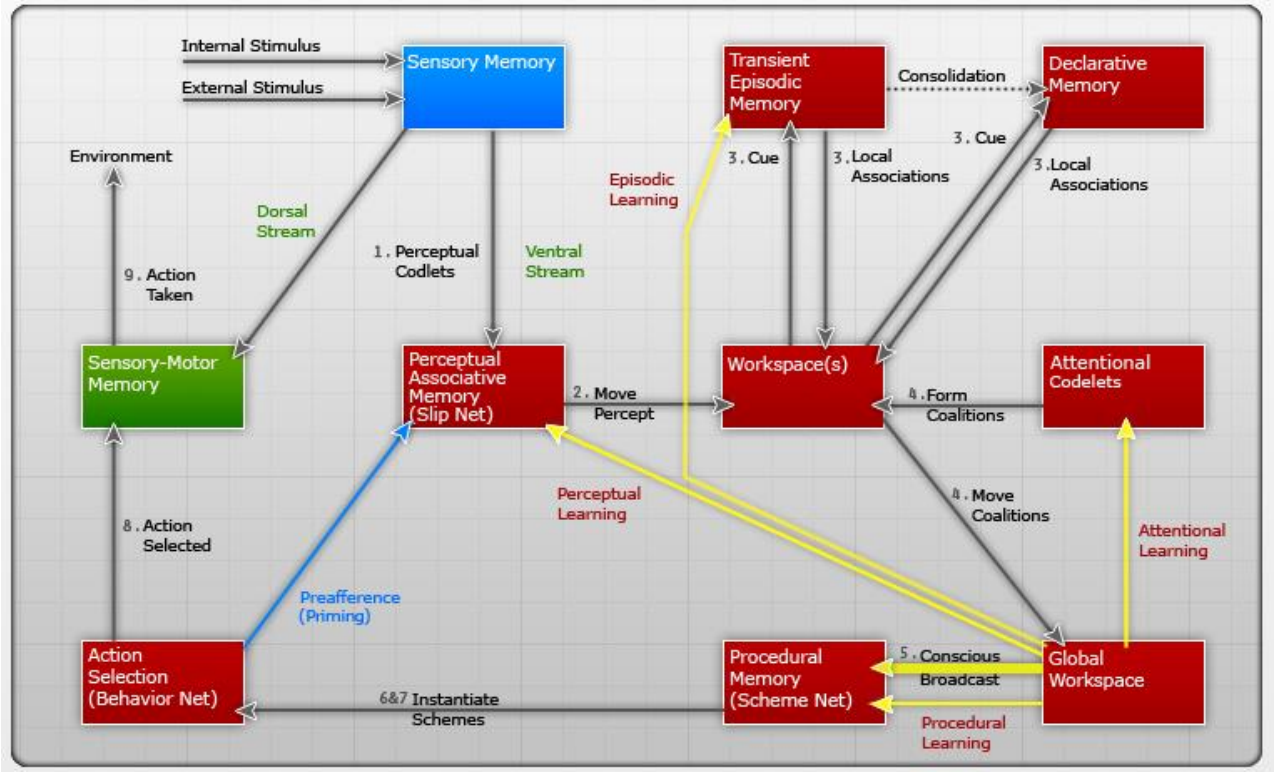
Aula 12

Atividade 1

O tutorial completo é muito longo, com 8 seções e diversos slides. Optei pelo estudo focado no módulo do Ciclo Cognitivo como recomendado. Links:

- **Tutorial completo**: http://ccrg.cs.memphis.edu/tutorial/tutorial.html
- **Ciclo Cognitivo:** http://ccrg.cs.memphis.edu/tutorial/synopsis.html



IDA (Intelligent Distribution Agent) foi desenvolvida para a marina americana para auxiliar na alocação de recursos humanos. LIDA (Learning IDA) é uma arquitetura que nasceu da IDA com a adição de diversos mecanismos. O ciclo cognitivo do LIDA pode ser subdividido em tres fases:

- **Fase de compreensão:** estímulos ativam detectores de baixo nivel na memorial sensorial (Sensorial Memory - SM). A saída deles afetam memória perceptual associativa (Perceptual Associative Memory - PAM) onde detectores de alto nível alimentam entidades abstratas como objetos, categorias, ações, eventos e etc. Os resultados são movidos para a área de trabalho (Workspace - W) onde produzem associações locais na memória episódica (Transient Episodic Memory - TEM) e na memória declarativa (Declarative Memory - DM). Essas associações são combinadas para gerar um modelo situacional que é a representação “do que está acontecendo agora”.
- **Fase de atenção (consciente):** Começa com a formação de coalizões no modelo situacional atual, que compete por atenção, que forma os elementos conscientes. Os selecionados fazem o *broadcast* global iniciando a fase de aprendizado e seleção. Novas entidades e associações, além do reforço das existentes, ocorrem de várias formas nas memórias.

- **Fase de seleção e aprendizado:** Em paralelo com tudo isso, planos de ações possiveis são instanciados da Memória Procedural (Procedural Memory - PM) e enviados para a seleção onde competem para serem o comportamento selecionado para o ciclo cognitivo corrente. O selecionado dispara a memória sensorial-motora para produzir um algoritmo adequado para execução que completa o Ciclo Cognitivo.

Acima, no modelo apresentado em imagem, temos o modelo do ciclo cognitvo do LIDA em uma diagrama de blocos com setas indicativas do fluxo e relação. Cada agente mantém uma iteração contínua na forma do ciclo sentir->conhecer->agir. As duas hipóteses que sustentam a arquitetura são (1) as funções cognitivas humanas funcionam em iterações com frequência de aproximadamente 10 hertz chamadas de ciclos cognitivos e (2) esses ciclos servem como "átomos" da cognição para formar composições mais elaboradas em processos superiores (*higher-level*).

Atividade 2

Seguimos o estudo pelo tutorial do LIDA e alides do professor:

- Tutorial: http://ccrg.cs.memphis.edu/assets/framework/The-LIDA-Tutorial.pdf
- Slides: http://faculty.dca.fee.unicamp.br/gudwin/sites/faculty.dca.fee.unicamp.br.gudwin/files/ia006/LIDA.pdf

Introdução: O tutorial do LIDA introduz o básico sobre design e implementação de agentes com base no modelo do LIDA utilizando-se do framework do LIDA. Atualmente, o framework é uma implementação parcial do modelo na forma de software utilizando Java.

O modelo LIDA: O modelo é amplo, conceitual e computacional cobrindo uma larga porção da cognição humana. Com base na teoria do **Global Workspace** de Baars, este implementa variadas teorias psicológicas e neuropsicológicas. Todo agente autônomo humano/animal/artificial precisa perceber (sentir) o ambiente e selecionar uma resposta apropriada (ação). Agentes mais sofisticados (humanos) processam essa percepção de forma a facilitar uma tomada de decisão. A "vida" do agente pode ser vista como a sequência contínua desses ciclos cognitivos.

O framework LIDA: Como agentes em software podem ser muito complexos, isso torna difícil a implementação e customização. Para isso, decidiram implementar um framework que acreditam ser a melhor arquitetura para software em larga escala promovendo reausabilidade e facilidade de customização.

As principais características da implementação padrão inclui: Um **módulo** (module) é um conjunto coeso de representações (**dados**) e suas operações associadas. Uma **tarefa** (task) é um algoritmo simples que pode executar repetidamente implementando um pequeno processo. Várias estruturas de dados implementam representações internas comuns. Um **gerenciador de tarefas** (task manager) suporta a execução simultânea de tarefas. Uma **interface gráfica** (GUI) configurável mostra o estado atual interno dos processos e os dados no sistema. Uma **fábrica** (factory) é usada para criar estruturas de dados e estratégias comuns, que encapsulam algoritmos comuns. Finalmente um **analisador** (parser) XML é utilizado para carregar e criar um agente a partir de um arquivo XML declarativo.

A relação entre o framework e o modelo pode ser resumida em: (1) a idéia genérica de um **módulo** é especificada pela interface **FrameworkModule;** (2) **Módulos** no modelo são normalmente especificados por uma interface, por exemplo, o módulo de **memória procedural** (Procedural Memory) está especificado pela interface **ProceduralMemory**; (3) **Processos dos módulos** são implementados usando as **tarefas** de vários tipos e são implementações da interface **FrameworkTask**; (4) O **tempo** no modelo é representado internamente em uma unidade chamada de **tick**; (5) **Execução assíncrona** no modelo é implementada pela execução concorrente de tarefas gerenciada pelo **TaskManager**; (6) **Comunicação entre módulos** baseiam-se em **listeners (padrão de projeto Observer)** que implementam a interface **ModuleListener**; (7) **Links e nós** são implementados usando **NodeStructures**; (8) A **ativação** de nós/links/outros é realizada através da interface **Activatible**; Essa implementação inclui também **estratégias** que controlam **o decaimento e a elevação do nível** desses elementos.

Elementos básicos do framework: Módulos, listeners, tasks, task spawners, task manager, nodes, links, node structures, activation, strategies, element factory e GUI.

Inicialização do framework: o pacote **initialization** contém as classes para a simulação. A classe **AgentStarter** provê métodos para rodar uma simulação. Também dispõe método principal (main) para especificar o **arquivo de configuração primário,** cujo nome padrão é **lidaConfig.properties**. Ao executar o **start** os passos que seguem são: (1) carrega definições dos tipos de elementos em um **arquivo secundário de configuração,** conhecido como factory data, especificado na propriedade **lida.elementfactory.data** no arquivo primário**;** (2) Cria instância da classe **Agent** com base no conteúdo do **arquivo de declaração do agente**. Esse arquivo é especificado na propriedade **lida.agentdata** do arquivo primário; (3) Se **lida.gui.enable** no arquivo primário igual **true**, cria a GUI do Framework usando as informações contidas nos **arquivos de configuração da interface gráfica**, especificados pelos parâmetros **lida.gui.panel** e **lida.gui.commands** no arquivo primário; (4) O agente é carregado; (5) GUI é carregada e mostrada.

Mais detalhes da inicialização:

- **Arquivo de declaração do agente:** especificado em **lida.agentdata** com os componentes do agente, com a tag raiz **<lida>** e as tags aninhadas:
    - **<gobalparams> ::** tag opcional com parâmetros de iniciação
    - **<taskmanager>** :: configuração do TaskManager
    - **<taskspawners>** :: formado com subtags **<taskspawner>,** determina os TaskSpawner disponíveis
    - **<submodules>** :: formado com subtags do tipo **<module>** e aninhamentos **<associatedmodule>, <initialtasks>, <initializerclass>** ou mesmo **<submodules>** que determinam as classes e parâmetros para criação
    - **<listeners>** :: forado por subtags **<listener>** que estabelecem as conexões entre os módulos do agente
    - Especificação formal e detalhada no arquivo LidaXMLSchema.xsd
- **A classe AgentXMLFactory:** carrega o arquivo de declaração do agente, analisa, cria componentes e monta tudo dentro de um **objeto Agent**. O método **getAgent** executa as seguintes ações:
    - Os parâmetros globais são lidos e adicionados na classe **GlobalInitializer**

    - **TaskManager** é criado
    - **TaskSpawners** são criados
    - Os módulos são criados (e métodos **init** de cada módulo são chamados)
    - As conexões **ModuleListener** são estabelecidas
    - Opcionalmente, os módulos são associados se especificado
    - Cada módulo que especifica uma classe **initializer** em sua declaração é reinicializada usando a classe especificada
    - Para cada módulo que possuir **tarefas iniciais** (initial tasks) especificadas em sua declaração, essas tarefas são criadas e agendadas para execução pelo **TaskSpawner** do módulo
- **O arquivo de definição da fábrica:** arquivo especificado **lida.elementfactory.data** contém definições diversas como **Strategy/Node/Link** e tipos de tarefa que devem ser adicionados ao **ElementFactory** na inicialização. As tags disponíveis aninhadas:
    - **<strategies>** :: formado com subtags **<strategy>** que definem os tipos **Strategy** a serem carregados em **ElementFactory**
    - **<nodes>** :: formado com subtags **<node>** que definem os tipos **Node** a serem carregados em **ElementFactory**
    - **<links>** :: formado por subtags **<link>** que definem os tipos **Link** a serem carregados em **ElementFactory**
    - **<tasks>** :: formado subtags **<task>** que definem os tipos **Task** carregados em **ElementFactory**
- **A interface inicializável:** vários elementos do framework implementam a interface **Initializable** inclusive algumas implementações padrões. A interface provê uma forma uniforme de inicializar elementos com parâmetros.

Implementações padrões dos módulos do LIDA: nesta versão estão inclusos:

- Environment (abstract)
- Sensory Memory (abstract)
- Perceptual Associative Memory
- Transient Episodic Memory
- Declarative Memory
- Workspace
- Structure-Building Codelets
- Attention Codelets
- Global Workspace
- Procedural Memory
- Action Selection
- Sensory-Motor Memory

Existem vários detalhes a serem mencionados, mas resumidamente: (1) o módulo **Environment** tem de ser provido pelo usuário pois o framework somente determina uma classe básica abstrata chamada **EnvironmentImpl**; (2) a memória sensorial é atualmente provida como uma classe abstrata dado que existe grandes variações de domínio para domínio; (3) a implementação padrão de seleção de ação, na versão atual, não contempla todos as funcionalidades descritas no modelo; (4) as duas memórias descritas no modelo, Transient Episodic Memory and Declarative Memory, foram implementadas com base na memória distribuída esparsa (Kanerva). Por fim, vale lembrar que diversos algortimos de aprendizado ainda não têm implementação padrão.

Atividade 3

Arquitetura baseada em Codelets: um Codelet é um pequeno trecho de código com função específica e bem definida, são utilizados em paralelo para dividir uma tarefa em porções menores em paralelo.

PAM - Perceptual Associative Memory: corresponde neurologicamente as partes de diferentes sensores corticoidais em humanos e permite o agente distinguir, classificar e identificar informações internas e externas, implementado no LIDA como uma **slipnet**.

Behavior Network: LIDA tem uma behavior network de Maes com alguma modificações como implementação do mecanismo de seleção de ações responsável por decidir o que será feito em seguida.

Global Workspace Theory: originalmente concebida por Baars como um modelo neuropsicológico e processos conscientes e inconsisentes, expandido para processos humanos de cognição e implementado no LIDA como modelo base, também chamado de **GWT**.

Como a arquitetura LIDA implementa a percepção do ambiente? Existe uma classe básica abstrata chamada **EnvironmentImpl** e é responsabilidade do usuário fazer a implementação no domínio em questão, no mínimo os métodos **getState** e **processAction** são necessários.

Como a arquitetura LIDA guarda/armazena informações de memória? As duas memórias descritas no modelo, **Transient Episodic Memory** e **Declarative Memory**, foram implementadas com base na memória distribuída esparsa (Kanerva).

Como a arquitetura LIDA seleciona as ações que serão implementadas no ambiente? Pela **Behavior Network** modificada, a ação mais apropriada é selecionada em resposta à situação atual percebida a cada fim do ciclo cognitivo.

Como a arquitetura LIDA executa a ação selecionada no ambiente? O módulo **Sensory-Motor-Memory** é responsável pela execução, com a entrada selecionada pela **Behavior Network** e analisando a **Sensory Memory** ocorre a implementação da lista de tarefas a serem executadas em cada ciclo cognitivo.