

Hildesheimer Informatik-Berichte

Christian Kröher

Control Action Types

Patterns of Central Control for Self-adaptive Systems

August 26, 2022

Report No. 1/2022, SSE 1/22/E

ISSN 0941-3014

Abstract

An essential building block for any Self-adaptive System (SaS) is its inherent control mechanism. This mechanism enables a SaS to modify its domain functionality with respect to changes in its environment autonomously. Distributed control and central control represent two complementary paradigms to establish this capability. The selection of one of these paradigms leads to significant trade-offs regarding certain software qualities when designing a SaS. A promising approach to minimize these trade-offs is an integration, which combines the individual benefits to achieve the best of both paradigms. However, establishing such a multi-paradigm control requires comprehensive knowledge about control options and their interactions, which is hardly available.

In this report, we build on our previous work and present a pattern catalog for integrating distributed and central control. We introduce these patterns by a taxonomy of Control Action Types (CATs). Each CAT describes a unique type of interaction between a central controller and a distributed controlled SaS to achieve a desired adaptation. Further, we identify involved trade-offs between these CATs. While other approaches exist, which propose their individual integrations, we aim at a systematic discussion of the range of multi-paradigm control for a SaS.

Keywords: Self-adaptive systems, distributed control, emergence, control unit, central control, multi-paradigm control, patterns

Contents

1	Introduction	5
2	Background	7
2.1	Multi-Paradigm Control	7
2.2	System Understanding	8
2.3	Running Example	10
3	Taxonomy	14
4	Patterns	16
4.1	Command	16
4.2	Constraint	18
4.3	Influence	20
4.4	Pseudo-Emergence	24
4.5	Isolation	27
5	Discussion	31
6	Related Work	33
7	Conclusion	36
	Bibliography	38

List of Figures

2.1	SaS control design space and trade-offs	7
2.2	System reference model	9
2.3	System reference model instance: example of a micro grid	11
4.1	Command pattern	17
4.2	Constraint pattern	18
4.3	Influence pattern	21
4.4	Pseudo-emergence pattern	25
4.5	Isolation pattern	28

List of Tables

5.1	Summary and classification of presented control action types	31
-----	--	----

Chapter 1

Introduction

The unique characteristic of any Self-adaptive System (SaS) is its autonomous adaptation to changes in its environment [WSG⁺13]. In general, this capability serves several system qualities, like scalability, resilience, and flexibility [KM07]. However, the extent of support for certain system qualities depends significantly on the selected mechanism to establish self-adaptation of a SaS. In particular, the specific way of establishing the control mechanism for observing and reacting to changes plays a fundamental role.

Distributed control and central control are two complementary paradigms to establish the control mechanism for a SaS and thereby its ability to self-adapt. The former paradigm splits the system control across the constituent entities of a SaS [GCGC07, KC03, DKJ18]. The latter paradigm adds a single controller, which manages all constituent entities of a SaS collectively [WSG⁺13, GCGL15, Boy76]. This results in individual benefits and drawbacks regarding certain software qualities. For example, distributed control provides higher resilience by avoiding any single point of failure (no single controller). In turn, central control enables higher predictability of the ultimate outcome of adaptations by avoiding any negotiations (no coordination of control between multiple entities). Hence, the selection of one of these paradigms leads to significant trade-offs when designing a SaS [Bor13].

A promising approach to minimize the trade-offs between distributed and central control is their integration. The aim of this integration is the combination of the individual benefits to achieve the best of both paradigms [KGS22, Bor13]. However, a reasonable integration forming a useful multi-paradigm control approach is challenging [MEHdH13]. For example, it is not trivial to find a suitable compromise between resilience (distributed control) and predictability (central control) of a SaS. Further, SaS engineering lacks a systematic understanding of control options [dLGM⁺13] as well as references for their interactions [BDMSG⁺09]. This absence of fundamental knowledge impedes profound design decisions towards a multi-paradigm control approach.

In this report, we build on our previous work [KSPS21, KGS22] as part of the DevOpt-project [Dev] and present a pattern catalog for integrating distributed and central control. We therefore extend our initial catalog [KSPS21] by explanations of our approach for multi-paradigm control and the underlying understanding of a general SaS. On this basis, we introduce the patterns by an extended taxonomy of Control Action Types (CATs). These CATs support the design of a multi-paradigm approach to SaS control, like design patterns do for object-oriented software development [GHJV94] or idioms for specific programming languages [Cop92]. Each CAT describes a unique

type of interaction between a central controller and a distributed controlled SaS to achieve a desired adaptation. Hence, we provide a systematic discussion of how centralized control can influence a SaS, while keeping as much of its autonomy as possible. This core contribution explicitly addresses the aforementioned obstacles when designing control for a SaS [MEHdH13, dLGM⁺13, BDMSG⁺09]. Further, we discuss the trade-offs between the CATs and the general benefits of this catalog for practitioners as well as researchers.

The aim of our pattern catalog is to provide a systematic overview of the range of possible approaches to establish multi-paradigm control for a SaS. Hence, we do not introduce fundamentally new control mechanisms. Our catalog categorizes existing approaches, like [KM90, GCS03, KTK⁺16], and enables the comparison of their properties on the level of abstract control action types. In this way, our contribution differs from other overviews on SaS control, which, for example, focus on the broader design of such systems [KTPR20], the interaction and distribution of control loop steps [dLGM⁺13], or the distribution of entire controllers [SMSc⁺10].

The remainder of this report is structured as follows. Chapter 2 presents the background for our pattern catalog. It explains our approach to multi-paradigm control and our general understanding of the target system we apply this approach to. We exemplify this explanation by a scenario from the electric grid domain, which we use as a running example throughout this report. Chapter 3 introduces the taxonomy for describing our patterns consistently. We define the individual dimensions of this taxonomy, which cover both general as well as multi-paradigm control aspects. In Chapter 4, we apply this taxonomy to describe the individual patterns. Further, we illustrate their application in the context of our running example. Chapter 5 summarizes and discusses the contribution of this report. In particular, we discuss the main advantages and disadvantages of the individual patterns, provide examples for their selection in practice, and explain the expected impact for current and future research. Chapter 6 delimits this overall contribution from related work, while Chapter 7 concludes this report with final remarks on future work.

Chapter 2

Background

The contribution of this report relies on a specific integration of distributed and central control mechanisms to achieve the desired multi-paradigm control approach. Hence, we start by outlining this integration and its motivation in Section 2.1. This basic introduction to our control approach leads to a particular understanding of a SaS. We explain this understanding along a system reference model in Section 2.2. This model introduces the generic elements of and their relations in a SaS to which we refer to in Chapter 4 to describe individual CATs. In Section 2.3, we instantiate this model to create a running example for the remainder of this report.

2.1 Multi-Paradigm Control

The main motivation for our pattern catalog presented in this report is based on our approach to integrate distributed and central control for complex SaS [KGS22]. This integration enables a SaS to easier satisfy multiple software qualities simultaneously than comparable systems, which rely on one of these control paradigms exclusively. The emergent properties arising from distributed control will enable a SaS to handle common situations in a resilient way autonomously. For special situations, which require global knowledge to be detected and to derive respective actions, a central controller for the SaS keeps an option to intervene.

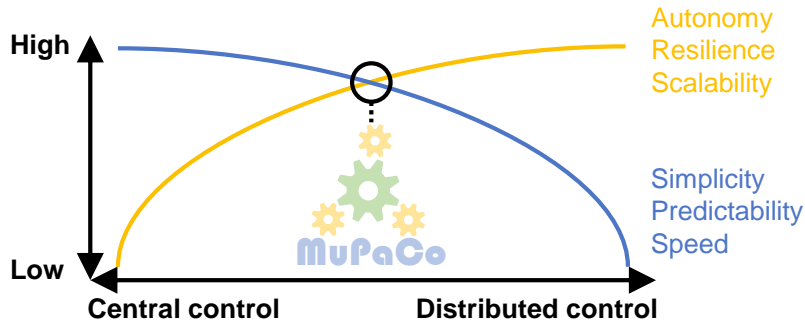


Figure 2.1: SaS control design space and trade-offs

Our Multi-Paradigm Control (MuPaCo) approach therefore aims at an optimum for design trade-offs regarding certain software qualities as illustrated in Figure 2.1. The more control is distributed among the constituent entities of a SaS, the higher is

their *autonomy* and, hence, the support for *resilience* [GCGC07] and *scalability* [KC03]. However, a high degree of distribution makes it difficult to reach global optimum, optimize simultaneously multiple properties, and balance the entire system fast after significant changes. Central control counteracts these downsides by its *simplicity* compared to multiple distributed control mechanisms and their coordination. This leads to high *predictability* of the ultimate outcome of adaptations [WSG⁺13, GCGL15] at comparably high *speed*. In turn, a high degree of central control lacks robustness against failure, requires support for potentially multiple adaptation approaches offered by SaS entities, and needs comparably high computational effort to manage all SaS entities simultaneously. The advantages of distributed control again compensate these disadvantages from strictly following central control.

The integration of basic distributed control with temporal, case-dependent central intervention is at the heart of our MuPaCo approach. In this way, we bring together recent advances in distributed, emergent systems engineering with well-established control theory to reach optimal design trade-offs. A significant step towards the realization of this approach is the overview of possible types of central control actions and their impact on a distributed controlled SaS presented in this report.

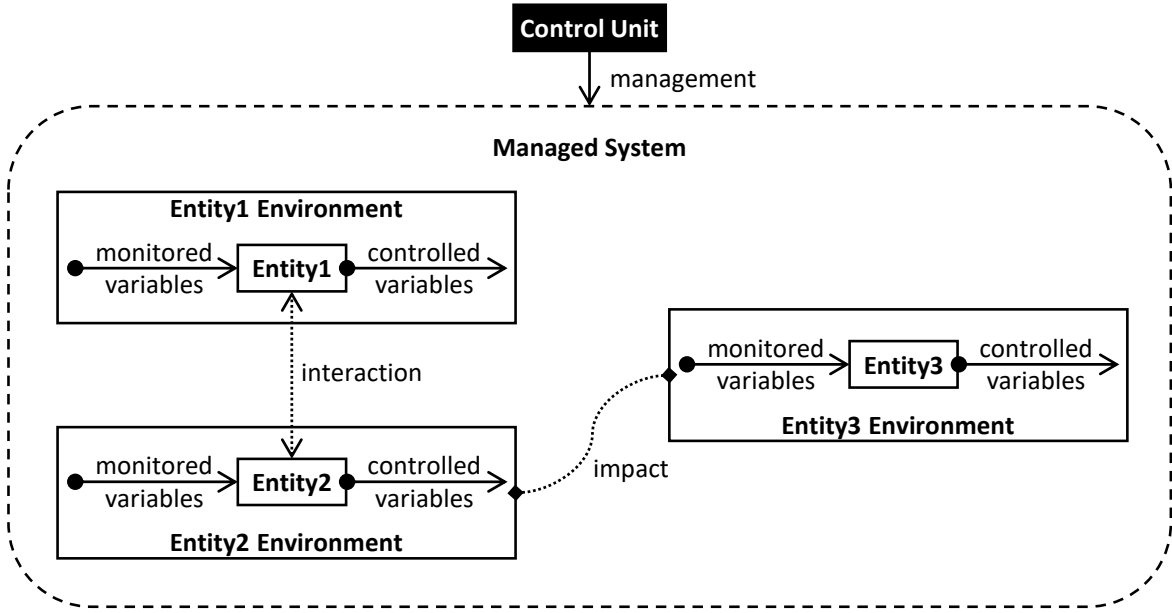
2.2 System Understanding

A typical SaS basically consists of a managing system and a managed system [KC03, WA13, QWG21]. The former system type often realizes a control loop, like MAPE [KC03], OODA [Boy76], or CARE [GCGL15], to adapt the domain functionality provided by the latter system type to changes in its environment. In Figure 2.2a, we adopt this basic differentiation in terms of a *control unit* representing the managing system for the *managed system*. We intentionally use a different term for the managing system as our focus is on executing certain adaptations via our CATs in this report. Based on this specific focus, we interpret our control unit as a special case of managing system, which is able to perform all of these CATs (indicated by the *ctrl*-attribute in Figure 2.2b). Hence, we assume that the control unit also provides upstream capabilities, like monitoring and analyzing the managed system, but do not further discuss them here.

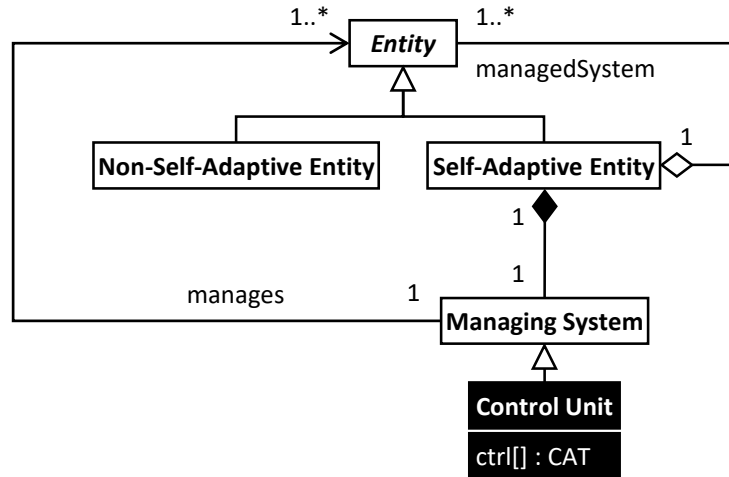
The basic constituent element of the managed system is called *entity* in Figure 2.2a. In general, an entity is a software system, which interacts with its local *entity environment*. It observes this environment via its *monitored variables* and reacts to the derived input by producing output, which it maps to its *controlled variables* [PM95]. As part of the managed system, these entities typically only provide certain domain functionality, but do not include self-adaptive capabilities [HTHJ09, ARS17]. However, some notions of a SaS consider its entities to be self-adaptive on their own [KC03, SMSc⁺10]. Hence, we differentiate between two specific types of entities as illustrated in Figure 2.2b:

- A *self-adaptive entity* pursues a specific goal autonomously. It therefore adapts itself to its local entity environment without the need of external guidance. A typical SaS represents an instance of such an entity in this understanding. This leads to a recursion where a self-adaptive entity simultaneously is an element of a managed system and consists of a managing system as well as, again, a managed system.

- A *non-self-adaptive entity* does not pursue a goal autonomously. It therefore only provides its domain functionality, but no self-adaptive properties on its own. Any instance of such an entity will only react to explicit calls for this domain functionality without consideration of the state of its local entity environment¹. This leads to the absence of any refinement of a non-self-adaptive entity relevant for our system understanding².



(a) System overview



(b) Core element types

Figure 2.2: System reference model

¹Except for those parts of the environment (monitored and controlled variables) relevant to observe such calls and react to them. However, it always provides a constant behavior even in situations that would require adaptation to improve it.

²A non-self-adaptive entity may of course be further refined into constituent elements, but these elements will not be a managing and a managed system as for self-adaptive entities.

Based on this differentiation, our reference model supports the following SaS variants:

- **Basic SaS** for which the control unit represents the managing system and a set of non-self-adaptive entities realizes the managed system; this represents a simple application of the reference model as the entities of the managed system cannot be further refined by re-applying this model
- **Complex SaS** in which the managed system consists of self-adaptive entities managed by a superior control unit; this represents a full-recursive application of the reference model as it can be re-applied to each entity of the managed system
- **Hybrid SaS** including non-self-adaptive and self-adaptive entities as part of their managed system and a superior control unit as managing system; this represents a partial-recursive application of the reference model as it can be re-applied to the self-adaptive entities of the managed system, but not to the non-self-adaptive ones

Independent of their specific type, relations between entities exist to provide the general domain functionality of the managed system collectively. We categorize these relations into direct interaction and indirect impact. Direct *interaction* describes any form of information exchange between at least two entities. However, the type of exchanged information is limited. We do not expect explicit control of one entity over others by sending commands or force a specific adaptation. This is an exclusive capability of the control unit. Interactions only convey information about entities, like their intention or state. For example, the interaction between *Entity1* and *Entity2* in Figure 2.2a may allow coordinating their future actions before their actual execution. In contrast, indirect *impact* occurs as an effect of one entity's actions via its controlled variables. For example, the impact relation in Figure 2.2a may indicate that *Entity3* observes (the result of) the execution of certain actions by *Entity2*. For both types of relations, the specific characteristics of their instances vary in practice. In general, we assume that at least one of these types exist between entities of a managed system, while any combination per entity is possible.

The presented reference model enables the derivation of SaS instances ranging from single, small-scale systems to complex systems of systems. Further, it supports both basic control paradigms to establish self-adaptation: the basic SaS variant represents a purely central control approach, while the complex SaS variant allows a purely distributed control by excluding the superior control unit. Our multi-paradigm control approach exists inherently in the complex SaS variant (without modifications) and the hybrid SaS variant. This generality of the reference model aims at its practical applicability as required in the DevOpt-project [Dev]. For the CATs in this report, we focus on the complex and hybrid SaS variants, which consist of distributed controlled entities and a superior central control unit.

2.3 Running Example

In the DevOpt-project [Dev], one of the target use case scenarios resides in the domain of smart energy supply. Figure 2.3 illustrates an electric grid as an example from this

domain. The *Micro Grid* represents a certain city district, which consists of a set of electric grid elements:

- The *Power Plant* acts as a pure energy provider for the entire district
- The *Factory* is a pure energy consumer as it does not generate any energy itself
- The *Wind Turbines* combine energy provision with its consumption when loading their local battery for later use of their generated renewable energy surplus
- The *Home A* and *Home B* also realize hybrid capabilities via their photovoltaic as local renewable energy generators as well as their batteries and electric cars as local consumers

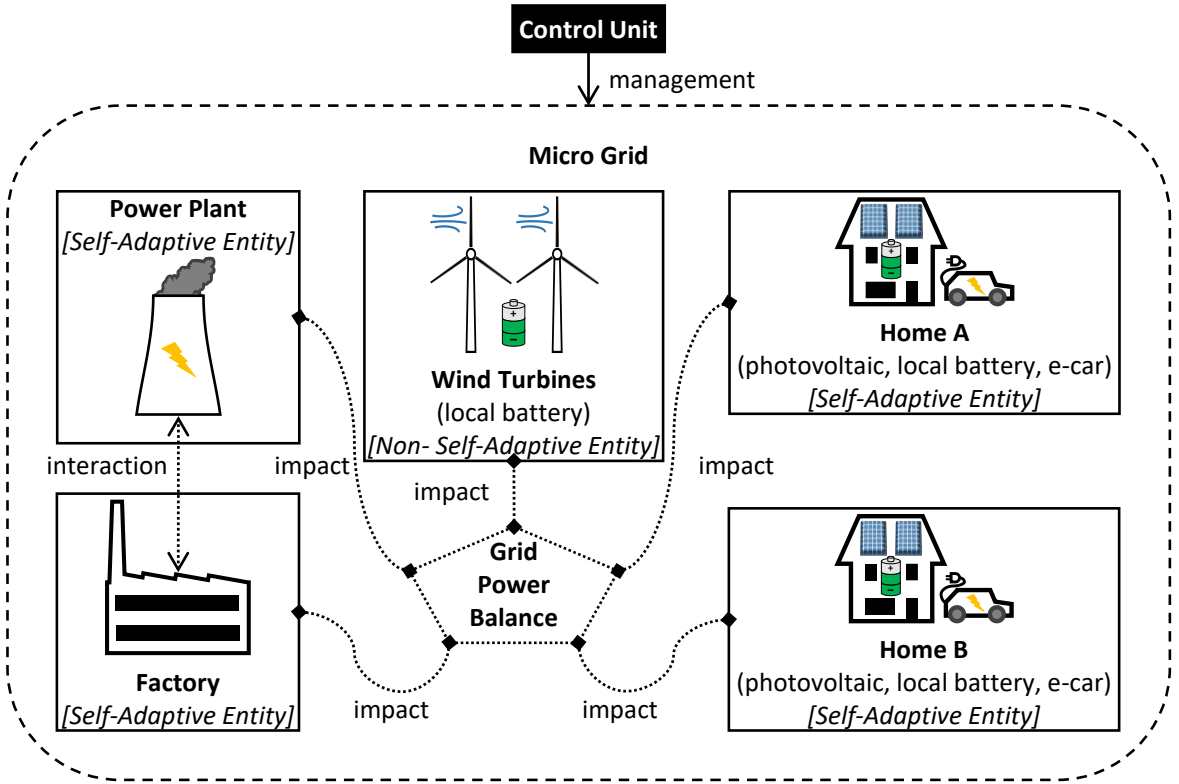


Figure 2.3: System reference model instance: example of a micro grid

We consider these electric grid elements as specific instances of the entities of our system reference model introduced in the previous section. In general, each of these elements monitors and controls its respective variables in its specific environment (no global grid knowledge). However, they vary with respect to their degree of self-adaptation and autonomy. The *Power Plant*, *Factory*, *Home A*, and *Home B* represent instances of self-adaptive entities as they typically pursue their individual goals. For example, *Home A* and *Home B* should offer a high level of comfort for its residents with minimal energy consumption. The *Factory* should reach the maximum of production with the available amount of energy. In turn, the *Power Plant* should provide the maximum of energy with minimal effort. In order to reach their goals, each of these

elements consists of further entities supervised by a certain form of managing system. In practice, such forms may range from automated control loops to manual control by humans. In contrast, the *Wind Turbines* are an instance of a non-self-adaptive entity due to their lack of an individual goal and a dedicated managing system. They only react to changing weather conditions (monitored variables) by blocking or releasing their rotor blades (controlled variables). If no other demand exists, they load their local battery until maximum, which then terminates the energy generation. A goal beyond mere energy production requiring adaptations by a more sophisticated local managing system typically do not exist for this entity here.

The *Grid Power Balance* is a means to connect all entities in Figure 2.3. It quantifies the overall energy flow between the entities as a single variable aggregating the individual energy surpluses (provisions) and deficits (consumptions). In this way, the *Grid Power Balance* realizes an *impact* relation of our system reference model. Any significant change in an entity power balance as an effect of its locally controlled variables will occur as a subsequent change of the *Grid Power Balance*. In turn, this subsequent change may affect other entities indirectly. Hence, this relation between all entities enables them to act on the current provision and consumption of energy in the grid. Further, the *interaction* relation between the *Power Plant* and the *Factory* enable their direct communication in special situations. For example, the *Factory* may inform the *Power Plant* about its plan to increase its production unscheduled in a certain timeframe in future. The *Power Plant* may then acknowledge or reject this plan depending on its capability to provide the necessary extra energy.

The individual capabilities of the entities as well as their relations result in the emergent behavior of self-stabilizing towards an optimal *Grid Power Balance* on a global level. This optimum typically is the provision of enough energy to satisfy all demands, while consuming all available energy to prevent its waste. For example, *Home A* may offer an energy surplus via its photovoltaic and due to the lack of active local consumers. The remaining entities in Figure 2.3 recognize this surplus as a consequence of the respective increase of the *Grid Power Balance*. Several alternative strategies exist to avoid wasting this surplus, which solely rely on the self-adaptation of individual entities realized by a distributed control paradigm:

- If *Home B* or the *Factory* increases its energy demand at the same time, the surplus could be used to balance the grid
- If the grid power is balanced (no simultaneously increasing demand):
 - The surplus could be an opportunity for the *Power Plant* to reduce its production
 - The surplus could be used to start other consumers, for example, in the *Factory* or any batteries

Each of these strategies is sufficient to reach an optimal *Grid Power Balance* again. However, comprehensive knowledge about the current situation in the entire grid would allow for a more qualified selection, like whether reducing the production of the *Power Plant* is the better choice than starting other consumers. A possible approach to always select the optimal strategy in any of such situations is to extend the basic self-adaptive and distributed approach by a central *Control Unit*. Hence, the *Micro Grid*

becomes a managed system that ensures reliably balancing its energy autonomously via distributed control. The *Control Unit* typically only performs optimizations, like selecting a globally optimal distribution of energy or prefer renewable energy to other sources. Further, the grid would always fall back on a reliable energy supply, if problems with the central *Control Unit* occur.

Chapter 3

Taxonomy

The characterization of the patterns as Control Action Types (CATs) uses a common schema for describing their properties consistently. We introduce this schema in this chapter by defining its constituent dimensions. These dimensions cover *general* and *multi-paradigm control* aspects as well as *demonstration* purposes. They are inspired by taxonomies used to characterize patterns in other research areas, like object-oriented design [GHJV94], programming languages [Mar96], or agent-based systems [SCH⁺02]. The result is a refinement of the taxonomy used in our previous work [KSPS21]. The explanation of this refined taxonomy relies on the system understanding and uses the terms as introduced in Section 2.2.

The *general* aspects of our taxonomy provide the following fundamental information about a CAT:

- **Name:** The unique label of the CAT, which enables its precise identification and conveys its core control mechanism. We introduce the name of the CAT in terms of the heading and as part of the first paragraph of the respective section describing the CAT in Chapter 4.
- **Intent:** The description of the control mechanism of the CAT. This includes the discussion of the intent and rationale to pursue this particular action. If necessary, we will consider additional problems motivating its application.
- **Applicability:** The discussion of any conditions that must be satisfied for a successful application of the CAT. This covers aspects ranging from general assumptions on the entire SaS to specific capabilities of the constituent entities. However, the amount and details of relevant aspects will vary with respect to the individual CAT. Further, we always assume upstream control capabilities to be available (cf. Section 2.2). For example, we will not discuss the steps of monitoring and analyzing the SaS for the selection of an appropriate CAT for a particular situation. Our focus is on the CATs and their application only.
- **Structure:** The description of the elements involved in the CAT along a graphical representation of their static relations. This always includes a control unit, which executes the CAT, and a target entity. For some CATs to work as intended, the relevant relations require additional auxiliary elements, e.g., to transmit the control action from the control unit to the target entity.

- **Dynamics:** The description of the dynamic relations between the elements introduced as part of the CAT’s structure. In particular, we discuss the communication among them along a graphical representation. Further, we consider the response of the (remaining) SaS.

The dimensions specific to *multi-paradigm control* discuss the following runtime properties of a CAT:

- **Delay:** The delay between the start of the central intervention and the point in time when the desired results have been achieved. Depending on the specific CAT, this may include the time for the response by the (remaining) SaS and, in particular, its self-adaptive entities.
- **Autonomy:** The extent to which the application of the CAT preserves the distributed self-organization of the SaS. This discussion always considers the directly affected target entity. Further, we discuss any potentially additional indirect impact on the remaining self-adaptive entities.

The *demonstration* aspects of our taxonomy cover the following practical information for a CAT:

- **Example:** The description of the application of the CAT in the context of our running example of an electric grid (cf. Section 2.3). In contrast to the following dimension, we illustrate the application of each CAT in the same context here to highlight their individual nature.
- **Usage:** The collection of known uses of the CAT in other literature. This also constitutes partially related work of this report specific to the respective CAT.

Chapter 4

Patterns

This chapter provides the detailed descriptions of the Control Action Type (CAT) patterns using the taxonomy defined in Chapter 3. These descriptions rely on our basic SaS understanding introduced in Section 2.2. Hence, we describe each CAT by an interaction between a control unit, which executes the CAT, and a self-adaptive entity, which represents the target of that control action. Following our multi-paradigm control approach (cf. Section 2.1), we assume the target entity as well as the remaining SaS entities to be distributed controlled, while the control unit deploys central control. The selection of a self-adaptive entity as the target of a central control action is subject to previous analysis and planning steps, which are out of scope of this contribution (cf. Section 2.2). Further, CAT descriptions do not consider adaptations of multiple entities simultaneously. For this purpose, the control unit either re-applies the same CAT to all target entities or executes different CATs for each target entity.

4.1 Command

Global knowledge about the current situation sometimes demands for immediately ensuring a specific adaptation of a particular entity. Hence, complex coordination tasks are not desired, like internal reasoning and decision-making by individual entities, or their interaction. In particular, in a global case of emergency, autonomous propagation of this situation between entities and their respective adaptations delay urgent reactions. The *command* pattern describes how to establish a central control mechanism for these types of situations. It bypasses the basic distributed control and the related self-adaptation processes in a SaS for the benefit of a fast and targeted adaptation.

Intent: A command represents an instruction that the target entity must carry out as specified immediately. It leaves no room for interpretation for that entity. While this erodes its autonomy [VDPV97], some safety critical situations and systems rely on the possibility of such interventions [KBB⁺16]. Commands may trigger a range of different adaptations of the target entity, like its re-configuration, forcing a specific single action, or disabling it completely. These adaptations aim at the target entity exclusively. In particular, commands do not affect the remaining entities directly.

Applicability: The control unit must know the supported commands of the target entity. The target entity must provide a corresponding interface that supports a command. Further, it must directly adhere to a received command. This requires an

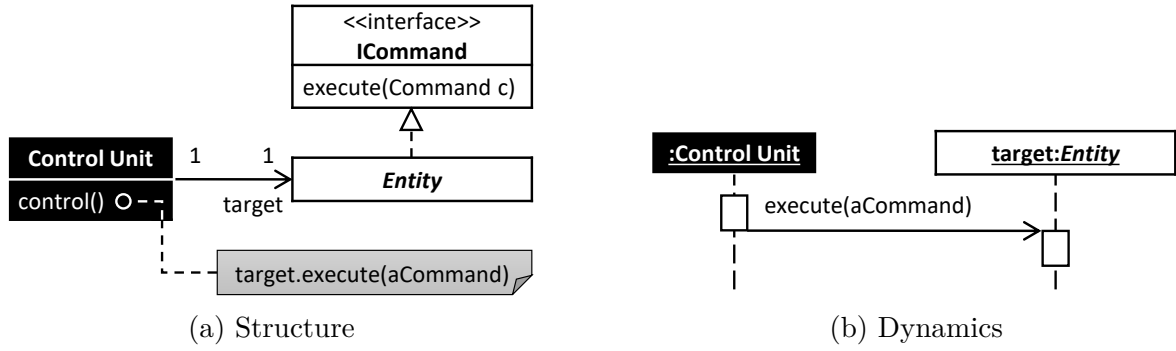


Figure 4.1: Command pattern

exclusive bypass for commands to omit the interpretation and autonomous derivation of adaptations an entity typically performs based on its usual input.

Structure: The participants of the command pattern consist of a single control unit and a single entity as illustrated in Figure 4.1a. That target entity implements the interface necessary to execute remote commands. An association between the two participants enables the direct transmission of such a command from the control unit to the entity. Communication from the entity to the control unit is not necessary. The control unit observes the result of the command by its usual monitoring of the target entity.

Dynamics: The control unit sends a command to the target entity as illustrated in Figure 4.1b. The entity processes the command and performs the desired adaptations accordingly. The command does not affect any other entities of the system. However, the resulting adaptations of the target entity may cause further adaptations of the remaining ones.

Delay: Sending a command directly to the target entity instantly triggers its respective adaptations. No further actions (and their delays) are significant to achieving the goal associated with a command. This explicitly excludes any subsequent adaptations of the remaining entities as consequences of the adaptation of the targeted one. These consequences may be an indirect effect of the pattern, but are not directly involved in it.

Autonomy: A command explicitly dictates the reaction of the target entity without any further room for interpretation. The result is a pure centralized control with a temporal, but complete loss of autonomy for this entity. All remaining entities maintain their autonomy as they are not affected by that command. Potential subsequent adaptations of other entities are the result of their autonomous reaction to the controlled adaptation of the target entity.

Example: The charging of the electric cars at *Home A* and *Home B* uses the entire renewable energy produced by the *Wind Turbines*. In the meantime, the *Factory* increases its energy demand due to rising production. The typical autonomous reaction of the *Power Plant* is to increase its energy production accordingly to keep the entire grid in balance. The *Control Unit* observes this self-adaptation and identifies the resulting reduction of the percentage of renewable energy in the grid. Hence, the control unit sends a command to each home, which forces the charging process of the electric cars to stop. The renewable energy from the *Wind Turbines* now covers the

increased demand of the *Factory*, which sustains the target percentage of renewable energy in the grid.

Usage: IBM introduces so-called touchpoints that employ commands to manage a hardware or software component of an autonomic computing system [IBM05]. Autonomic managers use these hardware or software components via their touchpoints, while each manager embodies a particular control loop to perform self-management tasks. Hence, an autonomic manager represents a control unit in our patterns with the ability to control one or more components (entities) via commands.

Said et al. use parameter tuning as well as activation and deactivation to adapt elements of a SaS [SKK⁺14]. Further, Mösch et al. propose a similar external parametrization of components in the context of organic computing [MLESA⁺06]. These actions represent specific examples of our general command pattern. In particular, setting (new) parameters for an entity or (de-)activating it explicitly are instructions that the target must carry out as specified.

Garlan et al. build their architecture-based self-repair approach on an adaptation framework, which assumes that the adaptation target provides a set of change operations [GCS03]. This provision of change operations equals the implementation of the command interface in our pattern.

Finally, Müller et al. present commands as part of a feedback control for adaptive systems [MPS08]. However, they do not explain these commands beyond their appearance as connective element between a corrector element and the executing system.

4.2 Constraint

Global knowledge about the current situation sometimes implies advantages for (parts of) the SaS, if a particular entity omits some of its adaptations. In particular, avoiding the typical behavior of an entity because of its self-adaption may contribute to a more suitable global reaction of the SaS to temporal events. An illustrative example is a highway, where the cars and construction workers of road works are self-adaptive entities. A partial speed limit for these road works forbids higher speed for the cars (while they are allowed to drive even slower) to reduce the risk of accidents for all entities. The *constraint* pattern enables such (temporal) restrictions on the adaptation and, hence, behavior of entities.

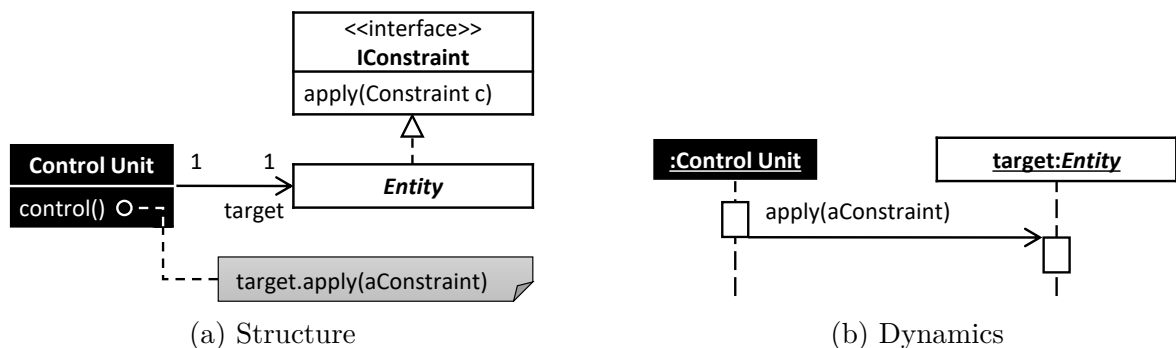


Figure 4.2: Constraint pattern

Intent: A constraint defines an explicit restriction on the adaptation space of the target entity. This restriction forbids the application of a subset of the adaptations available in that space immediately. Depending on the available adaptations, a constraint may restrict the set of executable actions or even the entire behavior of the target entity [Fis06, KBB⁺16]. Hence, the target entity autonomously selects the optimal adaptation as usual, but within the limits of the received constraint. The remaining entities remain unaffected.

Applicability: The control unit must know the adaptation space of the target entity and its supported expressiveness to define valid constraints. The adaptation space provides the basic operands and their values, like configuration parameters and their valid ranges, or the set of executable actions and their triggers. The supported expressiveness defines the available operators to connect operands and values for restricting this adaptation space. The target entity must provide a corresponding interface that allows the communication of such constraints. Further, it must be willing to adhere to a received constraint. This requires ignoring those adaptations during its adaptation planning, which the received constraint forbids explicitly.

Structure: The static view on the elements of the constraint pattern reveals the same participants as in the command pattern (cf. Section 4.1). The differences reside in the interface and the association between those elements. In Figure 4.2a, the target entity implements the interface necessary to receive remote constraints. Further, the association enables the transmission of such a constraint from the control unit to the entity. Communication from the entity to the control unit is, again, not necessary as the correct application of the constraint becomes observable via the usual monitoring of the target entity.

Dynamics: The control unit sends a constraint to the target entity as illustrated in Figure 4.2b. The target entity applies the constraint immediately. This application results in a constant restriction of its adaptation space as defined by the constraint. The removal of this restriction requires the transmission of new constraint information by the control unit. In particular, new constraint information may force the removal of an actively maintained restriction to fully release the adaptation space again. While a constraint does not affect the remaining entities of a SaS directly, they may react to the absence of the forbidden adaptations of the target entity.

Delay: Sending a constraint directly to the target entity instantly triggers its application and, hence, the defined restriction of the entity’s adaptation space. Further, this application must trigger a check, whether the current state of the target entity violates the constraint. In case of constraint violation, the target entity immediately adapts in accordance to its restricted adaptation space. If the target entity already satisfies the constraint, it does not perform further actions, but to maintain this satisfaction in future. Hence, in both cases, no significant delays for applying and satisfying the constraint occur.

Autonomy: The autonomy of the target entity is mostly preserved. It keeps its control authority within the limits of the restriction defined by the constraint. However, a restricted adaptation space partially limits the autonomy of the target entity as some potential adaptations are no longer available. In general, the more restrictive a constraint is, the higher is also the loss of autonomy for the target entity. The remaining entities maintain their autonomy. Hence, they may react autonomously to the results of the application of the constraint by the target entity.

Example: In the evening of a typical working day, the people return to *Home A* and *Home B* with their electric cars. The simultaneous charging of multiple electric cars results in a spike of energy consumption. Hence, the *Power Plant* would autonomously increase its energy production to keep the entire grid in balance. However, the *Control Unit* already has the information that the wind will increase during the night. This increase will enable the *Wind Turbines* to generate enough renewable energy to charge the electric cars. The control unit therefore sends a constraint to (the charging stations of) *Home A* and *Home B*, which reduces their energy consumption and consequently avoids the increasing production of non-renewable energy by the *Power Plant*. During the night, the control unit replaces this constraint by a less restrictive one to make use of the available wind energy for charging the electric cars.

Usage: Minsky proposes the usage of the Law-Governed Interaction (LGI) control mechanism to support self-healing in open distributed systems [Min03]. A law in the LGI control mechanism defines regulations for the interaction of agents of the system. In particular, regulations restrict the effect that events should have on an agent. While the restrictions apply to the interaction of agents exclusively, laws and regulations represent a special instance of constraints in the sense of our pattern catalog.

Kantert et al. introduce a system-wide control loop for guided self-organization [KTK⁺16]. The controller part of this control loop creates and applies norms to prevent negative emergent behavior of the system. A norm defines incentives and sanctions, which individual agents of the system perceive as a response to their contribution to the overall system behavior. Receiving incentives for beneficial or sanctions for malicious behavior both represent a form of continuous manipulation of self-adaptive elements towards a certain behavior. Hence, norms indirectly restrict the adaptation space as the agents typically strive for incentives and try to avoid those behaviors that cause sanctions. Synonyms for this subcategory of our general constraints are social constraints [PSA11], social laws [ST92] or, social control in general [HW11].

4.3 Influence

The direct control of an entity via commands and constraints sometimes is not an adequate choice. For example, these CATs restrict the autonomy of the target entity too much, their application does not lead to the desired result, or the target entity does not support them at all. For such situations, global knowledge about the relations between entities of the SaS allows to establish an indirect control. The *influence* pattern leverages this knowledge to manipulate the target entity via direct control of a related entity.

Intent: An influence aims at a specific adaptation of the target entity in an autonomous way. Hence, it does not directly address that entity, but causes a change in its environment, which triggers a particular reaction. A command or a constraint for an entity related to that target can cause such a change via interaction or impact as described in Section 2.2. This related entity becomes the controlled influencer to trigger a specific adaptation of the target entity. While this results in the respective loss of autonomy for the influencer, the target entity and all other entities of the SaS remain autonomous. In this way, this pattern also supports situations where a certain openness exists about which entity will ultimately react.

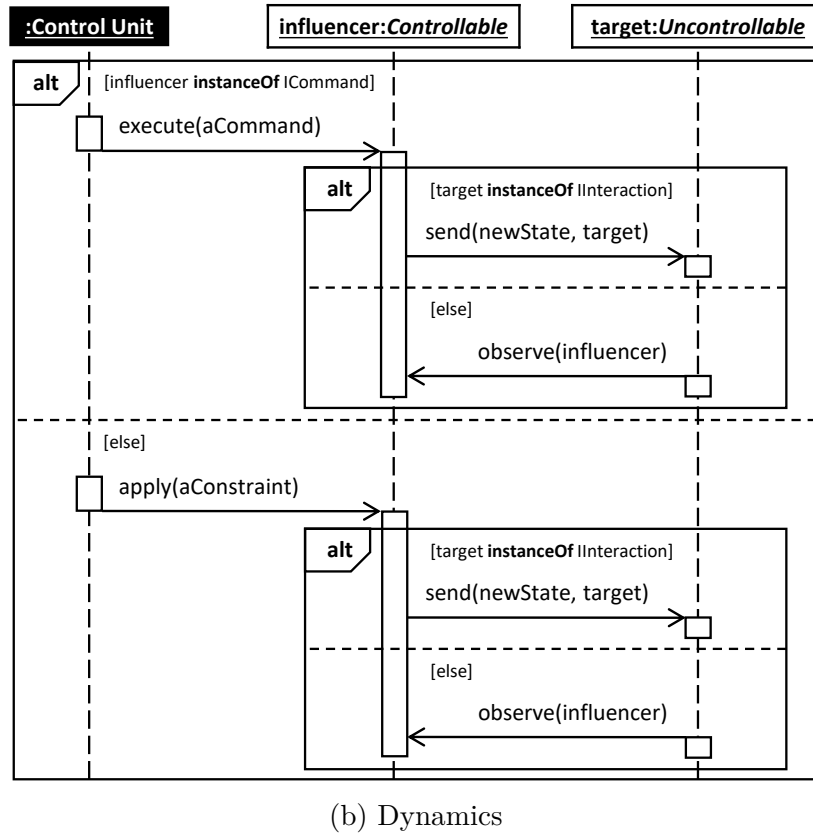
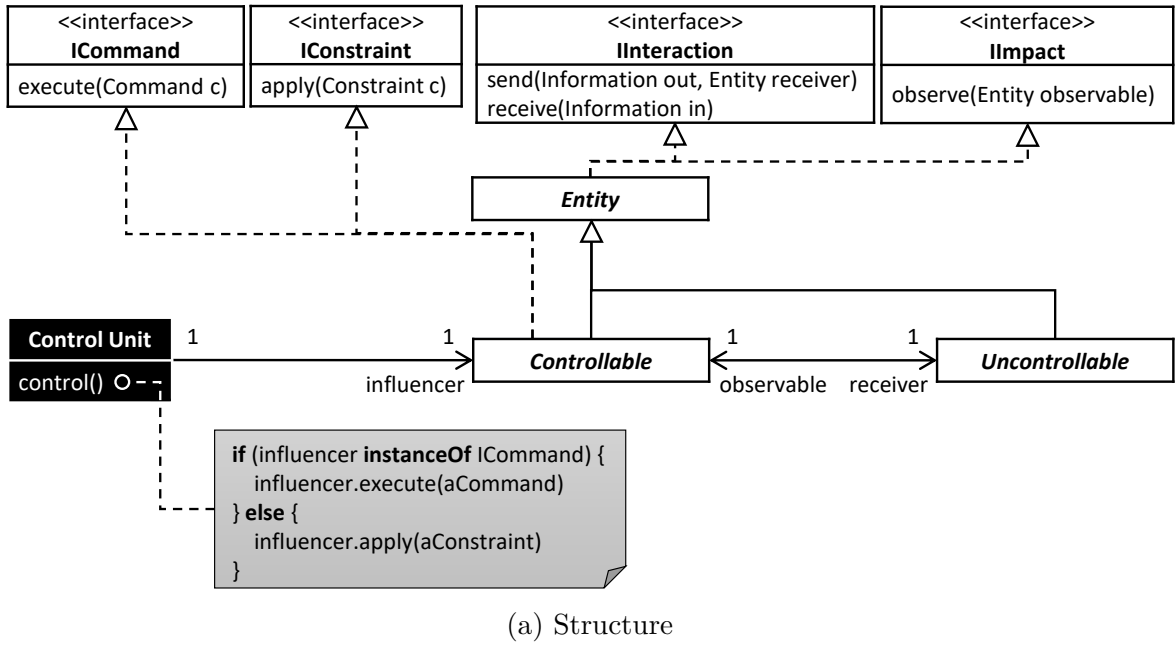


Figure 4.3: Influence pattern

Applicability: The control unit must know the entities of the SaS and their relations to identify an appropriate entity as influencer for the target entity. This identification process must guarantee that the influencer satisfies the following requirements:

- The entity must have an influencing relation to the target entity, which is:

- Either a direct interaction between that entity and the target entity
- Or an indirect impact as the target entity observes that entity
- The entity must support direct control via commands (cf. Section 4.1) or constraints (cf. Section 4.2)

Depending on the supported direct control of the selected influencer, the control unit must know the commands for that entity or its adaptation space. In particular, it must know the specific command or constraint for the influencer, which will indirectly trigger the desired adaptation of the target entity.

Structure: The influence pattern distinguishes between controllable and uncontrollable entities as illustrated in Figure 4.3a. The former implements at least one of the interfaces, which enable its external control via commands (cf. Section 4.1) or constraints (cf. Section 4.2). The latter does not provide any external control interfaces, but has an association to the former, e.g. to provide certain domain functionality collectively as described in Section 2.2. For this purpose, both entities implement at least one of the interfaces, which enable direct interaction or indirect impact. Hence, either the controllable entity can send its new state as a result of its adaptation to the uncontrollable receiver, or the uncontrollable entity can observe this adaptation on its own. In both cases, the association enables the uncontrollable entity to react to adaptations of the controllable one. A second association from the control unit to the controllable entity allows the transmission of commands or constraints depending on the interfaces implemented by the controllable entity. However, that entity does not represent the actual target for the control unit. The controllable entity acts as an influencer for the uncontrollable entity by leveraging the relation (association in Figure 4.3a) between them.

Dynamics: The control unit sends a command or a constraint to the influencer as described in Section 4.1 and Section 4.2, respectively. In Figure 4.3b, the decision on which of these previous CATs to use solely depends on the interface implemented by the controllable entity. If the influencer implements both interfaces, this decision must also consider which CAT results in an adaptation of the controllable entity that influences the uncontrollable target as desired. We omit this complex planning step in Figure 4.3b for brevity. The actual influence occurs depending on the interface, which realizes the relation between the two entities. On the one hand, the influencer may directly send its new state after its adaptation to the target¹, which reacts accordingly. On the other hand, the target observes the adaptation of the influencer, which triggers a corresponding reaction. In the same way, any remaining entity of a SaS with similar relations to the influencer or the target entity may react to their resulting adaptations.

Delay: Influencing a target entity consists of two fundamental steps: the direct control of the influencer and its subsequent influence on the actual target. The first step solely relies on CATs without any significant delay (cf. Sections 4.1 and 4.2). The second step leads to a delay until the target entity performs the ultimately intended adaptation. The extent of this delay depends on several factors, like the specific type of relation between influencer and target, the steps and processes defined by their interaction protocol, and the actual adaptation the target must perform. In particular, the

¹Another way of direct interaction is that the influencer sends its intent to adapt in a certain way before its actual adaptation.

actual adaptation may vary between an almost instant switch of a single configuration option to complex rebinding of several internal components. Further, the influencer may trigger adaptations of entities outside the scope of this pattern. This side-effect may result in additional influences on the adaptation of the target entity and, hence, the delay of this CAT.

Autonomy: The preservation of the autonomy of the involved entities depends on their respective role in this CAT. The influencer loses its autonomy at least partially in case of using constraints to apply the initial direct control (cf. Section 4.2). If the control unit uses commands for this control, the influencer loses its autonomy completely (cf. Section 4.1). The actual target in this CAT preserves its autonomy. While this CAT influences the target on purpose, it uses no other mechanisms than the usual relations between entities and their self-adaptation to induce that influence. Hence, the remaining entities also maintain their autonomy and may react autonomously to the results of the direct control of the influencer and its influence on the target entity.

Example: In this particular example, we define the *Factory* to be an uncontrollable entity, e.g. for compliance reasons. The remaining grid entities are controllable and, hence, represent potential influencers for the uncontrollable *Factory* by leveraging their direct and indirect relations. In this setting, the *Factory* increases its production unusually in the morning to compensate an error in its production system and the resulting loss of production during the night. This leads to an unscheduled increase in its energy demand without prior notice. The *Wind Turbines* as well as the photovoltaics at *Home A* and *Home B* already provide their weather-dependent maximum of renewable energy to the grid instead of loading their empty batteries. Hence, the autonomous reaction of the *Power Plant* is to increase its energy production to keep the entire grid in balance. The *Control Unit* observes the decrease of the ratio of renewable energy over the entire grid because of these local autonomous adaptations. Further, it detects the uncontrollable *Factory* to be the driving force for this global change. Based on previous interactions, the *Control Unit* is aware of the typical coordination activities between the controllable *Power Plant* and the *Factory*. It therefore selects the *Power Plant* as its influencer and sends a constraint, which prohibits its additional energy production. As part of applying this constraint, the *Power Plant* informs the *Factory* about the upcoming decrease of energy provision via their direct interaction relation. In the end, the *Factory* has no other reasonable choice than to reduce its production to a usual amount again.

Usage: Fisher defines influence in the context of emergent systems of systems as "*any mechanism by which one entity interacts with another in a way that changes the physical, informational, or emotional state of the other*" [Fis06]. This definition summarizes the basic principle of our eponymous pattern. However, Fisher does not include a particular control hierarchy. While our pattern assumes a control unit to initiate an influence explicitly via commands or constraints, Fisher considers all entities to be equal.

Johnson and Brown demonstrate the control of the collective behavior of robot swarms via the manipulation of their environment [JB15]. In particular, the relocation of their target(s) causes the robots (our entities in this report) to autonomously change their behavior accordingly. Considering the target of a robot to be part of its environment, the notion of influence by Richter et al. generalizes from the specific context of robotics. They discuss influencing an entity via its environment as a form of

central control in general [RMB⁺06]. Their goal of applying such influence is to disrupt emergent behavior of decentralized systems. For this purpose, Schmeck et al. include environmental influence in their architecture for controlled self-organization [SMSc⁺10].

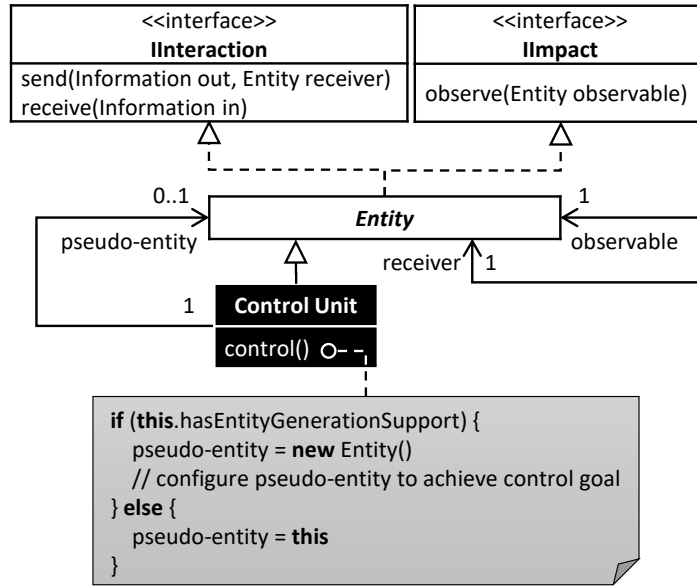
4.4 Pseudo-Emergence

In some SaS, neither direct control via commands or constraints nor indirect influence represent applicable CATs. Reasons range from being inadequate or even incapable to achieve the desired adaptation to a total lack of their support. In particular, in a complex SaS (cf. Section 2.2), related self-adaptive entities (again individual SaS) may not provide the necessary interfaces for such (external) control. However, global knowledge about the self-adaptive and emergent behavior of the entities enables applying control in an emergent way. In the *pseudo-emergence* pattern, the control unit leverages this knowledge to influence the target entity by integrating a pseudo-entity within the SaS.² Hence, this CAT is related to the influence pattern (cf. Section 4.3), but deploys a special entity for this purpose instead of selecting an influencer from the existing ones.

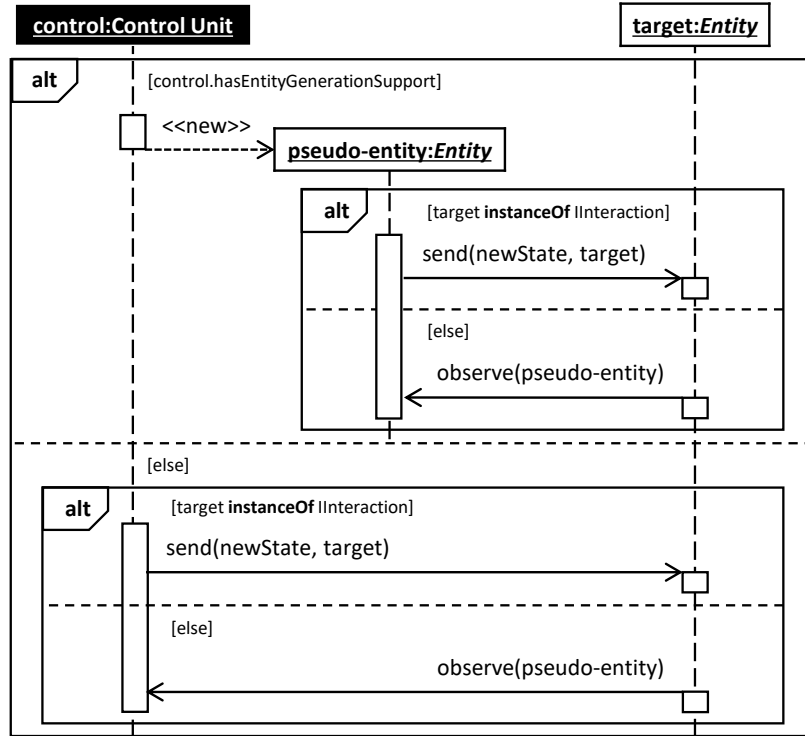
Intent: Pseudo-emergence describes the deliberate exploitation of a system’s self-adaptation to assert control. It does not directly address any entity of a SaS, but causes a change in the environment of a target entity to trigger its adaptation. For this purpose, it integrates a pseudo-entity into a SaS, which leverages the usual relations between entities to influence the target. The SaS and its entities perceive this integration as a typical system extension as the pseudo-entity imitates the behavior of a native type of entity. In particular, the target entity relates to the pseudo-entity in the same way as to any other instance of that native type. Hence, the target entity and all other entities of the SaS remain autonomous, which also results in a certain openness about which entity will ultimately react.

Applicability: The SaS must allow integrating additional entities at runtime. In particular, the SaS or the individual entities must provide a mechanism, which relates additional and existing entities by establishing direct interaction or indirect impact (cf. Section 2.2). The control unit must know the entities of the SaS and their relations to create stimulus-response-relations of the target entity. These relations define which information or action of which type of related entity triggers which adaptation of the target. The control unit uses this information to create the correct type and behavior for the pseudo-entity. It therefore must generate a new entity, configure it to achieve the desired adaptation of the target, and deploy it as pseudo-entity in the SaS. The control unit may alternatively deploy itself as such pseudo-entity. This alternative requires a control unit design, which allows using the same relations that typically exist between entities exclusively.

²The terms *pseudo-emergence* and *pseudo-entity* refer to the imitating nature of this pattern, which deploys a new entity to the emergent collective, while its only purpose is to execute a certain control instead of contributing to the domain functionality.



(a) Structure



(b) Dynamics

Figure 4.4: Pseudo-emergence pattern

Structure: The pseudo-emergence pattern treats all constituent elements of a SaS as generic entities. Hence, Figure 4.4a only includes such a single type of entity, which implements (at least one of) the interfaces for direct interaction and indirect impact to enable the association between its instances in the roles of receiver and observable (cf. Section 2.2). In order to illustrate both ways of defining the pseudo-entity, the control unit has an association to the generic entity and additionally inherits from

it. The association allows creating and configuring a new entity as pseudo-entity, if the control unit supports such entity (code) generation. The inheritance defines the control unit also to implement the interfaces, which usually enable direct interaction or indirect impact between entities only. Hence, the control unit can define itself as the pseudo-entity to send states of any kind to the target, or to perform arbitrary actions, which the target will observe. In both ways of defining the pseudo-entity, the self-association of the generic entity enables the target entity to react to adaptations of the pseudo-entity, which it perceives as a typical change in its environment.

Dynamics: The control unit either creates a new entity as the pseudo-entity or defines itself as pseudo-entity to relate with the target. In Figure 4.4b, we only indicate these preliminary steps, but omit implementation-specific details, like the exact way of configuring and integrating the pseudo-entity. The pseudo-entity then either directly sends information to the target entity, or performs actions, which the target entity will observe. In Figure 4.4b, the decision on how to initiate this self-adaptation of the target solely depends on the interface it implements. However, this decision must also consider the target’s adaptation space and the environmental conditions that enact specific adaptations of the target (stimulus-response-relations). The result of this decision process is a particular information or action, which will trigger the desired adaptation of the target and, hence, defines the specific type of relation to use. While the pseudo-entity must perform to observable action to create the respective stimulus for the target, it must not necessarily realize what it transmits as information only. For example, it may send a new state information to the target without actually switching to this state as the sole information about that adaptation is sufficient to trigger the target’s response (illustrated by the *newState* in Figure 4.4b). Independent of the particular way of influencing the target entity, it finally adapts in response. Further, all remaining entities of the SaS, which also relate to the pseudo-entity autonomously, may react to its information and actions.

Delay: In general, influencing a target entity leads to a delay until the target performs the desired adaptation (cf. Section 4.3). Pseudo-emergence requires the additional upfront integration of a pseudo-entity potentially including its creation and configuration. This integration extends the basic delay of the influence up to a significant degree, if creating a new entity as pseudo-entity is necessary for each control action. However, if the control unit integrates itself as pseudo-entity into the SaS constantly, all subsequent control actions become faster after this initial integration. Independent of the particular way of integration, the pseudo-entity may trigger adaptations of entities besides the targeted one. This may in turn cause additional influences on the target and, hence, on the delay of this CAT.

Autonomy: Pseudo-emergence represents the only CAT in this pattern catalogue, which completely preserves the autonomy of the entire SaS. It leverages the knowledge about and the mechanisms of the usual relations between entities and their self-adaptation. In particular, it only influences the target entity by manipulating its environment and lets it react to that influence independently. The creation and integration of the pseudo-entity to induce such influence has no impact on the autonomy. However, this preservation of autonomy results in an uncertainty on the details of the effect of an influence. Similar to the corresponding CAT in Section 4.3, any entity of the SaS may react autonomously to the results of pseudo-emergence and its influence on the target entity.

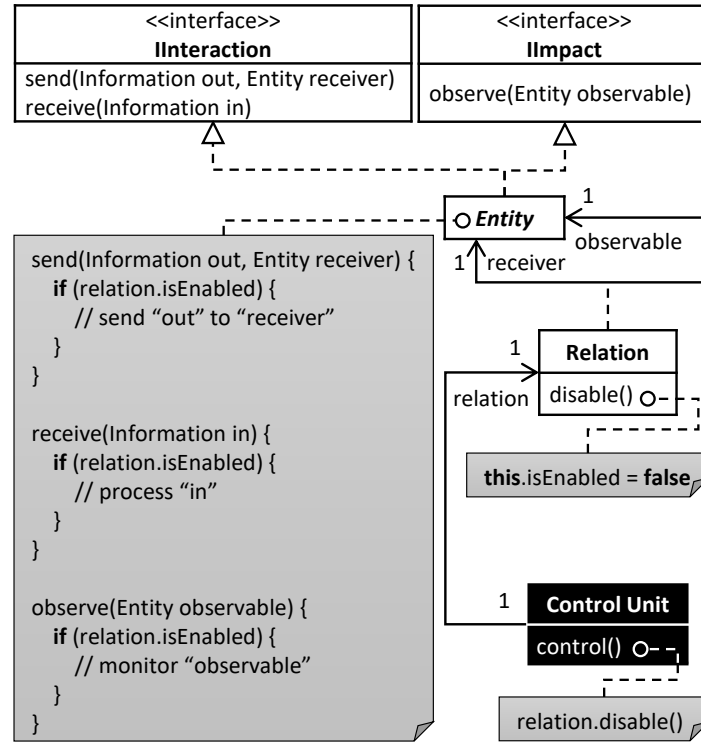
Example: The general policies for the photovoltaics at *Home A* and *Home B* define the provision of their surplus to the electric grid for further distribution, if no local demand exists. Hence, the photovoltaic at *Home A* provides its total energy to the grid as the residents are on vacation and the battery is already fully charged. The photovoltaic at *Home B* instead provides its total energy to local consumers. The additional charging of the battery using renewable energy at *Home B* is therefore not possible. The *Control Unit* observes these local situations and identifies a more reasonable use of the renewable energy at *Home A* by charging the battery at *Home B*. Due to missing interfaces to directly instruct the photovoltaic and the battery, the *Control Unit* establishes a pseudo-entity. The pseudo-entity acts as an energy consumer for the photovoltaic at *Home A* and as an energy provider for the battery at *Home B*. This temporal deployment of the pseudo-entity in the electric grid enables redirecting the surplus to the demand.

Usage: Karuna et al. present the design of a manufacturing coordination and control system based on the concepts of multi-agent systems and emergence [KVSG⁺05]. The inherent control mechanisms includes an activity, which shares the core idea and principles of our pseudo-emergence pattern. In this activity, a parent agent creates new agents on demand to prepare manufacturing resources (again represented as agents) for their later usage. This creation of new agents corresponds to the integration of pseudo-entities, while the preparation represents a special form of influencing the remaining system. Further, pseudo-emergence shares some concepts with influencing as described in [Fis06, RMB⁺06]. In particular, it extends this basic action by (optionally generating and) integrating a pseudo-entity as influencer into the system, which actually performs the influence.

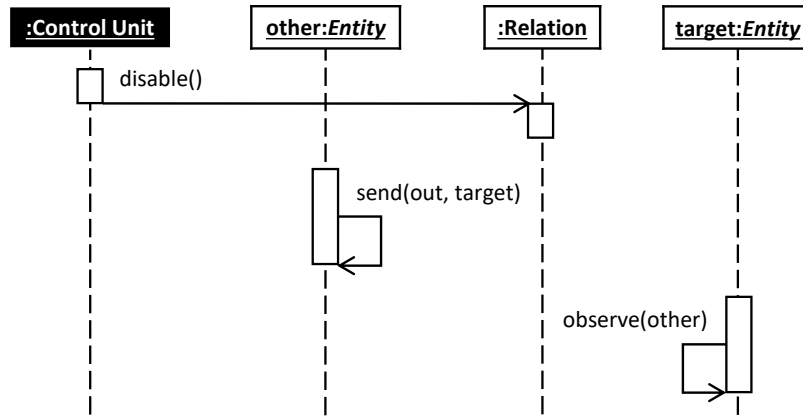
4.5 Isolation

Special situations sometimes require the complete exclusion of an entity from the SaS. For example, the target entity constantly disrupts (parts of) the SaS due to malfunction or bad faith, which also excludes other forms of interaction with that entity. Further, maintenance and testing of the target may require its temporary isolation to prevent side-effects on the remaining SaS during these tasks. The *isolation* pattern represents a CAT, which realizes such a form of control. In particular, it provides an alternative to a complete deactivation of the target entity. While deactivating the target results in a similar effect on the remaining SaS (the target entity is no longer available and the SaS must adapt to this new situation), the target may not support this direct control due to malfunction, bad faith, or missing interfaces. Maintenance or testing tasks may even require that the target entity remains active.

Intent: Isolation disables the relations between the target entity and the remaining ones to prohibit their direct interaction or any indirect impact. For this purpose, it must not trigger adaptations of entities. However, all affected entities perceive this disabling as a change in their environment, which may cause respective adaptations as sequential events. In contrast to the other CATs in this catalogue, these adaptations are not the intention of this pattern. The core of isolation is to prohibit any interactions or impacts between the target and the remaining entities of the SaS. Further, the target remains active either on purpose or due to its missing support for remote deactivation.



(a) Structure



(b) Dynamics

Figure 4.5: Isolation pattern

Applicability: The SaS must allow accessing its network infrastructure. In particular, the SaS or the individual relations must support a mechanism to disable specific relations between entities on demand. This mechanism may either be an exclusive capability of the control unit (e.g., in terms of a dedicated network management module), or may be available via application of other CATs to network management entities of the SaS. Indeed, the specific realization of this mechanism depends on the particular infrastructure of the SaS. The control unit must know the entities of the SaS and their relations. Further, it must identify the subset of relations from the target entity to the remaining ones and vice-versa to disable these relations only.

Structure: The focus of the isolation pattern is on the relations between entities. In order to establish such relations, the influence pattern already introduced the inter-

action and impact interfaces, which an entity typically implements (cf. Section 4.3). Figure 4.5a provides more details on that interface implementation in a comment related to the generic entity. These details outline a dependency of the execution of the core interaction and impact actions on the state of the involved relation introduced as eponymous association class. The main purpose of these details is to illustrate that entity interaction or impact only occurs, if the respective relation is enabled. Other ways of realizing equal dependencies exist, which may be more suitable for the particular SaS implementation at hand. The core of the isolation pattern is the association between the control unit and the relation (association class). It allows disabling a relation by setting its state accordingly. In Figure 4.5a, a simple Boolean variable represents this state. If set to false, it leads to the desired suppression of core interaction and impact actions of the related entities.

Dynamics: The default state of any established relation between entities always enables typical interactions and impacts as illustrated for previous patterns, like in Figure 4.3b or Figure 4.4b. Hence, we assume this default state for the relation between the target entity and the other entity in Figure 4.5b as an origin to describe the dynamics of the isolation pattern. In this dynamical view, the control unit disables the relation between the target entity and the other entity. The relation switches its state accordingly leading to a blocking of the interactions and impacts between these entities. Figure 4.5b illustrates this blocking by self-calls of the target and the other entity, which would otherwise have reached the other one, respectively. In order to isolate the target completely in practice, the control unit disables all other relations involving the target in the same way. The related other entities of the SaS may react to this isolation by individual adaptations.

Delay: The isolation of a target entity instantly applies to the entire SaS. In particular, neither disabling the relations between the target and the remaining entities nor blocking their interactions and impacts implies significant delays. Similar to the command pattern (cf. Section 4.1), this explicitly excludes any subsequent adaptations of the remaining entities as consequences of the isolation. These consequences may be an indirect effect of this pattern, which do not affect the target.

Autonomy: Isolation preserves the autonomy of individual entities as it aims at the relations between them only. It does not directly address an entity or its adaptation space. However, it restricts the autonomy of the SaS as a whole due to its active change of the network topology usually created collectively by the entities. The resulting loss of relations may trigger adaptations of the target as well as the remaining entities, but without further central intervention.

Example: On a calm day, the *Wind Turbines* suddenly start to provide a significant amount of energy to the grid. The *Factory* autonomously increases its production therefore above its normal level for this time of day. The *Control Unit* observes both abnormal behaviors and identifies a malfunction of the batteries at the *Wind Turbines*. This malfunction causes the batteries to supply their energy to the grid instead of saving it for the night. However, the *Wind Turbines* including their batteries do not respond to any external control action. Hence, the *Control Unit* decides to isolate the *Wind Turbines* as they should not have contributed to the grid power balance anyway (safe exclusion regarding the power supply of the remaining grid). It disables their impact relations to the other grid elements, which ultimately results in cutting of the energy flow from and to the *Wind Turbines*. The subsequent decrease of available

energy forces the *Factory* to decrease its production to its normal level. The later repair of the batteries also enables the relations between the *Wind Turbines* and the other grid elements again.

Usage: Kramer and Magee propose an architectural reference model for self-managed systems [KM07]. This model contains a component control layer for which the authors propose their change management algorithm [KM90] to ensure stable conditions before changing components. One example for such stable conditions is that components must be isolated before they can be safely removed. Our pattern targets exactly such an isolation, while the ultimate removal of an isolated component represents an extension.

Minsky includes message transformation and rerouting as possible regulations defined by LGI laws (cf. Section 4.2) for controlling the message exchange between entities [Min03]. Further, Richter et al. introduce the modification of the interconnection network of entities to influence the system structure and, hence, the global system behavior [RMB⁺06]. These approaches represent a weak form of isolation as defined by this CAT. In particular, both approaches do not aim at a complete isolation of an entity explicitly. However, we consider the basic mechanisms of these approaches to be capable of achieving this kind of complete exclusion.

Chapter 5

Discussion

The patterns of our catalog provide a systematic overview of the range of possible approaches to establish multi-paradigm control for a SaS. Each CAT defines a specific type of centralized control action with its individual impact on the general autonomy of the SaS entities. Table 5.1 summarizes these key properties from Chapter 4 and classifies them as advantage and disadvantage of the respective CAT. Hence, this table outlines their basic trade-offs. The details for an individual CAT can then be found in the respective section as listed in the table.

Table 5.1: Summary and classification of presented control action types

CAT	Advantage	Disadvantage
<i>Command</i> (Section 4.1)	Instant action with known unique result	Total loss of autonomy for target entity
<i>Constraint</i> (Section 4.2)	Instant action with known range of results	Partial loss of autonomy for target entity
<i>Influence</i> (Section 4.3)	Preservation of autonomy for target entity	Delayed action with uncertain range of results
<i>Pseudo-Emergence</i> (Section 4.4)	Preservation of autonomy for all entities	Delayed action with uncertain range of results
<i>Isolation</i> (Section 4.5)	Instant action with known unique result; preservation of autonomy for all entities	Total loss of target entity

In practice, Table 5.1 serves as a starting point to access the catalog and to decide which CAT to consider when designing a SaS with multi-paradigm control. For example, the control unit may represent the last instance for averting errors and failures in a SaS. A designer therefore may choose between *command* and *constraint*, which both enable immediate reactions. However, their results and inherent loss of autonomy for a target entity vary. Hence, the designer either has to balance between those properties or decides to integrate both CATs into the SaS. The latter requires the additional def-

initiation and realization of respective triggers to select the appropriate CAT for a given situation at runtime. If the self-adaptability of the entities in a SaS already provide an appropriate level of reliability and robustness, the control unit may be a means to optimize the SaS based on global knowledge additionally. In this case, the focus is on preserving autonomy rather than immediate corrections, which *influence* and *pseudo-emergence* guarantee in varying degree. Other reasons to select one or more of the CATs as well as other combinations are possible. *Influence* may prevent harm in an early stage by leveraging SaS autonomy. *Isolation* may optimize a SaS in its entirety by excluding a particular interfering entity.

In research, the pattern catalog contributes a systematic understanding of control options for multi paradigm control of SaS. Hence, it tackles the challenge of integrating distributed and central control [MEHdH13] and fills an already identified research gap [dLGM⁺13, BDMSG⁺09] (cf. Chapter 1). Further, this catalog is a source for new research questions, which are beyond the scope of this report. For example, the questions arise whether more CATs with different properties are needed or which combinations of them are meaningful with respect to the properties of the SaS and its application environment. In this sense, researcher may build on this catalog for classifying their approaches. This facilitates their comparison and results in a continuously maintained collection of CATs, which benefits research as well as practice.

Chapter 6

Related Work

In this report, we introduce Control Action Types (CATs) as patterns of interaction between a central controller and a distributed controlled SaS to achieve a desired adaptation. The particular focus of these patterns is on the application and execution of specific actions. Hence, we target the final step of an iteration of a SaS control loop exclusively, like *execute* [KC03], *act* [Boy76], or *enact* [GCGL15]. Our discussion of related work therefore starts with other patterns and templates for this particular step, but with a different focus or for a different type of target system. For complex autonomous or self-adaptive systems, we discuss relations to other generic descriptions of control aspects afterward. In general and to the best of our knowledge, no catalog of central control patterns for distributed controlled SaS in the presented form exists.

Krupitzer et al. differentiate 55 SaS design patterns from their systematic literature review into seven categories [KTPR20]. One of these categories focuses on the execution step of the MAPE [KC03] control loop exclusively. While the identified design patterns target decentralized coordination in general, the particular subset in the execution category provides an overview on different patterns for realizing adaptation (actions). However, as being an overview, the authors do not detail the individual findings in the way we describe our CATs. In contrast, we would add our report as another contribution to their execution category. In that sense, we must discuss the other papers in this category as follows:

- Abuseta and Swesi [AS15] present one design pattern for each step of the MAPE control loop. The pattern for the execution step describes how to establish the relations between the components that accomplish the execution activity. The authors do not discuss the possible types of actions to perform, like we do with our individual CATs.
- Said et al. [SKK⁺14] also propose one design pattern for each step of a control loop in the particular context of self-adaptive real-time embedded systems. In contrast to Abuseta and Swesi, the pattern for the acting step includes specific actions, which we generalize to our command pattern in Section 4.1 (see the usage dimension in that section for a detailed comparison).
- Iglesia and Weyns [IW15] formalize behavior specification templates for each step of the MAPE control loop. An important characteristic of the target systems of these formalizations is the changing availability of resources. Hence, their behavior template for the execution step focuses on adding and removing resources ex-

clusively. While we could perform the pure addition or removal via the command pattern (cf. Section 4.1), the behavior template includes additional activities to ensure correctness of these actions.

- Ramirez and Cheng [RC10] define four reconfiguration patterns to perform structural and behavioral adaptations of a SaS. These patterns focus on the internal procedures and states of the affected components as well as the entire system to perform the respective adaptations safely. Hence, we consider these reconfiguration patterns as potential approaches to realize the internal changes of target entities induced by applying our CATs.

In addition to the survey of Krupitzer et al., we identified the following work on adaptation actions and strategies. Salehie and Tahvildari differentiate twelve adaptation actions by their impact on a SaS, their execution time, required resources, and complexity [ST09]. In our understanding, these adaptation actions represent entire tactics, which include multiple actions in the sense of our patterns. Musli et al. identify three multi-adaptation patterns for the design of Cyber-Physical Systems (CPS) with self-adaptation capabilities [MMW⁺17]. These patterns define different combinations of multiple types of self-adaptation spanning multiple layers of a CPS. Hence, these patterns are significantly more complex than our CATs, which is partially due to the nature of their target system type. Alfonso et al. present four adaptation strategies derived from their systematic literature review on self-adaptation for Internet of Things (IoT) systems [AGCC21]. Each strategy describes a form of system adaptation as a reaction to particular events or changes in the system’s environment to maintain a certain Quality of Service (QoS). While our patterns also aim at triggering certain adaptations, we do not particularly focus on QoS of individual entities or the entire system besides their autonomy. However, the CATs in this report may support the operationalization of the strategies by Alfonso et al. in practice.

Besides generic descriptions for the realization of the execution step, other patterns focus on its distribution as part of an entire control loop. de Lemos et al. briefly introduce five patterns for interacting control loops as part of their broader research roadmap for engineering SaS in general [dLGM⁺13]. Each pattern describes a particular way of orchestrating entire MAPE control loops or their individual steps¹. Weyns et al. extend these basic descriptions using a specific graphical notation to capture the interactions and a common schema for describing the same patterns systematically [WSG⁺13]. Quin et al. map these patterns in [WSG⁺13] to coordination mechanisms, which further extends their description by how loop and step interactions are realized in other literature [QWG21]. In general, the individual patterns define different degrees of decentralization of control loops (steps) to manage adaptations in complex SaS sufficiently. In contrast, our patterns define different types of central control actions to achieve a desired adaptation of a particular entity within a SaS. Our focus is on the interaction between a central control unit and those distributed controlled entities. While the control unit may realize an entire control loop, our patterns contribute to the realization of the execution step exclusively.

An extension of distributing individual steps of a control loop is the distribution of the entire loop in terms of a controller. Schmeck et al. provide three different

¹While the authors focus on the MAPE control loop exclusively, the patterns are also applicable with other loops, like OODA [Boy76] or CARE [GCGL15].

architectural variants to realize observation and control in organic computing systems [SMSc⁺10]. Each variant defines a specific distribution of observer and controller instances with its individual impact on the autonomy of the system they observe and control. The authors therefore define a degree of self-organization as synonym for autonomy, which relates the number of controllers to the number of elements of the system. Based on this definition, the presence of a single controller for all elements implies central control and, hence, weak self-organization (no autonomy). If the number of controller instances is greater or equal to the number of elements, the system is strongly self-organized (autonomy). The impact on the autonomy of the SaS is also an important dimension, which we discuss for each of our patterns. However, we do not quantify the resulting degree of the autonomy, but describe it by the freedom that the target entity and the remaining system have for reacting to the application of a particular CAT. In general, our patterns define control actions instead of changing distributions of controlling elements in a system. Weyns and Andersson derive three architectural styles for self-adaptation in systems of systems from classic control architectures [WA13]. Each style provides its individual level of knowledge sharing and collaboration between the constituent systems and, in particular, between the managed systems and their controller. In general, the authors focus on the distribution of controller instances and the data flow within the resulting systems. The descriptions of our patterns also consider data flow between the central controller and the distributed controlled entities. However, our patterns define control actions instead of changing distributions of controlling elements in a system.

Finally, design patterns for the creation of entire control systems exist. Patikirikoralala et al. derive nine patterns to design control systems for SaS in their technical report [PCHW12b]². Wooldridge and Jennings report four different architectures for control systems of autonomous agents (which can be interpreted as special type of SaS) [WJ95]. Both the patterns as well as the architectures support realizing entire control systems, but do not consider individual control actions, like our CATs.

²The related symposium paper [PCHW12a] does not include these patterns.

Chapter 7

Conclusion

A reasonable approach to minimize the quality trade-offs between distributed and central control in Self-adaptive Systems (SaS) is to integrate both paradigms into a single multi-paradigm control solution. The patterns presented in this report contribute to such a solution. They define a set of Control Action Types (CATs), which describe specific forms of interactions between a central controller and a distributed controlled SaS. We describe each CAT along various dimensions of a taxonomy. This taxonomy covers general aspects, like the problem a particular CAT addresses, as well as aspects specific to multi-paradigm control, e.g., its impact on the autonomy of affected system entities. Further, a running example from the electric grid domain shows the application of each CAT.

Our pattern catalog supports practitioners in the design of their SaS when following a multi-paradigm control approach. We therefore summarized their most important trade-offs: the delay and predictability of control results in contrast to the loss of autonomy for the affected system (entities). In general, the faster and more precise a CAT is, the higher is the loss of autonomy. This summary enables a basic selection of one or more CATs during the design of a SaS. The main part of this report then provides their full details along references of their usage in related work. Hence, the pattern catalog also contributes to current and future research activities with a systematic overview of control options. Researcher may use this overview to identify research gaps and to classify their own approaches.

Our general vision is a comprehensive multi-paradigm control approach for SaS. It will combine higher level, central control with highly reliable, distributed controlled SaS entities. In particular, the central control mechanism will select the appropriate CAT for a certain situation as needed. The presented pattern catalog provides the foundation for this selection. Further, the details for each CAT will support their realization as part of a central control unit.

Acknowledgments

This work is partially supported by the DevOpt-project, funded by the German Ministry of Research and Education (BMBF) under grant 01IS18076A. Any opinions expressed herein are solely by the authors and not of the BMBF.

Bibliography

- [AGCC21] Iván Alfonso, Kelly Garcés, Harold Castro, and Jordi Cabot. Self-adaptive architectures in IoT systems: A systematic literature review. *Journal of Internet Services and Applications*, 12(1):1–28, 2021.
- [ARS17] Paolo Arcaini, Elvinia Riccobene, and Patrizia Scandurra. Formal design and verification of self-adaptive systems with decentralized control. *ACM Transactions on Autonomous and Adaptive Systems*, 11(4):1–35, 2017.
- [AS15] Yousef Abuseta and Khaled Swesi. Design patterns for self adaptive systems engineering. *International Journal of Software Engineering and Applications*, 6(4):11–28, 2015.
- [BDMSG⁺09] Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. *Engineering Self-Adaptive Systems through Feedback Loops*, pages 48–70. Springer, Berlin, Heidelberg, 2009.
- [Bor13] Etienne Borde. *Software Engineering for Adaptive Embedded Systems*, pages 159–190. John Wiley & Sons, Ltd, 2013.
- [Boy76] John Raymond Boyd. *Destruction and creation*, 1976.
- [Cop92] James Coplien. *Advanced C++ Programming Styles and Idioms*. Addison-Wesley, 1992.
- [Dev] DevOpt - DevOps for self-optimizing emergent systems. Online: <http://www.devopt-projekt.de/>.
- [DKJ18] Ali Dorri, Salil S. Kanhere, and Raja Jurdak. Multi-agent systems: A survey. *IEEE Access*, 6:28573–28593, 2018.
- [dLGM⁺13] Rogério de Lemos, Holger Giese, Hausi A. Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M. Villegas, Thomas Vogel, Danny Weyns, Luciano Baresi, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Ron Desmarais, Schahram Dustdar, Gregor Engels, Kurt Geihs, Karl M. Göschka, Alessandra Gorla, Vincenzo Grassi, Paola Inverardi, Gabor Karsai, Jeff Kramer, Antónia Lopes, Jeff Magee, Sam Malek, Serge Mankovskii, Raffaella Mirandola, John Mylopoulos, Oscar Nierstrasz, Mauro Pezzè, Christian Prehofer, Wilhelm Schäfer, Rick Schlichting, Dennis B. Smith,

- João Pedro Sousa, Ladan Tahvildari, Kenny Wong, and Jochen Wuttk. *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap*, pages 1–32. Springer, Berlin, Heidelberg, 2013.
- [Fis06] David Fisher. An emergent perspective on interoperation in systems of systems. Technical Report CMU/SEI-2006-TR-003, Carnegie-Mellon University/Software Engineering Institute, 2006.
- [GCGC07] Marie-Pierre Gleizes, Valérie Camps, Jean-Pierre Georgé, and Davy Capera. Engineering systems which generate emergent functionalities. In *International Workshop on Engineering Environment-Mediated Multi-Agent Systems*, pages 58–75, Berlin, Heidelberg, 2007. Springer.
- [GCGL15] Dieter Gawlick, Eric S. Chan, Adel Ghoneimy, and Zhen Hua Liu. Mastering situation awareness: The next big challenge? *ACM Special Interest Group on Management of Data Record*, 44(3):19–24, 2015.
- [GCS03] David Garlan, Shang-Wen Cheng, and Bradley Schmerl. *Increasing System Dependability through Architecture-Based Self-Repair*, pages 61–89. Springer, Berlin, Heidelberg, 2003.
- [GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
- [HTHJ09] Robrecht Haesevoets, Eddy Truyen, Tom Holvoet, and Wouter Joosen. Weaving the fabric of the control loop through aspects. In *1st International Workshop on Self-Organizing Architectures*, pages 38–65, Berlin, Heidelberg, 2009. Springer.
- [HW11] Christopher D. Hollander and Annie S. Wu. The current state of normative agent-based systems. *Journal of Artificial Societies and Social Simulation*, 14(2):1–24, 2011.
- [IBM05] IBM Corporation. An architectural blueprint for autonomic computing. Third edition. Technical report, IBM Corporation, 2005.
- [IW15] Didac Gil De La Iglesia and Danny Weyns. MAPE-K formal templates to rigorously design behaviors for self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems*, 10(3):1–31, 2015.
- [JB15] Matthew Johnson and Daniel Brown. Evolving and controlling perimeter, rendezvous, and foraging behaviors in a computation-free robot swarm. In *9th EAI International Conference on Bio-Inspired Information and Communications Technologies*, page 311–314, 2015.
- [KBB⁺16] Hermann Kopetz, Andrea Bondavalli, Francesco Brancati, Bernhard Frömel, Oliver Höftberger, and Sorin Iacob. Emergence in cyber-physical systems-of-systems (CPSoSs). In Andrea Bondavalli, Sara Bouchenak, and Hermann Kopertz, editors, *Cyber-Physical Systems of Systems*, pages 73–96. Springer, Cham, 2016.

- [KC03] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.
- [KGS22] Christian Kröher, Lea Gerling, and Klaus Schmid. Combining distributed and central control for self-adaptive systems of systems. In *1st DISCOLI Workshop on DIStributed COLlective Intelligence*, 2022. Accepted.
- [KM90] Jeff Kramer and Jeff Magee. The evolving philosophers problem: Dynamic change management. *IEEE Transactions on Software Engineering*, 16(11):1293–1306, 1990.
- [KM07] Jeff Kramer and Jeff Magee. Self-managed systems: An architectural challenge. In *2007 Future of Software Engineering*, page 259–268. IEEE, 2007.
- [KSPS21] Christian Kröher, Klaus Schmid, Simon Paasche, and Christian Sauer. Combining central control with collective adaptive systems. In *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion*, pages 56–61. IEEE, 2021.
- [KTK⁺16] Jan Kantert, Sven Tomforde, Melanie Kauder, Richard Scharrer, Sarah Edenhofer, Jörg Hähner, and Christian Müller-Schloer. Controlling negative emergent behavior by graph analysis at runtime. *ACM Transactions on Autonomous and Adaptive Systems*, 11(2):1–34, 2016.
- [KTPR20] Christian Krupitzer, Timur Temizer, Thomas Prantl, and Claudia Raibulet. An overview of design patterns for self-adaptive systems in the context of the internet of things. *IEEE Access*, 8:187384–187399, 2020.
- [KVSG⁺05] Hadeli Karuna, Paul Valckenaers, Bart Saint-Germain, Paul Verstraete, Constantin Bala Zamfirescu, and Hendrik Van Brussel. Emergent forecasting using a stigmergy approach in manufacturing coordination and control. In *International Workshop on Engineering Self-Organising Applications*, pages 210–226, Berlin, Heidelberg, 2005. Springer.
- [Mar96] Robert C. Martin. Design patterns for dealing with dual inheritance hierarchies in C++. In *2nd USENIX Conference on Object-Oriented Technologies*, volume 2, pages 1–11. USENIX Association, 1996.
- [MEHdH13] Frank D. Macías-Escrivá, Rodolfo Haber, Raul del Toro, and Vicente Hernandez. Self-adaptive systems: A survey of current approaches, research challenges and applications. *Expert Systems with Applications*, 40(18):7267–7279, 2013.
- [Min03] Naftaly H. Minsky. On conditions for self-healing in distributed software systems. In *2003 Autonomic Computing Workshop*, pages 86–92. IEEE, 2003.

- [MLESA⁺06] Florian Mösch, Marek Litza, Adam El Sayed Auf, Erik Maehle, Karl E. Großpietsch, and Werner Brockmann. ORCA - towards an organic robotic control architecture. In *Self-Organizing Systems, 1st International Workshop, IWSOS 2006, and 3rd International Workshop on New Trends in Network Architectures and Services, EuroNGI 2006*, pages 251–253, Berlin, Heidelberg, 2006. Springer.
- [MMW⁺17] Angelika Musil, Juergen Musil, Danny Weyns, Tomas Bures, Henry Muccini, and Mohammad Sharaf. *Patterns for Self-Adaptation in Cyber-Physical Systems*, pages 331–368. Springer, Cham, 2017.
- [MPS08] Hausi Müller, Mauro Pezzè, and Mary Shaw. Visibility of control in adaptive systems. In *2nd International Workshop on Ultra-Large-Scale Software-Intensive Systems*, page 23–26, New York, NY, USA, 2008. ACM.
- [PCHW12a] Tharindu Patikirikorala, Alan W. Colman, Jun Han, and Liuping Wang. A systematic survey on the design of self-adaptive software systems using control engineering approaches. In *7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 33–42. IEEE, 2012.
- [PCHW12b] Tharindu Patikirikorala, Alan W. Colman, Jun Han, and Liuping Wang. Technical report: A systematic survey on the design of self-adaptive software systems using control engineering approaches. Technical report, Swinburne University of Technology, 2012.
- [PM95] David Lorge Parnas and Jan Madey. Functional documents for computer systems. *Science of Computer Programming*, 25(1):41–61, 1995.
- [PSA11] Jeremy Pitt, Julia Schaumeier, and Alexander Artikis. The axiomatisation of socio-economic principles for self-organising systems. In *5th International Conference on Self-Adaptive and Self-Organizing Systems*, pages 138–147. IEEE, 2011.
- [QWG21] Federico Quin, Danny Weyns, and Omid Gheibi. Decentralized self-adaptive systems: A mapping study. In *16th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 18–29. IEEE, 2021.
- [RC10] Andres J. Ramirez and Betty H. C. Cheng. Design patterns for developing dynamically adaptive systems. In *ICSE 2010 Workshop on Software Engineering for Adaptive and Self-Managing Systems*, page 49–58, New York, NY, USA, 2010. ACM.
- [RMB⁺06] Urban Richter, Moez Mnif, Jürgen Branke, Christian Müller-Schloer, and Hartmut Schmeck. Towards a generic observer/controller architecture for organic computing. In *INFORMATIK 2006 – Informatik für Menschen*, volume 1, pages 112–119. Gesellschaft für Informatik eV, 2006.

- [SCH⁺02] Kurt Schelfthout, Tim Coninx, Alexander Helleboogh, Tom Holvoet, Elke Steegmans, and Danny Weyns. Agent implementation patterns. In *OOPSLA 2002 Workshop on Agent-Oriented Methodologies*, pages 119–130. ACM, 2002.
- [SKK⁺14] Mouna Ben Said, Yessine Hadj Kacem, Mickaël Kerboeuf, Nader Ben Amor, and Mohamed Abid. Design patterns for self-adaptive RTE systems specification. *International Journal of Reconfigurable Computing*, 2014, 2014. Article ID 536362.
- [SMSc⁺10] Hartmut Schmeck, Christian Müller-Schloer, Emre Çakar, Moez Mnif, and Urban Richter. Adaptivity and self-organization in organic computing systems. *ACM Transactions on Autonomous and Adaptive Systems*, 5(3), 2010.
- [ST92] Yoav Shoham and Moshe Tennenholtz. On the synthesis of useful social laws for artificial agent societies. In *10th National Conference on Artificial Intelligence*, page 276–281. AAAI Press, 1992.
- [ST09] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems*, 4(2), 2009.
- [VDPV97] H. Van Dyke Parunak and R. S. VanderBok. Managing emergent behavior in distributed control systems. Technical report, Industrial Technology Institute, Ann Arbor, United States, 1997.
- [WA13] Danny Weyns and Jesper Andersson. On the challenges of self-adaptation in systems of systems. In *1st International Workshop on Software Engineering for Systems-of-Systems*, page 47–51, New York, NY, USA, 2013. ACM.
- [WJ95] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.
- [WSG⁺13] Danny Weyns, Bradley Schmerl, Vincenzo Grassi, Sam Malek, Raffaella Mirandola, Christian Prehofer, Jochen Wuttke, Jesper Andersson, Holger Giese, and Karl M. Göschka. *On Patterns for Decentralized Control in Self-Adaptive Systems*, pages 76–107. Springer, Berlin, Heidelberg, 2013.