XCS-O/C: Das Extended Classifier System XCS in einer Observer/Controller-Architektur

Alexander Paßberger
Universität Passau

Passau, Deutschland
passbe04@gw.uni-passau.de

***Abstract*—Organic Computing stellt eine moderne Variante des Entwerfen autonomer Systeme dar, bei der Entscheidungen zur Laufzeit vom System selbst getroffen werden. Das Steuern der ausgeführten Aktionen benötigt in einem solchen System Möglichkeiten zur Selbstverbesserung. Ein möglicher Ansatz zur Problemlösung sind Learning Classifier Systeme, speziell das Extended Classifier System XCS. Die Arbeit liefert eine detaillierte Beschreibung der Abläufe im Extended Classifier System XCS und der nötigen Änderungen zum Integrieren in ein autonomes organisches System. Das Learning Classifier System wird hierzu in eine generische Observer/Controller-Architektur eingebettet, eine der fundamentalen Designarchitekturen im Bereich Organic Computing.**

I. EINFÜHRUNG

Tiere verarbeiten zum Überleben ständig die Signale aus ihrer Umgebung und treffen darauf basierende Entscheidungen, welche der Befriedigung ihrer Bedürfnisse dienen. Nur wenige Verhaltensweisen sind angeboren, die meisten werden durch Ausprobieren, sprich Erfahrungen, im Laufe des Lebens erlernt. Tiere können sich nur eine limitierte Anzahl an Erfahrungen merken. Unter der Annahme, dass Tiere nur inkrementell lernen, muss es ihnen möglich sein die wahrgenommenen Informationen zu filtern. Sie leiten aus ihren Erfahrungen allgemein geltende Regeln ab, anstatt sich alle Erfahrungen zu merken.
Des Weiteren sind Tiere in ihrer natürlichen Umgebung meistens auf sich alleine gestellt. Sie müssen selbst beurteilen, inwiefern ihre Entscheidungen den gewünschten Effekt haben. Da nicht auszuschließen ist, dass eine andere Verhaltensweise die Bedürfnisse besser befriedigt hätte, ist eine Bewertung der Effektivität im Verhältnis zu anderen Optionen notwendig, beziehungsweise eine Bewertung des Grades der Befriedigung. Die meisten Entscheidungen verursachen keinen direkten Ertrag, sondern dienen der Vorbereitung, um im nächsten Schritt, oder einem darauf folgenden, erfolgreich zu sein. Das Tier muss in der Lage sein, den Zusammenhang von vorhergegangen Erfahrungen in Beziehung zum Ertrag zu setzen, um Ausgangssituationen frühestmöglich zu erkennen.
Um dem aufgeführten deduktiven Ansatz zu folgen, müssen Tiere aus verschiedenen Bedingungen und Aktionen durch Erfahrung allgemein gültige Regeln ableiten, um ihr Überleben zu sichern. Ähnliche Voraussetzungen gelten für autonome Roboter, wie sie beispielsweise auf dem Mars eingesetzt werden. Dies ist als Animatproblem (animat engl. artificial animal) bekannt und stellt ein Grundlegendes Problem im Bereich des maschinellen lernen, speziell im Bereich des reaktionsbasierten Lernverfahrens dar. Eine allgemein gültige Repräsentationsmöglichkeit findet man am Beispiel von Bedingungs-/Aktionspaare bei denen die Bedingung bestimmten Aspekten der Umwelt und des internen Zustands entsprechen muss. Die ausgeführte Aktion verändert den internen Zustand oder führt eine Bewegung [1].

Learning Classifier Systeme (LCS) verbinden Lernverfahren aus den Reinforcement-Verfahren und genetische Algorithmen miteinander. Sie sind eine vielversprechende Problemlösung. LCS sollen eine Menge von Regeln, bzw. von Classifiern evolvieren, welche möglichst genau und generell sind, d.h. eine Menge, die möglichst viele Probleminstanzen korrekt klassifiziert [2].
Auch im Forschungsgebiet Organic Computing (OC) bezieht man sich beim Entwerfen von komplexen verteilten Systemen auf das Vorbild der Natur (organisch). So sollen die Systeme sich unter anderem selbst organisieren, selbst konfigurieren, selbst reparieren oder anpassen können. Dadurch werden die Herausforderungen für den Designer verringert, weil nicht länger jedes Verhalten in den Situationen einzeln spezifiziert werden muss. Stattdessen wird den Systemen ein gewisser Grad an Freiheit in ihrer Reaktion gelassen, welche nur durch Setzen weniger generalisierter Parameter beeinflusst wird [3]. Das Verwenden bestimmter selbstlernender Verfahren und Algorithmen innerhalb eines organischen Systems ist unabdingbar, wenn eine Anpassung und eine Selbstkonfiguration erreicht werden soll. Es besteht somit ein Schnittpunkt zwischen den maschinellen Lernverfahren und organischen Systemen. In dieser Arbeit wird insbesondere auf das Extended Classifier System (XCS), seinen Algorithmen zum Entwickeln von Classifiern und den nötigen Anpassungen zur Einbindung in ein organisches System eingegangen.

II. LEARNING CLASSIFIER SYSTEMS

A. Zweck von Learning Classifier Systemen

Learning Classifier Systeme gehören zu den maschinellen Lernverfahren und lösen sowohl Klassifizierungsprobleme, als auch Reinforcement-Probleme durch Evolvieren allgemein gültiger Regeln [2]. Erstmals 1976 von John H. Holland vorgestellt, entwickelten sich im Laufe der Zeit zwei Hauptvarianten von LCS: Das Michigan-style LCS und das Pittsburgh-style LCS. Im Folgenden wird nur auf das Michigan-style LCs eingegangen, das im Gegensatz zum Pittsburgh-style LCS nach jeder Zustandsveränderung immer nur eine Lösung durch iteratives Updaten der einzelnen Regeln evolviert und als Basis des Extended Classifier System XCS dient. LCS bilden die Umgebung meistens binär als String aus der Menge {0, 1, #} ab. Das Wildcard Symbol # zählt sowohl als Eins, als auch als Null [4].

B. Klassifizierungs- und Reinforcement-Probleme

Bei einem Klassifizierungsproblem soll für jede Probleminstanz $s \in S$ die zugehörige Klasse $a \in A$ gefunden werden. Es soll die korrekte Abbildung von S nach A, $SXA: s \rightarrow a$ aus dem Konzeptraum, demnach aus allen möglichen Abbildungen von S nach A gefunden werden.

Bei den Reinforcement-Problemen hingegen können Probleminstanzen von vorherigen Probleminstanzen und Aktionen abhängig sein. Sie liefern keine Klassenzugehörigkeit, sondern eine Bewertung der ausgeführten Aktion. Sie sind somit näher an der natürlichen Vorlage. Auch das in der Einführung veranschaulichte Animatproblem zählt zu den Reinforcement-Problemen.
Klassifizierungsprobleme lassen sich zu Reinforcement-Problemen umformulieren, indem man eine Belohnung für eine richtige Klassifizierung und eine falsche wählt. Deswegen wird häufig der Ausdruck single-step Reinforcement-Problem, bzw. mulit-step Reinforcement-Problem gewählt [2].

C. Aufbau moderner Learning Classifier Systeme

Ein modernes LCS besitzt grundsätzlich drei Komponenten: eine Performance Komponente zum Auswählen der aktuellen Umweltsituation entsprechender Classifier, sowie der als nächstes auszuführenden Aktion, eine Discovery Komponente, zum Erforschen neuer Regeln beziehungsweise Classifiern und eine Reinforcement Komponente, zum Updaten vorhandener Classifiern um allgemein gültige Regeln zu evolvieren. Dabei werden sämtliche Regeln in einer größenbeschränkten Population [P] gespeichert, wodurch eine Konkurrenz zum Verbleiben in der Population ausgelöst wird und schlechte Classifier durch die Discovery Komponente gelöscht werden. Die Größe für [P] kann je nach Anwendungsgebiet unterschiedlich gewählt werden. Sie sollte aber groß genug sein um alle Umweltsituation abzudecken. Ansonsten würde das LCS in einem Kreislauf von Löschen zu anschließendem Wiederentdecken sinnvoller Regeln laufen. Jeder, der aktuellen Situation σ entsprechender Classifier kommt bei jeder Iteration ins Matching Set [M]. Alle Classifier aus [M], welche die gleiche wie von der Performance Komponente ausgewählte Aktion ausführen kommen in das Action Set [A]. Ein einzelner Classifier besteht im einfachen Fall aus einem Tripel cl:= (C, a, s), wobei C für die Bedingung zur Ausführung eines Classifier steht (engl. condition), a für die auszuführende Aktion und s für den Bewertungswert der Aktion a (engl. strength value) [4]. Das Zusammenspiel der verschiedenen Komponenten ist in Abb.1 nochmals graphisch dargestellt. Die einzelnen Bereiche werden im Folgenden detailliert betrachtet.

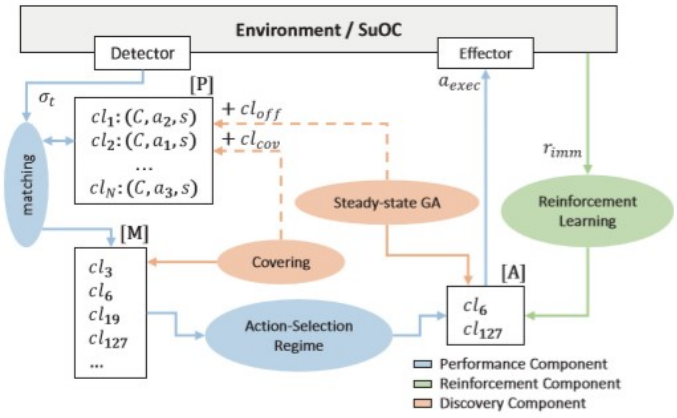

Abb. 1: Aufbau eines modernen LCS, entnommen aus [4]

1) Die Performance Komponente

Die Performance Komponente arbeitet, wenn der Detektor oder die Detektoren des Systems eine neue Umgebungssituation σ melden. Dies unterteilt sich in einem Matching-Prozess und einem Action-Selection-Prozess.
Beim Matching werden alle bis dato entwickelten Regeln cl ∈ [P] mit der vorliegenden Situation verglichen. Jeder Classifier, bei dem die Bedingung C erfüllt ist, wird daraufhin in das Match-Set [M] ⊆ [P] gesteckt. Im Allgemeinen enthält [M] Classifier welche mehr als eine verschiedene Aktion ausführen. Falls nicht, erzeugt die Discovery Komponente weitere Classifier.
An dieser Stelle greift der Action-Selection-Prozess. Er wählt zum einen die Aktion aus, welche das beste Ergebnis verspricht, zum anderen aber auch zufällig, um die vorliegende Situation zu erforschen und neue, unter Umständen auch negative Erfahrungen, zu erhalten. Das richtige Verhältnis zwischen der Nutzung bewährter Regeln und dem Erforschen neuer Regeln zu finden, ist eines der schwierigsten Designprobleme in der Verwendung eines LCS [4].

2) Die Reinforcement Komponente

Nach der von der Performance Komponente ausgewählten Aktion erhält die Reinforcement Komponente anschließend die Belohnung r_{imm}. Daraufhin wird von allen Classifiern aus dem Action Set [A] die Strengh Value s aktualisiert. Bei multi-step-Problemen muss die Belohnung auf alle Classifier aufgeteilt werden, die an der zum Erfolg geführten Kette beteiligt waren. Diese Herausforderung, im Englischen auch als credit-assignement-Problem bekannt, wird in klassischen LCS mittels des Bucket Brigade Algorithm (BBA) von John H. Holland gelöst. Aber auch neuere Algorithmen, wie der Q-learning Algorithmus oder der modifizierte Implicit Bucket Brigade Algorithmus, sind geläufig [4]. Eine mögliche Lösung des Problems findet sich im Abschnitt III bei der Aktualisierung der XCS Classifier Werte.

3) Die Discovery Komponente

Nachdem die Mechanismen zum Auswählen geeigneter Regeln, sowie zum Bewerten des Ertrags vorgestellt wurden, bleibt noch die Komponente, welche überhaupt erst Classifier erzeugt, offen: Die Discovery Komponente. Das Erzeugen neuer Regeln geschieht einerseits mittels Covering, wenn kein Classifier in [P] vorhanden ist, welcher der aktuellen Umweltsituation entspricht, beziehungsweise der Betrag von [M] kleiner als die festgelegte Untergrenze ist. Das zweite Modul, welches neue Regeln erzeugt ist der steady-state Genetic Algorithm (GA). Dieser wird, festgelegt durch einen Parameter, in größeren Zeitabständen periodisch angewendet.

Jede Umweltsituation $\sigma = (s_1, \dots, s_n)$ bei welcher kein Classifer cl ∈ [P] mit erfüllter Bedingung existiert, führt zu einem Stillstand des Systems. Um dies zu verhindern, erstellt die Discovery Komponente in solchen Fällen einen Classifier cl_{cov}, durch welchen die Hauptschleife normal weiter läuft. Die Anzahl an erzeugten Classifiern hängt von der Anzahl an möglichen Aktionen des Systems ab. Ebenso wird der Covering-Mechanismus ausgeführt, wenn die Anzahl an in [M] enthaltenen unterschiedlichen Aktionen einen festgelegten Grenzwert unterschreitet. Der neu erzeugten Regel wird für s ein Standardwert und für a eine beliebige in

[M] nicht enthaltene Aktion zugewiesen. Als Bedingung wird eine probabilistisch generalisierte Form der Ausgangssituation σ gewählt, welche auch bei benachbarten Situationen erfüllt wird.

Die zweite Komponente, der steady-state GA, ersetzt im Gegensatz zu einem normalen generational GA nicht den Großteil der vorhandenen Regeln durch die Standard Operatoren (Selektion, Mutation und Crossover). Er erzeugt zwei neue Classifier cl_{off}, indem er vorhandene Classifier kopiert, deren Bedingung mutiert und kreuzt. Die erzeugten Regeln werden dann eventuell zur Population [P] hinzugefügt. Der genetische Algorithmus wurde zu Beginn der LCS auf die gesamte Population [P] angewendet. In modernen LCS wird hauptsächlich ein steady-state niche GA verwendet, welcher auf [M] oder gar [A] arbeitet. Dabei haben Regeln mit einer höheren erwarteten Belohnung s eine höhere Wahrscheinlichkeit ausgewählt zu werden. Meistens wird für die Wahrscheinlichkeit eines Classifier vom GA ausgewählt zu werden folgende Gleichung verwendet:

$$p_{GA}(cl) = \frac{cl.s}{\sum_{cl' \in [A]} cl'.s}$$

Somit werden bessere Regeln bevorzugt genauer erforscht. Zu generelle Regeln, die ein schlechtes Ergebnis liefern, werden von der Reinforcement Komponente wieder herabgesetzt, sodass ihre Wahrscheinlichkeit zur weiteren Reproduktion wieder sinkt[4].

III. Das Extended Classifier Sytem XCS

Das Extended Classifier System XCS wurde 1995 von Stewart W. Wilson als Weiterentwicklung klassischer LCS postuliert. Wie jedes LCS sammelt es Informationen aus der Umwelt, führt darauf basierende Aktionen aus und erhält als Bewertung der ausgeführten Aktion eine Belohnung. XCS ist hingegen in der Lage Regeln zu evolvieren, welche die erwartete Belohnung nach Ausführung einer Aktion exakt vorhersagen. Klassische LCS entwickeln hauptsächlich Regeln mit hoher Belohnung [6]. Das Ziel von XCS ist es, eine möglichst genaue, vollständige sowie maximal generelle Zustands-Aktion-Abbildung, formal $X \times A \Rightarrow P$, zu konstruieren. X stellt die Menge aller möglichen Umweltzustände (vgl. S bei LCS) dar, A die Menge aller Aktionen (vgl. LCS) sowie P den Belohnungsraum [5]. Abbildung 2 zeigt das Extended Classifier System XCS in einer schematischen Darstellung (vgl. Abbildung 1 eines LCS in Abschnitt II. Learning Classifier Systems).

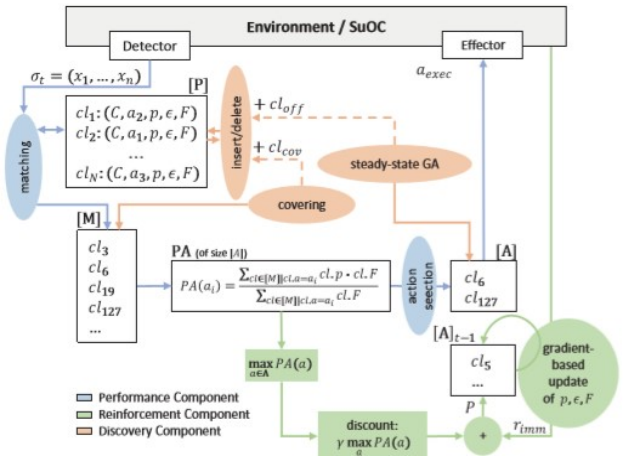

Abb. 2: Aufbau des Extended Classifier System XCS, entnommen aus [4]

A. Unterschiede zu einfachen Learning Classifier Systemen

XCS unterscheidet sich von anderen Learning Classifier Systemen hauptsächlich bei der Bewertung von ausgeführten Aktionen, bzw. deren Weiterverwendung. Anders als bei den klassischen LCS, die mit der Stärke einer Regel arbeiten, nutzt XCS vor allem die Genauigkeit der Vorhersage als Belohnung. Somit verbleiben auch Regeln in der Population, die zwar keine große Belohnung versprechen, diese aber genau vorhersagen können. Des Weiteren bildet der genetische Algorithmus bei XCS neue Regeln anhand des Action Sets [A], löscht vorhandene jedoch in der Gesamtpopulation [P]. Hierdurch lernt XCS besonders vollständig, da zur Produktion neuer Regeln lediglich Regeln aus der Nische der aktuellen Umweltsituation verwendet werden. XCS erfüllt somit die zwei Hauptanforderungen von Learning Classifier Systemen: Regeln so allgemein wie möglich zu halten, sowie die Zustands-Aktion-Abbildung vollständig und genau zu erlernen [2].

B. Classifier in XCS

Ein Classifier in XCS besitzt deutlich mehr Bestandteile als der eines klassischen LCS. Die Bedingung C, die Aktion a sowie die Vorhersage p entsprechen denen aus den LCS bekannten Eigenschaften. Zusätzlich wird der Vorhersagefehler ϵ , welcher den durchschnittlichen, absoluten Fehler der Vorhersage von der tatsächlichen Belohnung angibt, sowie die Fitness f , welche die Genauigkeit relativ zu anderen, überlappenden Classifiern angibt, gespeichert. Neben diesen fünf Hauptbestandteilen besitzt ein XCS Classifier noch weitere gespeicherte Parameter: Eine Action-Set-Größe as, welche den Durchschnitt der Action-Set-Größen angibt, in welchen der Classifier war. Einen Zeitstempel ts, der speichert wann der Classifier das letzte Mal vom GA ausgewählt wurde. Einen Erfahrungswert exp, der die Anzahl der Änderungen der Parameter des Classifier mitzählt, sowie eine Numerosität num, welche die Anzahl an syntaktisch äquivalenten repräsentierten Micro-Classifiern angibt [2]. Syntaktisch äquivalent bedeutet in diesem Zusammenhang, dass zwei Classifier die gleiche Umweltsituation abdecken, als auch die gleiche Aktion ausführen. Beim Einfügen neuer Classifier in [P] wird überprüft ob dieser bereits durch einen vorhandenen Classifier subsumiert wird und im Falle dessen die Numerosität des Subsumierenden Classifier erhöht, anstatt den neuen Classifier hinzuzufügen. Ein Micro-Classifier entspricht also einem Classifier mit $num = 1$. Bei der Berechnung der Größe von [P] werden die einzelnen Numerositäten zusammengezählt, ein Classifier mit $num > 1$ repräsentiert somit mehrere Micro-Classifier [4]. Es ergibt sich formell die Struktur $cl := (C, a, p, \epsilon, f, as, ts, exp, num)$.

C. Parameteraktualisierung in XCS

Nach dem Bilden des Match-Sets [M] wird ein Vorhersagevektor PA berechnet, welcher die Belohnung für jede mögliche Aktion bestimmt:

$$PA(a_i) = \frac{\sum_{cl \in [M] \mid cl.a=a_i} cl.p * cl.f}{\sum_{[M] \mid a} cl.f}$$

Dieser wird nach der Lernphase, in welcher das Classifier System eine zufällige Aktion auswählt, von der Performance Komponente genutzt. Es wählt dann die Aktion $a_{max} = \arg max_a\, PA(a)$. XCS aktualisiert daraufhin die Parameter der Classifier in der Reihenfolge:

1. Der Vorhersagefehlerϵ,
2. Die Vorhersage p
3. Die Fitness f.

Hierbei wird die Aktualisierung bei Klassifizierungsproblemen direkt nach den folgenden Regeln durchgeführt. Bei Reinforcement-Problemen hingegen auf das vorherige Action-Set $[A]^{t-1}$.
Für den Vorhersagefehler wird bei Reinforcement-Problemen zuerst ρ gesetzt:

$$\rho = r_{imm} + \gamma max_{a \in A} PA(a)$$

Bei Klassifizierungsproblemen wird ρ lediglich r_{imm} zugewiesen. Anschließend wird der Vorhersagefehler des Classifier mittels

$$cl.\epsilon \leftarrow cl.\epsilon + \beta(|\rho - cl.p| - cl.\epsilon)$$

für alle Classifier aus [A] aktualisiert [4]. β gibt dabei die Lernrate an und γ den Einfluss früherer Belohnungen. Je höher β ist, umso stärker werden neue Informationen im Verhältnis zu alten gewichtet, je höher γ, umso ältere Belohnungen [2].
Die Vorhersage p wird anschließend durch folgende, auf dem Q-learning Algorithmus basierende Formel, aktualisiert:

$$cl.p \leftarrow cl.p + \beta(\rho - cl.p)\ \forall\, cl \in [A]$$

Das Aktualisieren des Fitness-Attributes geschieht in mehreren Schritten. Zuerst wird die absolute Genauigkeit cl.k für jeden Classifier in $[A]^{t-1}$ berechnet. Danach werden die absoluten Werte in Action-Set relative Werte k‘ transformiert. Zuletzt wird der Fitnesswert auf Basis von k‘ aktualisiert. Für die Berechnung der absoluten Genauigkeit wird der Fehler-Toleranz-Parameter ε_0 eingeführt, welcher eine obere Grenze für ε angibt. Unter diesem Wert gilt eine Regel als völlig exakt. Die Parameter $\alpha\ und\ v$ werden zum Steuern des Einflusses einer falschen Vorhersage auf die Genauigkeit eines Classifier benutzt. Typische Werte sind $\alpha = 0.1$ sowie $v = 5$. Es ergibt sich:

$$cl.k = \begin{cases} \alpha\,(\frac{\varepsilon}{\varepsilon_0})^{-v}, & cl.\varepsilon \geq \varepsilon_0 \\ 1, sonst \end{cases}$$

Aus Folge dessen:

$$cl.k' := \frac{cl.k * cl.num}{\sum_{cl\,\epsilon\,[A]} cl.k * cl.num}$$

Die relative Genauigkeit spiegelt somit das Verhältnis der Genauigkeit eines bestimmten Classifier in Bezug auf alle Classifier mit der gleichen Aktion und in der gleichen, durch [A] gegebenen, Umgebungsnische wider. Mittels

$$cl.f = cl.f + \beta(cl.k' - cl.f), \forall\, cl \in [A]$$

wird zu guter Letzt die Fitness mit Hilfe der berechneten Genauigkeit k‘ aktualisiert [4].
Die weiteren Parameter werden nun in beliebiger Reihenfolge abgearbeitet. Die Action-Set-Größe as wird ebenfalls in Abhängigkeit von β aktualisiert, es gilt: $cl.as = cl.as + \beta(|[A]| - cl.as)$. Der Erfahrungswert $cl.exp$ wird bei jeder Aktualisierung eines Classifier um eins erhöht [2]. Der Zeitstempel wird vom genetischen Algorithmus gesetzt. Die Numerosität wird nur beim Erzeugen neuer Classifier verändert. Sie wird im nächsten Abschnitt detaillierter beschrieben.

D. Der steady-state niche GA

Der GA in XCS operiert auf dem aktuellen Action-Set [A] und wird periodisch ausgeführt, wenn die Ungleichung

$$t - \left(\frac{\sum_{cl\,\in[A]} cl.ts * cl.num}{\sum_{cl\,\in[A]} cl.num}\right) > \theta_{GA}$$

erfüllt ist. D.h. immer, wenn die durchschnittlich verstrichene Zeit aller Classifier in [A] den definierten Grenzwert θ_{GA} (üblicherweise auf 12 gesetzt) überschreitet [4].
Zuerst werden im klassischen XCS zwei Eltern-Classifier in Abhängigkeit von dem Fitnesswert f gewählt [2]:

$$p_s\,(cl) = \frac{cl.f}{\sum_{cl'\in[A]} cl'.f}$$

Dies ist als Roulette-Wheel-Selektion bekannt. In neueren XCS Versionen wird vor allem die Tournament-Selektion gewählt, welche im Allgemeinen bessere Ergebnisse liefert und somit als aktuellen Standard angesehen werden kann [4]. Es findet eine Art Wettbewerb zwischen einer bestimmten Anzahl s an Classifiern statt. Ausgewählt werden dann einfach die beiden Classifier mit dem höchsten Fitnesswert. Der Auswahldruck, und somit der durchschnittliche Fitnesswert der Gewinner, ist dabei umso höher, je größer s gewählt wird. Die Teilnehmer des Wettbewerbs werden zufällig aus [A] gewählt [7]. Von den beiden Eltern-Classifiern werden als nächstes zwei identische Kopien erstellt und deren Bedingungen gegenseitig mit einem Crossover Operator permutiert. Die Wahrscheinlichkeit x, dass ein Symbol der Bedingung permutiert wird im Normalfall auf 0,8 gesetzt. Zuletzt werden die Bedingungen der beiden Classifier individuell mutiert. Bei binären XCS wird mit der Wahrscheinlichkeit μ ein Bit von 0 auf 1, beziehungsweise von 1 auf 0 verändert. Im Falle einer Wildcard # wird das Bit mit gleicher Wahrscheinlichkeit auf null oder eins gesetzt. Ein geläufiger Standardwert ist $\mu = 0.04$. Die Werte für $p, \varepsilon\ und\ f$ werden auf den Durchschnitt der beiden Eltern-Classifier gesetzt, wobei die Fitness meistens noch mit Reduktionsfaktor $F_{reduction} = 0.1$ gewichtet wird. Solch ein Reduktionsfaktor existiert ebenso für ε. Dieser ist in der Regel auf eins gesetzt und wird somit nicht verwendet [4]. Der Wert für die Action-Set-Grüße as wird direkt vom Eltern-Classifier übernommen. Der Erfahrungswert exp und die Numerosität num werden auf eins gesetzt. Vor dem endgültigen Hinzufügen in [P] wird noch überprüft ob die erzeugten

Regeln nicht bereits von einer vorhandenen Regel überdeckt werden [2]. Ein Classifier überdeckt oder subsumiert einen anderen Classifier wenn er mehr Wildcards besitzt und sich alle restlichen Stellen gleichen. Außerdem muss der subsumierende Classifier ausreichend erfahren ($cl.\varepsilon < \varepsilon_0$), sowie hinreichend genau ($cl.exp > \theta_{sub}$) sein. θ_{sub} ist ein weiterer Parameter von XCS, welcher die mindestens notwendige Erfahrung angibt, damit ein Classifier andere subsumieren kann. Er wird üblicherweise auf 20 gesetzt [4].

E. Das Löschen aus [P] in XCS

Wenn die Population [P] die maximale Anzahl an Micro-Classifiern übersteigt, also $\sum_{cl \in P} cl.num > N$, werden $|P| - N$ Classifier aus [P] entfernt. Die Wahrscheinlichkeit zur Löschung einer bestimmten Regeln berechnet sich auf Grundlage der Action-Set-Größen der einzelnen Classifier cl.as, der mittleren Fitness der gesamten Population $\overline{f}$, sowie der Erfahrung cl.exp eines Classifer. Es ergibt sich für jeden Classifier die deletion vote:

$$vote(cl) = \begin{cases} cl.as * cl.num * \frac{\overline{f}}{\frac{cl.f}{cl.num}}, & \text{für } \wedge \frac{cl.\text{exp} > \theta_{del}}{\frac{cl.f}{cl.num} < \delta * \overline{f}} \\ cl.as * cl.num, & sonst \end{cases}$$

Mit

$$\overline{f} = \frac{\sum_{cl \in [P]} cl.f}{\sum_{cl \in [P]} cl.num}$$

θ_{del} ist ein weiterer Grenzwert welcher die minimale Anzahl an Aktualisierungen eines Classifier vorgibt, bevor dessen Fitness berücksichtigt wird. δ ist der Bruchteil von $\overline{f}$ welcher ein Micro-Classifier mindestens als Fitness überschreiten muss, damit sich seine Wahrscheinlichkeit zur Löschung erhöht. Die zu löschenden Regeln werden dann mit der bereits vorgestellten Roulette-Wheel Selektion bestimmt[4]. Es gilt:

$$p_{selection}(cl) = \frac{vote(cl)}{\sum_{cl' \in [P]} vote(cl')}$$

Es werden nur Classifier mit $cl.num = 1$ tatsächlich gelöscht, bei allen anderen wird cl.num lediglich um eins dekrementiert [4].

IV. XCS IN ORGANIC COMPUTING

Das grundlegende Ziel von Organic Computing besteht darin technische Systeme so zu designen, welche Aufgaben selbstständig erledigen können. Hierzu wird ein System mit den sogenannten Self-*Properties (Self-configuration oder auch Self-adaption, Self-organisation, Self-integration, Self-management, Self-healing, Self-protecting, Self-stabilising, Self-improving und Self-explaining) ausgestattet [8]. Häufig wird zum Steuern des SuOC (System under Observation and Control) eine Observer/Controller Architektur gewählt [9]. XCS lässt sich in solch einer Struktur mit einigen Veränderungen als Controller verwenden [4]. Die Anforderungen an ein Learning Classifier System erfüllen dabei die in Organic Computing gewünschten Eigenschaften der Selbstkonfiguration, Selbstintegration, sowie der selbstständigen Verbesserung der Entscheidungen. Die restlichen Anforderungen können als Regeln evolviert werden. Die nötigen Detektoren zum Erkennen eines Fehlers oder einer Gefahr, sowie die Möglichkeiten das System selbständig zu reparieren, bzw. angemessen darauf zu reagieren müssen von Beginn an gegeben sein. XCS kann also auf keine Umweltbedingung reagieren welche von σ nicht abgedeckt wird und auch keine Gegenmaßnahme ausführen welche nicht als Aktion zur Ausführung vorhanden ist.

A. Die Obersver/Controller –Architektur

Abb. 3 zeigt eine schematische Darstellung der Observer/Controller-Architektur. Auf die einzelnen Komponenten Observer, Controller und SuOC wird in den folgenden Abschnitten eingegangen.

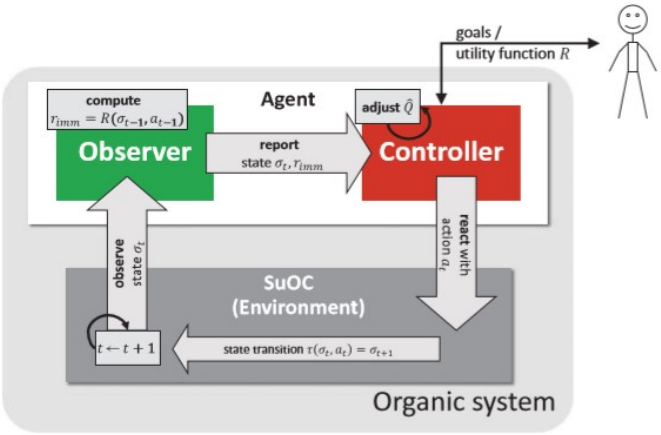

Abb. 3: Eine einfache Darstellung der Observer/Controller-Architektur, entnommen aus [4]

1) System under Observation and Control

Das System under Observation and Control (SuOC) bezeichnet das aktionsausführende, autonome System, bzw. eine Gruppe an autonomen Systemen. Damit es in einem organischen System integriert werden kann, muss sein Verhalten, als auch die Umgebungsbedingungen, beobachtbar sein. Des Weiteren muss seine Performanz sowohl gemessen, als auch während der Laufzeit dynamisch durch Variablen verändert werden können [9].

2) Observer

Der Monitor der Observerkomponente erhält die gesammelten Rohdaten des SuOC und generiert daraus eine Logdatei. Die gesamten Daten eines Zyklus werden an den Pre-processor übergeben. Dieser bereitet die vorhandenen Informationen durch Filtern, Glätten und Extrahieren abgeleiteter Attribute für den Data Analyser, den Predictor und dem Aggregator vor. Der Data Analyser nutzt verschiedene mathematische oder stochastische Methoden wie Cluster Computation oder Emergence Detection, zur Datenextraktion. Daraus wird eine systemweite Beschreibung des aktuellen Zustands erstellt.

Der Predictor erstellt eine Vorhersage zukünftiger Entwicklungen. Verwendete Vorhersagetechniken sind dabei systemabhängig. Der Aggregator bildet aus den Daten des Pre-Processors, dem Data Analyser und dem

Predictor Umgebungsparameter, welche an den Controller übergeben werden[9].

3) Controller

Der Controller ist intern in zwei Schichten aufgeteilt. Er beeinflusst das SuOC anhand der vom Observer erhaltenen Umgebungsparameter.

Die erste Schicht führt ein online-learning von geeigneten Aktionen für die vorliegende Umgebungssituation durch. Sie unterteilt sich in eine Mapping Komponente und einer Performance-Evaluationskomponente. Das Mapping Modul speichert Situation-Aktionspaare und reagiert auf eine vorliegende Situationen. Das Performanz Modul evaluiert die ausgeführte Aktion.

Die zweite Schicht führt eine off-line Optimierung durch. Sie besteht aus einer Adaptions-Komponente und einer Simulations-Komponente. Die Adaptions-Komponente ist für das Erstellen neuer Regeln und dem Löschen ineffizienter Regeln zuständig. Die Simulations-Komponente simuliert eine Aktion vor der tatsächlichen Anwendung und unterstützt damit das Entwickeln neuer Regeln [9].

Vergleicht man die Bestandteile des Controllers in einer Observer/Controller-Architektur mit der eines LCS sind deutliche Parallelen zu erkennen. Das Verwenden des optimierten Extended Classifier System XCS als Controller muss somit eine logische Konsequenz sein, um die Anforderungen an ein organischen Systems zu erfüllen.

B. XCS mit reellen Eingabewerten

Autonome Systeme agieren in der realen Welt. Ihre Umweltsituation lässt sich im Allgemeinen nicht binär abbilden. Daher werden vor der Eingliederung von XCS in die Observer/Controller-Architektur noch Änderungen am Extended Classifier System XCS vorgenommen. Diese ermöglichen dem Learning Classifier System das verarbeiten reeller Eingabewerte. Eine Anpassung der Eingabe-Schnittstelle, des Mutationsoperators und des Coveringverfahrens ist ausreichend. Alle anderen Komponenten werden direkt vom binären XCS übernommen.

1) Die Eingabe-Schnittstelle

Beim binären XCS wurde die Bedingung eines Classifiers als String aus {0, 1, #} dargestellt. Diese Darstellung wird durch eine Konkatenation aus Intervalltupel der Form $int_i = (c_i, s_i)$ ersetzt. Die Prädikate c_i und s_i sind reelle Zahlen. Eine Bedingung entspricht einer Umweltsituation genau dann wenn:

$$c_i - s_i \leq x_i < c_i + s_i, \forall x_i.$$

Das Prädikat c_i entspricht dem Zentrumswert, beziehungsweise den exakten Wert von int_i. Das Prädikat s_i bildet den maximalen Abstand von c_i und ist somit ein Delta-Wert. Es ergibt sich die Struktur $C = (c_0 s_0, \dots, c_n s_n)$ für die Bedingung eines Classifiers [10].

2) Der Mutationsoperator

Der Mutationsoperator arbeitet auf beiden Prädikaten der Intervalltupel. Bei Reellen Zahlen als Eingabe wird ein zufälliger Wert zwischen Null und einem vorher festgelegten Maximalwert m_0 gewählt. Dieser wird mit gleicher Wahrscheinlichkeit zu den Prädikaten Addiert, oder von diesen Subtrahiert. Ein typischer Wert für m_0 ist 0,1. Das Resultat ist eine Mutation durch zufällige Skalierung der Prädikate.

3) Das Covering

Wie im Standard XCS generiert der Covering-Mechanismus neue Regeln wenn die aktuelle Umweltsituation σ von keinem Classifier aus [P] abgedeckt wird. Der Wert des Zentrumprädikates c_i wird für jedes i auf den entsprechenden Wert x_i der vorliegenden Umweltsituation gesetzt. Die Werte für s_1 bis s_n werden auf zufällig generierte Zahlen zwischen Null und s_0 gesetzt. s_0 ist eine Konstante und erhält üblicherweise den Wert 0,5 [10]. Wie in XCS wird ein Classifier für jede mögliche Aktion generiert.

C. XCS-O/C

Um XCS als Controller in organischen Systemen zu verwenden werden die einzelnen Module in die beiden Schichten aufgeteilt. Die Perfomance-Komponente übernimmt die Aufgabe der Mapping-Komponente, die Reinforcement Komponente bildet die Perfomance-Evaluations-Komponente. Gemeinsam bilden sie die erste Schicht des Controllers. Der Covering-Mechanismus bildet zusammen mit einer Evolutionsstrategie die zweite Schicht. Der generische steady-state Nischenalgorithmus wird nicht verwendet [4]. Abbildung 4 zeigt eine schematische Darstellung der als XCS-O/C bezeichneten Architektur.

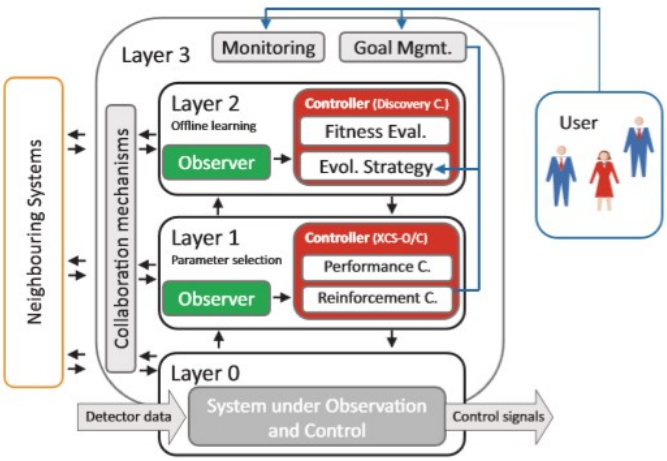

Abbildung 4: XCS integriert in die Observer/Controller-Architektur, entnommen aus [4]

1) Generalisierung ohne generischen Algorithmus

Der genetische steady-state Nischenalgorithmus wurde entfernt um neue Regeln erlernen zu können, ohne das SuOC direkt zu beeinflussen. Neue Regeln werden aus Folge dessen nicht mehr periodisch erstellt sondern ausschließlich durch den Covering-Mechanismus bei Bedarf. Da neue Regeln nicht mehr aus dem Action-Set [A] gebildet werden muss der Covering-Mechanismus für eine Generalisierung der Classifier sorgen. Dazu werden neue Regeln nicht mehr wahrscheinlichkeitsbasiert erstellt,

sondern in der Nähe liegende erweitert, bis sie die vorliegende Umweltsituation mit abdecken. Dies wird als widening Covering-Mechanismus bezeichnet. Da organische, autonome Systeme in der realen Welt agieren spielt der Sicherheitsaspekt eine entscheidende Rolle. Bei dem widening Covering-Prozess erstellte Regeln müssen deswegen vom Systemdesigner vorgegebene Akzeptanzmetriken erfüllen bevor sie zur Population hinzugefügt werden. Diese sind nicht-trivial und stark von der Umgebungsbeschreibung, als auch von der Ertragsfunktion abhängig. Gleichzeitig mit dem Covering-Mechanismus wird die Evolutionsstrategie der zweiten Schicht gestartet. Im Falle dass beide Mechanismen innerhalb der erlaubten Zeitspanne kein Ergebnis liefern sollte mindestens eine Standardregel vorhanden sein. Diese dient als Backup und kann immer als Notlösung ausgeführt werden. Trotz dieser Änderungen führt das Entfernen des genetischen Nischenalgorithmus zu einer niedrigeren Generalisierung im Vergleich zum Standard XCS. Ebenso wird die Fähigkeit auf nicht-deterministische Ereignisse zu reagieren verringert. Das Erkennen von Handlungsbedarf anhand der Fitnessentwicklung liegt im Aufgabenfeld des Systembedieners. Dieser kann durch verändern der Fitnessevaluierung oder der Akzeptanzkriterien Einfluss auf die Generalisierung des Systems nehmen. Es liegt somit in seiner Verantwortung eine ausreichende Generalisierung zu erzielen [4].

2) Die Evolutinonsstrategie

Die Evolutionsstrategie initialisiert als Erstes eine vorbestimmte Anzahl zufälliger Classifier. Auf diese werden, eine vorher festgelegte Generationenanzahl lang, die genetischen Operatoren (Selektion, Mutation und Crossover) angewendet. Der beste resultierende Classifier wird auf das Erfüllen der im Covering ebenfalls verwendeten Akzeptanzmetriken überprüft. Im Falle des Bestehens wird die Regel zu der Population [P] der ersten Schicht hinzugefügt. Insbesondere werden neue Classifier nicht anhand von bestehenden Regeln aus [P] oder [A] erzeugt, wodurch das Erforschen von Nischenbereichen ebenfalls vermindert wird. In allen Fällen unterliegen neu hinzugefügte Regeln vorher festgelegten Akzeptanzkriterien. Ein eventuell Sicherheitskritisches Trial-and-Error ausprobieren wird auf einen angemessenen Bereich beschränkt [4].

LITERARATURVERZEICHNIS

[1] Stewart W. Wilson, “Classifier Systems and the Animat Problem”, Machine Learning 2, Kluwer Academic Publishers, Boston, 1987, pp. 199-228

[2] Andreas Bernauer, “Das Learning Classifier System XCS”, Universität Tübingen, 2007

[3] Jürgen Brake, Moez Mnif, Christian Müller-Schloer, Holger Prothmann, Urban Richter, Fabian Rochner, Hartmut Schmeck, “Organic Computing – Addressing Complexitiy by Controlled Self-organization”, 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, Paphos, 2006

[4] Anthony Stein, “Reaction Learning”, Christian Müller-Schloer, Sven Tomforde, “Organic Computing – Technical Systems for Survival in the Real World”, 2017, pp.287-328

[5] Anthony Stein, Dominik Rauh, Sven Tomforde, Jörg Hähner, “Augmenting the Algorithmic Structure of XCS by Means of Interpolation”, Organic Computing Group, University of Augsburg, 2016

[6] Martin V. Butz, Tim Kovacs, Pier Luca Lanzi, Stewart W. Wilson, “How XCS Evolves Accurate Classifiers”, IlliGal Report No. 2001008, University of Illinois, 2001

[7] Brad L. Miller, David E. Goldberg, “Genetic Algorithms, Tournament Selection, and the Effects of Noise”, University of Illinois, 1995

[8] Sven Tomforde, Bernhard Sick, Christian Müller-Schloer, “Organic Computing in the Spotlight”, Intelligent Embeddes Systems Lab, University of Kassel, 2017

[9] Sven Tomforde, Holger Prothmann, Jürgen Branke, Jörg Hähner, Moez Mnif, Christian Müller-Schloer, Urban Richter, Harmut Schmeck, “Observation and Control of Organic Systems”, Organic Computing — A Paradigm Shift for Complex Systems, Springer, 2011, pp 325-338

[10] Stewart W. Wilson, “Get Real! XCS with Continuous-Valued Inputs”, Pier Luca Lanzi, Wolfgang Stolzmann, Stewart W.Wilson, “Learning Classifier Systems”, Springer, 1999, pp 209-219