Lecture

Organic Computing II

Summer term 2019

Chapter 3: Quantitative Organic Computing

Lecturer: Anthony Stein, M.Sc.

Slides have been created in cooperation with Prof. Dr. Sven Tomforde (Uni Kassel),
Prof. Dr. Müller-Schloer (Uni Hannover) and Prof. Dr. Jörg Hähner (Uni Augsburg)

Motivation

ORGANIC COMPUTING

Organic Computing means:

- Using concepts such as self-organisation in technical systems.
- As a result:
  - Traditional design-time decisions are moved to runtime.
  - From the engineer to the systems themselves.
  - Goal: Higher robustness, reduced complexity.

Engineering means:

- Quantification of system properties
- “Prove” that an organic system is better that any other.

→ We have to define the basic “ingredients” of OC systems.
→ We have to quantify the required aspects.

ORGANIC COMPUTING

What makes an OC system special?

- Emergence may occur.
- The system is self-managing.
- This self-managing process requires objectives or a utility.
- Systems come with a control mechanism that acts autonomously.
- This control mechanism can be distributed in various ways.
- It recovers the system after disturbances occurred.
- Self-organisation takes place to change the system's structure.
- Self-adaptation takes place to change the system's configuration.

Agenda

ORGANIC COMPUTING

Content

- Motivation
- Autonomy and self-organisation
- Quantification of self-organisation
- The survival cycle of an organic system
- Robustness
- Autonomy
- Conclusion and further readings

Goals

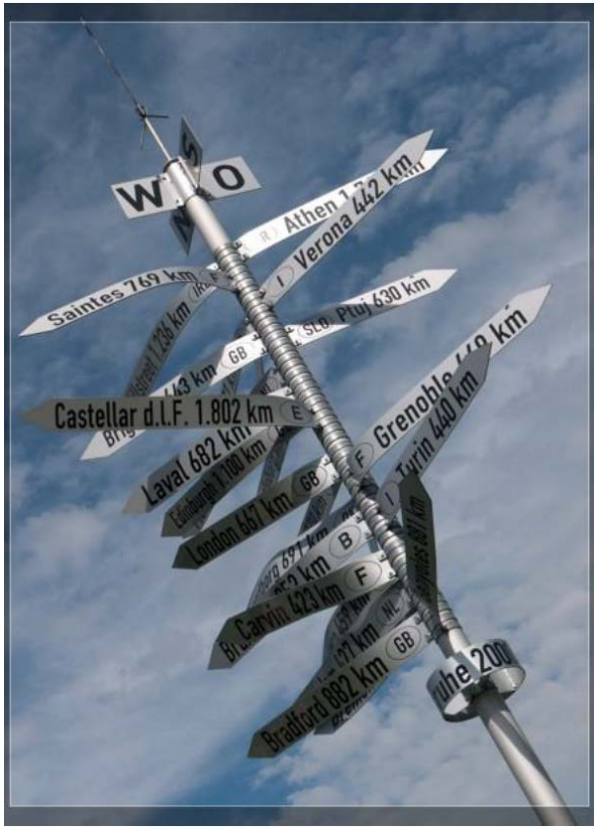
Students should be able to:

- Define what the terms self-organisation, autonomy, adaptability, utility, robustness, disturbance, and variability mean.
- Explain the behaviour of an organic system according to a state space model.
- Describe the OC survival cycle.
- Quantify robustness, self-organisation, and autonomy.

Agenda

ORGANIC COMPUTING

- Motivation
- Autonomy and self-organisation
- Quantification of self-organisation
- The survival cycle of an organic system
- Robustness
- Autonomy
- Conclusion and further readings

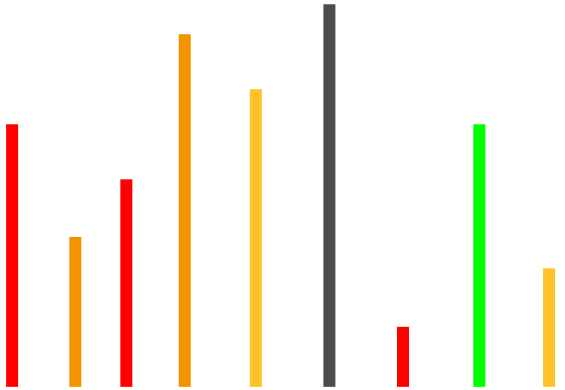


Example: An ordering game

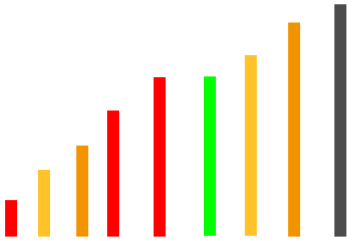
ORGANIC COMPUTING

- Properties of each stick
  - Length
  - Colour
- Ordering objectives
  - increasing length
  - decreasing length
  - colour clusters

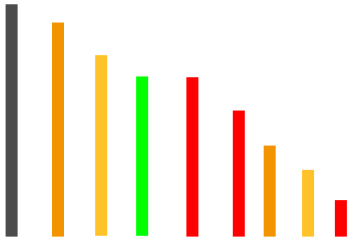
Ordering activity?



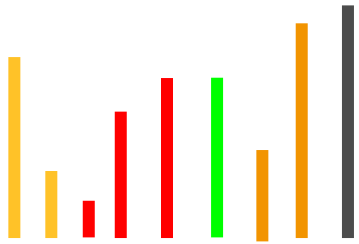
a)



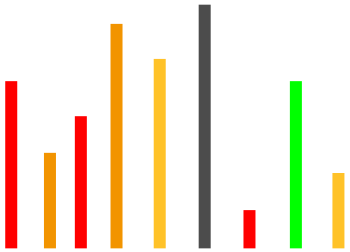
b)



c)

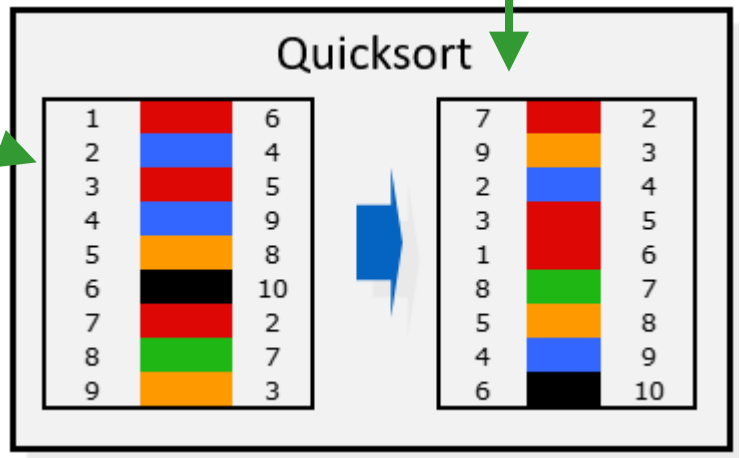


Example: An ordering game (2)

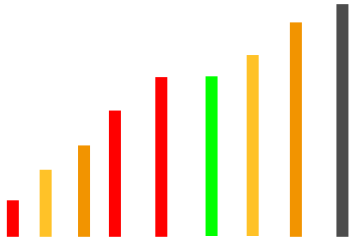


Global objective a := increasing length

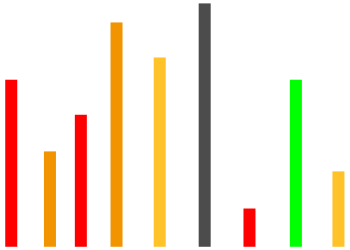
1 Observation
2 Model building
3 Simulation



4 Enactment



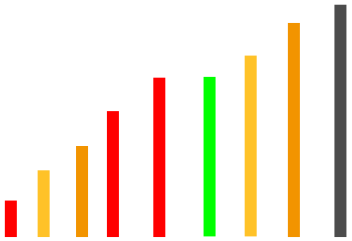
Example: An ordering game (3)



Global objective a := increasing length

1. Local objective: right > myself
2. Local observation: right < myself
3. Local decision:
   if right < myself then Switch places;
   else: nil
4. Local enactment
5. Go to 2

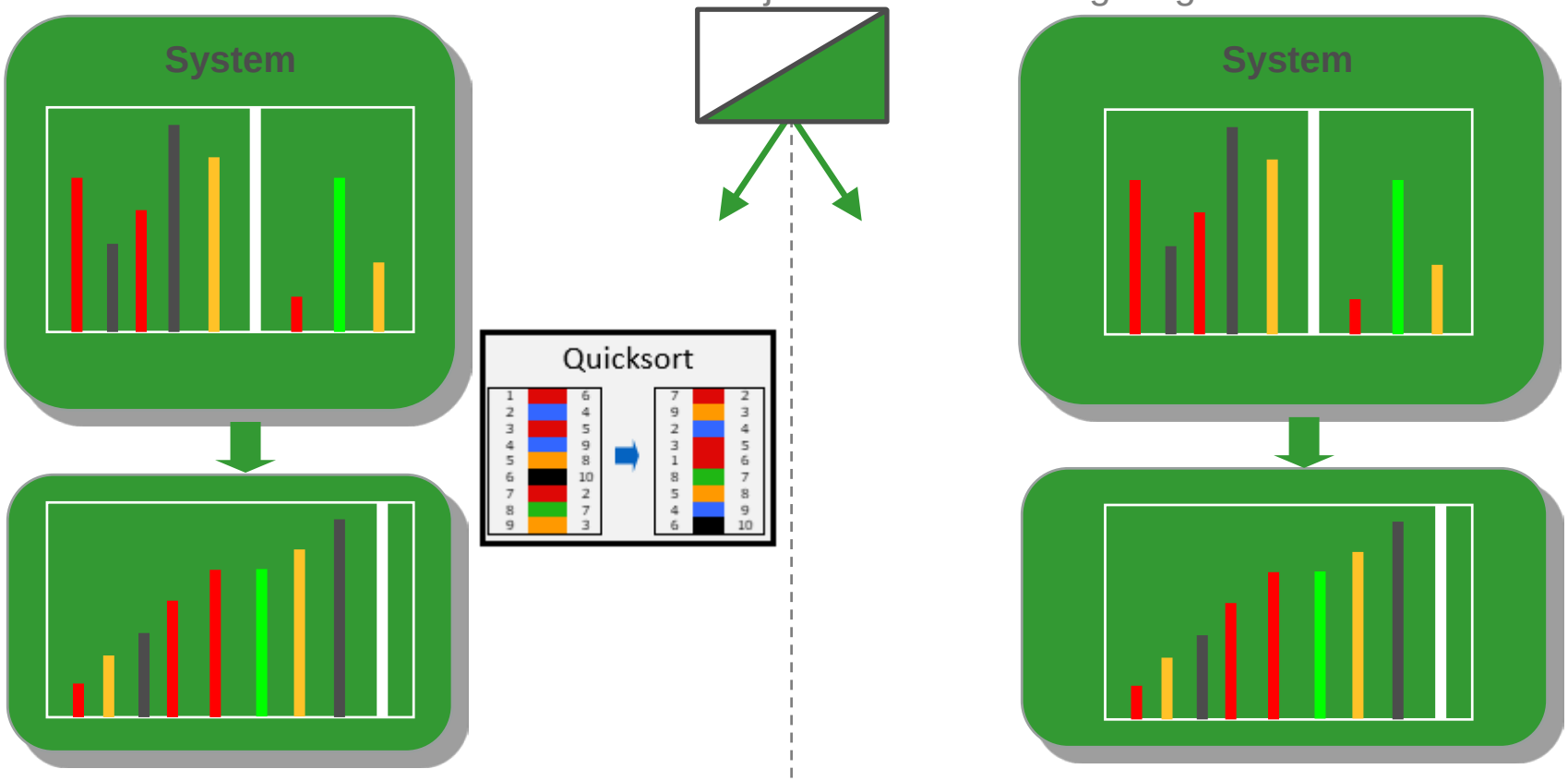
An “intelligent system“ that autonomously acts in the “game“.



Example: An ordering game (4)

Question: Is this self-organisation or autonomy?

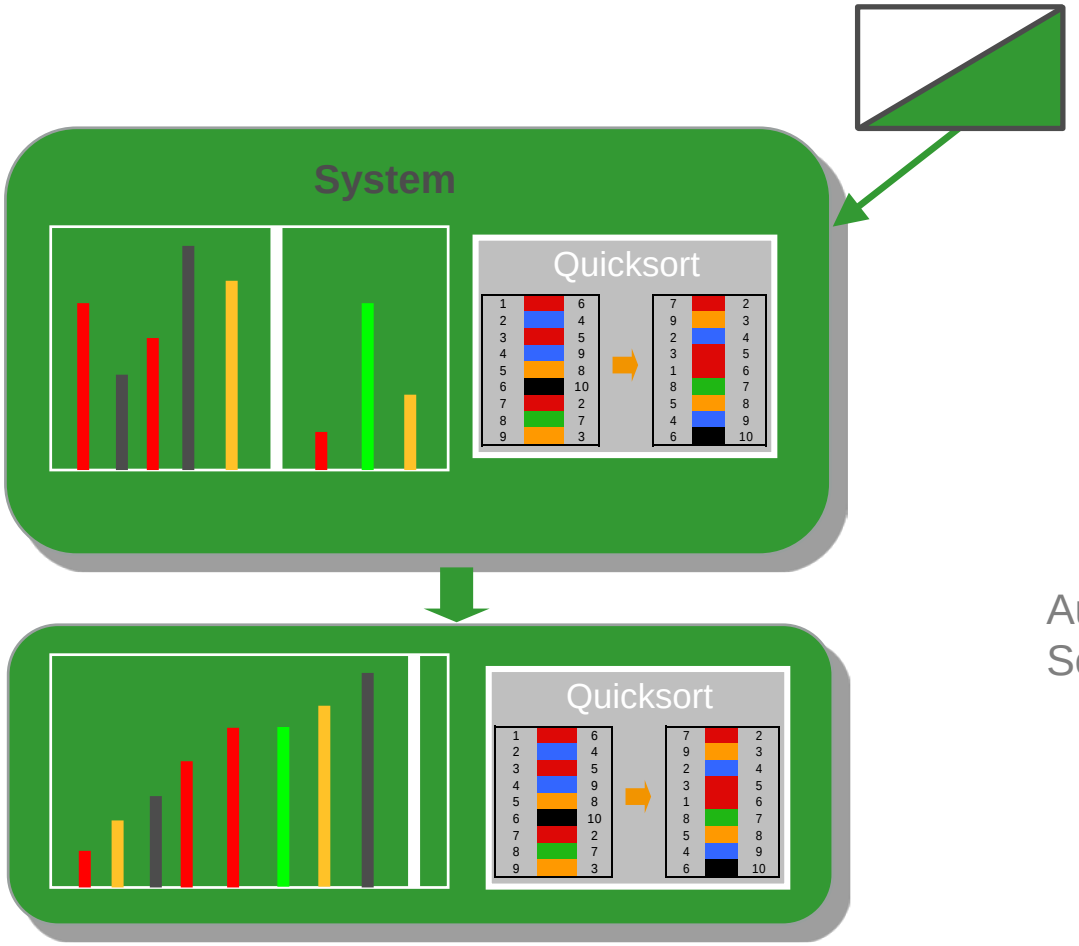
Global objective := increasing length



Autonomy No
SO? No

Autonomy? Yes
SO? Yes

Example: An ordering game (5)



Global objective
:= increasing length

Autonomy? yes
Self-organisation? maybe

ORGANIC COMPUTING

What can we learn from these examples?

- Preliminary definitions
  - Autonomy:
    - A System changes its structure without explicit external control.
    - There must be some kind of internal control mechanism!
  - Self-organisation:
    - The internal control mechanism is distributed (to a certain degree).

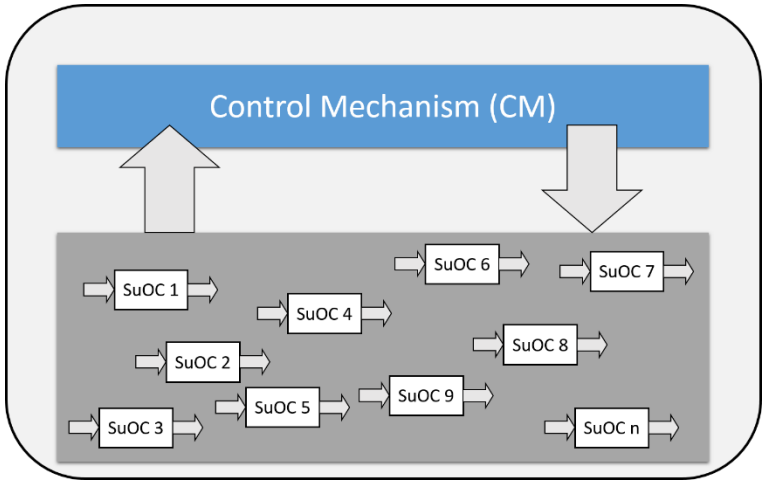
ORGANIC COMPUTING

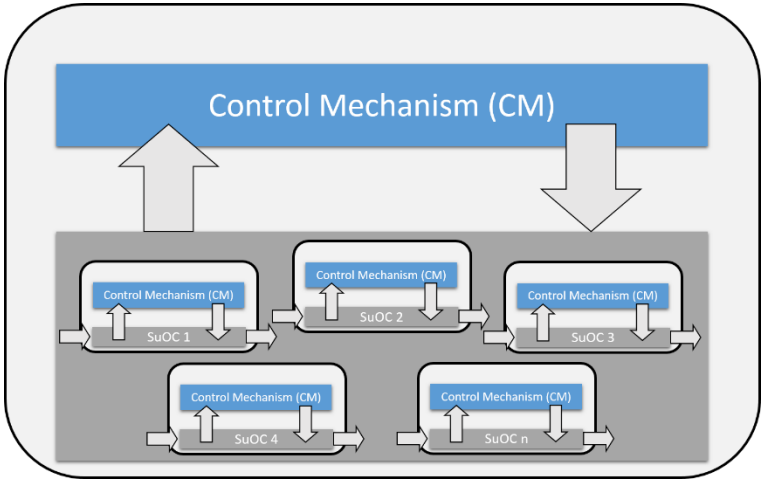
What is self-organisation?

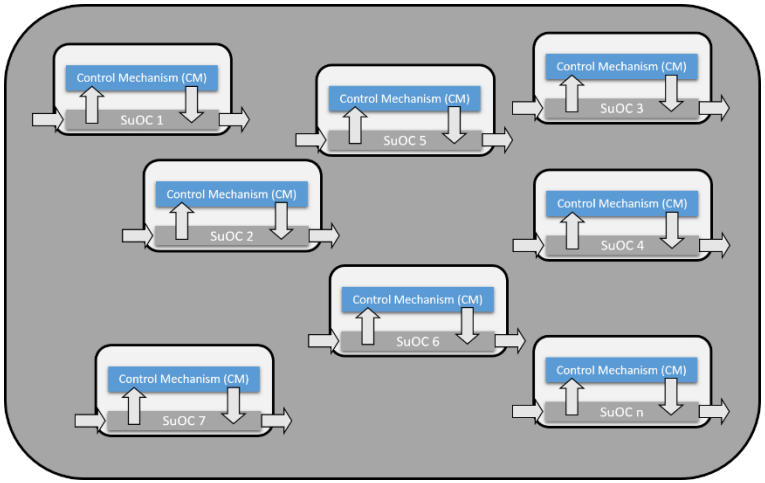
- Intuition suggests that a self-organising system is
  - a multi-element system (consisting of m elements, m > 1)
  - which needs no (little) external control to restructure itself (i.e., it has a high degree of autonomy).
- Common assumption:
  - System consists of one or more productive parts (i.e., the m elements) and an internal control mechanism (CM).
  - CM of a self-organising system is (to a certain degree) distributed over the *m* elements.
- The control mechanism (CM) can be:
  - centralised (one CM)
  - distributed over the m elements (m CMs)
  - distributed over a hierarchy of CMs.

Distribution of the control mechanism

ORGANIC COMPUTING



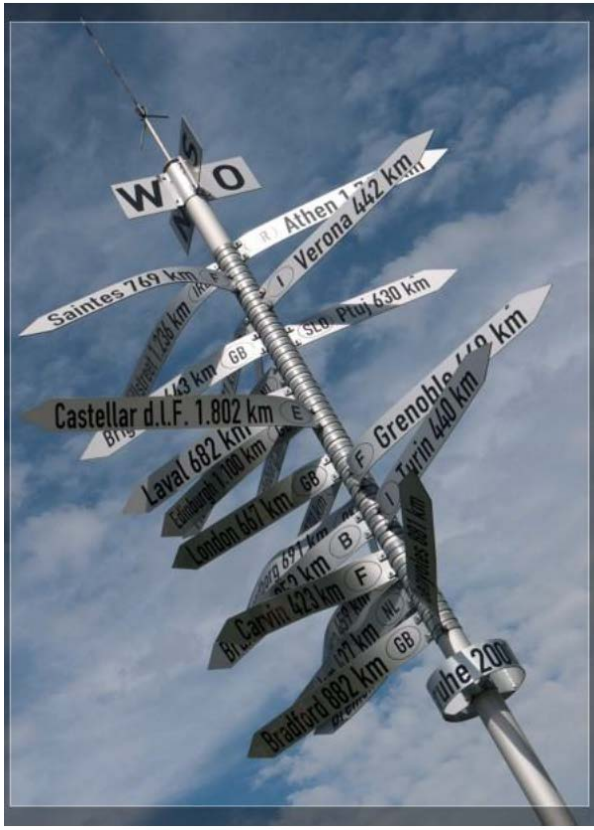




Agenda

ORGANIC COMPUTING

- Motivation
- Autonomy and self-organisation
- Quantification of self-organisation
- The survival cycle of an organic system
- Robustness
- Autonomy
- Conclusion and further readings



ORGANIC COMPUTING

Quantification: static degree of self-organisation

- Let S be an adaptive system consisting of m elements (m > 1) with large degrees of autonomy (α and β) and fully or partially distributed k control mechanisms CM (k ≥ 1) leading to a degree of self-organisation of (k : m).
- S is called strongly self-organised, if k = m, i.e. the degree of self-organisation is (m : m).
- S is called self-organised, if k > 1, i.e. it has a medium degree of self-organisation (k : m).
- S is called weakly self-organised, if k = 1, i.e. there is a central control mechanism and the degree of self-organisation is (1 : m).
  → Assumption: all CM are internal!

Static degree of self-organisation (2)

ORGANIC COMPUTING

Assumption:

- Exclude pathological organisational forms, e.g. the concentration of all CMs on a single controller, hidden CMs, etc.

Distinguish between following terms:

- Systems can be modified at runtime in terms of (i) structure and/or (ii) parameter values.
- Self-configuration relates to a (re-) parameterisation of a system.
- Self-organisation relates to change of the structure of a system (i.e. of components and their links).
- Self-management comprises self-configuration, self-organisation, and possibly further self-* mechanisms.

ORGANIC COMPUTING

Static degree of self-organisation

- Assumes full access to the design concept.
- Needed to identify and count the CM structures.
- A quantification of self-organisation from an external point-of-view, i.e. without interfering with the system's logic and design, needs a different approach.
- In the following, self-organisation is considered as a process at runtime instead of as a static value.

Self-organisation as a process

- Self-organisation: system's ability to modify its structure autonomously and in order to optimise a certain utility function.
- System is expected to make use of its self-organisation capability in response to certain events rather than as a permanent change.

ORGANIC COMPUTING

When is self-organisation observed at runtime?

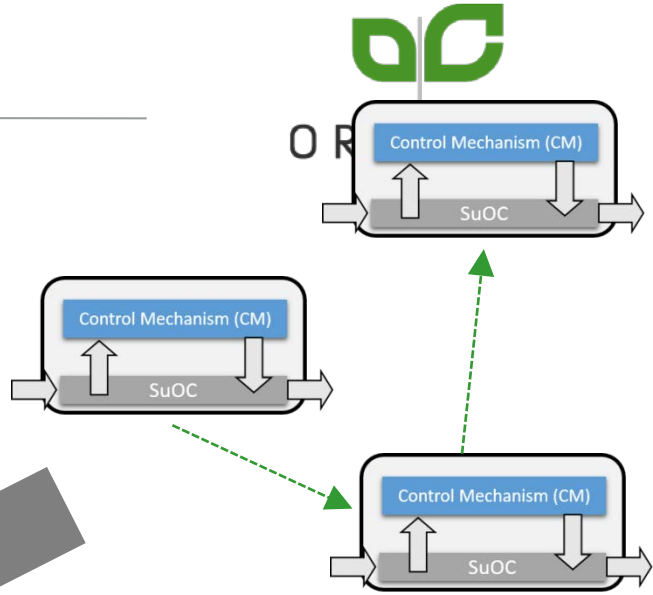
- When the system increases its degree of organisation and builds up a structure (i.e. at system start),
- When a disturbance happens and the system recovers to its previous degree of organisation.

Approach

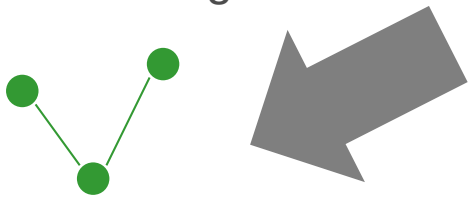
- Internal values are not accessible.
- Observe each (distributed) component of the entire system.
- Relationships between distributed components of the entire system define the structure of the system.
- Relationships are established and altered using communication.

→ Observe the communication and estimate relationships out of this behaviour.

Self-organisation as a process (2)



- Build a graph G = (V, E)
  - Each component/entity is modelled as a node.
  - Each relationship is modelled as an edge.



- From this graph, we can derive two aspects of self-organisation:
  1. The changes between two consecutive points in time that signalise that the structure in the system has been altered. This information is used to quantify to which degree self-organisation processes occurred in the system.
  2. The difference between the possible relationships and the actually utilised relationships over time. This information signalises to which degree the system makes use of its organisation potential.

ORGANIC COMPUTING

Approach: build a graph at both time steps (i.e. t_0 and t_1)

- Assumption: all communication is observable from the external and this communication is purpose-oriented.
- Communication is encoded with origin and destination.
- Messages are distinguishable at a semantic level: Those related to structure information are turned into an edge of the graph.
- Message model: unicast (or multicast) messages.
  → Broadcasts for bootstrapping reasons only (neglected here).

$$\Delta(G_1, G_2) = \frac{|\{e_{ij} : e_{ij} \in E_1 \oplus e_{ij} \in E_2\}|}{0.5 * (|V_1| + |V_2|)}$$

- Value of Δ : quantifies self-organisation that occurred in the system in the considered time frame.

E_k : set of edges of the k-th graph (i.e. G_k)
V_k : set of nodes of the k-th graph (i.e. G_k)
Each edge e_{ij} : directed connection from node i to j

Quantifying the potential for self-organisation

ORGANIC COMPUTING

- Isolated information of Δ just highlights that something happened.
- Good indicator when observing just one system to decide whether self-organisation appears or not.
- Goal: compare different systems
  - Put information of how much self-organisation has been observed into relation to the possible decision potential.
  - Quantify to which degree the system made use of its ability to self-organise.
- Required: estimate the decision potential (corresponds to the possible communication partners). Two approaches are possible:
  1. information is available by design or
     → designer specifies possible communication partners for each node.
  2. it has to be approximated at runtime.
     → method needed.

Approximation of self-organisation potential

ORGANIC COMPUTING

Approach:

- Continuously observe all structure-related communication of the node under consideration and analyse each message.
- Maintain a list of interaction partners I_j (either as origin or as destination of the observed messages) for each node j.
- Every time, a new interaction partner is observed, add it to the list.
- Pessimistic alternative, a worst case estimation is possible by assuming that each node is able to interact with each other node, resulting in a fully connected graph.

Please note: approach only results in meaningful information after observing the system for a certain period of time. Comparing the actual relationships to those ever observed in the system has only impact after a certain amount of change has occurred.

ORGANIC COMPUTING

Approach (ctd.):

- At a certain point t_{quant} consider set of interaction partners I_j.
- Build directed graph: edges connecting node j and its partners.
- Result of this process is reference graph G_r.
- Instead of the previous approach, where two distinct points in time are represented as graphs in terms of snapshots of conditions, we now consider the entire process.
- Compare all observed changes over time (i.e. the number of modified edges) to the reference graph to come up with a (dynamic) degree of self-organisation until a certain time t_{quant}.

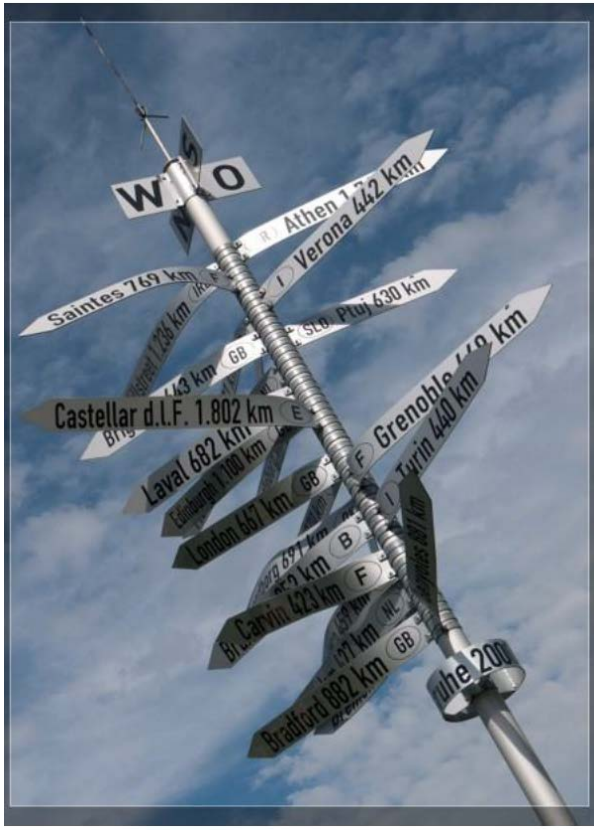
$$SO(t) = \frac{|E_t|}{|E_r|}$$

Set E_t contains all edges that have been changed from system start to time t, and E_r contains all edges of the reference graph. Please note that by definition $|E_r| > |E_t|$.

Agenda

ORGANIC COMPUTING

- Motivation
- Autonomy and self-organisation
- Quantification of self-organisation
- The survival cycle of an organic system
- Robustness
- Autonomy
- Conclusion and further readings



ORGANIC COMPUTING

Motivation

- Systems face the challenge to survive in an ever-changing world.
- Behaviour can be modelled as survival cycle.
- Model is used for formalisations and metrics afterwards.

An initial model

- State machine
- External events change the state of the system in a pre-programmed way.

Example: state machine with 2 states.

- Automaton encodes a predefined sequence of steps to be executed depending on the received input information (the events).
- There is no way to express preferences or ambiguities, though, since all transitions are deterministic and all states are equivalent.

ORGANIC COMPUTING

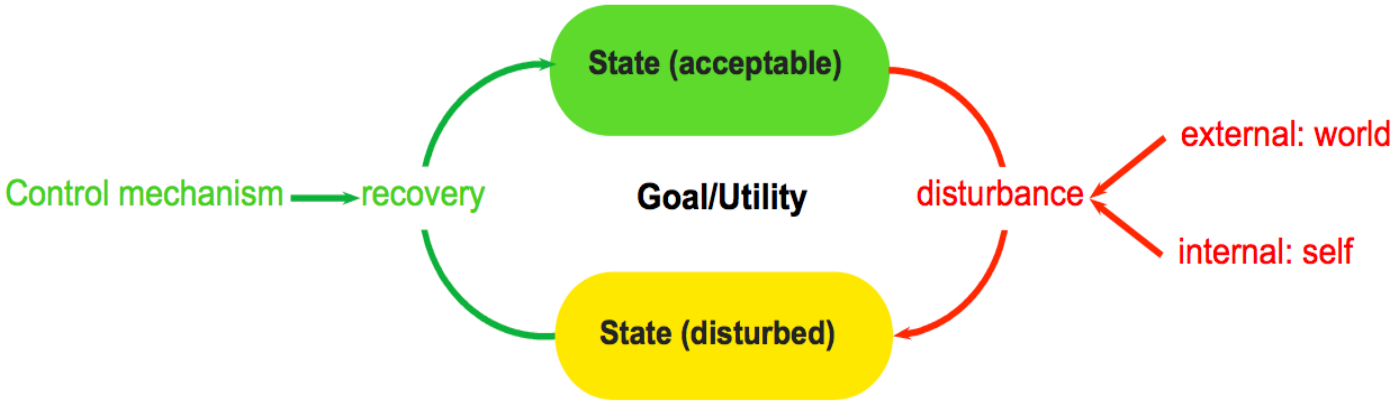
An OC approach

- Adopt the method of a state description, add preferences!
- Perspective of an agent:
  - has the goal to survive.
  - there are certain system states more desirable than others.
- Agent is at any given time in a state **z**.
- Moved to a different state by an event. Origin can be:
  - internal (e.g. a malfunction of the agent itself) or
  - external, i.e. a change in the environment.
- A state so far is neutral and does not indicate a value (i.e. good or bad).
- Needed: a state's utility in relation to a given goal.

Survival cycle (3)

ORGANIC COMPUTING

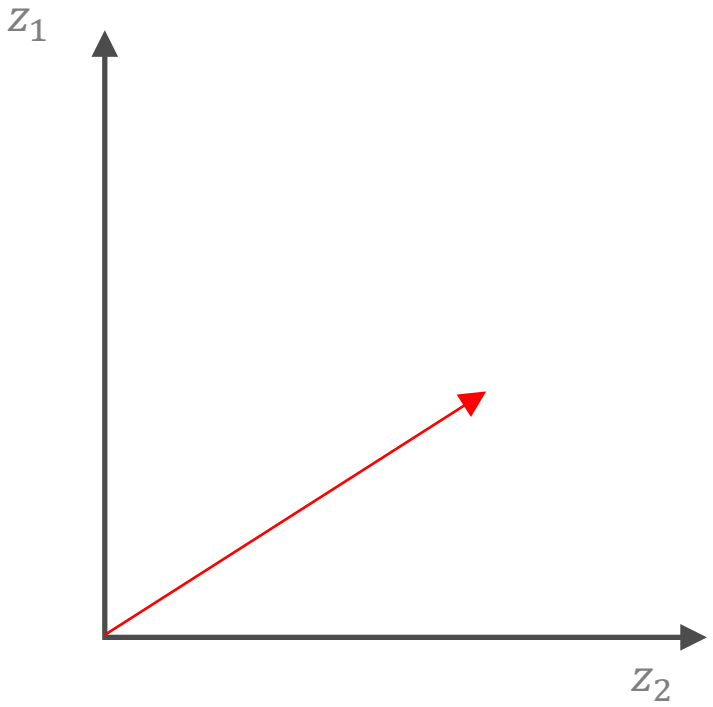
- Goals distinguish good states from bad ones (same holds for influences):
  - In relation to a given goal, a state is acceptable or disturbed (i.e. not acceptable).
  - An OC system always tries to stay in an acceptable state.
  - A disturbance causes a deviation from the acceptable state, the control mechanism will initiate recovery actions to guide the system back into an acceptable state.



ORGANIC COMPUTING

The system state

- Given by a vector **z**.
- With single attributes as components.
- Any useful metric to characterise the system state possible.
- But:
  We are interested only in observable metrics (by system itself).



ORGANIC COMPUTING

- Example:
  - State of a robot with a GPS localisation system might be defined as its x and y coordinates.
  - Robot is able to determine these coordinates itself: x and y are self-observables and part of the state vector.
  - Without a GPS: coordinates are known only from the point of view of an external observer.
    → Not part of the state vector.
- Further examples of state vectors are:
  - Ordering game: z = (length of stick 1, colour of stick 1, length of stick 2, colour of stick 2, …)
  - Traffic controller: z = (traffic flow in cars/hour for all turnings of the intersection)
  - Router: z = (length of message queues, link status)

State (3)

ORGANIC COMPUTING

- Observables might originate from the “world”.
  - I.e. perceived via the external sensory equipment of the agent.
  - Or from its internal sensors (indicating e.g. the battery level, the functional status of a motor, or the length of a task queue).
- In a stricter sense, all observables are internal since even perceptions stemming from the external world are first transferred (possibly under some distortions) into the agent’s internal world model before they can be used for control purposes.
- This is equivalent to the “beliefs” in the BDI agent model.
- At any given time t, the system S is in a state **z**(t).
- If there are n observable attributes used to describe the state of S, **z**(t) is a vector in an n-dimensional state space Z^n.

ORGANIC COMPUTING

- State vector of a system S is a neutral and (as far as possible) objective statement about the knowledge (or beliefs) of S.
- It becomes a statement about values (good or bad or something in between) by assigning values to the state.
- Assignment is achieved by a utility function η.
- The function η maps the system state into a set of real numbers **u**:

$$\mathbf{u} = \eta(\mathbf{z});\ u_i \in R$$

- **u** is used to represent the objectives to be achieved by the control mechanism of S.
- Objectives can be a maximisation or a minimisation.
- **u** can consist of different utilities u1, u2 … resulting from different state attributes and their combinations.
- Utilities can even contradict each other (multi-criterial optimisation).

ORGANIC COMPUTING

Examples for utility assignments:

- Ordering game (with increasing order): Let **u** denote the number of local mis-orderings (i.e. the number of locations i where the height of stick i + 1 is lower than the height of stick i); then reduce **u** to zero!
- Distributed workers having to work on a number of tasks: Let **u** be the aggregated differences in the number of tasks per worker; make **u** as small as possible.
- Vehicle passing a network (from A to B): Let t_{travel} be the travel time from A to B and t_{opt} the optimal realistic travel time, then define the utility as **u** = $t_{travel} - t_{opt}$ and minimise **u**.

→ An additional conflicting objective could be to minimise the fuel consumption of the vehicle.

Utility (3)

ORGANIC COMPUTING

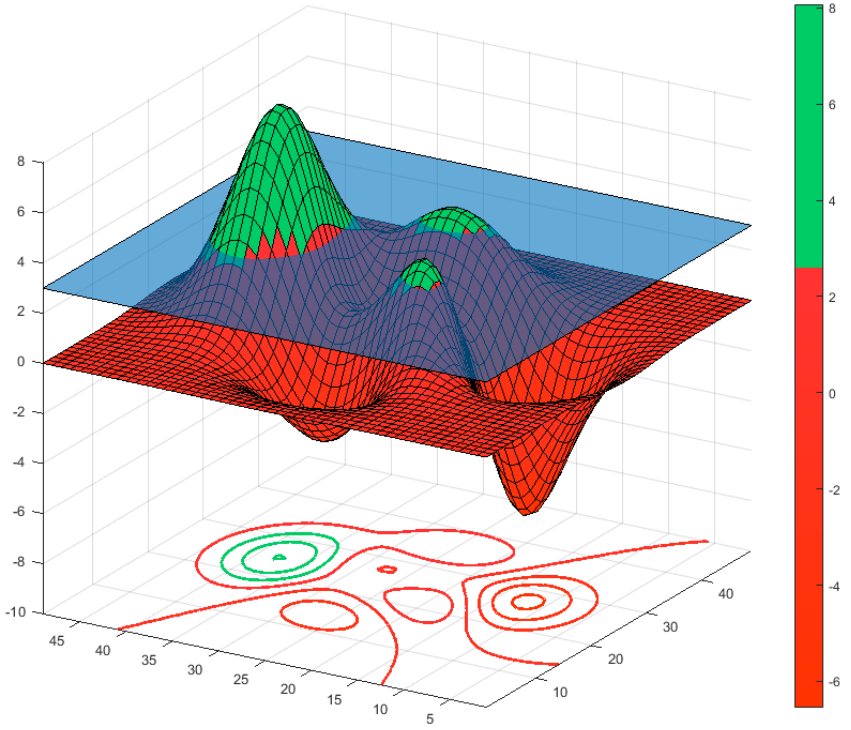
- Utilities can be used to visualise the problem faced by the agents.
- If each state is mapped onto a utility value, we obtain for an n-dimensional state space an n+1-dimensional fitness landscape.
- The agents navigate in this landscape and try to assume a state with a high utility (corresponding to a high fitness).
- In case of a 2-dimensional state space, the utility or fitness landscape is a 3-dimensional surface (utility = fitness).

Utility (4)

ORGANIC COMPUTING

Example:

- 3-dimensional fitness landscape for a 2-dimensional state space.
- Fitness threshold (e.g. an acceptance threshold) is a plane (blue) cutting the fitness “mountains” horizontally.
- All states with a utility higher than the threshold are acceptable (green).

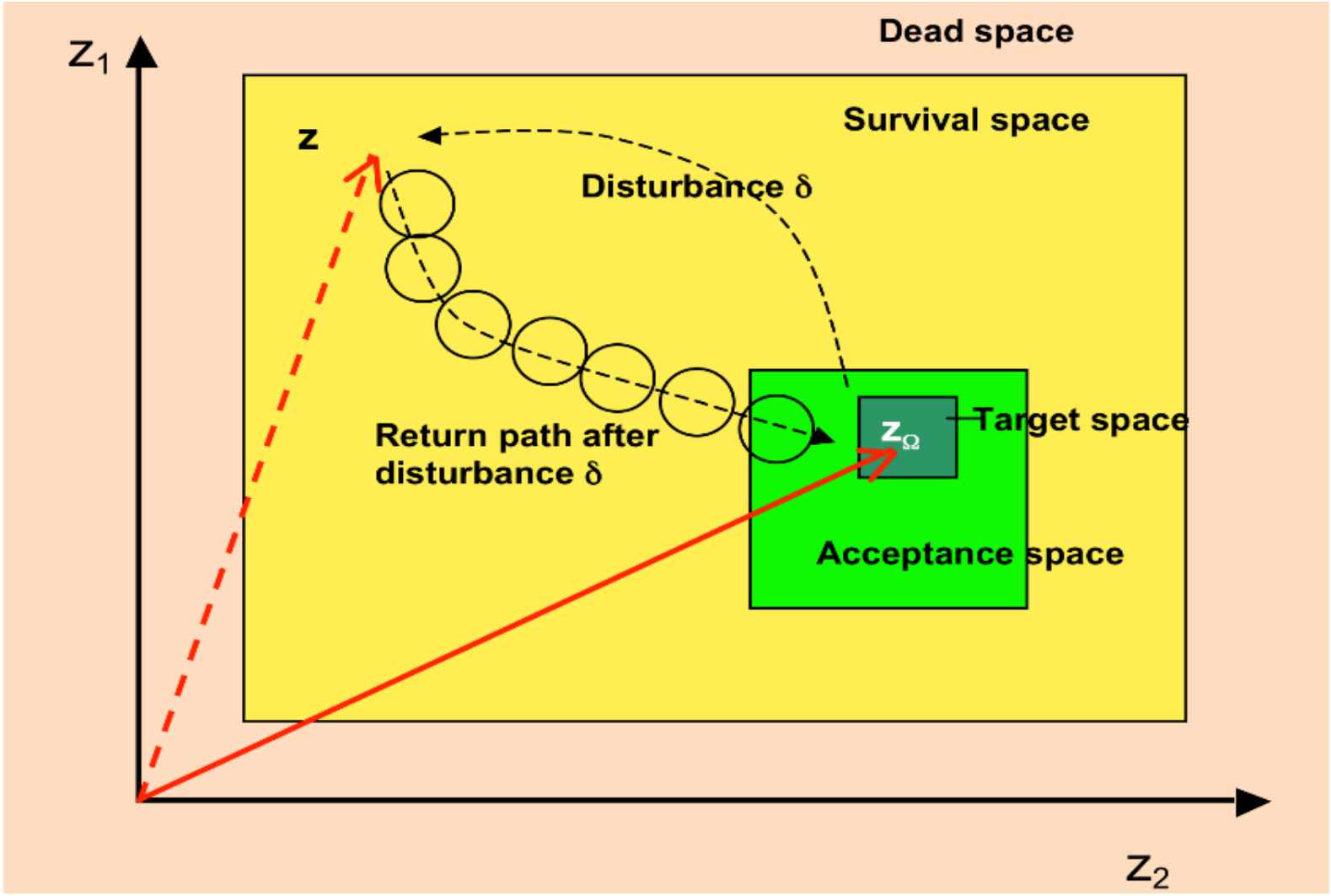


ORGANIC COMPUTING

From state and utility to state spaces

- State vector of a system S is a neutral and (as far as possible) objective statement about the knowledge (or beliefs) of S.
- Utilities are used by control mechanism to compare the current state with an ideal system state.
- If the actual utility is lower than the desired one, the control mechanism has to trigger actions to reduce the difference, i.e. to optimise the system.
- Needed: Thresholds categorising behaviour.
- Goal:
  - Classify states according to utility!
  - Provide different state spaces that require different actions from the control mechanism.

State spaces (2)



State spaces (3)

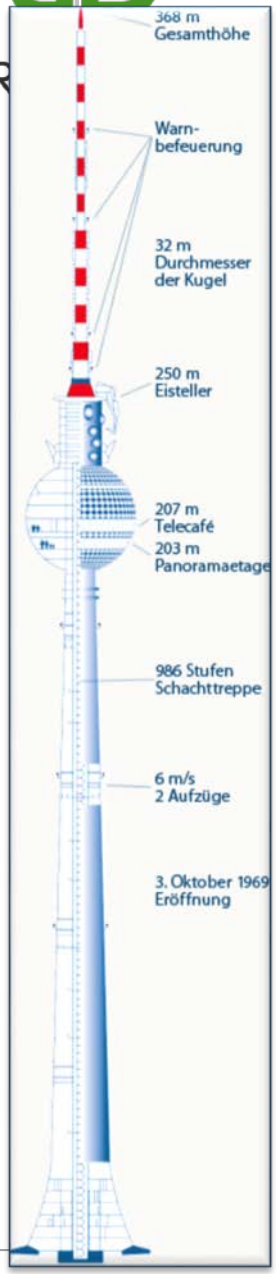
ORGANIC COMPUTING

Distinguish state spaces:

- Target Space: All states with a utility u ≥ u_{target} belong to the target space Z_{Ω} (might also be just *one* globally optimal state).
- Acceptance Space: We denote those system states as acceptable whose utility is greater than or equal to u_{acc}. The set of all acceptable system states is called acceptance space $Z_{acceptable}$.
- Survival Space: An OC system will always try to return to an acceptable state. Such a recovery is possible only from a certain subset of states, the Survival space.
- Dead Space: Any disturbance δ moving **z** outside the survival space will be lethal for S: Such states belong to the Dead space.

Example: Broadcast Tower

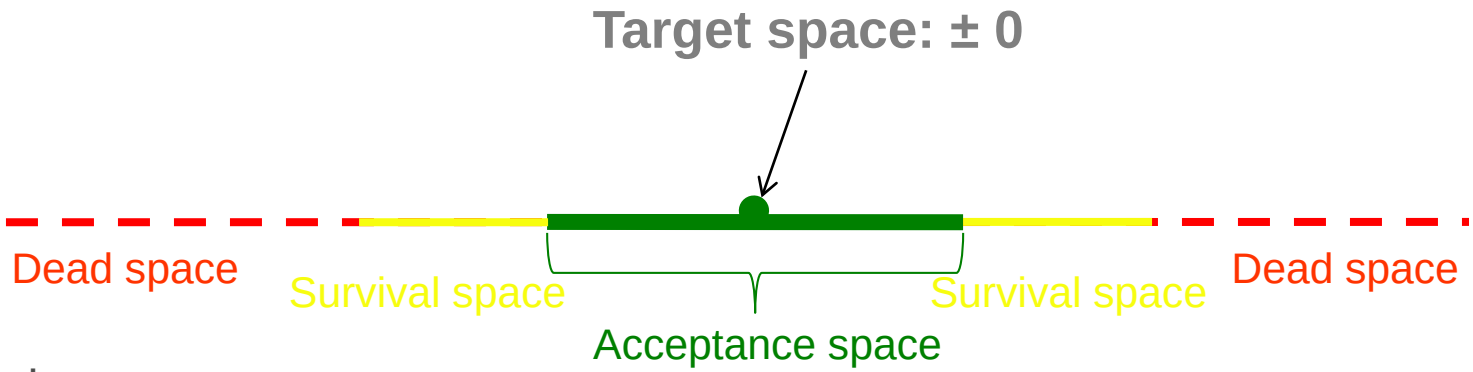
- Tower moves continuously (deflection)
  - Without human perception
  - On top with a radius of about 60cm
  - At the tower cafe with $r = 15cm$
  - The frequency of changing the deflection is about 7 to 10 seconds
- Goal (target): deflection = 0!
- Acceptance: deflection= ±60 cm (still open to the public)
- Survival: deflection >60cm and <120cm (closed to the public)
- Damage*: deflection> ±120 cm



Example: Broadcast Tower (2)

ORGANIC COMPUTING

- Formalisation:



- Dead space:

ORGANIC COMPUTING

Disturbances

- The system might be disturbed by environmental influences or disturbances δ.
- A disturbance δ changes the state z(t) into the disturbed state $z_\delta(t) = \delta(z(t))$.
- As a result: utility changes to $u_\delta = \eta(z_\delta(t))$.
- If $u_\delta < u_{acc}$: the system is outside the acceptance space.
- Examples for such a disturbance:
  - Sudden relocation of a stick (in the ordering game).
  - Failure of some component (e.g., in the travel scenario, a vehicle could get a flat tire or a road might be blocked by an accident).

Disturbance (2)

ORGANIC COMPUTING

Notion of disturbance

- Also includes changes in the evaluation criteria.
- Example: a change from an ascending to a descending ordering objective or from sorting with respect to height to grouping with respect to colour.
- Corresponds to a relocation of target and acceptance space and not a state change of the system!
- For control mechanism: Same concept.
  → It must become active until **z** is again within $Z_{\text{acceptable}}$.
- We refer to a shift in the utility (i.e. a change of the goals given by the user) as **flexibility**.

An OC system is *flexible* if its control mechanism guides the system back to (at least) an acceptable state after a change of the utility occurred (i.e. the state-utility mapping results in modified values); within a predefined time period.

Recovery

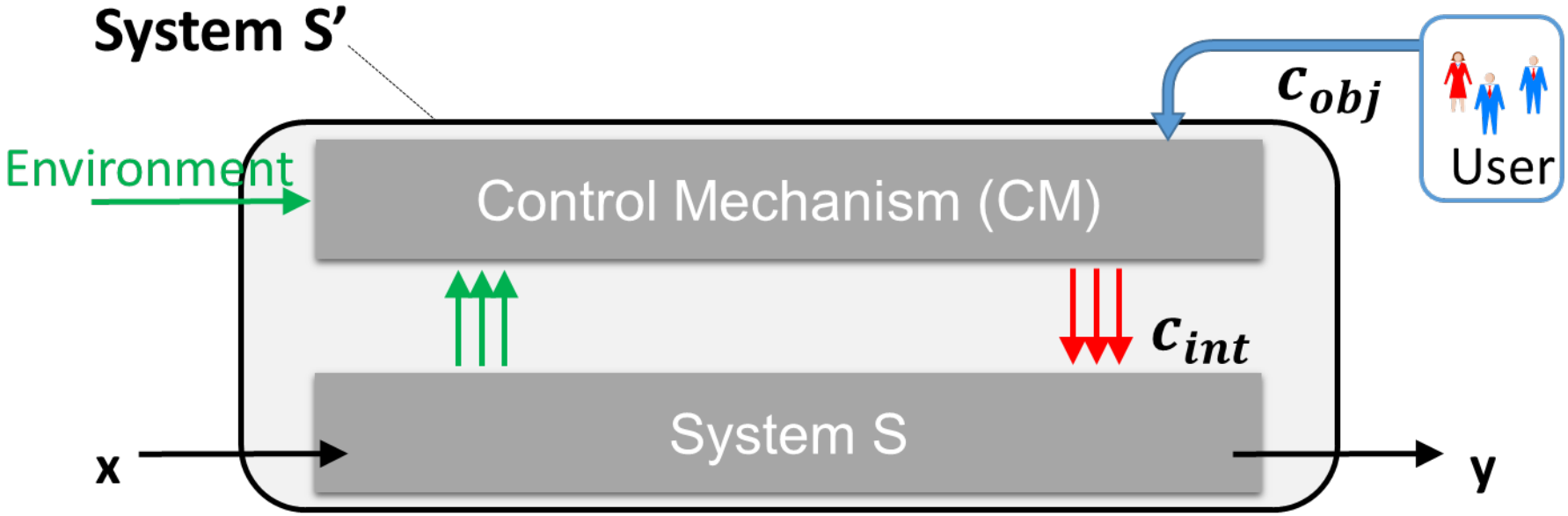
ORGANIC COMPUTING

- System state varies over time.
- Consequently: utility differs as well.
- Control mechanism is responsible to guide the system back to the acceptance space after a drop in utility (i.e. from survival space).
- Usually the utility drop occurs very fast after the disturbance while the recovery takes much more time.
- Reasons:
  - Recovery process being typically very noisy and interrupted by setbacks.
  - If the disturbance is still in force while the CM works against it.
- In state space:
  - Recovery of a system is observed as movement of the state vector back into acceptance/target space.
  - Symbolised by the circles.

ORGANIC COMPUTING

Control Mechanism (CM)

- Recovery is not magic and comes for free.
- Recovery is done by built-in CM in organic systems.
- CM influences behaviour of system S by changing some of its attributes.
- System S is called “System under Observation and Control” (SuOC).
- $\mathbf{CS_{int}}$ constitutes the *configuration space* of the SuOC.
- SuOC is the productive system, which does the “actual” work.
- It receives input signals **x** and transforms them into output signals **y**.
- CM initiates control actions (i.e. it issues control signals $\mathbf{c_{int}}$ to S).
- S together with the CM constitute a system S’, which again could be controlled by a higher-level CM via the control signals $\mathbf{c_{obj}}$.



- Important: CM does not change any environmental parameter directly. Even the system state can be influenced only indirectly by setting c_{int}.