based on monitored rule conformance; the Controller (H_3) may adapt its device-management strategy or exclude non-conforming devices accordingly (included in step 4); the Rule Manager (H_2) may also adapt its incentive strategy for H_3;
8. The Comfort Solver (H_1) receives comfort indicators (provided evaluation from H_2) and aggregates them into a comfort evaluation estimate. Based on these indicators, it may then adjust the rules G_R to better achieve the intermediary goals G_{Tmp} and G_L and hence the target goal G_{Cmf} (part of step 2). It also forwards the comfort evaluation to the User, possibly indicating the causes of failures (provided evaluation from H_1);
9. The User may change the goal G_{Cmf} to a more realistic one, based on the evaluation and recommendations s/he receives; or on external factors, like preferences and other contextual pressures.

(2) Goal-based interactions & holonic properties:

Complexity from simplicity is reached here by integrating several goal-oriented self-* loops into the holonic OC system S. Namely, for clarity, we chose in the example above to describe the overall goal-resolution process as a sequence of steps following a top-down path followed by a reverse bottom-up path through the smart-home system. In reality, this process is actually composed of several resolution sub-processes, each implemented as a feedback loop that targets a different (sub-)goal. These self-* sub-processes are implemented by different sub-holons of the OC system S: Comfort Solver H_1, Rule Manager H_2, and Production System H_3 (in turn

composed of a Controller and smart devices sub-sub-holons). These self-* sub-processes execute in parallel and coordinate with each other:

- Input goal from the User (holon) into the OC system S: the User inputs the Comfort goal G_{Cmf} (request) into the OC system and adjusts it based on evaluation feedback (reply) – steps 1 and 9;
- The Comfort Solver H_1: resolves the Comfort goal G_{Cmf} (target) by transforming it into Comfort Rules G_R (causes) and adjusting these based on feedback related to their efficiency from the Rule Manager H_2 – steps 2 and 8;
- The Rule Manager H_2: ensures that the Comfort Rules G_R are fulfilled by imposing a reward and penalty strategy onto the Controller in H_3 and adjusting this strategy based on feedback on its efficiency – steps 3 and 7;
- The Controller H_3: manages the smart devices based on the Comfort Rules G_R and adjusts its control strategy based on the capabilities of devices and on the reward/penalty input from the Rule Manager – steps 4 and 6;
- Output goals of the OC system: the OC system holon S requires an aggregate Prosumption goal G_{Pros} for achieving the User's goal G_{Cmf}; this goal G_{Pros} will be provided by another OC system, as discussed in examples 3 and 4.

Integrating these relatively simple control loops together leads to an overall OC system that can achieve a target Comfort goal (abstract, descriptive, human-oriented) by requiring a Prosumption goal (concrete, prescriptive, technical).

Semi-isolation is achieved within each holon by limiting the kinds of goals it can be required to pursue – namely, G_{Cmf} for the Comfort Solver and well-defined rules G_R for the Rule Manager, and the Controller. Hence, these holons can each optimise their internals for these types of goals only. In the Production System, semi-isolation is also achieved by screening devices before they join the smart-home, to ensure for instance that they are trustworthy and can be managed by the Controller. Furthermore, entities external to the OC system S, such as grid controllers, have no visibility over these internal devices, for privacy preservation; unless special permission is given. Semi-isolation also prevents the internal controllers of the OC System S from interfering directly with external controllers, such as those of another OC System S'. All exchanges with external entities pass through the membrane of S and hence are subject to its border control mechanisms. Among others, this can help prevent cascading self-adaptations, since the failing of a device in a smart home S should not be signalled directly to devices within another smart home S'. External entities, such as S', only become aware of, or impacted by, such internal events within S, indirectly, via the influence that such internal events may have on the aggregated change of the system S, which is visible from S' (see Abstraction below) – e.g. a failure of a device in S impacts the aggregated prosumption of S.

Abstraction is reached by representing each holonic level via its goals and hiding its internal details. Hence, the entire OC system S representing the smart home is seen from the grid only via its aggregate prosumption G_{Pros}; and seen by the user only via its ability to reach the Comfort goal G_{Cmf}. This goal-based abstraction also applies to the system's internal sub-holons, where, for instance, the home Controller is seen by the Rule Manager only via its conformance to rules; the Rule Manager

only via its ability to enforce rules G_R; and the Comfort Solver via its ability to find efficient rules G_R that lead to the Comfort goal G_{Cmf}. This facilitates the modelling and management of each level. For instance, the Controller uses knowledge on how well devices achieve goals and not on how they achieve them. This also allows each holon to reach its goals via a specific internal organisation, favouring diversity and local optimisation.

Progressive reactivity is tuned so that higher-level self-* loops, such as the User updating G_{Cmf}, are slower than intermediate goals, such as the control loops that translate G_{Cmf} to G_R; and then to rule enforcement. These are in turn slower than lower-level control loops, like the ones performing device management. Indeed, if the user updated G_{Cmf}, based on perceived temperature, faster than thermostats could reach a comfortable temperature, then oscillations may occur. Also, when devices of known types – like temperature or luminosity regulation equipment – join or leave the home, only the Controller reacts to take them into account – meaning, to detect and to manage them. The rest of the OC system S remains unchanged, unless devices cause changes that are visible at the aggregate level, like breaking the rules (in which case the Rule Manager intervenes, while the Comfort Solver may still remain unaware of such events).

Application scenario 2. Multi-goal conflict within a smart home

(1) Overview and selected scenario: This example introduces into the previous OC system S an additional target goal, which is requested by an external stakeholder and creates a *conflict* with an internal goal. For instance, based on example 1, the new target goal can represent the external power prosumption rules G_{PR}, which the grid manager imposes in order to avoid blackouts. These external rules G_{PR} cause a conflict with the comfort rules G_R (since they impose incompatible prosumption requirements on the smart devices).

Note that by conflicting with G_R, the external rules G_{PR} also cause an indirect conflict with G_{Cmf} (since G_R cause the achievement of G_{Cmf}). However, in principle, G_{PR} do not necessarily need to conflict with G_{Cmf} – for instance, in cases where G_{Cmf} can be achieved via an alternative set of rules $G_{R''}$ that require limited power consumption – e.g. using oil-based heating to maintain temperature, and using sunlight and gas lamps to ensure luminosity.

Fig. 5.22 generalises this specific case to External Rules and Internal Rules, in order to only focus on *where* the conflict is addressed from an architectural outlook. The Conflict Resolution holon in Fig. 5.22 is inserted in-between the Comfort Solver holon H_1 and the Rule Manager holon H_2 that were designed in example 1 (Fig. 5.21), in order to produce a set of coherent rules G_{Rc} for the Rule Manager. Fig. 5.23 depicts a simplified view of the example in Fig. 5.21 and shows how the additional conflict-resolution holon H_{CR} is inserted in-between H_1 and H_2.

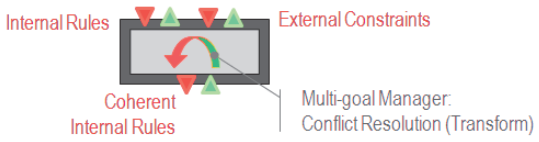


Fig. 5.22: Conflict resolution between target goals – where goals can be internal and/or external, and can represent high-level objectives, rules, constraints, policies or actions

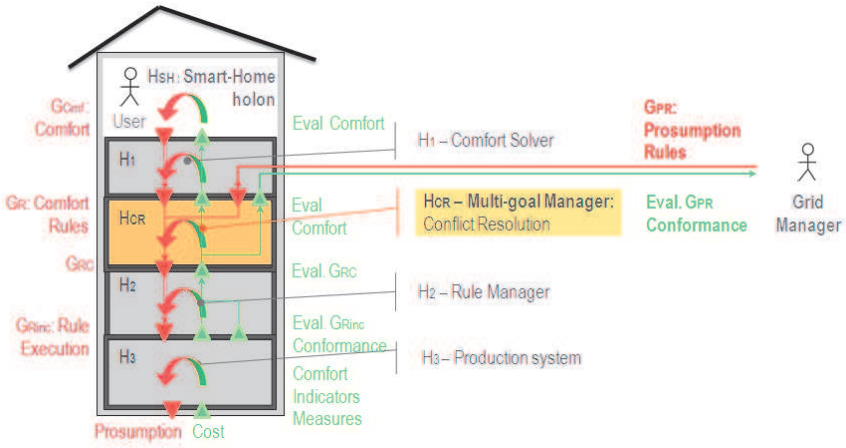


Fig. 5.23: Conflict resolution between comfort and prosumption rules

The same design applies, and should be inserted in the overall system design, whenever multiple conflicting goals are requested from any of the (sub-)holons, at any level. The actual conflict resolution strategy, or algorithm, is not essential here; nor is its particular design (e.g. centralised, decentralised, or hierarchical – see [FDD12] for a discussion on design patterns for conflict resolution). These aspects will depend on each system, case-by-case. In our example, a compromise will have to be found to ensure minimal comfort while not jeopardizing the grid.

(2) Goal-based interactions & Holonic Properties:

Complexity from simplicity is achieved here by inserting this new holon in-between the Comfort Solver H_1 and the Rule Manager H_2 (in example 1). Depending on its complexity, the Multi-goal Manager (Fig. 5.22) can be implemented as a monolithic centralised algorithm, with or without out self-* capabilities, or via various decentralised design patterns [FDD12], or as a holonic system.

Semi-isolation ensures that the Multi-goal Manager (for Conflict Resolution) only has to deal with conflicts inherent in the predefined types of its input goals and hence can optimise for this limited domain.

Abstraction means that the Multi-goal Manager holon is viewed externally as a conflict-resolution function, transforming certain types of conflicting input goals into a set of coherent output goals of different types. Hence, this holon can be discovered, (re)used and evaluated dynamically, across various systems that require this goal-transformation function, irrespectively of its internal design.

Progressive reactivity should be tuned so that the conflict-resolution implementation reacts to changes in the conflicting goal inputs faster than these inputs can change; and slower than the lower-level holon to which the coherent goals are sent.

APPLICATION SCENARIO 3. MARKET-ORIENTED ENERGY DISTRIBUTION

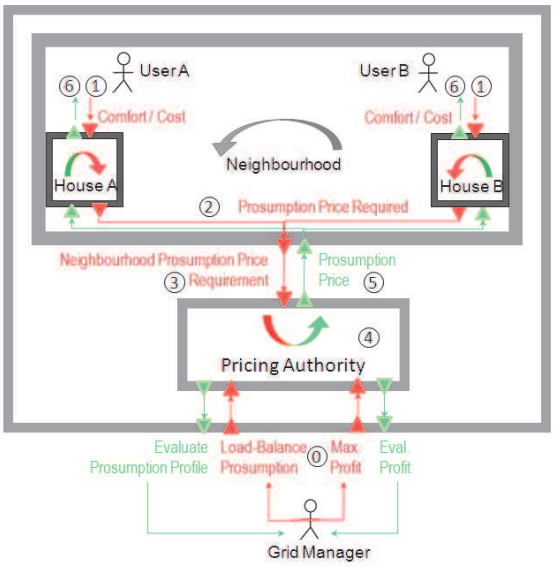
(1) Overview and selected scenario: This example increases the scope and the level of abstraction at which we analyse and design our system to the neighbourhood level. Here, smart houses like the ones designed in examples 1 and 2 represent mere prosumers, characterised by the amount of power they consume from or produce into the neighbourhood micro-grid. The new challenge here is to regulate the prosumption of each house so that the overall grid consumption and production level out; otherwise, the difference must be prosumed from the larger grid, or blackouts may occur. This challenge is expressed as a new goal: $G_{ProsN} = \{[Production == Consumption], Neighbourhood_N, every 0.5s\}$.

This example illustrates a market-oriented solution for achieving G_{ProsN}. It is based on offer-and-demand price settings, which aim to self-regulate the prosumption of smart houses. This approach assumes that high power consumption prices will deter consumption and encourage production at the house level; and vice-versa. This represents a top-down goal definition and enforcement, via price regulation, which can impact smart home internals by introducing new conflicts (as in example 2). In turn, this may trigger bottom-up adaptation in house-level prosumptions, which impact the overall neighbourhood prosumption. Such adaptations can lead to price readjustments, which again influence house-level prosumptions (yoyo effect). This approach is based on an example solution from the smart grid literature – i.e. the PowerMatcher [KWK05].

Fig. 5.24 depicts a specific scenario within this example, involving two houses (A and B). Each house is modelled as a holon, with an input comfort/cost goal (provided), an output prosumption goal (required) and an output prosumption cost goal (required). For simplicity, we only concentrate in this example on the part of the process where each smart home requires a price for its predicted prosumption (via its prosumption cost goal) and adapts its prosumption accordingly (automatically or by asking the user); the actual electricity prosumption is not shown in Fig. 5.24. The smart home model depicted here is an abstraction of the more detailed design in examples 1 and 2.

All houses are integrated together into a neighbourhood holon, with an output prosumption price goal (required). Another holon is the Pricing Authority, which offers a prosumption pricing goal (provided). Requests for this goal result in a reply

that specifies the price for the required prosumption. If the houses accept to prosume from this provider then they must pay the quoted price.



Step N°	Step Name
⓪	The grid manager sets load-balancing goals that impact the prosumption pricing scheme
①	Each User sets a Comfort/Cost goal G_{Cmf} for their smart house
②	Each smart home translates G_{Cmf} into a prosumption goal G_{Pros}
③	The neighbourhood aggregates prosumptions G_{ProsN} and requires a price quote
④	Pricing Authority calculates prosumption prices
⑤	Prosumption prices are returned to the smart houses
⑥	Each User / smart house decides whether or not to adapt its prosumption
⑦	Not shown: smart houses actually prosume electricity at the price set previously

Fig. 5.24: Top-down facilitation of bottom-up coordination

When the grid manager sets G_{ProsN} (step 0 in Fig. 5.24), for prosumption load-balancing, the prosumtion pricing scheme of the Pricing Authority is regulated accordingly (not discussed here). After that, the prosumption pricing process for the smart homes involves the following steps:

1. Each house is requested to reach a user-defined comfort goal G_{Cmf};
2. The smart houses (A and B) translate their comfort goals into prosumption requirements G_{Pros}, as in example 1.
3. The neighbourhood holon aggregates these into a prosumption requirement G_{ProsN} and forwards a price request to the Pricing Authority;

4. The Pricing Authority calculates global energy prices;
5. Prosumption prices are fed-back to the houses;
6. Prosumption prices are reformatted as necessary and fed-back to the users. Each house (internal self-* loop) decides to buy/sell energy at the given price or to update requirements; users may update comfort goals, as before. Afterwards the actual prosumptions are performed, measured and balanced, if needed, from the external grid (not shown in Fig. 5.24, for simplicty).

(2) Goal-based interactions & Holonic Properties:
Complexity from simplicity is reached by integrating multiple self-* prosumers (i.e. smart houses structured as in Example 1) and regulating their behaviour to meet grid- and house-level goals.

Semi-isolation is achieved as smart home internals only interact with the environment via energy prosumptions, bidding requests and prices; their internal self-* policies do not depend directly on those of other households, only via the global prices. Similarly, the Pricing Authority's internal strategies can optimise its goals, such as profit, based on prosumption requests, while abiding legal regulations.

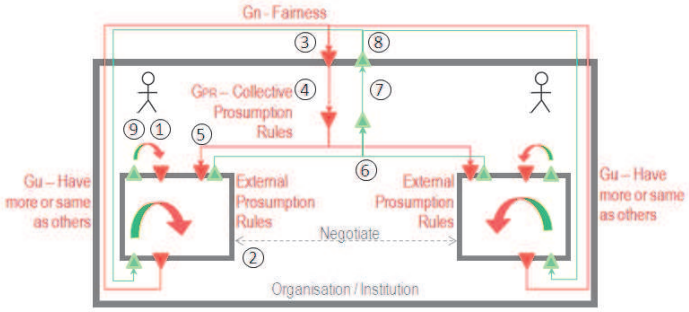
Abstraction is achieved by representing each home, and neighbourhood, only via their aggregate bidding requests and resulting prosumptions. This facilitates the management of large numbers of diverse households, which may join or leave the grid without impacting the global management scheme.

Progressive reactivity is tuned so that the times of the bidding and price-setting self-* processes, and of the actual prosumption, amount to sufficiently small intervals, for instance 0.5s, for allowing grid controllers to react so as to achieve balance.

APPLICATION SCENARIO 4. SELF-GOVERNING ENERGY COMMONS

(1) Overview and selected scenario: This example discusses a solution based on self-governance as an alternative to the prosumption management approach in example 3. It shows how global goals, such as *fairness* in this case, can be defined by users, or community members (bottom-up). Such global goals aim to help find a balance among conflicting individual goals – for instance, everyone wants to appear better off than their neighbours, yet also to avoid race conditions that would lead to resource depletion (tragedy of the commons, see section 5.2). Global goals can then be formalised as rules for regulating member behaviour (top-down) [DP14]. Subsequent evaluations of individual goals can then lead to redefining collective goals and rules (yoyo).

Fig. 5.25 illustrates the process for two users. The multi-layer holonic design for translating collective goals into rules and then into rule enforcement is similar to the one discussed in example 1 and hence simplified here. The scenario can be described via the following steps:



Step N°	Step Name
①	Each User wants to have more or the same as the other users – i.e. sets goal GU
②	All users negotiate in order to find a compromise and avoid an escalating situation
③	The user collective defines a fairness goal Gn
④	Gn is translated into prosumption rules GPR
⑤	The rules GPR regulate the prosumption behaviour of each smart home
⑥	Each smart home is evaluation and its conformance to GPR rewarded / sanctioned
⑦	GPR are adapted if deemed inefficient for attaining Gn
⑧	Fairness evaluation is broadcasted to all smart homes
⑨	Each User evaluates its fairness benefits and possibly renegotiates Gn

Fig. 5.25: Bottom-up collective goal definition, enforced top-down

1. Each user u in the shared neighbourhood n wishes to have more or at least as much comfort as their neighbours (at every instant t): $G_u = (better-than-others; home_u; t)$;
2. Users soon realise that aiming to achieve their individual goals G_u leads to escalating conflicts and race conditions detrimental to the entire community. Hence, they negotiate in order to find a compromise;
3. The compromise is defined as a collective fairness goal $G_n = (fairness; n; m)$, which aims to reach equal comfort for all users in the neighbourhood n over medium intervals m;
4. G_n is translated into prosumption rules: $G_{PR} = (rules_{PR}; n; t)$;
5. The prosumption rules G_{PR} are enforced into each household;
6. The conformance of prosumptions in each household with respect to the rules G_{PR} is evaluated; and rewards / sanctions are distributed to households accordingly;
7. The effectiveness of the rules G_{PR} to reach the fairness goal G_n is assessed; and the rules updated accordingly.
8. Fairness evaluations, based on house comfort measurements, are made available to the collective;

9. Each user evaluates their position with respect to the overall fairness; and may renegotiate (step 2); or leave the collective.

(2) Goal-based interactions & Holonic Properties:
Complexity from simplicity occurs here by integrating smart houses within a self-governing community, with multiple goals, where houses are already self-* systems of systems (as in example 1).

Semi-isolation is achieved in two interrelated ways. Firstly, the community members that share the collective goal are identified and grouped into a neighbourhood holon where they can self-organise and self-govern semi-independently from external authorities or other members. Secondly, external authorities allow the community to self-organise and self-govern, without interfering in their local regulations, as long as they meet external policies and norms – such as constitutional laws [JPD15]. If individual goals evolve and are no longer represented by the collective goal, the community holon may be dismantled, yet the smart homes remain.

Abstraction is achieved at the collective level by modelling each house only via its aggregate energy production and appropriation, and its conformance to the prosumption rules. The translation of the fairness goal into rules, as well as their enforcement and monitoring, can be achieved by member representatives, by electronic institutions, or both – yet this is not visible at the presented abstraction level. Also, individual users see the entire collective only via its fairness indicator, without access to details of other house profiles. At a higher abstraction level, different communities in the grid may self-organise based on different rules, resulting in diversity and plurality across the integrated community grid system.

Progressive reactivity should be tuned so that member prosumptions are faster than the evaluation of their rule conformance; which is faster than the evaluation (and update) of prosumption rules; in turn faster than the fairness goal management.

5.3.6 Conclusions

This section focused on some of the particularities of *complex OC systems* and the challenges inherent in their modelling, development and administration. It argued that as the complexity of OC systems and of their execution environments raises, dynamic *(self-)integration* becomes an essential capability for their success. This means that OC systems should be able to integrate (themselves) from resources discovered opportunistically, at runtime, in order to achieve goals specified by their stakeholders. The complexity of the resources that such OC systems integrate can vary quite widely, from 'traditional' components with no self-* capabilities, through single and collective self-adaptive systems (such as discussed in Sections 5.1 and 5.2 of this chapter) and all the way to entire systems-of-systems (the focus of this section).

To support the engineering process of complex OC systems that offer such (self-) integration functions, this section proposed a conceptual architectural model based on a new paradigm – *goal-oriented holonics* (or Goal-oriented Things-of-Things

– GoTT). This paradigm was defined by merging two important concepts. Firstly, it defined *goals* as first class modelling elements, representing explicit reference points for the systems to achieve when everything else may change in their internal construction and external environments (though goals may also change). Secondly, it identified several *key properties of holonic designs* observed in complex natural systems: constructing *complexity from simplicity* via encapsulated hierarchical structures; which feature *semi-isolation* among their components (or holons); which *abstract* their internal structures and behaviours for external observers and users; and, which feature suitable *time-tuning* among their dynamic self-* processes, in order to avoid oscillations and divergent behaviours. The conceptual model provided pointers on how to use goal-orientation in order to achieve such holonic properties in artificial OC systems.

Finally, the conceptual model was illustrated with several concrete examples from the smart micro-grid domain.

It is essential to note that the proposed architectural model presented represents research *work in progress*; rather than well-established, thoroughly-validated knowledge. So far, it is based on observations from related scientific domains that face similar complexity control problems, notably including natural systems, such as individual organisms and societies. These observations are merged with concepts from adjacent Information and Communication Technology (ICT) areas, such as Software Engineering (e.g. formal requirements, modularisation, and dynamic binding); self-adapting, self-organising and Multi-Agent Systems (e.g. goal-orientation and decentralised self-* functions); and concrete complex system designs in various application domains, featuring hierarchical or multi-layered structures and characterised by abstraction, semi-isolation and/or control time-tuning (e.g. communication protocol stacks, automates, robotics, or energy systems).

Therefore, this section represents a research direction requiring significant further work, which may, in fact, invalidate some of the initial assumptions and generic design. Our objective here was merely to provide a general intuition on how complex systems could be successfully designed and managed based on the goal-oriented holonics paradigm. If this research direction triggers and/or leads the way for further research, that may require a refinement, or even a rectification of this paradigm, then, we would have achieved our objective of helping to address the complexity problem in engineered OC systems.

Further Reading

For further details on hierarchical and holonic systems, mainly inspired from living systems, social organisations and economics, the reader may refer to [Sim62], [Sim96] and [Koe67]. [Kou+17a] provides a broad introduction to the novel area of Self-Aware Computing Systems, including many concerns that are relevant to the development of Organic Computing sys-

tems – e.g. goal-orientation, learning, knowledge representation, reasoning, self-adaptation, reporting, architectural aspects, testing, performance optimisation and so on. Similarly, [LMD13] offers a practical introduction to Autonomic Computing, concentrating on the core principles, design and implementation methods; and including monitoring, analysis, planning and execution functions, which are also relevant to the development of Organic Systems.
Finally, a lot of specific research and associated literature is available on each one of the concerns discussed in this section, including goals and requirements engineering (e.g. [Yu+11]) Software Engineering for Self-Adaptive Systems (e.g. [Che+09]), Component-Oriented Software engineering (e.g. [Szy97]); and adjacent topics, such as machine learning, evolutionary computing, search-oriented optimisation, multi-criteria decision-making, and so on – the curious reader is encouraged to use any library or online search engine to explore the vast body of ever-increasing literature available on these topics.

References

[AF06] D. P. Anderson and G. Fedak. 'The computational and storage potential of volunteer computing'. In: *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*. Vol. 1. IEEE. 2006, pp. 73–80.

[Akk10] J. op Akkerhuis. 'The Operator Hierarchy. A chain of closures linking matter, life and artificial intelligence'. PhD thesis. Radboud University Nijmegen, 2010.

[Ang+06] C. Anglano, J. Brevik, M. Canonico, D. Nurmi and R. Wolski. 'Fault-aware scheduling for Bag-of-Tasks applications on Desktop Grids'. In: *2006 7th IEEE/ACM International Conference on Grid Computing*. IEEE. 2006, pp. 56–63.

[Ang+08] C. Anglano, M. Canonico, M. Guazzone, M. Botta, S. Rabellino, S. Arena and G. Girardi. 'Peer-to-peer desktop grids in the real world: the ShareGrid project'. In: *Cluster Computing and the Grid, 2008. CCGRID'08. 8th IEEE International Symposium on*. IEEE. 2008, pp. 609–614.

[AP08] A. Artikis and J. Pitt. 'Specifying open agent systems: A survey'. In: *International Workshop on Engineering Societies in the Agents World*. Springer. 2008, pp. 29–45.

[Bal+13] T. Balke, C. da Costa Pereira, F. Dignum, E. Lorini, A. Rotolo, W. Vasconcelos and S. Villata. 'Norms in MAS: definitions and related concepts'. In: *Dagstuhl Follow-Ups* 4 (2013).

[Ber+10] Y. Bernard, L. Klejnowski, J. Hähner and C. Müller-Schloer. 'Towards trust in desktop grid systems'. In: *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE Computer Society. 2010, pp. 637–642.

[Bil93] D. Billington. 'Defeasible logic is stable'. In: *Journal of logic and computation* 3.4 (1993), pp. 379–400.

[Bin06] K. Binmore. 'Origins of fair play: Volume 614 of papers on economics and evolution'. In: *Max Planck Institute of Economics.[Online] Available at https://papers. econ. mpg. de/evo/discussionpapers/2006-14. pdf* (2006).

[BPV09] G. Boella, G. Pigozzi and L. Van Der Torre. 'Normative systems in computer science-ten guidelines for normative multiagent systems'. In: *Dagstuhl seminar proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2009.

[Bro86] R. A. Brooks. 'A robust layered control system for a mobile robot'. In: *IEEE Journal of Robotics and Automation* 2.1 (Mar. 1986), pp. 14–23. ISSN: 0882-4967.

[Bur+08] S. Burmester, H. Giese, E. Münch, O. Oberschelp, F. Klein and P. Scheideler. 'Tool support for the design of self-optimizing mechatronic multi-agent systems'. In: *Int. J. on Software Tools for Technology Transfer* 10 (3 2008), pp. 207–222. ISSN: 1433-2779.

[BZ96] J. C. Bennett and H. Zhang. 'WF 2 Q: worst-case fair weighted fair queueing'. In: *INFOCOM'96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE*. Vol. 1. IEEE. 1996, pp. 120–128.

[CB11] R. Centeno and H. Billhardt. 'Using incentive mechanisms for an adaptive regulation of open multi-agent systems'. In: *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*. Vol. 22. 1. Citeseer. 2011, p. 139.

[CBH11] R. Centeno, H. Billhardt and R. Hermoso. 'An adaptive sanctioning mechanism for open multi-agent systems regulated by norms'. In: *2011 IEEE 23rd International Conference on Tools with Artificial Intelligence*. IEEE. 2011, pp. 523–530.

[CBL04] A. J. Chakravarti, G. Baumgartner and M. Lauria. 'Application-specific scheduling for the organic grid'. In: *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*. IEEE Computer Society. 2004, pp. 146–155.

[CCD98] R. Conte, C. Castelfranchi and F. Dignum. 'Autonomous norm acceptance'. In: *International Workshop on Agent Theories, Architectures, and Languages*. Springer. 1998, pp. 99–112.

[CF10] C. Castelfranchi and R. Falcone. *Trust theory: A socio-cognitive and computational model*. Vol. 18. John Wiley & Sons, 2010.

[Che+09] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi and J. Magee. *Software Engineering for Self-Adaptive Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–26. ISBN: 978-3-642-02161-9. DOI: 10.1007/978-3-642-02161-9_1. URL: http://dx.doi.org/10.1007/978-3-642-02161-9_1.

[CHK05] H. Choset, S. Hutchinson and G. Kantor. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

[Cho+07] S. Choi, H. Kim, E. Byun, M. Baik, S. Kim, C. Park and C. Hwang. 'Characterizing and classifying desktop grid'. In: *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid'07)*. 2007.

[Cho+08] S. Choi, R. Buyya, H. Kim, E. Byun and J. Gil. 'A taxonomy of desktop grids and its mapping to state of the art systems'. In: *Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Tech. Rep* (2008).

[DDL12] B. Debbabi, A. Diaconescu and P. Lalanda. 'Controlling Self-Organising Software Applications with Archetypes'. In: *Sixth IEEE International Conference on Self-Adaptive and Self-Organizing Systems, (SASO 2012), Lyon, France, September 10-14, 2012*. 2012, pp. 69–78. DOI: 10.1109/SASO.2012.21. URL: http://dx.doi.org/10.1109/SASO.2012.21.

[Des16] J.-L. Dessalles. 'A Cognitive Approach to Relevant Argument Generation'. In: *Principles and Practice of Multi-Agent Systems - 2015 International Conference*. Ed. by M. Baldoni, C. Baroglio and F. Bex.

Springer, 2016. URL: http://www.dessalles.fr/papers/Dessalles_16061002.pdf.

[Dia+16] A. Diaconescu, S. Frey, C. Müller-Schloer, J. Pitt and S. Tomforde. '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, Sept. 12-16, 2016*. 2016, pp. 100–109. DOI: 10.1109/SASO.2016.16. URL: http://dx.doi.org/10.1109/SASO.2016.16.

[Dia+17] A. Diaconescu, K. L. Bellman, L. Esterle, H. Giese, S. Goetz, P. Lewis and A. Zisman. 'Generic Architectures for Collective Self-Aware Computing Systems'. In: *Self-Aware Computing Systems*. Ed. by S. Kounev, J. O. Kephart, A. Milenkoski and X. Zhu. Springer International Publishing, 2017. DOI: 10.1007/978-1-4471-5007-7.

[DKS89] A. Demers, S. Keshav and S. Shenker. 'Analysis and simulation of a fair queueing algorithm'. In: *ACM SIGCOMM Computer Communication Review*. Vol. 19. 4. ACM. 1989, pp. 1–12.

[DP14] A. Diaconescu and J. Pitt. 'Holonic Institutions for Multi-scale Polycentric Self-governance'. In: *Coordination, Organizations, Institutions, and Norms in Agent Systems X - COIN 2014 International Workshops, COIN@AAMAS, Paris, France, May 6, 2014, COIN@PRICAI, Gold Coast, QLD, Australia, December 4, 2014, Revised Selected Papers*. 2014, pp. 19–35. DOI: 10.1007/978-3-319-25420-3_2. URL: http://dx.doi.org/10.1007/978-3-319-25420-3_2.

[Ede+16] S. Edenhofer, S. Tomforde, J. Kantert, L. Klejnowski, Y. Bernard, J. Hähner and C. Müller-Schloer. 'Trust Communities: An Open, Self-Organised Social Infrastructure of Autonomous Agents'. In: *Trustworthy Open Self-Organising Systems*. Ed. by W. Reif, G. Anders, H. Seebach, J.-P. Steghöfer, E. André, J. Hähner, C. Müller-Schloer and T. Ungerer. Cham: Springer International Publishing, 2016, pp. 127–152. ISBN: 978-3-319-29201-4. DOI: 10.1007/978-3-319-29201-4_5. URL: http://dx.doi.org/10.1007/978-3-319-29201-4_5.

[FDD12] S. Frey, A. Diaconescu and I. M. Demeure. 'Architectural Integration Patterns for Autonomic Management Systems'. In: *9th IEEE International Conference and Workshops on the Engineering of Autonomic and Autonomous Systems (EASe 2012), Novi Sad, Serbia, 11-13 April*. 2012.

[Fre+15] S. Frey, A. Diaconescu, D. Menga and I. M. Demeure. 'A Generic Holonic Control Architecture for Heterogeneous Multiscale and Multiobjective Smart Microgrids'. In: *TAAS* 10.2 (2015), 9:1–9:21. DOI: 10.1145/2700326. URL: http://doi.acm.org/10.1145/2700326.

[GRB07] G. Governatori, A. Rotolo and L. A. BIO. 'Norms, Beliefs, Intentions in Defeasible Logic'. In: *Normative Multi-agent Systems, Dagstuhl Seminar Proceedings*. Vol. 7122. 2007.

[Har68] G. Hardin. 'The tragedy of the commons. The population problem has no technical solution; it requires a fundamental extension in morality.' In: *Science (New York, NY)* 162.3859 (1968), p. 1243.

[HBO10] R. Hermoso, H. Billhardt and S. Ossowski. 'Role evolution in open multi-agent systems as an information source for trust'. In: *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*. International Foundation for Autonomous Agents and Multiagent Systems. 2010, pp. 217–224.

[Hew91] C. Hewitt. 'Open information systems semantics for distributed artificial intelligence'. In: *Artificial intelligence* 47.1 (1991), pp. 79–106.

[HL04] B. Horling and V. Lesser. 'A survey of multi-agent organizational paradigms'. In: *The Knowledge Engineering Review* 19.04 (2004), pp. 281–316.

[HW11] C. D. Hollander and A. S. Wu. 'The current state of normative agent-based systems'. In: *Journal of Artificial Societies and Social Simulation* 14.2 (2011), p. 6.

[Jac93] O. Jacobs. *Introduction to Control Theory*. 2nd. Oxford, UK: Oxford University Press, 1993.

[Jai+96] R. Jain, G. Babic, B. Nagendra and C. Lam. 'Fairness, call establishment latency and other performance metrics'. In: *ATM-Forum 96 (1173), 1–6*. Tech. Rep. ATM Forum/96-1173, ATM Forum Document. 1996.

[JDT11] J. Jiang, V. Dignum and Y. H. Tan. 'An Agent Based Inter-organizational Collaboration Framework: OperA+'. In: *2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*. Vol. 3. Aug. 2011, pp. 21–24. DOI: 10.1109/WI-IAT.2011.165.

[JPD15] J. Jiang, J. Pitt and A. Diaconescu. 'Rule Conflicts in Holonic Institutions'. In: *3rd Workshop on Fundamentals of Collective Adaptive Systems (FoCAS 2015), part of 2015 IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops (SASOW 2015)*. Sept. 2015, pp. 49–54. DOI: 10.1109/SASOW.2015.13.

[Kan+14a] J. Kantert, S. Bödelt, S. Edenhofer, S. Tomforde, J. Hähner and C. Müller-Schloer. 'Interactive Simulation of an Open Trusted Desktop Grid System with Visualisation in 3D'. In: *2014 IEEE Eighth International Conference on Self-Adaptive and Self-Organizing Systems*. IEEE. 2014, pp. 191–192.

[Kan+14b] J. Kantert, L. Klejnowski, Y. Bernard and C. Müller-Schloer. 'Influence of Norms on Decision Making in Trusted Desktop Grid Systems-Making Norms Explicit.' In: *ICAART (2)*. 2014, pp. 278–283.

[Kan+14c] J. Kantert, H. Scharf, S. Edenhofer, S. Tomforde, J. Hähner and C. Müller-Schloer. 'A Graph Analysis Approach to Detect Attacks

in Multi-agent Systems at Runtime'. In: *2014 IEEE Eighth International Conference on Self-Adaptive and Self-Organizing Systems*. IEEE. 2014, pp. 80–89.

[Kan+15a] J. Kantert, S. Edenhofer, S. Tomforde, J. Hähner and C. Müller-Schloer. 'Defending autonomous agents against attacks in multi-agent systems using norms'. In: *Proceedings of the 7th International Conference on Agents and Artificial Intelligence*. 2015, pp. 149–156.

[Kan+15b] J. Kantert, H. Spiegelberg, S. Tomforde, J. Hähner and C. Müller-Schloer. 'Distributed rendering in an open self-organised trusted desktop grid'. In: *Autonomic Computing (ICAC), 2015 IEEE International Conference on*. IEEE. 2015, pp. 267–272.

[Kan+15c] J. Kantert, S. Wildemann, G. von Zengen, S. Edenhofer, S. Tomforde, L. Wolf, J. Hähner and C. Müller-Schloer. 'Improving reliability and endurance using end-to-end trust in distributed low-power sensor networks'. In: *International Conference on Architecture of Computing Systems*. Springer. 2015, pp. 135–145.

[Kan+16] J. Kantert, L. Klejnowski, S. Edenhofer, S. Tomforde and C. Müller-Schloer. 'A Threatmodel for Trust-based Systems Consisting of Open, Heterogeneous and Distributed Agents'. In: *Proceedings of the 8th International Conference on Agents and Artificial Intelligence (ICAART 2016), Volume 1, Rome, Italy, February 24-26, 2016*. 2016, pp. 173–180. DOI: 10.5220/0005696801730180. URL: http://dx.doi.org/10.5220/0005696801730180.

[KC03] J. Kephart and D. Chess. 'The Vision of Autonomic Computing'. In: *IEEE Computer* 36.1 (2003), pp. 41–50.

[Kep+17] J. O. Kephart, A. Diaconescu, H. Giese, A. Robertsson, T. Abdelzaher, P. Lewis, A. Filieri, L. Esterle and S. Frey. 'Self-Aware Computing Systems'. In: ed. by S. Kounev, J. Kephart, A. Milenkoski and X. Zhu. Springer Verlag, Berlin Heidelberg, 2017. Chap. Self-adaptation in Collective Self-aware Computing Systems, pp. 401–435. ISBN: 978-3-319-47474-8. URL: http://www.springer.com/us/book/9783319474724.

[Kle14] L. Klejnowski. 'Trusted community: a novel multiagent organisation for open distributed systems'. PhD thesis. Hannover, Univ., Diss., 2014, 2014.

[Koe67] A. Koestler. *The Ghost in the Machine*. 1st ed. GATEWAY EDITIONS, Henry Regnery Co., 1967.

[Kou+17a] 'Self-Aware Computing Systems'. In: ed. by S. Kounev, J. Kephart, A. Milenkoski and X. Zhu. Springer Verlag, Berlin Heidelberg, 2017. ISBN: 978-3-319-47474-8. URL: http://www.springer.com/us/book/9783319474724.

[Kou+17b] S. Kounev, J. O. Kephart, A. Milenkoski and X. Zhu. *Self-aware Computing Systems*. Berlin / Heidelberg, DE: Springer Verlag, 2017.

[KWK05] J. K. Kok, C. J. Warmer and I. G. Kamphuis. 'PowerMatcher: Multiagent Control in the Electricity Infrastructure'. In: *Proceedings of*

the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems. AAMAS '05. The Netherlands: ACM, 2005, pp. 75–82. ISBN: 1-59593-093-0. DOI: 10.1145/1082473.1082807. URL: http://doi.acm.org/10.1145/1082473.1082807.

[Lam01] A. van Lamsweerde. 'Goal-oriented requirements engineering: a guided tour'. In: *Proceedings Fifth IEEE International Symposium on Requirements Engineering*. 2001, pp. 249–262. DOI: 10.1109/ISRE.2001.948567.

[LB01] C. Landauer and K. L. Bellman. 'New Architectures for Constructed Complex Systems'. In: *Appl. Math. Comput.* 120.1-3 (May 2001), pp. 149–163. ISSN: 0096-3003. DOI: 10.1016/S0096-3003(99)00240-4. URL: http://dx.doi.org/10.1016/S0096-3003(99)00240-4.

[LB99] C. Landauer and K. L. Bellman. 'Problem Posing Interpretation of Programming Languages'. In: *Proceedings of the Thirty-Second Annual Hawaii International Conference on System Sciences-Volume 3 - Volume 3*. HICSS'99. Washington, DC, USA: IEEE Computer Society, 1999, pp. 3007–. ISBN: 0-7695-0001-3. URL: http://dl.acm.org/citation.cfm?id=874070.876075.

[Lew+17] P. Lewis, K. Bellman, C. Landauer, L. Esterle, K. Glette, A. Diaconescu and H. Giese. 'Towards A Framework for the Levels and Aspects of Self-Aware Computing Systems'. In: *Self-Aware Computing Systems*. Ed. by S. Kounev, J. O. Kephart, A. Milenkoski and X. Zhu. Springer International Publishing, 2017. DOI: 10.1007/978-1-4471-5007-7.

[LMD13] P. Lalanda, J. A. McCann and A. Diaconescu. *Autonomic Computing - Principles, Design and Implementation*. Undergraduate Topics in Computer Science. Springer, 2013. ISBN: 978-1-4471-5006-0. DOI: 10.1007/978-1-4471-5007-7. URL: http://dx.doi.org/10.1007/978-1-4471-5007-7.

[MF05] S. Mcgregor and C. Fernando. 'Levels of Description: A Novel Approach to Dynamical Hierarchies'. In: *Artificial Life* 11.4 (2005), pp. 459–472. DOI: 10.1162/106454605774270615.

[Nas51] J. Nash. 'Non-cooperative games'. In: *Annals of mathematics* (1951), pp. 286–295.

[New03] M. E. Newman. 'The structure and function of complex networks'. In: *SIAM review* 45.2 (2003), pp. 167–256.

[Nut01] D. Nute. 'Defeasible logic'. In: *International Conference on Applications of Prolog*. Springer. 2001, pp. 151–169.

[Nut88] D. Nute. 'Defeasible reasoning: a philosophical analysis in prolog'. In: *Aspects of Artificial Intelligence*. Springer, 1988, pp. 251–288.

[Nut94] D. Nute. *Defeasible logic, Handbook of logic in artificial intelligence and logic programming (vol. 3): nonmonotonic reasoning and uncertain reasoning*. 1994.

[NWH02] L. Nolle, K. C. P. Wong and A. A. Hopgood. 'DARBS: A Distributed Blackboard System'. In: *Research and Development in Intelligent Systems XVIII*. Springer London, 161170, 2002.

[OG05] M. Oussalah and N. Griffiths. 'Cooperative clans'. In: *Kybernetes* 34.9/10 (2005), pp. 1384–1403.

[Osb04] M. J. Osborne. *An introduction to game theory*. Vol. 3. 3. Oxford University Press New York, 2004.

[Ost+90] E. Ostrom et al. *Governing the commons: The evolution of institutions for collective action*. 1990.

[PSA11] J. Pitt, J. Schaumeier and A. Artikis. 'The axiomatisation of socio-economic principles for self-organising systems'. In: *Self-Adaptive and Self-Organizing Systems (SASO), 2011 Fifth IEEE International Conference on*. IEEE. 2011, pp. 138–147.

[Raw71] J. Rawls. *A Theory of Justice*. Belknap Press of Harvard University Press, 1971.

[RG91] A. Rao and P. M. Georgeff. 'Modeling Rational Agents within a BDI Architecture'. In: *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR91)*. Morgan Kaufmann, Mar. 1991, pp. 473–484.

[RNI95] S. Russell, P. Norvig and A. Intelligence. 'A modern approach'. In: *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs* 25 (1995).

[RZ94] J. S. Rosenschein and G. Zlotkin. *Rules of encounter: designing conventions for automated negotiation among computers*. MIT press, 1994.

[Sar05] G. Sartor. 'Legal reasoning'. In: *A Treatise of Legal Philosophy and General Jurisprudence* (2005).

[SC11] B. T. R. Savarimuthu and S. Cranefield. 'Norm creation, spreading and emergence: A survey of simulation models of norms in multi-agent systems'. In: *Multiagent and Grid Systems* 7.1 (2011), pp. 21–54.

[Sim62] H. A. Simon. 'The Architecture of Complexity'. In: *American Philosophical Society* 106 (1962).

[Sim96] H. A. Simon. *The Sciences of the Artificial*. MIT Press, 1996. ISBN: 9780262264495. URL: https://books.google.fr/books?id=k5Sr0nFw7psC.

[Sin+13] M. P. Singh, M. Arrott, T. Balke, A. K. Chopra, R. Christiaanse, S. Cranefield, F. Dignum, D. Eynard, E. Farcas, N. Fornara et al. *The uses of norms*. Vol. 4. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[Sin99] M. P. Singh. 'An ontology for commitments in multiagent systems'. In: *Artificial intelligence and law* 7.1 (1999), pp. 97–113.

[Smi80] R. G. Smith. 'The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver'. In: *IEEE Transactions on Computers* C-29.12 (Dec. 1980), pp. 1104–1113. ISSN: 0018-9340. DOI: 10.1109/TC.1980.1675516.

[Ste+14] J.-P. Steghöfer, G. Anders, J. Kantert, C. Müller-Schloer and W. Reif. 'An effective implementation of norms in trust-aware open self-organising systems'. In: *Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2014 IEEE Eighth International Conference on*. IEEE. 2014, pp. 76–77.

[Szy97] C. Szyperski. *Component Software: Beyond Object-oriented Programming*. ACM Press, 1997.

[TB95] R. Tuomela and M. Bonnevier-Tuomela. 'Norms and Agreements'. In: *E. J. of Law, Philosophy and Computer Science* 5 (1995), pp. 41–46.

[Ten00] D. Tennenhouse. 'Proactive Computing'. In: *Communications of the ACM* 43.5 (2000), pp. 43–50. ISSN: 0001-0782. DOI: http://doi.acm.org/10.1145/332833.332837.

[Tom+11a] S. Tomforde, H. Prothmann, J. Branke, J. Hähner, M. Mnif, C. Müller-Schloer, U. Richter and H. Schmeck. 'Observation and Control of Organic Systems'. In: *Organic Computing - A Paradigm Shift for Complex Systems*. Ed. by C. Müller-Schloer, H. Schmeck and T. Ungerer. Autonomic Systems. Birkhäuser Verlag, 2011, pp. 325–338.

[Tom+11b] S. Tomforde, H. Prothmann, J. Branke, J. Hähner, M. Mnif, C. Müller-Schloer, U. Richter and H. Schmeck. 'Observation and control of organic systems'. In: *Organic Computing—A Paradigm Shift for Complex Systems*. Springer, 2011, pp. 325–338.

[Tom+14] S. Tomforde, J. Haehner, H. Seebach, W. Reif, B. Sick, A. Wacker and I. Scholtes. 'Engineering and mastering interwoven systems'. In: *Architecture of Computing Systems (ARCS), 2014 27th International Conference on*. VDE. 2014, pp. 1–8.

[Tom+16] S. Tomforde, S. Rudolph, K. Bellman and R. Wurtz. 'An Organic Computing Perspective on Self-Improving System Interweaving at Runtime'. In: *2016 IEEE International Conference on Autonomic Computing (ICAC)*. July 2016, pp. 276–284. DOI: 10.1109/ICAC.2016.15.

[UG13] A. Urzică and C. Gratie. 'Policy-based instantiation of norms in MAS'. In: *Intelligent Distributed Computing VI*. Springer, 2013, pp. 287–296.

[VBH08] P. Valckenaers, H. V. Brussel and T. Holvoet. 'Fundamentals of Holonic Systems and Their Implications for Self-Adaptive and Self-Organizing Systems'. In: *Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops*. SASOW '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 168–173. ISBN: 978-0-7695-3553-1. DOI: 10.1109/SASOW.2008.29. URL: http://dx.doi.org/10.1109/SASOW.2008.29.

[Von63] G. H. Von Wright. 'Norm and action: a logical enquiry'. In: *Routledge & Kegan Paul, London / UK* (1963).