

HIVE Kernel Locale Facilities

Jim Purbrick

New Files and Classes

- Agent.h
 - Agent::EnvRefCount – An <Environment, Count> pair that is used to keep track of the number of references made to an Environment. When a joinEnvironment(...) call is made to join an Environment that is already replicated, its reference count is incremented. leaveEnvironment(...) decrements the count on an Environment and will only delete it when its reference count is zero. This allows multiple clients to share a single Agent, or two Locales to share an Environment (for example in the case of a mirror boundary). This is a private class used for implementation only.
- Item.h
 - BoundaryData – Boundary information stored in Environments. Includes the name of the target Environment, a transform from the origin of the source to the origin of the target, a factor to apply to awareness across the boundary, a cost associated with the target environment. It also specifies an abstract visual representation of the target that is rendered if the target is not replicated.
- Locale.h
 - Locale – An <Environment, Transform> pair. Includes other attributes like maximum awareness and minimum depth to the Locale from the current Locale and the name and abstract representation of the environment used if it's not replicated.
 - LocaleMap – A collection of Locales describing the surrounding Locales viewed from a current position. From a client perspective, LocaleMap is given an Agent and name of a starting Environment at construction time and manages all joinEnvironment(...) and leaveEnvironment(...) calls for the client. testMove(...) can be used to see if a boundary has been crossed and setPosition(...) used to set the current position to the resulting, or any other, Environment. Internally, updateMap() builds the map which includes all Locales reachable by following boundaries found in the currently replicated environments. The method also calculates the minimum number of boundaries that must be crossed to reach a Locale (its depth) and the maximum awareness of a Locale based on the awareness transforms of all the boundaries crossed to reach it. These attributes can be used to decide if an Environment should be replicated or not.
 - NStepReplicator – A child of LocaleMap that overrides updateMap() to replicate all Environments reachable by crossing N or less boundaries from the current Locale.
 - NNearestReplicator – A child of LocaleMap that overrides updateMap() to replicate the nearest N Environments. Distance is measured in terms of boundaries crossed.
 - NMostAwareReplicator – A child of LocaleMap that replicates the N Locales which the client is most aware of.
 - CostBenefitReplicator – A child of LocaleMap that replicates Locales by spending a budget. Locales are assigned a value determined by dividing their maximum awareness by their cost. The class replicates high value Locales until their total cost equals its budget.
- List.h
 - List::Comparison – An abstract interface that can be implemented to sort a list in a specific way. The child classes LocaleMap use this interface to sort the list of Locales contained in the LocaleMap in a number of different ways.

Possible Improvements

The cost attribute of boundaries and Locales can currently be set at construction time to demonstrate replication policies, but it should be tied to Environment activity or geometric complexity. This could be done by Environments multicasting their activity to a group representing those agents not joined to the Environment, but interested in it, LocaleMaps polling Environments for their activity when they are considering joining, or Environments posting their activity to some other orthogonal service on a server. Alternatively Environments could just keep the costs of boundaries up to date and these costs will be found during the next call to `LocaleMap::updateMap()`. Similarly visual abstractions could be dynamically updated to show the activity of the locale they represent by getting bigger, changing height or size etc.

Dynamic visual abstractions could be implemented in a similar way to make the abstraction indicate the activity of the locale by changing colour, shape or size.

New more sophisticated replication policies could be implemented and used instead of the very simple current strategies. Also classes to combine replication policies through Boolean operators would be useful – allowing combinations like “All Locales one step away and as many high awareness Locales as possible with the current bandwidth”.

A public version of `LocaleMap::updateMap()` could be added. Calling this method from the client after each move would allow replication to change as a user moves around a single Locale rather than just when boundaries are crossed. This could be useful for replication policies based on Euclidean distances or positions.

The integration of the full spatial model with the boundary awareness transforms. Ideas like progressive meshes, transparency filters across boundaries to create windows or the use of colour fading could be included with this to provide more visual clues about awareness, auras and foci.

The efficiency of `testMove(...)` is important as it is called every time a client moves. It currently checks the vertices in each polygon are collinear, calculates the normal and then performs a collision test. The first two tests could be done when the polygon is created and stored with the vertices in a Polygon class. This would allow `testMove(...)` to do less work.

`NNearestReplicator` should be changed to sort Locales by shortest Euclidean distance to boundaries. It would need to calculate the distance to each boundary and keep track of each Locale it has already replicated, as it's likely that a number of the nearest boundaries will be to the same Locale – in the case of bounding box boundaries defining an extent for example. It may be possible to add some of the information needed to calculate the distances to the new Polygon class to improve the efficiency of `updateMap()` and allow it to be called on a per move basis.