Organic Computing: Zusammenfassung

Alexander Osiik
alexander.osiik@student.uni-luebeck.de
Organic Computing (WS 2019/2020)
Institute of Computer Engineering, University of Lübeck

6. Februar 2020

1 Einführung

Computersysteme wrden immer komplexer. Die Netzwerke von kleinen, eingebetteten Systemen werden immer größer und komplexer. Es stellt sich die Frage, wie man unter solchen Bedingungen die Komplexität von

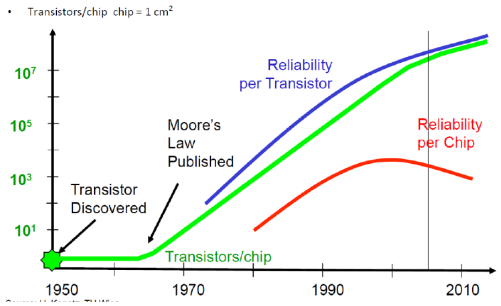
- Design
- Implementierung
- Management
- Adaptation etc.

betrachtet und verbessert.

In Organic Computing möchte man deswegen selbst **organisierte Systeme** entwickeln, die automatisch höherlevelige **Ziele** erfüllen und sich an die **Wünsche des Nutzers** anpassen. Dabei orientiert man sich oft an **lebenden Organismen**.

1.1 Motivation

Man möchte bei komplexen Systemen herausfinden, wie sie **funktionieren**, wie man sie **designed** und wie man sie **überwacht**.
Moore'sches Postulat



Bei komplexen Systemen wie autonomen Fahrzeugen stellen sich viele Fragen.

- Wie **überwacht** man die Komplexität eines solchen Systems?
- Wie stellt man sicher, dass es keine **ungewollte Interaktion** zwischen zwei Systemen gibt?
- **WER** kontrolliert das Auto?
- Wer wird bei Fehlern zur **Rechenschaft** gezogen?

Man möchte sich die Entwicklung komplexer Systeme dort abgucken, wo es schon erfolgreich war (zB. Gehirn).

Komplexe Systeme entwickeln sich meistens durch **natürliche Selektion / Evolution**, oder durch tiefgründige **Forschung**, wo mehr und mehr komplexe selbst-organisierte Systeme gefunden werden können.

1.2 Emergenz

Unter Emergenz versteht man die spontane **Entwicklung von Systemcharakteristiken**, die sich durch Interaktion mehrerer Entitäten entwickeln. Diese emergenten Eigenschaften können nicht bei einer **isolierten** Entität erklärt werden.

- Interaktion zwischen mehreren Entitäten
- **keine** zentrale Kontrollinstanz
- resultierende Patterns sind nicht vorher explizit programmiert

Die Herausforderungen liegen darin, das emergente Verhalten **integrativ** zu verstehen (micro→macro) oder das Verhalten individueller Elemente herauszufinden (macro→micro).
Beispiele: Shockwave Traffic Jam, V-Formation Vögel

1.3 Traditionelle Perspektive

"Deterministic Centralized **(DC)** Mindset"

- **D**: Relevanz von Randomisierung und Rauschen nicht berücksichtigt
- **C**: Pattern wird durch Koordinator generiert

Eine falsche Annahme ist, dass bei systematischen oder systemweiten Effekte nur ein einziger Aktor betrachtet wird.

1.4 Organic Computing

Bei Organic Computing betrachtet man das Zusammenspiel von (eingebetteten) Systemen mit bestimmten Eigenschaften, wie **viele Entitäten**, **Mobilität**, **Vernetzung** und spontane lokale Interaktionen **(Emergenz)**.

Was wir dafür brauchen sind viele, selbst-organisierte Entitäten, **robuste** Verhalten, flexible Verhalten, **Adaption** an dynamische Umweltfeatures und **Vertrauen** sowie **Kontrolle**.

Kann man ein selbst-organisierenden und adaptierenden Mechanismus erstellen, ohne alle mögliche Zustände zu kennen?

1.4.1 Meta-Design

- Der Ingenieur macht Gesetze, kein Micromanagement
- Fuzzy-Platzierung statt rigide
- Adaptieren und selbst regulieren statt Kontrolle und Redesign

1.5 Verhalten von Maschinen

Das Verhalten von Maschinen ist ein kompliziertes wissenschaftliches Feld, sobald es genauso schwer zu verstehen ist wie ein biologisches System.

1.5.1 Verhalten

Verhalten ist, wie sich ein Lebewesen in bestimmten Situationen oder als Reaktion auf einen bestimmten Stimulus verhält.

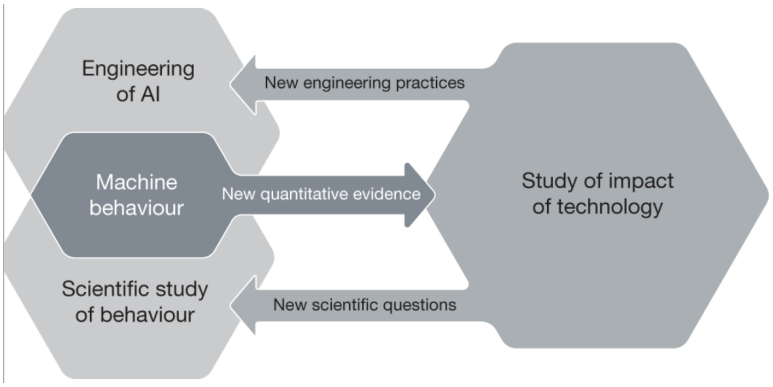
Es gibt sozial geprägtes Verhalten, Verhaltenswissenschaften (Psychologie, Sozialwissenschaften, Neuralwissenschaften)

1.5.2 Tierverhalten (Ethologie)

Wissenschaftliche und objektive Forschung von Tierverhalten unter natürlichen Voraussetzungen. Verhalten wird als adaptives evolutionäres Extra behandelt.

1.6 Komplexität und Opazität von Algorithmen

Architektur und Training können sehr simpel sein, jedoch ist das Resultat sehr komplex, eher eine **black box**. Exakte funktionale Prozesse sind **hart zu interpretieren**, sogar von Wissenschaftlern, die diese Algorithmen erstellen! Sourcecode und Modell sind nur bedingt prädiktive Kraft.



Man soll zunächst **individuelles**, **kollektives** und dann **hybrides Mensch-Maschine** Verhalten studieren!

2 Von Bewegung zu Intelligenz

Organic Computing ist Überwachung von komplexen technischen Systemen, und wie man diese Systeme **selbst-x**(adaptivm optimierend, reparierend etc.) macht. Man will also, dass diese System **intelligent** sind; sich in nicht antizipierten Situationen clever verhalten.
Intelligente Entscheidungsfindung ist das angemessene Verhalten in neuen Situationen, wobei man die Handlungsalternativen basierend auf den eigenen Präferenzen aussucht.

Doch was ist die **Grundlage** der Intelligenz, und wie können wir sie **finden** oder **integrieren**?

2.1 Wichtigkeit der Bewegung

Die Wahrnehmung der eigenen Bewegung und die anderer Entitäten ist Grundlage der Intelligenz. Es ist eine wichtige Kompetenz, die über das Überleben in der Wildnis entscheidet. Was sich bewegt macht einen **lebendigen** Eindruck.

2.1.1 Randomisierte Bewegung

Die **randomisierte** Bewegung, bei der ein System sich in irregulärer Weise bewegt.

$$\sigma_s = \sqrt{\langle s_N^2 \rangle} = \sqrt{N} \quad \text{Bewegung nach Münzwurfmodell}$$

Beim Random Walk wächst die Distanz zum Ursprung proportional mit der Zeit. Die unterliegende Wahrscheinlichkeitsdichte ähnelt der eines Diffusionsprozesses.
Brown'sche Bewegung: Zufällige Bewegung von Partikel in einer Flüssigkeit abhängig von Temperatur Bewegung.
Drunkard's Walk: Man startet eine Bewegung an Position c_0, jeder Schritt c_i hat die **Länge L**. Nach N Schritten ist die Distanz $\sqrt{N}L$ erreicht.
Weitere Variationen: Brown'sche Bewegung mit Drift:

$$\frac{\triangle x}{\triangle t} = \mu + \sigma \frac{\triangle W}{\triangle t}$$

2.1.2 Ziel-orientierte Bewegung

Beispiel: Bakterielle Chemotaxis
Hierbei haben Bakterien zwei Bewegungstypen:
Geradlinige Bewegung (Rotation CCW), und **Taumeln**(Rotation CW). Der **Bias** ist hier die zeitliche Wahrnehmung, ob die Situation sich verbessert (weiterschwimmen) oder nicht (Taumeln). Die Modellierung erfolgt anhande von

- Intrazellulären Modell
- Zellulären Model
- Populationsmodel (interzellulär)

2.2 Interaktion

Es gibt grundsätzlich zwei Formen der Interaktion

- Explizit: Es wird ein beabsichtigtes Signal gesendet
- Implizit: Physischer Kontakt, Kommunikation über die Umgebung

Die Signale können **Visuell**(Glühen, Farben), **Auditiv**(Heulen, Rufen), **Taktil**(Anfassen) und **Chemisch**(Pheromone, olfagorisches) sein

Bewegung nach Münzwurfmodell Kommunikation/Population braucht man denn? Wieviel Neuronen braucht man für Intelligenz? Wieviele Leute für eine gute Party?
Beispiel: Das Drei Körper Problem aus der Physik (Modellierung von 3 frei beweglichen abhängigen Partikeln)

2.2.1 Simple Interaktion

N Partikel, die sich zufällig bewegen, Gruppen detektieren, stoppen, und eine gewisse Zeit warten. Wie aggregiert man alle N Partikel in ein Cluster?
Ansatz: In großen Clustern länger bleiben.

2.2.2 Kollaboration

Bei zwei Agenten hat jeder die Wahl zwischen zwei möglichen Optionen:
"**Cooperate**" oder "**Defect**"

	C	**D**
C	R, R	S, T
D	T, S	P, P

Das **Iterated Prisoner's Dilemma** untersucht das obige Verhalten nach mehreren Runden. Hierbei ist die Frage, welche Strategie am schlausten ist (Keine Perfekte Information vorhanden, da Entscheidungsbaum simultan!):

- Revenge!
- Sofort Heimzahlen, aber 2.Chance geben
- Immer Nett sein
- Random

2.2.3 Evolution

Idee: Wir benutzen künstliche Evolution für IPD um Strategien zu entwickeln und zu schauen, welche Strategie überlebt. Die getesteten Stratefien sind

- Konditionale Kooperator
- betrügerische Überläufer
- Purer Überläufer

Man beobachtet jeweils eine zyklische Dominanz.

2.2.4 Grün-Bart-Effekt

Der Grün-Bart-Effekt ist ein Gedankenexperiment in verwendete Evolutionsbiologie selektiv zu erklären Altruismus unter den Individuen einer Spezies. Zwei sich erkennende Individuen benehmen sich **vetternwirtschaftlich**.

Dazu gehört

- Fakultatives Helfen: Nur GB helfen
- Obligatorisches Helfen: Immer helfen, aber nur GB profitiert
- Fakultatives Schaden: Immer den ¬GB schaden
- Obligatorisches Schaden: Immer schaden, aber GB immun

2.3 Emergente Taxis

Kann ein Kollektiv von simplen Agenten zusammen etwas erreichen, was ein einzelner nicht schafft?
taxis: Bewegung als Antwort auf einen Stimulus

Bei der **Emergent Phototaxis** möchte man

1. Die Kollision mit anderen Robotern vermeiden
2. Die Kohärenz beibehalten
3. Roboter, die Kohärenz erlangen, bewegen sich random.

Fazit: Es funktioniert!

3 Kollektive Intelligenz

In Agentensystemen gibt es wenig bis **gar keine zentralisierte Kommunikation** oder Kontrolle. Es können nicht alle Agenten reden.

Exemplarische Design Probleme, die verteilte Computersysteme beinhalten, sind

- Kontrollsystem für Routing in einem Netzwerk
- Routing in einem Stromnetz
- Verkehrskontrolle

3.1 El Farol Bar

Das El Farol Bar Problem ist ein Problem aus der Spieltheorie. Jeden Donnerstag wollen die Bewohner in die El Farol Bar. Die Bar ist klein, und niemand hat Spaß, wenn es zu voll ist.
Wenn weniger als 60% gehen, haben alle mehr Spaß, als sie zu Hause gehabt hätten. Vice Versa.
Problem: Jeder entscheidet zur selben Zeit!

3.2 Weisheit der Menschenmenge

Idee: Man sammle Meinungen von einer Anzahl an Experten. Diese Meinungen werden verarbeitet (Mittel/Median). Die "**kollektive**" Lösung ist meistens **besser**, als die "**individuelle**".
Francis Galton, Plymouth 1906

Die Effektivität der Weisheit wird oft auf das **Gesetz der Großen Zahlen** reduziert, bei dem der Durchschnitt eines statistischen Experiments den Erwartungswert μ erreicht, je mehr Versuche gemacht werden.
Hier ist es jedoch wichtig, dass die **Unabhängigkeit** jeder Wertung vorliegt, und Experten sich nicht gegenseitig beeinflussen. Das **Mitteln** reduziert zufällige Fehler, aber nicht **systematische Fehler**.

3.3 Kollektive Entscheidungsfindung

3.3.1 Best-Of-n Problem

Man möchte aus n Alternativen die beste finden.

- jede Option hat die Qualität $\rho_i \in (0, 1]$
- Mehrheit $M \geq (1 - \delta)N$ wird gesucht, mit $0 < \delta << 0.5$

Wie lange braucht der Prozess zum konvergieren?
Wie oft wird die Mehrheit erreicht?

3.3.2 Ising Model

Populäres Modell, welches aus dem Ferromagnetismus kommt. Jede Zelle eines Grids hat dabei die Zustände $s = \pm 1$. Nachbarn wollen dabei im gleichen Zustand sein (Energieminimierung). Ein spontaner Wechsel ist abhängig vn der Temperatur möglich.

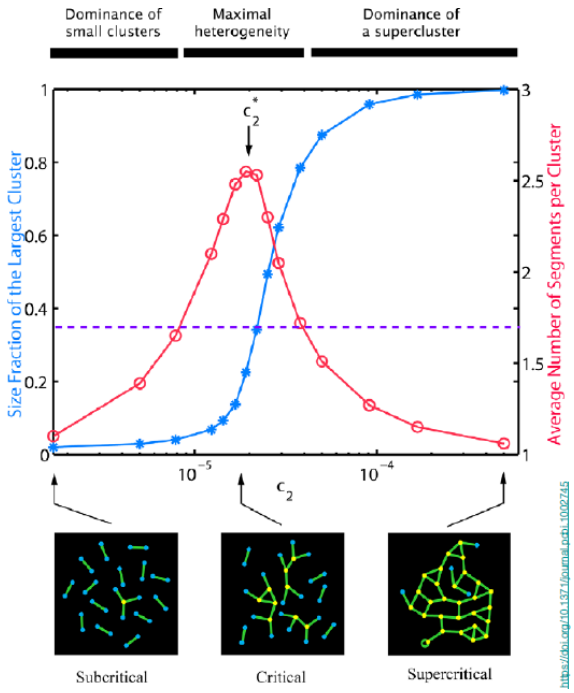
- Magnetisierung $M = \sum_i s_i$
- Energie $E = \sum_{\langle ij \rangle} s_i s_j$

Mit dem **Metropolis Algorithmus** wird pro Zeiteinheit ein Spin geflippt und geguckt, ob die Energie sich reduziert. Die Minimierung wird akzeptiert, wenn:

1. $\triangle E \leq 0$: Exploitation mit WS $= 1$
2. $\triangle E < 0$: Exploration mit Wahrscheinlichkeit $A = \exp(-\beta \triangle E)$, $\beta = \frac{1}{T}$

3.3.3 Perkolation (Versickerung)

Perkolation ist der Wasserfluss durch ein solides Substrat. **Perkolationstheorie** wird als Model für viele Szenarien verwendet, zb: Ausbreitung von **Epidemien**, **Waldbrände**, **Leitungsfähigkeit**...
Dabei beobachtet man ein interessantes, nicht-lineares Verhalten. Es gibt eine **kritische Wahrscheinlichkeit** p^*, wodurch ein kleines Cluster zu einem maximal großem Cluster mutiert.
Beispiel: Ein Feuer würde bis p^* ausgehen, danach jedoch immer durchbrennen.



3.3.4 Individuelle Entscheidungsfindung

Idee: Es gibt **Beweise** für eine der Alternativen, welche verarbeitet und **akkumuliert** werden bis eine **Schwelle** überschritten wird, nach der die **Entscheidung** getroffen wird.

3.4 Geschwindigkeit vs. Genauigkeit

Man kann schnell und ungenau sein, oder langsam und genau. Dies hängt nicht nur von Reaktionszeit ab!

Bei Kollektiven bekommt man meist Informationen durch seine Nachbarn. Beim **Ising-Modell** sind die Nachbarn fix, bei regulären Schwärmen bewegen sich die Nachbarn ständig! Die **lokale Stichprobe ist repräsentativ** für die gesamte Situation.

- Majority Rule (MR): Zähle die Meinungen der Nachbarn, bleib bei der Mehrheit (schneller)
- Voter Model (VM): Nimm einen zufälligen Nachbarn und übernimm seine Meinung (bei großen Schwärmen genauer)

4 Modellierung von OC-Systemen

Start: Man möchte komplexe Systeme überwachen.

- Was ist ein **System**?
- Was ist ein **komplexes System**?
- Wie können wir sie überwachen und **Performance** feststellen?

4.1 Technische Begriffe

4.1.1 System

Ein System ist eine Menge detaillierter Methoden, Prozeduren und Routinen, welche eine bestimmte **Aktivität** ausüben. Es ist eine organisierte **zweckgebundene Struktur**, die aus von einander abhängigen **Entitäten** abhängt. Diese Elemente beeinflussen sich gegenseitig und **erhalten die Existenz des Systems.**

Das **lineare System** ist dabei die simpelste Form eines Systems. Eine Änderung/Störung löst eine Antwort des Systems aus, welche proportional zur Eingabe ist.

$$y = Ax + B$$

Bei **nicht-lineare Systemen** ist die Systemantwort dementsprechend nicht proportional zur Eingabe, aber dennoch **vorhersehbar**.

$$y = \sin x$$

Nicht-lineare chaotische Systeme scheinen unvorhersehbar zu sein, obwohl deren Systemgleichungen deterministisch sind. Ein solches System ist stark von den **Anfangsbedingungen** abhängig.

$$x_{n+1} = rx_n(1 - x_n)$$

4.1.2 Komplex vs Kompliziert

Bei komplizierten Systemen ist die Lösungshierarchie **flach** (Evolution macht komplizierte Strukturen), während sie bei komplexen Systemen **vertikal**(Emergenz macht komplexe Strukturen) ist.

4.1.3 Systemperspektive

Betrachtung aller Verhalten des Systems als ein ganzes im Kontext seiner Umgebung ist die Systemperspektive. Bei der Systemperspektive soll das System bei weiterer Betrachtung nicht weiter reduziert werden.

4.1.4 Komplexität

Eigenschaft eines Systems, **alle** seine Eigenschaften nicht durch **einen** einzigen Formalismus adäquat beschreiben zu können.

4.1.5 Modellierung

Ein Modell ist eine vereinfachte Formalisierung der durch den Modellbauer verstandenen Welt (nicht zwingend realitätsgetreu). Für agenten-basierte Modelle modelliert man **Probleme**, **nicht Systeme**!

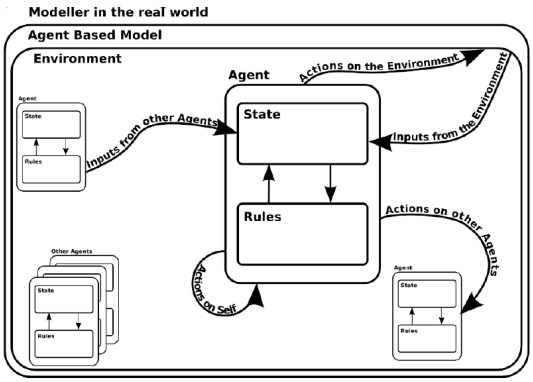
Beim **Top Down** Ansatz wird das System vom ganzen ins kleine modelliert. Voraussetzung dafür ist, dass das Verhalten bekannt ist, welches man reproduzieren möchte. Man braucht also ein gutes **Verständnis vom gesamten System** und wie die Komponenten **interagieren**.

Beim **Bottom Up** Ansatz wird das Verständnis vom System jedoch mehr gestärkt. Man versteht, wie relativ einfache Einheiten miteinander vernetzen, und wie ein **deterministischer Regelsatz** zur Entwicklung **komplexer Verhalten** führt.

Zu den agenten-basierten Ansätzen gehören **KI** (Agent ist autonome Einheit, welche Probleme löst), **Multi-Agenten Systeme** (Verteilte Kontrolle, kollektive Zielverfolgung) und **agenten-basierte Modellierung**.

4.2 Agenten

Ein Agent ist ein **eingekapselter** Computer, welcher in einer Umgebung ausgesetzt ist. Er kann durch **flexible, autonome Aktionen** in dieser Umgebung ein vorgegebenes Ziel verfolgen.



4.2.1 Zustand / State

Ein Zustand ist das Wissen über den Status der Umgebung. Dieses kann **öffentlich** oder **intern** sein. Zu jedem bestimmten Zeitpunkt ist der Zustand s als Element der Zustandsmenge $S = \{s_1, s_2, \dots\}$ (Intern: $I = \{i_1, i_2, \dots\}$) vorgegeben.

4.2.2 Regelnsatz

Neben Zuständen verfügt der Agent über einen Regelsatz, welche die interne Beschreibung der Prozesse darstellt. Dies können zB Transitionsregeln sein. Dabei gibt es die Varianten

- regelbasiert

- abhängig von mehreren konkurrierenden Zielen
- Heuristiken
- Machine Learning

4.2.3 Aktion

Eine Aktion hängt von anderen Agenten, internem Zustand (indirekte Umgebung) und dem Regelsatz ab. Eine Aktion beeinflusst dabei genau diese.

4.2.4 Umgebung

Eine Umgebung ist alles, was kein Agent ist, aber relevant für das Modell ist. Sie gibt **Informationen**, **Struktur** und **Stimuli**. Sie beeinflusst die Agenten, und wird von ihnen beeinflusst.

4.2.5 Zeit

Bei Agentensystemen betrachtet man in der Regel **diskrete Zeit**. Ein Zeitschritt ist dabei alles zwischen zwei Zeitpunkten, in diesem Zeitschritt funktioniert alles **simultan**. Agenten-basierte Systeme laufen **parallel** und **asynchron**, die Auswahl der Erstberechnungen muss also **randomisiert** erfolgen, um **Unabhängigkeit** zu schaffen.

4.2.6 Agententypen

- **Agenten mit Wahrnehmung (Reflex Agenten):**
  Sensordaten sind Wahrnehmung, aus der der Zustand der Umgebung bestimmt wird. Danach erfolgt eine Aktion.
- **Agenten mit internen Zuständen:**
  Agent hat internes Gedächtnis um den Zustand des Systems zu speichern. Der neue Zustand ist vom alten Zustand abhängig und der Wahrnehmung. Abhängig von **neuem** Zustand wird dann die Aktion ausgeführt.

4.3 Multi-Agenten-Systeme

Eine Menge von Agenten die in der gleichen Umgebung koexistieren. Dadurch können Aufgaben auf mehrere Agenten **aufgeteilt** werden, um die **Systemperformance** zu erhöhen, oder es können **komplexe Probleme** gelöst werden, die eine einzelner Agent nicht lösen kann.

Wie koordiniert man die Agenten durch Kommunikation und Interaktion?

Wichtige Eigenschaften sind:

- Die Agenten sind **autonom**
- Ihnen steht nur **lokale** Information zur Verfügung
- Das System ist **dezentralisiert**

- Zur Kollaboration und Koordination müssen Aktionen **geplant**, **mitgeteilt** und **verhandelt** werden.

4.4 Koordinationsmechanismen

Bei der **Korrelation** gibt es nur eine lose/zufällige Verbindung zwischen bestimmten Events. Es herrscht **keine Kausalität**.

Koordination ist eine Eigenschaft der Agentengruppe, die eine Aufgabe löst. Es ist ein **kausaler** Prozess, der aus Kommunikation der Agenten entsteht.

4.4.1 Kommunikationstypen

Die Kommunikation von Agenten kann durch **Kommunikationstopologie**(Zentral, Dezentral) und **Informationsfluss**(Direkt, Indirekt) kategorisiert werden.

- **Topologie:**
  Zentral: kein Informationsfluss zwischen Slaves
  Dezentral: Peer2Peer
- **Informationsfluss:**
  Direkt: Agent zu Agent, ohne Umgebung zu berücksichtigen
  Indirket: Durch die Umgebung (Constraints durch Master)

4.4.2 Typen von Koordinationsmechanismen

- **Markierung-basiert:** (Zeichen, Signal)
  Markierungen können von anderen Agenten observiert werden, andere Agenten können Tags beeinflussen oder hinzufügen (Vertrauen und Sicherheit)
- **Token-basiert:**(alles, was Agenten in einer Gruppe teilen können)
  Für jedes Token gibt es nur wenige Instanzen, Agenten mit Token haben **exklusiven** Zugriff. (Selbstschutz und Rollenverteilung)
- **Markt-basiert:** (Kaufen, Verkaufen)
  Agenten kommunizieren durch Transaktionen (Ressourcen Zuweisung)

Beispiel: Ameisen nutzen Pheromone zur Kommunikation. Pheromone ist die akkumulierte Erfahrung aller Ameisen. Diese "Information" kann **aggregiert** werden oder **verdampfen**. Jeder Agent kann auf die Information zugreifen und Information hinzufügen. **(Dezentralisiert, Indirekt, Markierung-basiert)**

4.4.3 Self X (OC in anderen Ländern)

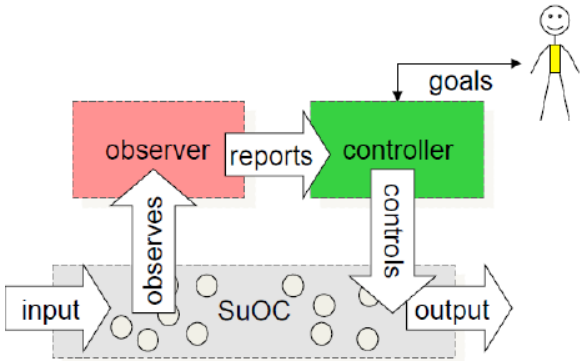
- **Self-Healing:** Das System kann sich von Fehlern erholen
- **Self-Protection:** Das System identifiziert und wehrt sich gegen Angriffe
- **Self-Optimizing:** Das System finden ein optimales Verhalten für ein Szenario
- **Self-Adaptive:** Das System passt sich an Veränderungen der Umwelt an

5 Architekturen

Man möchte das komplexe System, was man beobachtet, auch **beeinflussen**. Man möchte damit auch sicherstellen, dass das System innerhalb der **gegebenen Parameter** operiert.

5.1 Observer-Controller Architektur

5.1.1 System under Observation and Control (SuOC)



Start: Ein einzelnes System, bestehend aus einer Gruppe von autonomen Robotern, welches als komplett bezeichnet werden kann.
Input sind alle Informationen (**Umgebungseffekte**), die ein SuOC zum reagieren braucht, aber nicht beeinflussen kann.
Output sind alle Informationen (**Parameter**) die vom SuOC bestimmt werden und von außen zugänglich sind.

Ein SuOC muss dabei folgende Anforderungen erfüllen:

- **Ziele** erreichen und **Constraints** einhalten
- Verhalten von SuOC muss **beobachtbar** sein
- Performance muss **messbar** sein
- Muss **dynamische Kontrollfaktoren** besitzen

Der **Observer** (Beobachter) muss den aktuellen Systemzustand messen und evaluieren. Aus diesen Informationen muss er den nächsten Zustand vorhersagen können.

Der **Controller** "überwacht" den Prozess. Er **beeinflusst** das System so, dass die **Ziele erreicht** und die **Einschränkungen eingehalten** werden.
Er unterbricht dabei **unerwünschtes emergentes Verhalten** und **rekonfiguriert** das System, um der Entwicklung von unerwünschtem Verhalten entgegenzuwirken.

Möglichkeiten, das SuOC zu beeinflussen, sind:

- Verhalten von Agenten ändern (**Regelsatz**)

- Das **Netzwerk** modifizieren um das Agentenverhalten zu ändern
- Die **Anzahl** an Agenten(typen) erhöhen oder verringern
- Die **Umgebung** modifizieren

5.1.2 Observer Modelle

Der Controller kann dem Observer vorgeben, nach welchem **Observer-Modell** er handeln soll. Das Observer-Modell entscheidet darüber, welche **Features** überwacht werden solle, wie die **Daten analysiert** werden und wie der **zukünftige Zustand** vorhergesagt wird.

5.1.3 Observer

Der Aufbau des Observers ist wie folgt:

1. **Monitor:** Überwachung der ausgewählten **Parameter** zu vorgegebener **Sampling-Rate**. Diese werden in einem **Log-File** festgehalten, welche als **Basis für Vorhersagen** genutzt wird.
2. **Pre-Prozessor:** Bereitet die Daten für das Verarbeiten vor (Filterung und Aggregation)
3. **Data Analyzer:** Generiert den aktuellen Systemzustand (Parameter, entstehende Effekte). Die Methoden der Analyse hängen dabei vom gewählten Observer-Modell ab
4. **Predictor:** Vorhersage des zukünftigen Systemzustands. Erlaubt dem Controller, Entscheidungen nicht nur abhängig von der Vergangenheit zu treffen. Dadurch wird die Reaktionszeit reduziert (Proaktives Verhalten)
5. **Aggregator:** Kombiniert die oben genannten Daten zu "**Situationsparametern**", die dem Controller übergeben werden.

5.1.4 Controller

Der Aufbau des Controllers ist wie folgt:

1. **Mapping:** Implementiert Reaktion A_i auf bestimmte Situationsparameter C_i. Erlaubt auch **erlernte Reaktionen** für unbekannte Situationen
2. **Performance Evaluation:** Evaluation basiert auf Adaption von Selektoren. Abhängig davon wird das **Mapping** aktualisiert. **Online Learning** mit Historie von Aktions- und Situationsparametern
3. **Simulation Model:** Erlaubt das generieren und evaluieren neuer Regeln. Diese werden durch Modifikation anderer Regeln oder Neuschöpfen hinzugefügt, nachdem sie in der **Simulation erfolgreich** waren (**Offline Learning**). Bietet dadurch mehr Systemsicherheit.

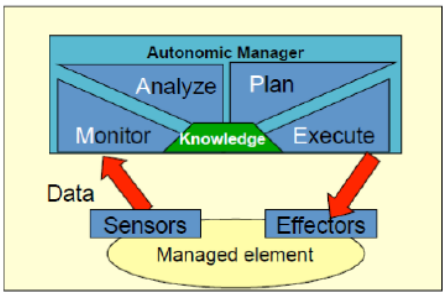
Die Organisation kann dabei **zentral**, **dezentral** oder **multi-level** erfolgen.

5.1.5 Beispiel: Aufzugskontrolle

Observer observiert en Bunching Effekt, Controller sorgt dafür, dass die Aufzüge äquidistant verlaufen.

5.2 Monitor-Analyze-Plan-Execute

Historie: Problem bei IBM. Rechnersysteme werden komplexer und die Administration wird komplizierter. Man benötigt ein Paradigmenwechsel, um weniger Leute einstellen zu müssen und Kunden zufrieden zu machen.



5.2.1 Autonomic Computing

Computer, die sich **selbst verwalten** können, wenn ihnen ein **high-level Ziel** gegeben wird. Ähnlich dem menschlichen Nervensystem sollen auch Computern oder Verteilten Systemen **vitale Funktionen** zur Verfügung stehen.
Siehe auch: **Self-CHOP Prinzipien**

5.2.2 Architektur

- **Autonome Systeme** sind ein Verbund von interagierenden autonomen Elementen
- **Autonome Elemente** besitzen Ressourcen und stellen Services zur Verfügung (für Nutzer oder andere AE). Sie interagieren mit anderen AE um Ziele zu erreichen. Sie haben einen oder mehrere zu verwaltende Elemente.
- Managed Elements (ähnlich wie SuOC) können ihren eigenen Kontrollzyklus haben. Beispiel hierfür wären Hardware- oder Softwareressourcen.

Der Autonome Manager ist für die Automation des IT Management Funktionalität verantwortlich. Er ist in der Regel für vier Aufgaben zuständig:

- **Überwachen** von ME
- **Analyse** von Daten
- **Planen** von Aktionen
- **Ausführen** von Aktionen

5.2.3 Monitor

Automatische **Kollektion**, **Aggregation**, **Korrelation** und **Filterung** von Informationen über die Managed Ressource. Die Daten werden gesammelt, bis ein Symptom entdeckt wird. Die beobachteten Symptome werden zur Analyse weitergereicht.

5.2.4 Analyse

Zustäandig für Einhaltung von Beschränkungen. Observiert und analysiert Symptome, lernt Systemverhalten und sagt zukünftige Zustände voraus.

5.2.5 Plan

Generiert und wählt Prozeduren aus die die Managed Ressource beeinflussen. Generiert einen Plan der von GuideLines abhängt.

5.2.6 Execute

Zeitplanung und Ausführung der geforderten Systemänderungen. Kann mehrere oder einen einzigen **Managed Ressource** beeinflussen.

5.2.7 Knowledge Source

Repositorium für Symptome, Guidelines, Änderungsanfragen etc. Kann von vielen AM aus zugegriffen werden.

5.3 Fazit

- "Wissen" ist über das O/C System verteilt
- MAPE: Sequenz von Operationen, O/C: Architekturkomponenten
- Bei O/C liegt der Fokus auf definierten Zielen
- MAPE hat keine expliziten **Offline Learning** Regeln
- Bei beiden gibt es keine Interaktion zwischen autonomen und organische Elementen

6 Schwarm Intelligenz (ACO)

Wie kann man neue Mapping Regeln erstellen?
Wie können wir die Auswahl an Mapping Regeln verbessern?
$\Longrightarrow$ Anpassung, Optimierung, Lernen

6.1 Von Ameisen zu Algorithmen

Rückblick: Ameisen Kolonie, Verhalten bei Binary Bridge Problem
Erweiterung: Ein Pfad ist doppelt so lang wie der andere. Im Endeffekt ist die Pheromon-Konzentration auf dem kürzeren Pfad höher. Der längere wird trotzdem ab und zu gewählt.

6.1.1 Double Bridge Experiment

2. Erweiterung: Der kurze Pfad wird erst später eingefügt. Die Ameisen vergessen jedoch nicht den langen Pfad, da die Pheromon-Konzentration dort höher ist.
Als Stochastisches Modell: $p_{is}(t)$ als Wahrscheinlichkeit, an Stelle i den kürzeren Pfad zu nehmen.

$$p_{is}(t) = \frac{(\frac{1}{t_s} + \phi_{is}(t))^\alpha}{(\frac{1}{t_s} + \phi_{is}(t))^\alpha + (\frac{1}{t_l} + \phi_{il}(t))^\alpha}$$

Zeit für den kürzeren Pfad ist $t_s = \frac{l_s}{v}$
Pheromon-Konzentration auf kurzem und langem Pfad $\phi_{is}, \phi il$
$\alpha = 2$ experimentell bestimmt

Die Pheromonkonzentration wird bestimmt durch

$$\frac{d\phi_{is}}{dt} = \psi p_{js}(t - t_s) + \psi p_{is}(t) \quad (i = 1, j = 2; i = 2, j = 1)$$

$$\frac{d\phi_{il}}{dt} = \psi p_{jl}(t - rt_s) + \psi p_{il}(t) \quad (i = 1, j = 2; i = 2, j = 1)$$

6.1.2 Double Bridge Experiment als diskretes Model

$$p_{is}(t) = \frac{(+\phi_{is}(t))^\alpha}{(\phi_{is}(t))^\alpha + (t_l + \phi_{il}(t))^\alpha}$$

Update von Pheromonen-Konzentrationen:

$$\phi_{is}(t) = \phi_{is}(t-1) + p_{is}(t-1)\psi_i(t-1) + p_{js}(t-1)\psi_j(t-1)$$

$$\phi_{il}(t) = \phi_{il}(t-1) + p_{il}(t-1)\psi_i(t-1) + p_{jl}(t-r)\psi_j(t-r)$$

$$(i = 1, j = 2; i = 2, j = 1)$$

6.1.3 Kürzeste Pfade in Graphen

Ein Graph $G = (N, A)$ ist gegeben mit Menge der Knoten N und Menge der unidirektionalen Kanten A. Es besteht das Risiko, dass ungewollte Zyklen entstehen, wenn Ameisen immer Pheromone ablegen.
Ein Ansatz ist, dass die Ameisen ihre Pheromone nur auf dem **Rückweg** ablegen (**unnatürlich**).

Noch eine Erweiterung wäre, den Ameisen eine **Erinnerung** an den **bisherigen Pfad** und die entstandenen **Kosten** zu geben. Dadurch werden Zyklen detektiert. Die Menge der abgelegten Pheromone ist proportional zur Qualität der Lösung.

6.1.4 Simple Ant Colony Algorithm

Die Pfadwahl ist **probabilistisch** und hängt von der Pheromon-Konzentration ab. Auf dem Weg zum Ziel werden keine Pheromone abgelegt. Die Sequenz der **Wegpunkte** wird gespeichert. Durch **Backtracking** werden die Pheromone aktualisiert, die **Kosten** für den gefundenen Pfad werden berechnet und die **Evaporation** der Pheromone wird simuliert.

Die Pheromon-Konzentration τ_{ij} von i zu j ist initial $= 1$.
Die Wahrscheinlichkeit, dass die Ameise k einen Pfad $i \to j$ wählt, wird berechnet als

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}^\alpha}{\sum_{l \in N_i^k} \tau_{il}^\alpha}, & \text{if } j \in N_i^k \\ 0, & \text{if } j \notin N_i^k \end{cases}$$

N_i^k sind alle Knoten die von i erreichbar sind.

Zyklen werden eliminiert, indem der Pfad nochmal durchsucht wird und Sequenzen zwischen zwei Zahlen eliminiert.

Update der Pheromon-Konzentration durch ablegen (kann von Länge des Pfades abhängen)

$$\tau_{ij} \leftarrow \tau_{ij} + \triangle\tau^k$$

Evaporation der Pheromone dann durch

$$\tau_{ij} \leftarrow (1 - p)\tau_{ij}$$

6.1.5 Fazit

Update basierend auf der Qualität der Lösung konvergiert zur optimalen Lösung. Für größere Werte α, p konvergiert S-ACO jedoch schneller auf sub-optimalen Lösungen.

6.2 ACO Metaheuristik

Eine Metaheuristik ist ein allgemein anwendbares, generisches, nicht vom Problem abhängendes Schema zur Entwicklung heuristischer Prozesse. Es ist eine

allgemein anwendbare Heuristik, die (iterativ oder konstruktiv) problemspezifische Heuristiken zu vorteilhaften Bereichen des Suchraums führt.

Künstliche Ameisen generieren potentielle Lösungen, indem sie zufällig durch einen verbundenen Graphen $G_C = (C, L)$ laufen.
Die Knoten sind Komponenten, G_C ist der Konstruktionsgraph, L sind die Verbindungen.
Die Beschränkungen Ω sind Verhaltensregeln der Ameisen.
Für jede $c_i \in C$ und eine Verbindung l_{ij} wird eine Pheromon-Konzentration τ_i gesetzt.
Hinzu kommt die Komponente η für jedes c_i und l_{ij}, welche die heuristische Information ist.
Die Ameise k sucht den Graphen nach einer optimalen Lösung $s^* \in S^*$.
Darüber hinaus besitzt er die Memory M^k, die wichtig ist, um Beschränkungen zu erfüllen und die aktuelle Lösung zu evaluieren.
Es hat einen Anfangszustand und Endzustand, wobei der aktuelle Zustand $x_r = \langle x_{r-1}, i \rangle$ ist.

```
procedure ACOMetaheuristic
  ScheduleActivities
    ConstructAntSolutions
    UpdatePheromones
    DaemonActions % optional
  end-ScheduleActivities
 end-procedure
```

6.3 Traveling Salesman Problem

Ziel: Finde den kürzesten Pfad, der alle Städte verbindet (Funktion f). Jede Stadt kann nur ein mal besucht werden (Beschränkung Ω). Eine Ameise wandert von Stadt zu Stadt, bis alle Städte durch sind. Die Entscheidung, in die Stadt i zu gehen, hängt davon ab, ob man schon drin war, oder von der Anzahl der Pheromone.
Formal: Gewichteter Graph $G = (N, A)$, wobei $(i, j) \in A$ ein Gewicht d_{ij} hat.

6.3.1 Ant System Algorithmus

```
procedure ACOMetaheuristic
  set parameters, initialize pheromone trails
  while (termination condition not met) do
        ConstructAntSolutions
        UpdatePheromones
end-procedure
```

Folgendes Vorgehen:

1. Setze alle Konzentrationen auf $\tau_0 = \frac{m}{C^{nn}}$
2. in jedem Schritt bewegt sich die Ameise gemäß **random proportional rule**
$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}]^\alpha \cdot [\eta_{il}]^\beta}$$
Als Heuristik wird der Abstand der Knoten gewählt
$$\eta = \frac{1}{d_{ij}}$$
Hierbei $\alpha = 1$ und $\beta = 2 - 5$ ganz gut
3. Update der Pheromone ist dann Evaporation
$$\tau_{ij} \leftarrow (1 - p)\tau_{ij}$$
und erhöhte Konzentration (je nachdem wie oft der Pfad begangen wurde)
$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \triangle \tau_{ij}^k$$
Wobei $\triangle \tau_{ij}^k = \frac{1}{C^k}$, C^k kosten der Tour

Weitere Optimierungen:

- 2 Exchange
- Mehr Fokus auf Ausnutzen der Erfahrung
- Pheromon-Konzentration für die Ameise mit der besten Tour Updaten.
- Pheromone früher evapoieren

6.4 Anwendungsbeispiele

6.4.1 Vehicle Routing Problem

Es gibt ein Depot mit dem Index 0. Es gibt n Kunden, wobei jeder Kunde i die Nachfrage b_i nach dem selben Produkt hat. Es gibt mehrere Transportvehikel, alle haben Kapazität B. Es gibt verschiedene Kosten zwischen Knoten d_{ij}.

6.4.2 VRP mit Zeitfenster

Hierbei hat jeder Kunde noch ein vorgegebenes Zeitfenster. Dabei sind die Constraints hierarchisch, d.h. man minimiert lieber die Anzahl der Vehikel statt die Reisezeit.

6.4.3 AntNet

Best-Effort-Routing gesucht in IP Netzwerken. Dabei soll das Verhalten adaptiv zum aktuellen Datenverkehr reguliert werden.

Das Netzwerk ist charakterisiert durch

- Bandbreite und Latenz von Links (Kanten)
- Router (Store-And-Forward, Buffer, Routing Lookup)

Der ACO kann hierbei wie folgt angewendet werden:

1. Die Prozedur ist das Aussenden der Ameisen zu einer Destination. Die Ameise sucht Pfad mit minimalen Kosten, basierend auf der Heuristik des Routers und der Memory der Ameise.
2. Die Modellierung erfolgt durch Graph G_C von Routern. Jeder Router erhält eine Pheromone Matrix, die angibt, wie gut es ist von i nach d via j zu gehen.

Und so weiter.

7 Analyse von OC Systemen

Wie kann man ein OC System/ SuOC verstehen?
Von wo kommen die kreativen Ideen?
Kann Wissenschaft automatisiert werden?

7.1 Wissenschaftliche Bemühungen

Herausforderungen bei Analyse von OC Systemen:

- Systeme sind komplex und nicht-linear
- Es herrscht selbst-Organisation und viele Interaktionen
- Es ist schwer, alle Effekte zu antizipieren

$\Longrightarrow$ Kreativität ist gefordert!

7.1.1 Wie entdeckt man?

Wie und woher kommen sie Ideen zu wissenschaftlichen Entdeckungen? Zufall? Gibt es einen Algorithmus dafür?

"Art of Thought"

1. Preparation: Am Problem arbeiten
2. Inkubation: Problem verinnerlichen
3. Andeutung: Spüren, dass eine Lösung auf dem Weg ist
4. Erleuchtung: Kreative Idee wird bewusst
5. Verifizierung: Lösung wird geprüft

7.1.2 Hypothese

Eine Hypothese ist eine gebildete Vermutung in der empirischen Wissenschaft. Sie ist **testbar**(Es gibt evtl. Gegenbeispiele, Plausibilität) oder **falsifizierbar**.

Was kommt zuerst: Hypothese oder Daten? Formuliert man die Hypothese, weil man Daten gesehen hat (**Induktion**)? Sammle ich Daten weil ich eine Hypothese habe (**Deduktion**)?

7.1.3 Norwood Russell Hanson

Norwood Russel Hanson war ein amerikanischer Philosoph. Seine wissenschaftliche Entdeckung war die Argumentation von Daten zu Hypothesen. Er entwickelte das **Hypothesis Catching**

1. Fange eine Hypothese
2. Teste sie

3. Verfeinere oder Verwerfe
4. Fang nochmal an!

Markov Ketten können benutzt werden, um auf micro-level einen **individuellen Agenten** zu beschreiben oder auf macro-level das ganze **SuOC** zu beschreiben.

Selbst-Organisation basiert auf 4 Komponenten:

- positive und negative Rückkopplung
- Fluktuationen (zufällige Ereignisse)
- Mehrere Interaktionen
- Balance zwischen Exploitation und Exploration

Um nun OC Systeme zu analysieren, muss man zunächst **inkrementell** OC Systeme **modellieren**. Man sucht für das vorliegende System und Szenario ein **key feature** heraus, welches die relevanteste Eigenschaft des Systems ist. Danach versuchen wir, **Markov Zustände** $s \in S$ zu finden, die die Klasse der konfigurierten Zustände $C' \subset C$ repräsentiert. S ist dann nach key-feature $s_f \in S_f$ strukturiert, $s \in S \backslash S_f$

Wir möchten also

1. eine Markov Kette erstellen,
2. die Rückkopplung überprüfen
3. verfeinern und dann mit 2. weitermachen, oder mit 1. von vorne starten

7.1.4 Key Feature Transitionen

Eine **bool'sche Funktion** f bestimmt, ob ein Agent a das **key feature** hat. Die Teilmenge der Agenten mit key feature zur Zeit t ist definiert durch

$$A_f(t) = \{a | f(a,t)\}$$

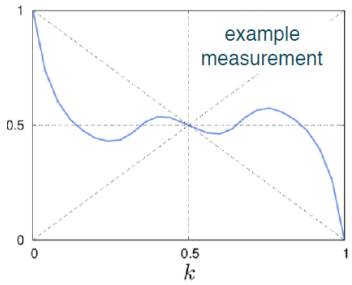
Die Verhältnis von Schlüsselmerkmalen k errechnet sich durch

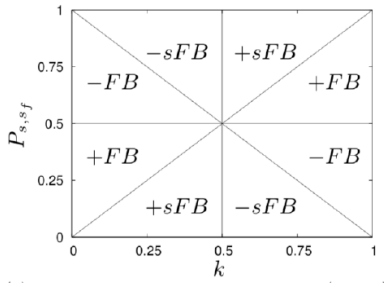
$$k = \frac{|A_f(t)|}{|A(t)|}$$

Wir sagen dann, dass $s_f \to s$ und $s \to s_f$ Key-Feature-Transitionen sind, weil sich dann das Schlüsselmerkmal mind. eines Agenten ändert mit Auswirkung auf das Gesamtsystem.
Die Wahrscheinlichkeit für eine solche Transition mit einem Verhältnis k ist

$$P_{s,s_f}(k) = \frac{P(s \to s_f, k)}{P(s \to s_f, k) + P(s_f \to s, k)}$$

Die Transitions-Wahrscheinlichkeit $P_{s,s_f}(k)$ kann auch **empirisch** gemessen werden und durch einen Octant analysiert werden:





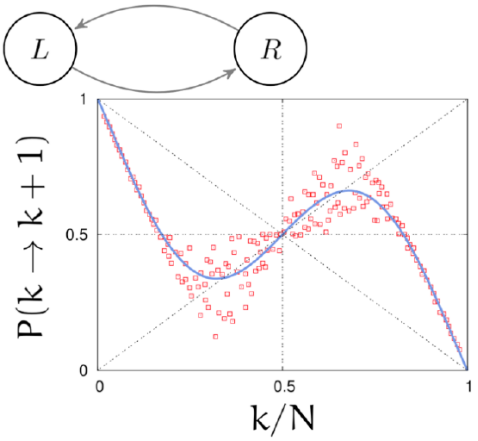
7.2 Exemplarische Analyse

Heuschrecken Bewegung

$$x_i(t+1) = x_i(t) + v_0 v_i(t)$$

$$u_i(t+1) = \begin{cases} G(L_i(t), R_i(t)), & \text{with probability } P_d \\ -u_i(t), & \text{with probability } P_n \\ u_i(t), & \text{else} \end{cases}$$

$$G(L,R) = \begin{cases} -1, & L > R \\ +1, & R > L \\ u_r, & R = L \end{cases}$$



7.2.1 Interpretation

Micro-lvel Markov Ketten reflektieren nicht alle relevanten Systemeigenschaften. Das Wechseln als eine Minorität führt echtes Feedback ein, kleine Komponenten zeigen Fluktuationen, die als negatives Feedback beitragen.

8 Robustheit und Vertrauen

Woher wissen wir, ob das System **sicher** ist?
Wie verhält sich das System bei Fehlern und Störungen?
Wie Flexibel ist es?
Können wir dem System trauen?

8.1 Robustheit

Unter **Robustheit** versteht man die Fähigkeit eines Systems, auch im Falle von Parameterwechsel das gewünschte Verhalten zu garantieren. Die Funktion wird beibehalten, egal ob sich die **interne Struktur** ändert oder die **Umgebung**.

Ursachen für Störungen und Fehler sind in der Regel

- ein falsches Systemmodell
- Numerische Fehler
- Varianz der Parameter
- oder einfach Abnutzung

Unter **Flexibilität** versteht man die Fähigkeit des Systems, sein Verhalten anzupassen, wenn sich Parameter oder Ziele ändern.

8.1.1 System Modell

Ein System S ist ein Tupel $S = (\mathcal{I}, \mathcal{O}, \beta)$

- $\mathcal{I}$ ist das **Input** Interface:

  $$\mathcal{I} = R_s \cup C_c,$$

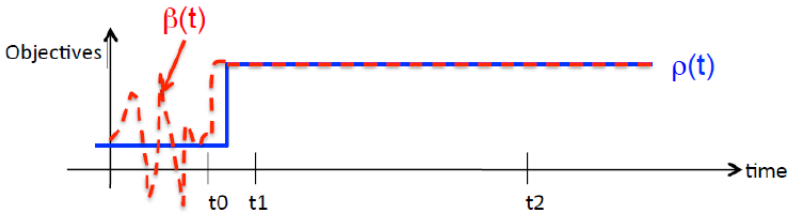
  wobei $r \in R_s$ der rohe Sensor Input ist, und $c \in C_s$ der Control Input
- $\mathcal{O}$ ist das **Output** Interface. $o \in \mathcal{O}$ sind alle Parameter, die vom System beeinflusst werden.
- β ist das **Verhalten**. Es definiert die Relation zwischen R_s und O_s zum Zeitpunkt t und Antwort bei $t + 1$. Es kann auch **zeitvariant** sein:

  $$\beta(t) \subseteq R_S \times O_s$$

- Dazu gibt es noch die Verhaltensspezifikation $\rho \subseteq R_s \times O_s$, welches die Performace Function ist.

8.1.2 Adaptivität

Fähigkeit des Systems, sich an verschiedene Verhaltensspezifikationen ρ anzupassen.
Das System ist **adaptiv** wenn es ein $t_1 \geq t_0$ gibt, sodass für alle t, $t_1 \leq t \leq t_2$ gilt, dass $\rho(t)$ und $\beta(t)$ erfüllt sind.



Zur Evaluation erweitern wir das Systemmodell. Jedes System hat neben der Beschreibung $S = (\mathcal{I}, \mathcal{O}, \beta)$ auch den **aktuellen Zustand** $\mathbf{z}(t)$. Hinzu kommen die Parameter der **Umgebung** und die **interne Struktur** des Systems.

Zur Evaluation wird der Zustandsraum auf den Objective Space **gemappt**. Anhand der numerische Werte kann beurteilt werden, ob das System sich noch im Akzeptierten Zustandsraum befindet.

8.1.3 Störung und unterteilter Zustandsraum

Der Zustandsraum kann in 4 Zonen eingeteilt werden:

- idealer Zustandsraum (Target Space TS)
- akzeptabler Zustandsraum (Acceptance State, AS)
  Zustände, die einen Evaluationswert innerhalb eines Grenzwerts haben
- Überlebenszustand (Survival State, SS)
  Das System kann von SS zu AS durch eine Kontrollsequenz noch zurückkehren
- Todeszone (Dead Space, DS)
  Kein zurück

Eine **Störung** δ kann intern oder extern vom System sein und führt zu einer Systemtransition von (t) zu $\delta(\mathbf{z}(t))$

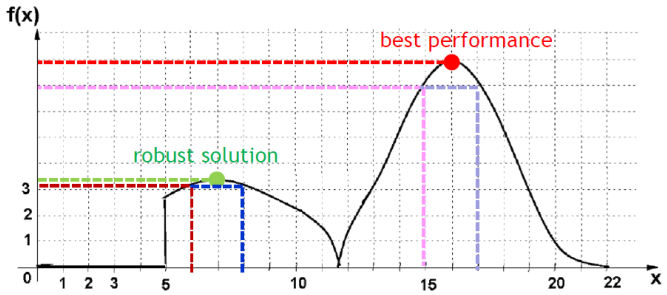
Die Robustheit wird dann anhand der nichtleeren Menge $D = \{\delta_0, \delta_1, \dots\}$ gemessen.
Es ist **stark robust** in Bezug auf D wenn es für alle $\delta \in D$ im idealen Zustandsraum (TS) bleibt, sogar wenn sich Parameter ändern.
Das System ist **schwach robust** in Bezug auf D wenn es für alle $\delta \in D$ im akzeptablen Zustandsraum bleibt (AS)

Ein System ist **adaptiv**, wenn es in Bezug auf D ohne Fremdeinwirkung in AS zurückkehren kann. Es besitzt **Potenzial zur Adaptivität** wenn es nur mithilfe eines **externen Einflusses** zurückkehren kann.

Wichtig:
Flexibilität bedeutet, dass ein System S nach Änderung der **Spezifikation** oder Ziele zu AS zurückkehrt.
Robustheit bedeutet dass ein System S nach einer **Störung** zu AS zurückkehrt.

8.1.4 Arten von Unsicherheiten

- **Aleatorische Unsicherheiten** (=stochastische) durch **natürliche Variabilität** der Prozesse. Können nicht vom Menschen beeinflusst werden. Diese Unsicherheiten werden als Wahrscheinlichkeitsfunktion modelliert. Das benötigt jedoch eine Menge Daten, die oft nicht verfügbar ist.
- **Epistemische Unsicherheiten** (=subjektive) durch **unvollständiges Wissen** über den Prozess und die Modelle. Dazu gehören Unsicherheiten in
  - Parameter: Input und Systemzustand
  - Model: unvollständig oder falsch
  - Szenario: relevantes Szenario, was nicht betrachtet wurde
  - Numerik: Rundungsfehler



Um eine möglichst robuste Lösung zu bestimmen, versucht man das **minimal mögliche Resultat zu maximieren**. Dies ist vor allem für Probleme nützlich, wo eine **absolute Zuverlässigkeit** benötigt wird. Dabei gibt es einen Tradeoff zwischen **Qualität** und **Varianz** der Lösung.

Maximierung der erwarteten Performance:

$$f_{eff} = E(f(x+\delta)) = \int_{-\infty}^{+\infty} p(\delta) f(x+\delta) d\delta$$

Da f oft unbekannt ist, schätzt man die Funktion mit **Monte-Carlo Estimation**, "empirisch ermitteltes Mitteln", ab. Ein Wert x_0 kann durch n Samples $\hat{x}_i = x_0 + \delta_i$ in der Nachbarschaft von x_0:

$$f_{eff}(x_0) = \frac{1}{n} \sum_{i=1}^{n} f(\hat{x}_i)$$

8.1.5 Sampling

- **Zufällig**
- **Antithetisch:** $+\delta$ un $-\delta$, Mittelwert 0
- **Rasterbasiert:** Samples in kleinen Subgruppen, repräsentativ

- **Latin Hypercube Sampling:** Aufteilung in k Teile, wobei jedes Teil nur von einem okkupiert wird

Ein Design ist **robust**, wenn die Varianz der Performance Function minimal ist. Ein Design ist **zuverlässig**, wenn die Wahrscheinlichkeit von Systemversagen minimal ist.

8.1.6 Zuverlässigkeit bestimmen

Reliability Based Design Optimization:
Man möchte die Fitnessfunktion $f(\mathbf{x}, \delta)$ so minimieren, sodass alle J Constraints $g_J(\mathbf{x}, \delta)$ (Abstand zu *infeasable space*) eingehalten werden (Linear Programming?):

$$min\ f(\mathbf{x}, \delta)$$

$$\text{such that } Prob(g_k(\mathbf{x}, \delta) \geq 0) \geq R_j,\ j \in \{1, 2, \ldots, J\}$$

R_j: W'keit dass Constraint j erfüllt ist.

Solution Based:
Man generiere n Lösungen $x + \delta$ und prüft für jede Lösung on $g_j(\mathbf{x}, \delta)$ erfüllt. Misst man r_j erfolgreiche Fälle, so kann man die $Prob-$Funktion zu

$$\frac{r_j}{n} \geq R_j$$

abschätzen. Natürlich muss man dabei nach Lösungen suchen, wo $R_J \neq 1$, um den Abstand zu minimieren. Dabei ist das **globale** Optimum gar nicht so wichtig, da es in der Praxis weniger signifikant ist. Es wird nicht die **Sensibilität** des Systems betrachtet.

8.1.7 Robustheit - 2 Typen

Typ 1: minimiert f_{eff} in einer δ-Nachbarschaft

Typ 2: minimiert prozentuale Abweichung $|f_p(x) - f(x) \leq \eta|$: $f(x)$ soll besser sein als die gestörte Funktion $f_p(x)$

8.1.8 Separated Consideration

Neue Fitnessfunktion, die objective function und constraint violatoin trennt:

$$F(\mathbf{x}) = \begin{cases} f(\mathbf{x},\ \text{for } g_j(\mathbf{x}) \geq 0, \forall j \in \{1, 2, \ldots, J\}) \\ f_{max} + \sum_{j=1}^{J} |g_j(\mathbf{x})|,\ \text{else} \end{cases}$$

8.1.9 Pareto-Optimal und Paretodominanz

Eine Lösung x^* ist Pareto-Optimal, wenn es kein $\mathbf{x} \in S$ gibt, welches besser ist (Optimalität) und es in einer Dimension echt besser ist (?). Das x muss also in einigen Dimensionen besser sein, aber im Schnitt schlechter abschneiden.

8.2 Vertrauen

"*Vertrauen ist eine Wette mit bestimmten Risiko, nämlich dem, was mit durch Vertrauen gewonnen und verloren werden kann.*"

"*Die Bereitschaft, sich für andere verwundbar zu machen, auf Grundlage der Erwartung, dass der Andere eine Aktion ausführt, die dem Vertrauensgeber wichtig ist.*"

Nutzer eines OC-Systems müssen Kontrolle aufgeben, da das System **selbstorganisiert** ist. Auch der Entwickler kann aufgrund der **Komplexität** des OC Systems keine klassischen Methoden anwenden. Im System selbst muss eine Einheit entscheiden, ob es einer anderen Einheit trauen kann.

8.2.1 Aspekte von Vertrauen

- **Korrektheit:** Tut, was es soll. **Formale Methoden**
- **Sicherheit:** Autorisierter Zugriff. **Authentität**, **Autorisierung**, **Integrität**
- **Sicherheit:** Unerwartete Nebeneffekte. **Systemgrenzen**
- **Verfügbarkeit:** Verfügbar oder Erreichbar? **MTTF**, **MTTR**, **MTBF**
- **Robustheit:** Funktionalität unter verschiedenen Konditionen
- **Privatsphäre:** Nutzung privater Daten. **Persönliche Informationen**
- **Performance:** gewünschte Antwortzeiten
- **Glaubwürdigkeit:** Kommunkation mit Nutzer transparent? **Falsche Agenten**
- **Nutzbarkeit:** User Interface?

8.2.2 Wie misst man Vertrauen?

In verteilten OC Systemen wird Vertrauen auf die Verfügbarkeit reduziert, die von anderen Knoten observiert werden kann.

Dafür verwendbare Metriken wären die Anzahl der angekommenen **Delayed ACK**:

- Mittel: $\frac{\sum_{i=1}^{n} x_i}{n}$
- Gewichtetes Mittel: $\frac{\sum_{i=1}^{n} \frac{i}{n} x_i}{\sum_{i=1}^{n} \frac{i}{n}}$
- Invers Gewichtetes Mittel: $\frac{\sum_{i=1}^{n} \frac{n-i}{n} x_i}{\sum_{i=1}^{n} \frac{n-i}{n}}$

9 Anwendungen

9.1 Urban Traffic Control

Verkehrskontrolle und -management ist durch die hohe Dynamik sehr komplex. Es interagieren viele Teilnehmer parallel zueinander, wobei die Teilnehmer aufgrund verschiedener Ziele oft im Konflikt zueinander stehen.

Ein zentralisierter Ansatz skaliert nicht, da die benötigte Verarbeitung der Datenmanegen in Echtzeit schwer möglich ist.
Ein dezentralisierter Ansatz hingegen reduziert die Komplexität des Optimierungsproblems, und führt zu kürzeren Reaktionszeiten.

Zur Optimierung des Verkehrsflusses müssen

- adaptive Lernmechanismen für Ampelschaltungen,
- selbst-organisierte Koordination von grünen Wellen,
- aktives und adaptives Verkehrsleitsystem

betrachtet werden.
Entworfen wird ein OC System mit mehrschichtiger Architektur mit einer **reaktiven** Schicht (aktuelle Verkehrslage), einer **reflektiven** Schicht (Optimierung, falls Layer 1 Hilfe benötigt) und einer **Nutzerschicht** (Ziele durch Operatoren).

9.1.1 Decentralized Progressive Signal System

DPSS ist ein Prozess der auf drei Schritten basiert:

1. **Partnerschaft:** Kreuzungen finden und kollaborieren mit Partnern. Diese sind vom Demand abhängig! Stärkster Fluss wird bestimmt.
2. **Zykluszeit:** Kollaboratoren einigen sich auf einen gemeinsamen Zyklus. Maximum der benötigten Zykluszeiten entlang der Verkehrsrichtung muss erfasst werden!
3. **Signal Plan:** Auswahl der Pläne, Berechnung des Versatzes (grüne Welle)

9.1.2 Hierarchical Progressive Signal System

Lokale Lösungen sind oft suboptimal. Der Fluss einer Straße ist oft nicht so groß wie de Fluss mehrerer Parallelstraßen.

1. **Netzwerkgraph** wird erstellt, der die Netzwerktopologie repräsentiert, Gewichte an Kanten sind der aktuelle Flow.
2. **Verkehrsstrom** bestimmen anhand einer Heuristik
3. **Strömungssystem** wird bestimmt, wobei Ströme sich nicht überschneiden dürfen.

9.2 Trust Communities in offenen verteilten Systemen

Komponenten in OC Systemen sind **unzuverlässig**, mit gewissen Unsicherheiten muss gerechnet werden. Die Idee ist, dass Agenten das Verhalten anderer bewerten und Trust Communities formen, um maliziöse Teilnehmer auszugrenzen und die Kommunikation zu vertrauten Agenten zu vereinfachen (weniger Overhead).

9.2.1 Stereotypische Verhalten

- Adaptiver Agent: Kooperativ. Reputation relativ zur Systemauslastung
- Free Rider: Lehnt alle Job Requests ab, erhöht Systemauslastung
- Egoist: Akzeptiert Requests, gibt aber falsche Werte aus, verringert Nutzbarkeit
- Listige Agenten: Periodisch kooperativ/egoistisch, verringert Nutzbarkeit
- Altruisten: Akzeptieren jeden Request, auch den von maliziösen Agenten

9.2.2 Implicit Trust Community

Dezentraler Ansatz, der nicht unbedingt auf Gegenseitigkeit beruht. Reputation wird in $Rep \in [-1, +1]$ gewertet, wobei < 0 **ausgeschlossen** wird. Agenten, dessen Reputation höher als ein Trust Treshold sind ($Rep > TT^{sub}$) werden als **kooperative** Agenten gewertet.

Dieser Ansatz konvergiert schnell und funktioniert in der Regel gut in Sensornetzen, da auch da Kommunikation auf lokaler Interaktion basiert.

9.2.3 Explicit Trust Community

Hierbei gibt es eine explizite **Mitgliedschaftsfunktion**. Agenten eröffnen zunächst ein iTC, wählen dann in ihrere Gemeinschaft einen **Trust Community Manager (TCM)**, der dann die Reputation der Agenten administriert und mit allen Agenten teilt. Dadruch wird eine erhöhte Performanz versprochen (Reduktion der Sicherheitsmechanismen).

Jedoch können Probleme auftreten:

- **Vertrauenspanne:** Viele Angreifer treten der Gemeinschaft bei, die allen Teilnehmern eine schlechte Bewertung geben, sodass niemand mehr jemanden traut. (**Schlammschlacht**)
- **Geheime Absprachen:** Ähnlich wie oben, nur dass sich Teilnehmer gegenseitig gut Bewerten und die Reputation erhöhen. Dadurch wird ein Beitritt in eine Gemeinschaft erleichtert.

9.2.4 Normative Trust Community

Ganz anderer Ansatz: Es gibt eine globale Instanz (Norm Manager) der die Systembelastung und Vertrauensbrüche observiert.

- Vertrauensgraph: Wer traut wem?
- Jobgraph: Wer gibt wem Jobs?
- Führt Graph-Analysen durch (Schadhaftes Verhalten erkennen)

9.3 Electric Power Management

Ein rein reaktives Verhalten reicht nicht aus, um Angebot-Nachfrage Balance aufrechtzuerhalten. Aufgrund von **witterungsbedingten Energiequellen** ist das Angebot unsicher. Es werden proaktive Funktionen benötigt.

Idee: **Repeated Scheduling**. Alle 15min wird anhand eines Models abgeschätzt, wie weit die benötige Nachfrage sinkt oder steigt.

Virtual Power Plants sind zielorientierte und kurzlebige Koalitionen von Kraftwerken. Sie können ohne fremde Hilfe auf kleine **Veränderungen** reagieren, **unabhängig** handeln und helfen dabei, das Energienetz in **microgrids** aufzuteilen.

Die (A)VPPs müssen dabei non-dispatchable Kraftwerek enthalten und dürfen in der internen Struktur nicht homogen sein, da sich sonst Cluster von unähnlichen Systemen mit verschiedenen Zuverlässigkeiten bilden.