NeuPaths Overview

Introduction

Imagine a computer program that operates like the human mind, where hundreds, thousands, even millions of cells work together to solve complex problems. This concept is now possible using a new programming paradigm called *Cellular Programming*. With cellular programming, computational tasks are performed by autonomous, distributed execution units working in concert to process data. While exhibiting some characteristics of Kahn Process Networks and Flow-Based Programming, Cellular Programming is a clean-sheet design inspired by the human nervous system. *Cells* are interconnected by *Synapses* and organized into clusters (see Figure 1.) The cells in a cluster process *Stimuli* that traverse the synapses, producing new stimuli in response. Clusters are also interconnected by synapses, and together form a *System*.

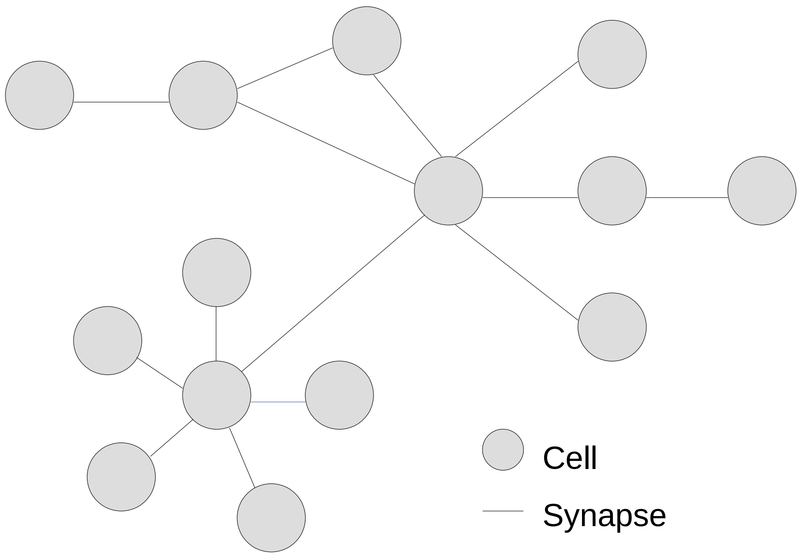

Figure 1: Example of Cellular System

In its simplest form, a cell could be viewed as one or more subroutines whose input and output parameters are the stimuli arriving and leaving the cell respectively. However, a cell could be logically much more, or less. For example, a cell could represent an entire CPU, or it could represent just one transistor of a CPU. The granularity and nature of processing is entirely at the discretion of the cellular programmer.

While the synapses govern the paths that stimuli can follow, a cell knows nothing about the cellular system structure. A cell is only concerned with consuming and producing certain stimuli. A cell advertises *Subscriptions* for stimuli. These subscriptions propagate through the cellular system over the synapses. Each cell’s nucleus learns the synapse(s) over which to forward stimuli that satisfy subscriptions.

NeuPaths is an API and runtime environment for Cellular Programming. It is written in pure Java SE, making it possible to achieve "write once, run anywhere." If necessary, Java technologies such as JNI and FFM can be used for system-level interfacing. Porting to other programming languages is under consideration.

A cellular system has very powerful characteristics:

- A system has no start or finish; its cells are always available to process stimuli.
- A system is a collection of autonomous cells. There is no coordination between cells other than the production and consumption of stimuli. There is also no central control of a cellular system.
- The location of a cell is transparent. Every cell can be hosted locally or remotely. Changing the location of a cell involves only changing its synapses, and the processing performed by a cell is independent of its location.
- All stimuli processing in a system is asynchronous, performed in near real-time.
- A system can be dynamic; cells can join and leave the system at any time.
- Support for redundancy and fault-tolerance is built into the system. Cells can perform redundant services, and down-stream cells transparently filter redundant stimuli.
- The interconnections between cells have no limitations. The system forms an undirected multi-graph that can contain cycles.

In summary, a cellular system operates in a decentralized, flexible, parallel, distributed and asynchronous manner. Sounds a bit like the human nervous system, right?

The ultimate goal of cellular programming is true visual programming. However, before that can happen, the foundation must be laid. The *NeuPaths* ecosystem will continue to grow as the runtime system matures. Tools will be created to deploy, debug and tune a cellular system. Eventually, an IDE will be created for visual cellular programming.

Cellular System Architecture

The following is an example of a cellular system architecture (see Figure 2.) *Cells* (C_i) and their *Synapses* ($S_{i,j}$) form an undirected multigraph. Synapses are the glue that holds the system together; they determine the possible paths for *Stimuli* to follow. Depending on the system architecture, a stimulus may arrive at an interested cell over multiple paths. For example, if cell C_8 were interested in stimuli from cell C_3, the stimuli would take two paths: $C_3 \rightarrow C_4 \rightarrow C_6 \rightarrow C_8$ and $C_3 \rightarrow C_4 \rightarrow C_7 \rightarrow C_8$. The *NeuPaths* runtime would deliver one copy of each stimulus to C_8 (i.e. duplicates are filtered.) This example assumes that the related *Subscriptions* are in the global *Domain*. Subscriptions are another element of the cellular architecture; they pull stimuli through the system to their destination. Domains and subscriptions are discussed in a later section.

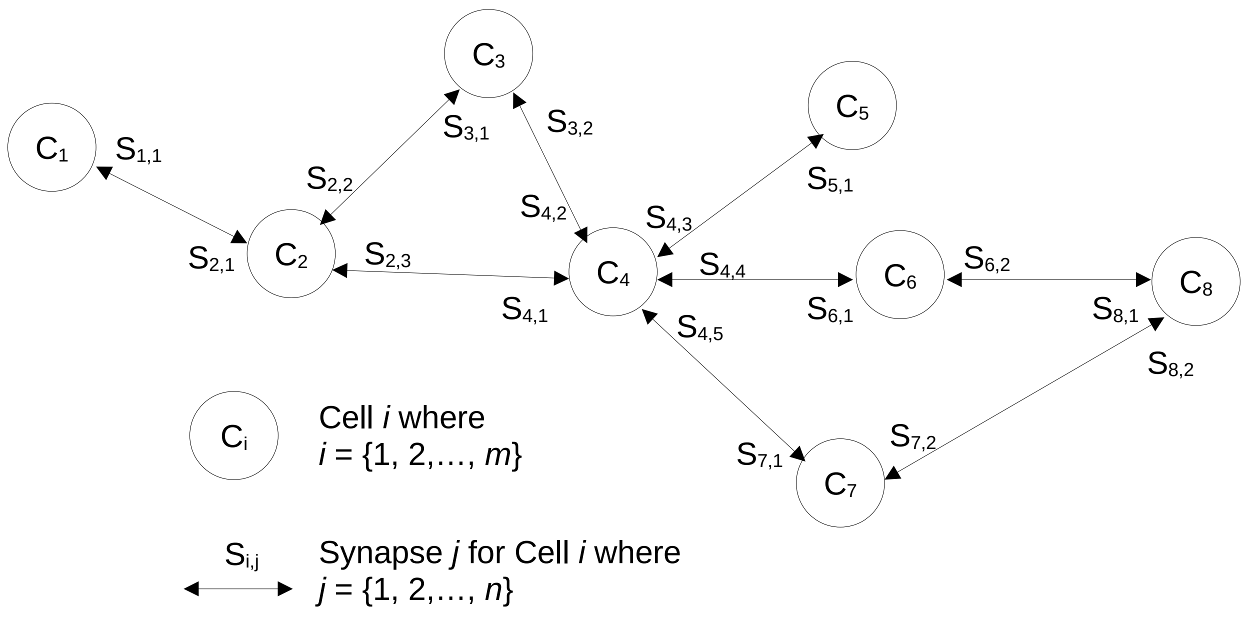

Figure 2: A Cellular System Architecture

Cell Anatomy

Each cell contains one or more *Activators* that evaluate stimuli as they arrive. Activators process stimuli on their *Receptors* and advertise new stimuli on their *Transmitters* (see Figure 3.) Each activator has its own set of receptors, transmitters and subscriptions. The names of receptors and transmitters are not required to be unique among a cell's activators; activators can share receptors and transmitters as long as they agree on the stimulus type.

Receptors

Receptors are named, type-safe receptacles for stimuli. They consume stimuli that satisfy a subscription.

Transmitters

Transmitters are named, type-safe emitters of stimuli. They forward stimuli to interested synapses.

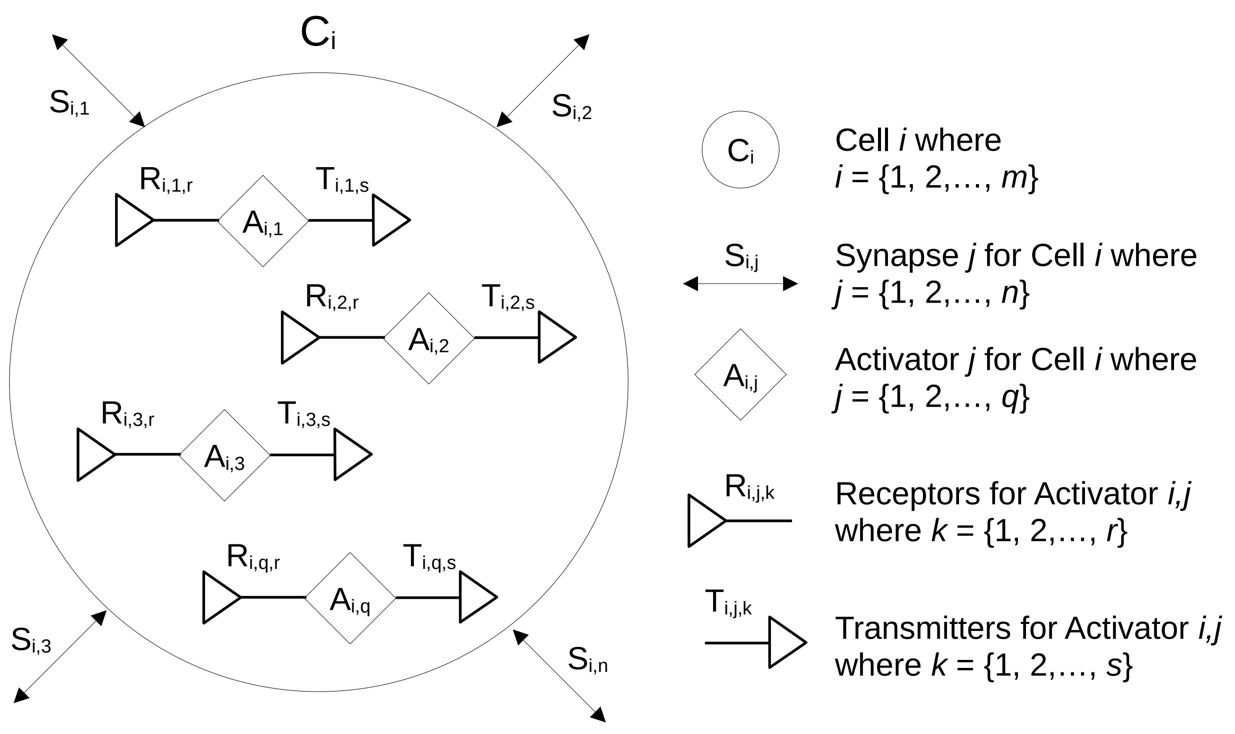
C_i
$S_{i,1}$
$S_{i,2}$
$R_{i,1,r}$
$A_{i,1}$
$T_{i,1,s}$
$R_{i,2,r}$
$A_{i,2}$
$T_{i,2,s}$
$R_{i,3,r}$
$A_{i,3}$
$T_{i,3,s}$
$R_{i,q,r}$
$A_{i,q}$
$T_{i,q,s}$
$S_{i,3}$
$S_{i,n}$
C_i Cell i where $i = \{1, 2,\dots, m\}$
$S_{i,j}$ Synapse j for Cell i where $j = \{1, 2,\dots, n\}$
$A_{i,j}$ Activator j for Cell i where $j = \{1, 2,\dots, q\}$
$R_{i,j,k}$ Receptors for Activator i,j where $k = \{1, 2,\dots, r\}$
$T_{i,j,k}$ Transmitters for Activator i,j where $k = \{1, 2,\dots, s\}$

Figure 3: Cell Anatomy

Activators

Activators do the heavy lifting in a cellular system; all stimuli processing is performed by an activator. Consequently, a majority of the design and coding will involve activators. Think of an activator as a black box with inputs and outputs. The inputs are specified as a set of receptors, while the outputs are specified as a set of transmitters. Subscriptions pull stimuli from producing cells to the receptors. Once all of an activator’s receptors have stimuli, the activator evaluates the stimuli.

Synapses

Synapses bind together the cells in a system. They are an abstraction for the underlying technologies used to transport stimuli. Synapses participate in a single domain and will only transport stimuli that satisfy a subscription in that domain. The one exception is that any synapse will transport stimuli that match a subscription in the global domain.

Stimuli

Stimuli are atomic data elements that are transported by synapses. A stimulus is a unique instance of a *Stimulus Type*, which defines the content and layout of the data. Even if two stimuli of the same type share the same value, they are considered unique to the *NeuPaths* runtime. Receptors and transmitters are type-safe receptacles of stimuli; they will only accept stimuli of a designated type.

Domains

Cells can be logically grouped together by named domains. Synapses and subscriptions are tagged with a domain name, and stimuli will only be transported over synapses with a domain that matches the subscription domain. The one exception to this rule is that stimuli matching subscriptions in the global domain will be transported over any applicable synapse. Since cells can have any number of synapses, and each synapse has a domain, a cell can participate in multiple domains.

Subscriptions

Subscriptions express interest in stimuli that are emitted by cell transmitters. A subscription specifies the producing cell (C), producing transmitter (T) and consuming receptor names (R). In essence, a subscription is a contract: (C, T) $\rightarrow$ R. This contract implies that the transmitter and receptor stimulus types match. The producing cell and producing transmitter names may specify regular expressions, making it possible to aggregate stimuli from multiple cells. A subscription also specifies the domain in which the stimuli will be transported. The global domain indicates that stimuli should pass to all domains.

Subscriptions are a key element in a cellular system. Stimuli will only transit where a subscription has previously passed. A subscription is published by a cell on all of its synapses. The neighboring cells make note of the subscription and forward it to their neighbors. In this fashion, a subscription propagates throughout the cellular system. When a subscription specifies a domain, it will only propagate over synapses in the same domain.

Recall that a cellular system is dynamic. Since cells can join and leave the system at will, it is necessary to update subscriptions periodically. Each cell has a subscription refresh period. This feature can be disabled if the cellular system is known to be static after creation.

Designing a Cellular System

Designing a cellular system, or a subsystem, is generally performed in three phases:

1. Creating the **Data Flow Model**. This phase identifies the cells, stimuli and dependencies in the system. Interconnections are not important at this stage; you can assume that the stimuli will arrive. Activators can be identified in this phase as well.
2. Creating the **Cluster Model**. This phase identifies the synapses as interconnections. The system can be partitioned into domains to compartmentalize functionality and services. Data transportation is not important at this stage; synapses show connectedness but do not imply communication topology. This phase can influence system performance greatly.

3. Creating the **Network Model**. This phase identifies where the system lives. It may reside entirely on a single host computer, on several host computers in a single intranet, or spread across nodes in multiple interconnected data networks. This model dictates the synapse names (specifications) that are used. This phase must determine the best data transport type(s) to use and identify network and security issues.

Performance and Resources

A cellular system can consume a lot of resources: Each cell consists of several threads, each synapse consumes network resources, and receptors and transmitters use heap memory. The cluster model can also have a large impact on resource utilization. Subscribing to stimuli from distant cells uses more resources than from a nearby cell (in terms of network and processor utilization.) A system with a complex model that includes cycles will rely heavily on duplicate stimuli detection, which requires resources to manage (in terms of heap and processor utilization.)

Use of domains can influence system performance. Subscriptions periodically propagate throughout a system, which can add significantly to system traffic. Subscriptions in the global domain will reach every cell in a system. For large systems, this can consume important resources. Subscriptions in a particular domain will only reach cells participating in the domain, and since stimuli only go where subscriptions have gone, the stimuli will also remain within the domain.

The tuning of performance and resource utilization is an iterative process. The cluster model can be refined, and in most cases refinement will not change the data flow model. The subscription refresh and duplicate detection intervals can be tuned for each cell independently. The number and location of JVM instances can be refined to improve processor and memory utilization.

Why Use NeuPaths?

NeuPaths is designed to process data as fast and efficiently as possible. Speed is achieved by processing data in (near) real-time as it becomes available. Speed is also achieved by increasing parallelism using cells. Efficiency is achieved by reducing CPU utilization when there is currently no data to process. This is accomplished by eliminating polling loops and excessive synchronization (mutexes, locks, semaphores, etc.) Cells operate autonomously in a cluster, waiting for stimuli to arrive. While waiting, a cell has no work to perform, making it possible for the host OS to quiesce the cell. As stimuli arrive, the host OS schedules the cell to run, at which time the cell can decide if it has enough data to process. When a cell has the full complement of its stimuli, its activators are triggered.

NeuPaths provides an event-driven data-flow execution model. Parallelism is leveraged by decomposing an algorithm into discrete processing units that are hosted locally and/or remotely. Synchronization is maintained by nature of the data-flow; an activator will not process until it has all of the requested stimuli.