Simulation von Wissen -

Konstruktion eines kognitiven Agenten (2)

>> Achtung : Skript gibt den mündlichen Vortrag nur unvollständig wieder !!! <<<

Einführung

Nachdem die Entscheidung gefällt wurde, die weitere Entwicklung eines *kognitiven Agenten* (CA, CognitiveAgent) im Rahmen des durch das *Planet Earth Simulator Projekt* vorgegebenen Rahmen vorzunehmen (siehe Schaubild), und ferner das dazu definierte Projekt *SimWorld + SimCognitiveAgent* in zwei Teilprojekte aufgespalten worden ist, von dem das Teilprojekt *SimWorld* an eine andere Gruppe ausgelagert wurde, verbleibt nun als Teilprojekt das Projekt *SimCognitiveAgent* als zu lösende Aufgabe.

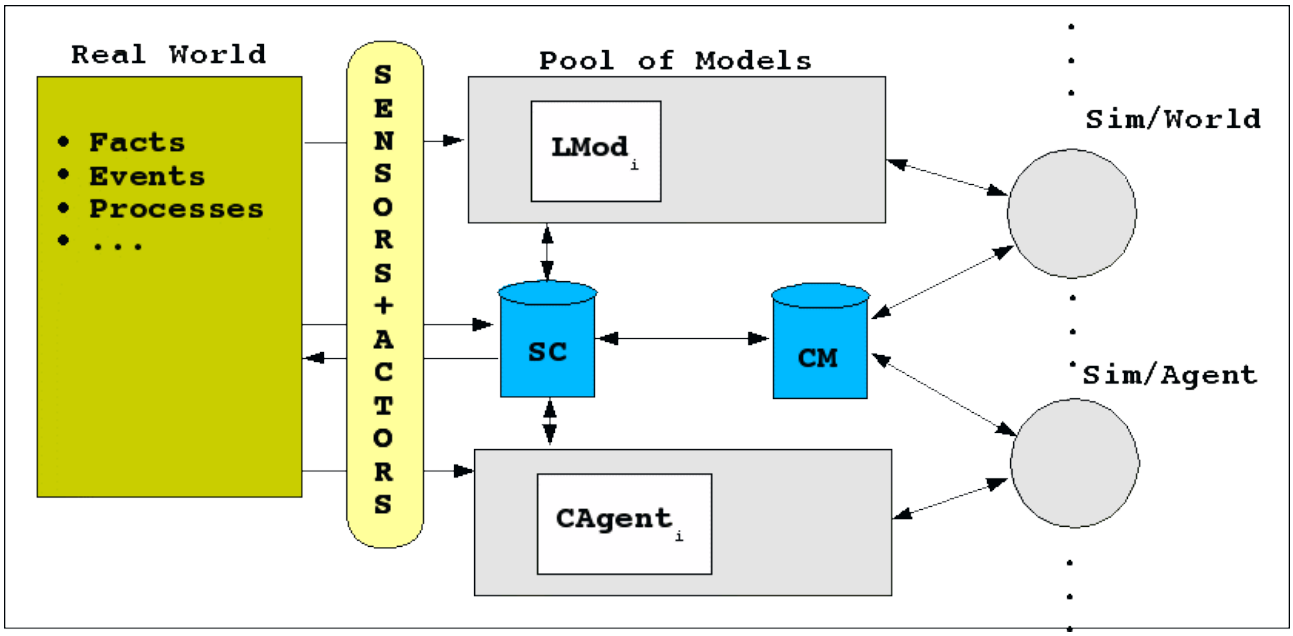

Bild: Erweitertes Planet Earth Simulator Projekt

Teilprojekt *SimCognitiveAgent*

Zur Schnittstelle zwischen *SimCognitiveAgent* und *SimWorld*

Die Ausgangslage für das Teilprojekt SimCognitiveAgent ist im nachfolgenden Schaubild festgehalten:

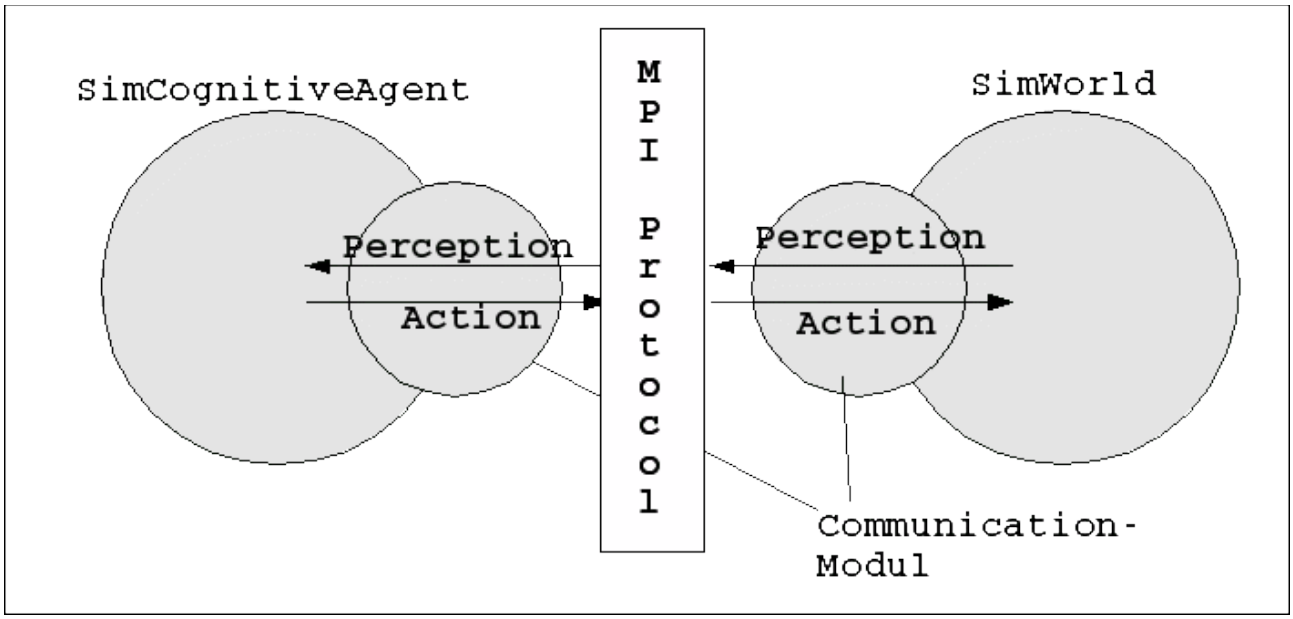

Bild: SimCognitiveagent mit Simworld

Die Wahrnehmung eines kognitiven Agenten --realisiert als CognitiveAgent-- ist an eine bestimmte simulierte Welt --Simworld-- gebunden. Entsprechend werden auch alle Aktionen des kognitiven Agenten zu der Welt geleitet, aus der die Wahrnehmungen stammen.

Die technische Realisierung dieser Kommunikation läuft --wie auch im Falle des Teilprojektes *SimWorld*-- über das *MPI-Protokoll* in der Realisierung von *mpich* oder *lam*. Für den Programmierer eines CognitiveAgents heisst dies, dass er C++-Strings senden und empfangen kann. Das Format dieser Strings regelt das *SimCognitiveAgent-Protocol*.

Für die Festlegung der Entwicklung eines CognitiveAgent soll die *Roadmap* für CognitiveAgents dienen (siehe nachfolgende Tabelle). Dabei wird Phase 1 übersprungen und gleich Phase 2

anvisiert.

In Phase 2 der Roadmap wird angenommen, dass die **Welt** aus *Objekten* besteht , die *Positionen* in einer *Fläche* einnehmen; jedes Objekt kann *Eigenschaften* haben. Vom **kognitiven Agenten** wird angenommen, dass seine *Bewegungen zufällig* sind ; Objekte für *Nahrungsaufnahme* können *erkannt* und *konsumiert* werden; Energieverlust unter einem Schwellwert führt zum *Tod. Gedächtnisinhalte* sollen noch nicht vorhanden sein.

Phase	***Sim/Welt***	***Sim/CAgent***	***Memory von Sim/CAgent***
1	Einfaches Echo	Kann Nachrichten senden und empfangen	Leer
2	Die Welt besteht aus *Objekten*, die *Positionen* in einer *Fläche* einnehmen; jedes Objekt kann *Eigenschaften* haben	*Bewegungen* sind *zufällig*; Objekte für *Nahrungsaufnahme* können *erkannt* und *konsumiert* werden; Energieverlust unter einem Schwellwert führt zum *Tod*	Leer
		Bewegungen können *Gedächtnisinformtionen nutzen*	*Imitierendes passives* Gedächtnis für Positionen und Objekte an diesen Positionen
			Imitierendes und *aktiv expandierendes* Gedächtnis; Unterscheidung von '*registriert*' und'*konstruiert*'

Diese groben Angaben sind offensichtlich durch eine Reihe weiterer Annahmen zu ergänzen bzw. zu präzisieren. Dabei ist zu beachten, dass es bis zu einem gewissen Grade eine *Korrespondenz* gibt zwischen den Eigenschaften der Welt und den kognitiven Fähigkeiten des kognitiven Agenten:

Nr.	*SimCognitiveAgent*	*SimWorld*
1	Sensorische Inhalte gepackt in einen String	1. Packen eines Strings 2. Setzt Wissen voraus, welche Inhalte relevant sind 3. Setzt Wissen voraus, welche Sensoren mit welchem Format bedient werden müssen 4. Setzt Wissen voraus, an welcher Position in der Welt sich die Sensoren befinden und wie sie angeordnet sind 5. Setzt Wissen voraus, wann und wie sich Eigenschaften und Anordnung der Sensoren ändern können

Im Fall der sensorischen Inhalte wird deutlich, dass die SimWorld sehr viel ' wissen' muss darüber, welche sensorischen Inhalte unter welchen Bedingungen von den *Sensoren* des CognitiveAgent übermittelt werden. D.h. man benötigt eine *standardiesierte Beschreibung* eines *Agentenkörpers*, der mindestens eine Beschreibung seiner *Sensorik* beinhaltet. Diese Beschreibung koennte etwa eine Struktur sein der Art:

```
struct AgentBody {
  sensor_visual vs1;
  sensor_acoustical ac1, ac2;
  ...
}

struct sensor_visual {
  ...
}
```

usw.

Entsprechend ergeben sich auch zahlreiche Anforderungen aus der Tatsache von Aktionen:

Nr.	*SimCognitiveAgent*	*SimWorld*
2	Aktionen gepackt in einen String	6. Entpacken eines Strings 7. Setzt Wissen voraus, welche Pattern mit welchen Aktionen in der Welt zu verknüpfen sind 8. Setzt Wissen voraus, welche Aktionen mit welchen Vorgängen in der Welt zu verknüpfen sind 9. Setzt Wissen voraus, welche Vorgänge in der Welt 'kompatibel' sind und welche nicht

Was immer auch in einem Aktions-String kodiert werden mag, SimWorld muss diese Pattern dekodieren in dem Sinne, dass geklärt werden muss, welche Aktion gemeint ist und es muss für jede Form von Aktion feststehen, welche *Wirkung* diese Aktion in der Welt besitzt, d.h. welche *Vorgänge* werden durch eine Aktion ausgelöst.

Man benötigt also eine Art Liste, in der für jede mögliche Aktion und deren Parameter festgelegt wird, was dies in einer bestimmten Welt bedeuten soll.

Zum Testen kann man sich natürlich eine '*DummySimWorld*' bauen, in der man per Hand alle Perzeptionen und Aktionen simulieren kann.

Zur Struktur des SimCognitiveAgent Nr.1

Nimmt man also jetzt an, dass Sinneseindrücke wie auch Aktionen in Form von Strings kodiert werden, stellt sich jetzt die Frage, welche weitere Struktur einen CognitiveAgent als SimCognitiveAgent auszeichnet.

Wir werden so vorgehen, dass wir zunächst eine möglichst einfache Struktur vorgeben und dann die Frage stellen, inwieweit man an diesem Beispiel sowohl schwaches wie auch starkes Lernen einführen könnte.

Bezüglich des Gedächtnisses ('memory') wird angenommen, dass dieses zu Beginn keine Funktion hat. Es gibt aber zwei Systemzustände, die von Bedeutung sind: die *Energiebilanz ('energy budget')* sowie die (innere) *Uhr ('clock').* Die Uhr steht für ein periodisches Ereignis, das zur Steuerung von Prozessen genutzt werden kann. Ein solcher Prozess ist die Veränderung der Energiebilanz. Die *Energiebilanz* vermindert sich in Abhängigkeit von der Zeit ('energy--') wie auch in Abhängigkeit von Bewegungen. Erhöht werden kann sie nur ('energy++') durch

Nahrungsaufnahme, sprich durch eine Aktion *essen ('eat')*

Bezüglich der möglichen *Aktionen ('actions')* wird angenommen, dass es nur zwei Aktionen zu Beginn gibt: (i) +/- *essen ('eat')* sowie +/- *bewegen ('move').* Sofern Bewegung stattfinden soll muss noch entschieden werden, in welche *Richtung ('direction')* bewegt werden soll und wie gross die *Distanz ('distance')* sein soll, die zurückgelegt wird.

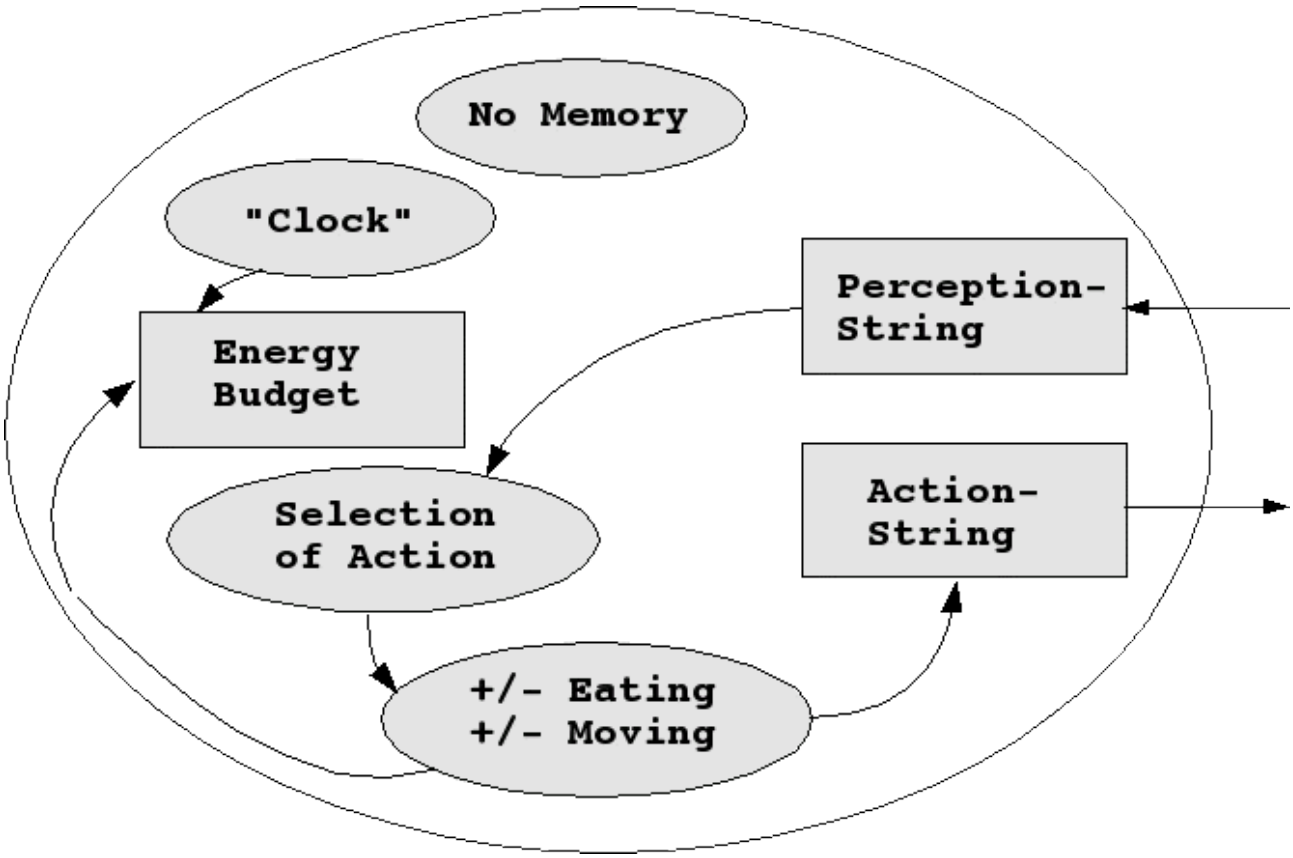

Bild: Minimale Struktur eines CognitiveAgent

Sofern etwas *'Essbares' ('eatable')* wahrgenommen wird, wird die Aktion essen ausgeführt; falls nichts Essbares wahrgenommen wird, entscheidet der Zufall, ob eine Bewegung ausgeführt werden soll oder nicht. Soll eine Bewegung ausgeführt werden, entscheidet wieder der Zufall, welche der möglichen Richtungen gewählt wird. Der Betrag der Wegstrecke wird standardmässig mit '1' angenommen (unterstellt man eine einfache 2-dimensionale Kästchenwelt, dann würde dies bedeuten, dass der Agent sich 1 'Kästchen' weiter bewegt) (siehe Aktivitätsdiagramm im folgenden Schaubild).

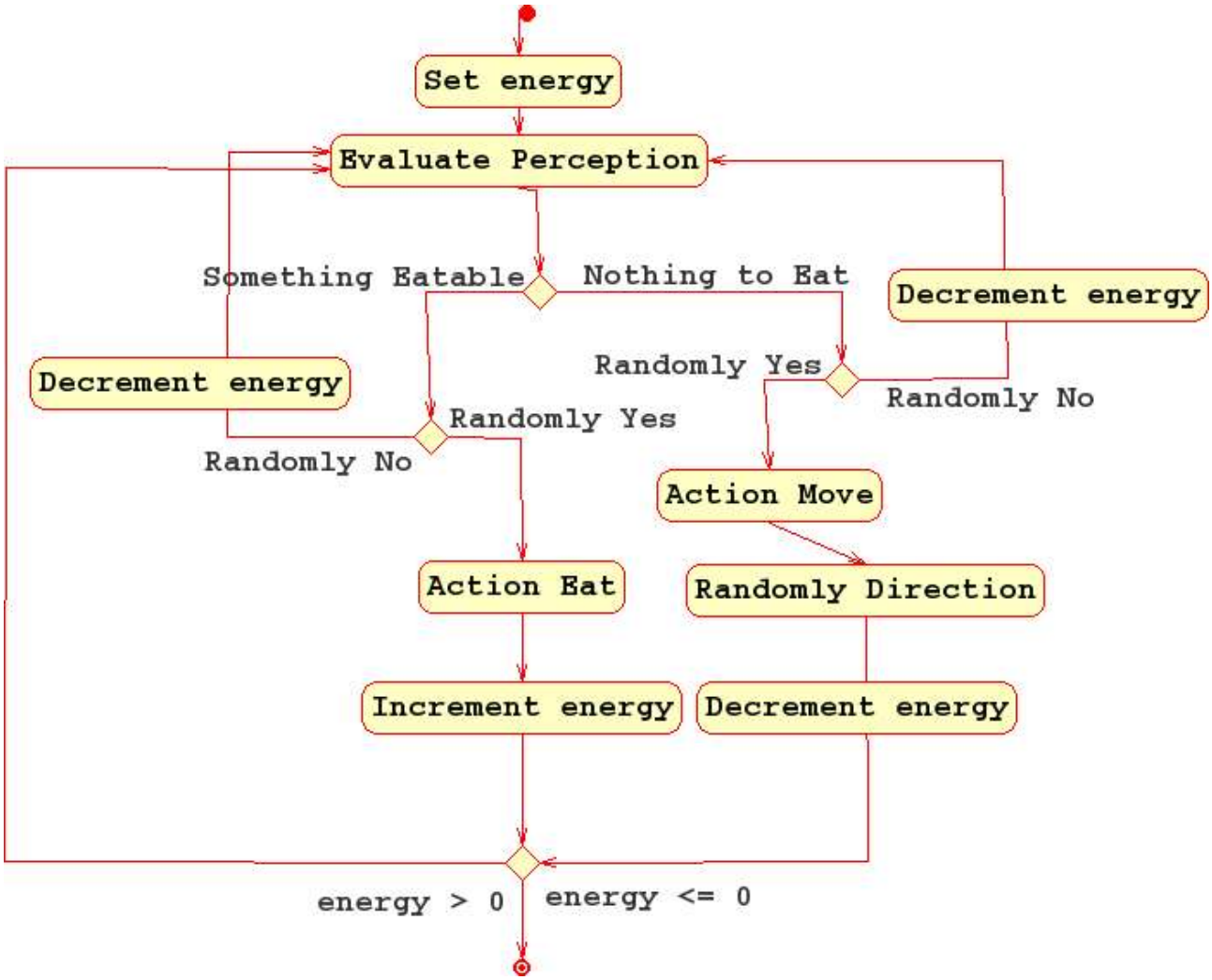

Bild: Aktivitätsdiagramm zum CognitiveAgent Nr.1

Zur Implementierung des SimCognitiveAgent Nr.1

Prinzipiell könnte man den Cognitiveagent Nr.1 mit jeder beliebigen Programmiersprache programmieren. Es soll hier aber überlegt werden, inwieweit das theoretische Interesse daran, unterschiedliche Formen des Lernens zu untersuchen, bestimmte Programmierweisen favorisieren oder nicht.

Im Falle des *schwachen Lernens* kann man davon ausgehen, dass die generelle Lernfunktion *fixiert* ist; nur die Struktur des Gedächtnisses kann sich ändern.

Im Falle des *starken Lernens* kann sich neben dem Gedächtnis auch die Struktur der generellen Lernfunktion ändern. Im letzteren Fall wäre es günstig, wenn die generelle Lernfunktion so kodiert wäre, dass eine Meta-Lernfunktion --z.B. ein genetischer Algorithmus-- die generelle Lernfunktion als ' Input' bekommen könnte, um diese Funktion verändert als ' Output' wieder ausgeben zu können.

Letztere Überlegung würde bedeuten, dass die generelle Lernfunktion als ein *Anweisungstext (:= Programm, Algorithmus)* --dies könnte auch eine Datenstruktur sein-- zu verstehen ist, der von einem *Prozessor (:= Interpreter)* 'abgearbeitet' würde. Der Anweisungstext wäre dann einfach ein *Text* bzw. eine *Zeichenkette*, die von dem genetischen Algorithmus verändert werden könnte.

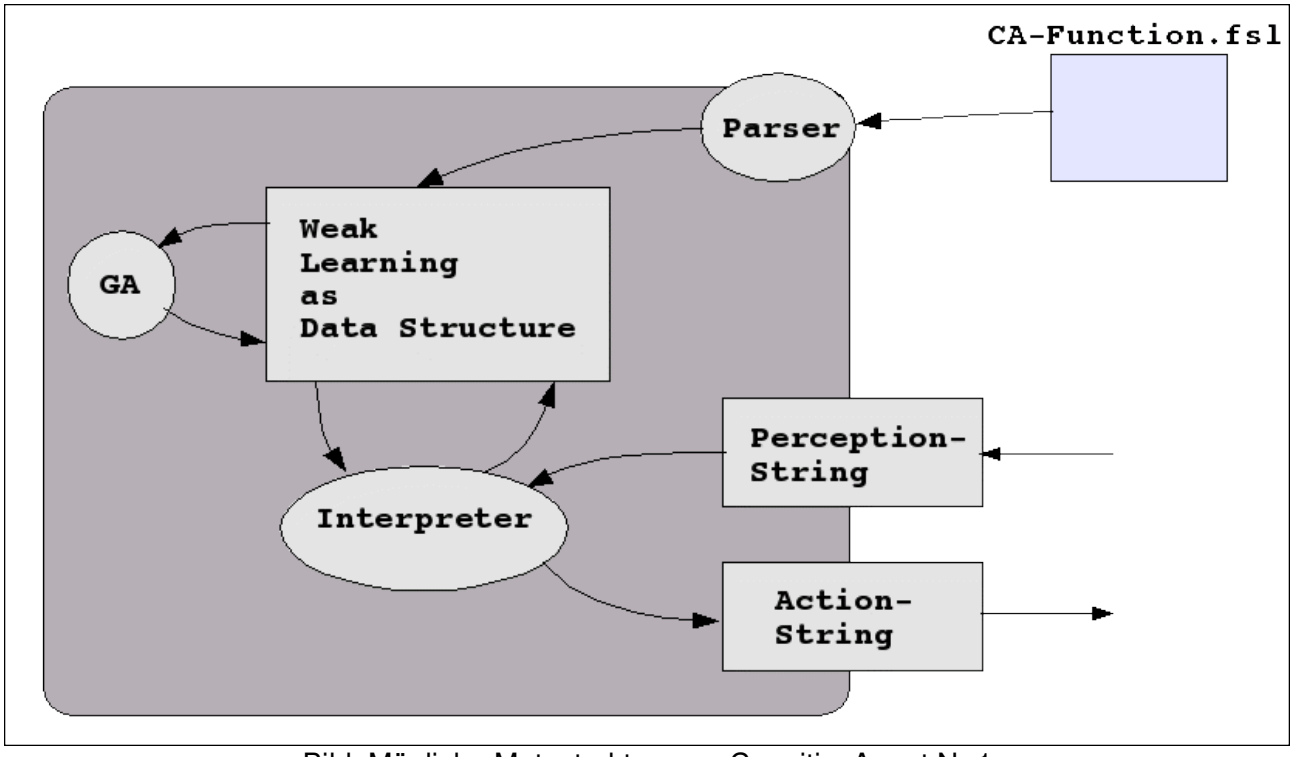

Bild: Mögliche Metastruktur zum CognitiveAgent Nr.1

Mit diesen Überlegungen rückt die Struktur des simCognitiveAgent wieder in die Nähe der Struktur einer Simworld, d.h. eine Beschreibungsdatei --z.B. eine .fsl-Datei-- enthält die Struktur der schwachen Lernfunktion eines SimCognitiveAgent. Diese Beschreibungsdatei wird von einem *Parser* eingelesen und in eine geeignete *Datenstruktur* umgesetzt. Die Grundstruktur dieser Datenstruktur besteht aus Systemen, die miteinander vernetzt sind. Diese Systeme sind vielfach interpretierbar; ein System kann ein *Neuron* repräsentieren; dann entspräche die Datenstruktur einer *neuronalen Struktur,* also einem möglichen *Gehirn*. Ein System kann aber auch eine *Regel* interpretieren; dann entspräche die Menge der Systeme einem *Expertensystem*. Beide Optionen

sollen hier momentan zugelassen werden.

Entscheidend ist hier nur, dass es einen 'Prozessor', einen *Interpreter* gibt, der diese Datenstruktur interpretierend abarbeiten kann und unter Berücksichtigung aktueller sensorischer Daten (:= Input) einen aktuellen Output in Form von Aktionen berechnet.

Ferner gibt es eine Funktion GA --die man als *genetischen Algorithmus* konzipieren könnte--, die ebenfalls auf der Datenstruktur operieren kann, und zwar in dem Sinne, dass die Funktion GA die Datenstruktur abändern darf (Insertion, Deletion, Change of Values...).

Es fragt sich dann, wie man die beiden Funktionen *Interpreter* und *GA* implementieren soll. Folgende Strategien bieten sich an: (i) klassisch *prozedural* (etwa mit 'C'), (ii) klassisch *objektorientiert* (z.B. mit C++) oder (iii) klassisch *regelbasiert* (etwa mit Prolog oder CLIPSE).

Prinzipiell sind alle drei Varianten möglich und aus Sicht der Berechenbarkeit gleichwertig. Die Frage ist, welche der drei Varianten ist *intuitiver* zu handhaben und welche ist *schneller*. Für die Zukunftwäre auch noch die Realzeittauglichkeit zu bedenken).

Hier sollen speziell die beiden Varianten (ii) mit C++ sowie (iii) mit CLIPS näher untersucht werden.