Aula 8 – Clarion

Introdução:

CLARION (**C**onnectionist **L**earning with **A**daptive **R**ule **I**nduction **On**-Line) é uma arquitetura cognitiva desenvolvida por pesquisadores do *Rensselaer Polytechnic Institute*. Seus idealizadores afirmam que a arquitetura tem como objetivo "investigar as estruturas fundamentais da mente humana através da sintetização de várias idéias intelectuais em um modelo de cognição coerente e unificado".

O objetivo desta atividade é estudar esta arquitetura através dos tutoriais fornecidos por seus desenvoledores.

A versão mais recente do ***CLARION*** utiliza a *C#* .NET portanto a execução dos tutoriais requer um mínimo de conhecimento de programação orientada a objetos. Detalhes a respeito da linguagem e paradigma não serão documentados na atividade que tem como foco a arquitetura ***Clarion*** em si.

Os tutoriais são relativamente breves e servem para explicar o funcionamento e aplicação de conceitos fundamentais da arquitetura. O foco deste trablalho será documentar novos conceitos conforme surgirem e o funcionamento e motivo de cada um deles na forma como for introduzido nos codigos dos tutoriais.

Objetos Descritivos x Objetos Funcionais

O primeiro conceito relativo à arquitetura apresentado no tutorial é a noção e destinção entre objetos descritivos e funcionais. Segundo a documentação, um **objeto descritivo** pode ser compreendido como um objeto que tem como propósito, como o próprio nome diz, descrever algo. Os exemplos de objetos descritivos fornecidos são pares dimensão-valor, agentes e os chamados ***chuncks***, que podem ser ações e objetivos por exemplo.

A princípio este conceito não parece muito intuitivo uma vez que espera-se que o agente seja muito mais do que uma simples descrição de algo. Contudo, a noção fica menos nebulosa com o conceito complementar de objeto funcionais. Um **objeto funcional** tem a capacidade de realizar tarefas específicas. Exemplos são redes de decisão, modulos meta-cognitivos e drives.

O Mundo

Na arquitetura, existe um objeto de fundamental importância: a instância singleton da classe **World.** O objeto por si só não possui propriedades ou métodos públicos, mas a classe possui diversos métodos estáticos visíveis que trabalham sobre ele exclusivamenete. O mundo basicamente contém todos os objetos descritivos e as formas para acessá-los e cria-los. Os métodos para criação de objetos descritivos são:

```
World.NewAgent
World.NewDeclarativeChunk
World.NewDimensionValuePair
World.NewDistributedDimensionValuePair
World.NewExternalActionChunk
World.NewGoalChunk
World.NewGoalStructureUpdateActionChunk
World.NewNACSReasoningActionChunk
World.NewNACSRetrieveActionChunk
World.NewParameterChangeActionChunk
World.NewSensoryInformation
World.NewWorkingMemoryUpdateActionChunk
```

Tutorial 1 – Configurando e utilizando um ACS

O objetivo deste tutorial é criar um agente que tem a capacidade de responder 'Hello' ou 'Goodbye' dependendo do input fornecido (saudações 'Hello' ou 'Goodbye' respectivamente). O agente utilizará uma rede neural com aprendizado baseado em reforço para aprender a resposta correta para cada saudação.

Para isso, é necessário:

1- Definir o que existe no mundo que pode ser percebido pelo agente. Neste caso, as saudações 'hello' e 'goodbye'.

2- Criar um agente
3- Definir uma rede de decisões para o agente que tenha capacidade de perceber os elementos do mundo e fornecer uma resposta ao que for percebido.
4- Repetidamente fornecer saudações para o agente, verificar a resposta dada e fornecer um feedback para o aprendizado reforçado.

Definindo objetos no mundo

Para nosso objetivo, precisamos definir as saudações que o agente perceberá e as ações que pode tomar dada uma saudação.

Clarion possui um importante conceito de par dimensão-valor que é útil para definir a existência de objetos em diferentes dimensões do mundo. Utilizaremos estes para definir as saudações:

```
DimensionValuePair hi = World.NewDimensionValuePair("Salutation", "Hello");
DimensionValuePair bye = World.NewDimensionValuePair("Salutation", "Goodbye");
```

Estes pares são objetos descritivos do mundo e podem ser percebidos por um agente existente no mesmo, como veremos adiante.

Para definir as ações que um agente pode tomar, utilizaremos a classe ExternalActionChunk, que pode ser utilizada pela rede neural (definida mais adiante) como output.

```
ExternalActionChunk sayHi = World.NewExternalActionChunk("Hello");
ExternalActionChunk sayBye = World.NewExternalActionChunk("Goodbye");
```

Criando um Agente

No contexto de Clarion, assim como todos os outros objetos descritivos um agente faz parte do singleton mundo (World). Para criar um novo agente inserido no mundo, utiliza-se o método estático World.NewAgent. Este método possui diversos parâmetros, mas para este agente simples forneceremos apenas o nome do agente:

```
Agent John = World.NewAgent("John");
```

Após inicializar o agente poderíamos anexar a ele diversos objetos funcionais para torna-lo mais complexo e realizar as tarefas necessárias. Neste exemplo, queremos apenas adicionar uma das várias redes neurais que a biblioteca de Clarion fornece para que o agente aprenda e forneça respostas aos estímulos (saudações) percebidos.

```
SimplifiedQBPNetwork net = AgentInitializer.InitializeImplicitDecisionNetwork(John, SimplifiedQBPNetwork.Factory);
```

A classe SimplifiedQBPNetwork representa uma rede do tipo '**Q**-learning **B**ack**p**ropagation **Network**', onde Q-learning é uma técnica de aprendizado por reforço no contexto de aprendizado de máquina.

Com a rede neural em mãos, podemos definir as entradas e saídas da mesma.

```
net.Input.Add(hi);
net.Input.Add(bye);
net.Output.Add(sayHi);
net.Output.Add(sayBye);
```

O objeto funcional **net** está agora pronto para ser utilizado pelo agente. Contudo, o agente só passará a utilizar este objeto depois que sinalizarmos que ele está de fato pronto. Quando isso acontece, o objeto se torna imutável (a menos de seus parâmetros de afinação) por isso deve ser completamente definido antes de ser utilizado. Em Clarion, isso é feito utilizando o seguinte método do agente:

```
John.Commit(net);
```

Percepção, ação e feedback

Agora que temos o agente criado e os elementos do mundo definidos, podemos iniciar um loop de percepção, ação e feedback para John. Para isso, fornecemos a John um informação sensorial criada da seguinte forma:

```
SensoryInformation si = World.NewSensoryInformation(John);
```

John é passado como parâmetro para o método de criação de informação sensorial porque o mundo é percebido de diferentes formas para cada agente e porque existe uma relação entre a informação sensorial com o estado interno do agente que a percebe (memória, drives, objetivos).

Para definir uma informação sensorial, inserimos objetos descritivos na mesma. Neste caso, queremos inserir uma das saudações aleatoriamente com 50% de chance de cada uma ocorrer a cada ciclo. Adicionamos ambas as informações mas apenas a escolhida terá um valor de ativação alto.

```
Random rand = new Random();

if (rand.NextDouble() < .5)
{
     si.Add(hi, John.Parameters.MAX_ACTIVATION);
     si.Add(bye, John.Parameters.MIN_ACTIVATION);
}
else
{
     si.Add(hi, John.Parameters.MIN_ACTIVATION);
     si.Add(bye, John.Parameters.MAX_ACTIVATION);
}
```

Com a informação sensorial criada, John pode percebe-la e escolher uma ação de acordo com seu estado interno.

```
John.Perceive(si);
ExternalActionChunk chosen = John.GetChosenExternalAction(si);
```

Agora que sabemos a resposta de John para a informação sensorial fornecida, podemos dar feedback a fim de afinar a rede neural implícita que criamos para o agente. O valor numérico (padrão) do feedback que podemos fornecer varia de 0 a 1, onde 0 representa o feedback mais negativo e 1 o mais positivo, tendo 0.5 como um feedback aproximadamente neutro. Analisamos então o objeto ***chosen*** e verificamos se ele corresponde ao valor correto dada a informação sensorial que utilizamos (a que tem valor de ativação máximo). Se o agente acertou, fornecemos feedback positivo e caso contrário negativo.

```
if (chosen == sayHi)
{
    if (si[hi] == John.Parameters.MAX_ACTIVATION)
    {
        John.ReceiveFeedback(si, 1.0);
    }
    else
    {
        John.ReceiveFeedback(si, 0.0);
    }
}
else
{
    if (si[bye] == John.Parameters.MAX_ACTIVATION)
    {
        John.ReceiveFeedback(si, 1.0);
    }
    else
    {
        John.ReceiveFeedback(si, 0.0);
    }
}
```

Fazendo estes passos dentro de um loop de execução, o agente irá eventualmente aprender qual resposta deve ser dada para cada saudação. Mesmo que não acerte 100% das vezes, depois de cem mil execuções aproximadamente apenas uma em mil aproximação é incorreta.

Tutorial 2 – Configurando e utilizando a estrutura de objetivos

Este tutorial explica como inicializar uma estrutura de objetivos e criar ***chuncks*** que realizam ações sobre estes objetivos. Pouca informação a respeito de como utilizar na prática estas estruturas são fornecidas, mas a forma básica de criação e inicialização será abordada. Nos tutoriais seguintes será abordada uma aplicação prática destas estruturas

A estrutura de objetivos

A estrutura de objetivos de um agente é uma coleção que contém ***chuncks*** de objetivos: objetos da classe ***GoalChunk***. Podemos acessar esta coleção diretamente através do agente, para isso utilizamos o método GetInternals da classe Agent e o enumerador Agent.InternalWorldObjectContainers da seguinte forma:

```
IEnumerable<GoalChunk> gsContents = (IEnumerable<GoalChunk>) John.GetInternals(Agent.InternalWorldObjectContainers.GOAL_STRUCTURE)
```

Podemos também recuperar do agente o objetivo atual do agente:

```
GoalChunk g = World.NewGoalChunk();
```

Criando e afinando um GoalChunk

Sendo um objeto descritivo, uma instância da classe GoalChunk deve ser criada através do singleton World, da seguinte forma:

```
GoalChunk g = World.NewGoalChunk();
```

Para determinar como os GoalChunk de um agente se comportam precisamos alterar configurações no subsistema motivacional do agente em questão e não as instâncias de GoalChunk em si. Duas configurações podem ser feitas:

1) Comportamento da coleção de objetivos (estrutura de objetivos): pilha ou lista

```
John.MS.Parameters.GOAL_STRUCTURE_BEHAVIOR_OPTION = MotivationalSubsystem.GoalStructureBehaviorOptions.STACK;
John.MS.Parameters.GOAL_STRUCTURE_BEHAVIOR_OPTION = MotivationalSubsystem.GoalStructureBehaviorOptions.LIST;
```

2) Nível de ativação do GoalChunk atual do agente: configurada na inicialização do GoalChunk ou ativação total.

```
John.MS.Parameters.CURRENT_GOAL_ACTIVATION_OPTION = MotivationalSubsystem.CurrentGoalActivationOptions.FULL;
John.MS.Parameters.CURRENT_GOAL_ACTIVATION_OPTION = MotivationalSubsystem.CurrentGoalActivationOptions.ACTUAL;
```

Adicionando e removendo um objetivo manualmente

Podemos simplesmente adicionar manualmente um objetivo (GoalChunk) ***g*** à estrutura de objetivos de um agente da seguinte forma:

```
John.SetGoal(g, 1);
```

Onde o segundo parâmetro é o nível de ativação deste objetivo que deve variar de 0 a 1.

Para remover um objetivo (GoalChunk) ***g*** da estrutura, utilizamos o método ResetGoal

```
John.ResetGoal(g);
```

Contudo, estes métodos manuais dificilmente serviriam para problemas mais complexos. Uma alternativa à adição/remoção manual de um objetivo é a utilização das ações de objetivo (goal actions) da arquitetura. Este conceito é representado pela biblioteca através da classe GoalStructureUpdateActionChunk. A interpretação para esta nomenclatura não tão intuitiva deve ser algo como "Action Chunck de atualização de estrutura de objetivo". Cada um destes *chuncks* pode possuir mais de uma ação, que corresponde a uma operação sobre um determinado GoalChunk **g**. As ações possíveis são adição de GoalChunk (SET), remoção de GoalChunk específico (RESET), remoção de todos os GoalChunk (RESET_ALL) e remoção de todos os GoalChunk seguida de adição (SET_RESET).

```
GoalStructureUpdateActionChunk gAct = World.NewGoalStructureUpdateActionChunk();
gAct.Add(GoalStructure.RecognizedActions.SET, g);
```

Se adicionarmos um destes *chuncks* na camada de saída de um componente como de uma rede neural **net** no ACS, as ações adicionadas serão executadas quando o chunk for selecionado.

```
net.Output.Add(gAct);
```

Tutorial 3 – Intermediate ACS Setup

Refinamento de parâmetros para execução de tarefas

Para tarefas específicas, configurações diferentes da padrão serão necessárias. Para isso, os diversos componentes da biblioteca Clarion podem ser refinados. É possível alterar parâmetros locais e globais dos componentes, como veremos a seguir.

Alterando parâmetros globalmente

Classes que possuem parâmetros globais os têm expostos em variáveis estáticas com nome GlobalParameters (por design da arquitetura). Portanto, se querermos alterar parâmetros globais de uma classe devemos buscar por esta variável estática e modificar suas propriedades de acordo com o desejado.

Estas propriedades são utilizadas durante a criação de um componente. Quando um componente utiliza um destes parâmetros globais, ele cria uma cópia local do parâmetro global utilizando o valor deste no momento da inicialização. Se modificarmos um parâmetro global, os componentes criados antes da modificação não serão alterados. Apenas componentes que forem criados após a modificação refletirão a mudança. Abaixo alguns exemplos de modificação de parâmetros globais:

```
ActionCenteredSubsystem.GlobalParameters.PERFORM_RER_REFINEMENT = false;
RefineableActionRule.GlobalParameters.GENERALIZATION_THRESHOLD_1 = -.2;
```

Outra característica importante a respeito de parâmetros globais é a relação de herança entre classes. Se uma classe A possui diversas classes herdeiras, se modificarmos algum de seus parâmetros globais ele será refletido em todas estas classes. Se, contudo, quisermos modificar apenas uma de suas classes herdeiras, podemos fazê-lo modificando o mesmo parâmetro utilizando a classe herdeira.

Alterando parâmetros localmente

Mesmo podendo modificar parâmetros em qualquer ponto da hierarquia, é necessário que possamos alterar parâmetros de instâncias específicas de cada classe. Para isso, utilizamos o atributo ***Parameters*** da instância. Além disso, vale notar que independentemente do momento em que estes parâmetros locais sofrerem mudanças, elas serão refletidas no comportamento do agente. Estes parâmetros não fazem parte da imutabilidade que um componente sofre após um ‘commit’, portanto podem ser modificadas a qualquer momento.

Memória de trabalho

A memória de trabalho no contexto do Clarion é uma coleção de ***chuncks***. Quaisquer tipos de chuncks podem ser adicionados à memória de trabalho.

Podemos acessar a coleção de chuncks da memória de trabalho da mesma forma que faríamos com objetivos:

```
IEnumerable<Chunk> wmContents = (IEnumerable<Chunk>) John.GetInternals(Agent.InternalWorldObjectContainers.WORKING_MEMORY);
```

Assim como na estrutura de objetivos, é possível adicionar ou remover chuncks da memória de trabalho manualmente ou através de ações de forma muito similar ao que foi feito na estrutura de objetivos. Para a adição manual, passamos como parâmetros o chunck **ch** a ser adicionado e seu valor de ativação:

```
John.SetWMChunk(ch, 1);
```

Para a remoção manual passamos apenas o chunck ch que desejamos remover:

```
John.ResetWMChunk(ch);
```

Para aplicar ações, utilizamos action chuncks específicos para a Memória de Trabalho assim como fizemos com aqueles específicos para objetivos, definindo a operação a ser realizada e o chunck **ch**.

```
WorkingMemoryUpdateActionChunk wmAct = World.NewWorkingMemoryUpdateActionChunk();
wmAct.Add(WorkingMemory.RecognizedActions.SET, ch);
```

O enumerador RecognizedActions possui as mesmas ações que o enumerador para os objetivos possui.