Powering the AI Revolution™

Psyclone 2.0

Beta 3 Technical Whitepaper

By Thor List, CTO

1 July 2014

Contents

Introduction

Psyclone 2.0 provides a platform for running large systems of heterogeneous modules communication using heterogeneous data with near-instantaneous dynamic re-programming of local or global data flows.

It is a complete rewrite of the original Psyclone 1.x system from scratch. Very little code has been reused to ensure a very lean implementation that focusses on performance and usability while preserving the strengths of previous versions. The API for writing modules is, however, very similar and still very easy to use.

The main purpose of Psyclone is, as before, to make it easy for a single individual or groups to design, implement, deploy, test, and maintain, very large multi-module systems with many interacting parts. While much of the original Psyclone 1.x concepts remain in the 2.0 implementation, new designs have also been introduced. With Psyclone 2.0 this goal is better realized than ever, and the vastly improved efficiency and greatly lowered communication latency should make Psyclone the preferred platform for even those seeking strict adherence to time protocols.

This Whitepaper assumes familiarity with the main Psyclone 1.x concepts, but the key points should be understandable to anyone with some knowledge of networking and software development.

About this Beta Release

The current release is still in beta and although the development has focused predominantly on stability some instability is still to be expected.

The main reasons for releasing the early beta version are to get user feedback on stability, usability, new features and to direct the roadmap for the not-yet-implemented new features.

The main system features included in this beta release are:

- Windows and Linux 32 and 64-bit interoperability support
- Psyclone *Nodes* and *Spaces*
- Internal and External modules
- Whiteboards and Catalogs
- The full PsyProbe 2.0 web interface
- System-wide performance monitoring
- Full context, types, topics and signals implementation

The user can do:

- Internal and External modules
- Custom Whiteboards and Catalogs
- Custom web interfaces and services
- Add any size and number of data chunks to messages

A number of features and functions will be added in future releases, including:

- Feeds
- Interfaces
- System-wide performance adjustment
- Module migration
- Automatic load-balancing
- Other OS’es and programming languages
- Scripted modules support

... and more.

For more information and a more comprehensive list of existing, new and coming features, please see the *Psyclone 2.0 Presentation*.

Psyclone 2.0 System Overview

Psyclone is a general-purpose platform for deploying multiple processes that can interact via powerful message and stream communications channels and protocols. For module interaction Psyclone employs a simple but powerful publish-subscribe mechanism that enables a user to quickly create and connect modules and make them interact. This can be done dynamically at runtime and via the Psyclone main initialization file (PsySpec).

The term "Psyclone system" is used to refer to the base Psyclone system plus any user-created components that participate in the end-user's deployed system.

All interaction in a Psyclone system happens using discrete communications based on *subscriptions* and *filters*. A basic unit of communication is a Psyclone message, each of which have parameters such as a *type*, a *topic*, a *time-to-live*, etc.

The most basic type of user-defined component in a Psyclone system is the *module*. Modules can have any number of user-defined parameters which can be accessed by the module itself and by other modules, even those running on other computers. A module can post (produce) and get triggered (receive) messages from anywhere in a Psyclone system. A module is triggered when messages of the types to which it has subscribed are posted by some component in the system.

When modules post a message of a particular type the message is automatically routed to whoever has subscribed to its type. Any type of data of any size can be put into a message; the data will be kept in a single block of memory for efficiency, which is greatly improved over Psyclone 1.x.

The main configuration file for a Psyclone system is called the PsySpec. It contains a list of a system's modules and components, relevant information regarding these, as well as partial or full data flow specification, according to the end-user needs, initial subscriptions for each module, and more. The full list of nodes in the system and process spaces used is specified here and nodes can be provided as either required or ad-hoc. The latter allows nodes to dynamically join and leave the system throughout the system lifetime.

Messages

All components in a Psyclone system have subscriptions, which are based on *types*, *topics,* and *signals*. Types are dot-delimited strings; subscriptions can specify either the full type or use wildcards to reference groups of types. Topics can be used to segment data temporally, i.e. a module may only be interested in camera data referencing a specific object or event. Signals are low-level events (lower than subscription messages) which can be used to synchronise many modules at the same time, and can be either "tempo" signals (with a fixed timing like a musical conductor), or event signals.

Every subscription is context-sensitive; as the global system contexts change so will the data flow mapped out by the subscriptions. The system can have a number of globally active contexts at any given time. When a context *A.B.C* is active, all other *A.x.x.x.x.x* contexts are inactive. A context change is system-wide and near-instantaneous, affecting all subscriptions, and thus all data flows, in the whole system which are sensitive to the changed context.

As already mentioned, when modules post a message the message will be automatically routed to whoever has subscribed to its type. Messages in Psyclone 2.0 live for a finite amount of time inside the *shared memory*, as specified in the PsySpec with the time-to-live (ttl) parameter value. If ttl = 0 the message will only be delivered to whoever has subscribed to it and after this seize to exist. If a ttl has been provided, it will be delivered the same way, but the receiver will be notified of a unique reference which can be used to retrieve the message again until the ttl expires.

Triggers – the specified conditions which specify when a module should handed a particular set of messages – can now have filters, anything from *maxage* to filtering on a specific custom user-defined key. *Trigger groups* can be used to wait until *all* or *some* of the specified triggers have arrived before handing them to the module.

Whiteboards, Streams & Catalogues

A Whiteboard, Stream or Catalogue (see below) can be referenced in the *To* field of a message. Rather than being actual message containers, as in Psyclone 1.x, Psyclone 2.0 Whiteboards, Streams, and Catalogues in Psyclone 2.0 are *indexers* of the messages. System modules can make specific requests to these components based on desired messages types, referencing the way each of these components index messages.

Data Communication

Modules in Psyclone 2.0 run alongside Nodes in the system. A system typically consists of a number of Nodes, each one running on a different computer running any of the supported operating systems. A system can have hundreds or thousands of Nodes and millions of modules, all running at the same time and communicating freely.

Modules in Psyclone 2.0 run inside process *Spaces*, which isolate runtime memory. The *Root* Space can run inside the Node process, but most commonly modules run in other spaces created automatically by the Node. If a module crashes, only the modules that live in the same space as the crashed module will go down – modules in other spaces are unaffected. The crashed modules can then be restarted automatically with all their saved state data intact. One Node can have a large number of Spaces and these will be automatically instantiated and maintained as specified in the main configuration file (PsySpec).

Third-party software can participate in the system using the same API to connect to the local Node. All communication between other local modules, and locally running third party software, happens through shared memory. Communication with modules running on other nodes happens through a configurable number of network protocols.

As mentioned, messages have a number of fields which can be filled by any data the user may wish to transport. Once the message has been constructed it is, however, one single chunk of memory, so using, copying, and moving is now significantly faster in Psyclone 2.0.

Other new features include the introduction of Feeds which are sequential data either from outside into the Psyclone system or from inside to the outside. This could be a camera, an audio source, an RSS feed, etc.

When a module posts a message it will go into the local node's shared memory bank and remain there until the message's time-to-live field says to destroy it. All modules attached to the local node will communicate via the shared memory and can recall any message still retained in the shared memory using its unique id. Modules can also query data from Whiteboards and Catalogs which may contain aggregated data or access to third party data sources.

When data needs to travel between nodes to other computers it can use a number of communication protocols. For guaranteed communication the nodes use TCP and for non-guaranteed communication (allowing for dropped messages when busy) the nodes use UDP.

Interfaces will provide access to a Psyclone 2.0 system internal services from the outside, via HTTP, Telnet, or similar protocols, and can be specified using subscriptions (triggers, posts, etc.). This can allow easy third-party access to the

runtime system, and will provide service to e.g. Android and iPhone SDKs so they don’t have to run a full Node locally.

The system is designed to allow other networking technologies to be added in the future such as Infiniband, I^2C and CAN Bus.