Semantic Design for Nu Game Engine (using Sedela)

let World = Axiom "The world value."

let Game = (GameAddress : Address<Game>; Simulant)

let Screen = (ScreenAddress : Address<Screen>; Simulant)

let Layer = (LayerAddress : Address<Layer>; Simulant)

let Entity = (EntityAddress : Address<Entity>; Simulant)

let Simulant = (SimulantAddress : Address<Simulant>)

let Dispatcher = Axiom "Specifies the shape and behavior of a simulant."

let getGame : World -> Game = Axiom "Get the global game handle."

let getScreens : World -> List<Screen> = Axiom "Get all screen handles belonging to the global game."

let getLayers : Screen -> World -> List<Layer> = Axiom "Get all layer handles belonging to the given screen."

let getEntities : Layer -> World -> List<Entity> = Axiom "Get all entity handles belonging to the given layer."

let tryGetParent : Simulant -> World -> Maybe<Simulant> = Axiom "Attempt to get the parent of a simulant."

let getChildren : Simulant -> World -> List<Simulant> = Axtion "Get the children of a simulant."

let getProperty : String -> Simulant -> World -> Any = Axiom "Get the property of a simulant."

let getDispatcher : Simulant -> World -> Dispatcher = Axiom "Get the dispatcher belonging to a simulant."

let getPropertyDefinition : String -> Dispatcher -> World -> PropertyDefinition = Axiom "Get property definition of dispatcher."

let getBehaviors<A, S :> Simulant> : Dispatcher -> World -> List<Behavior<A, S>> = Axiom "..."

let PropertyDefinition =

(Type : Axiom "A value type.",

Default : Any)

let Event<A, S :> Simulant> =

(Data : Any,

Publisher : Simulant,

Subscriber : S,

Address : Address<A>)

let Behavior<A, S :> Subscriber> =

Event<A, S> -> World -> World

Nu Script Semantic Design

let script (str : String) = Axiom "Denotes script code in str."

witness Monoid =

| append = script "**+**"

| empty = script "[empty -t-]"

witness Monoid =

| append = script "**\***"

| empty = script "[identity -t-]"

witness Monad =

| pure = script "[fun [a] [pure -t- a]]"

| map = script "map"

| apply = script "apply"

| bind = script "bind"

witness Foldable =

| fold = script "fold"

witness Functor2 =

| map2 = script "map2"

witness Summable =

| product = script "product"

| sum = script "sum"

let Property = Axiom "A property of a simulant."

let Relation = Axiom "Indexes a simulant or event relative to the local simulant."

let get<a> : Relation -> Property -> a = Axiom "Retrieve a property of a simulant indexed by Relation."

let set<a> : Relation -> Property -> a -> a = Axiom! "Update a property of a simulant indexed by Relation, then returns its value."

let Stream<a> = Axiom "A stream of simulant property or event values."

let getAsStream<a> : Relation -> Property -> Stream<a> = script "getAsStream"

let setAsStream<a> : Relation -> Property -> Stream<a> = script "setAsStream"

let makeStream<a> : Relation -> Stream<a> = script "makeStream"

let mapStream<a, b> (a -> b) -> Stream<a> -> Stream<b> = script "map"

let foldStream<a, b> : (b -> a -> b) -> b -> Stream<a> -> b = script "fold"

let map2Stream<a, b, c> : (a -> b -> c) -> Stream<a> -> Stream<b> -> Stream<c> = script "map2"

let productStream<a, b> : Stream<a> -> Stream<b> -> Stream<(a, b)> = script "product"

let sumStream<a, b> : Stream<a> -> Stream<b> -> Stream<Either<a, b>> = script "sum"

Semantic Design for Observable Property Bag Simulations (now implemented by Nu)

let PropertyChangeHandler<Key> = Simulation<Key> -> Simulation<Key> -> Simulation<Key>

and PropertyChangeUnhandler<Key> = Simulation<Key> -> Simulation<Key>

and Simulation<Key> = Axiom "A simulation in terms of an observable property bag."

let getPropertyOpt<Key, A> : Key -> Simulation<Key> -> Maybe<A> =

Axiom "Obtain a simulation property associated with the given key if it exists."

let setPropertyOpt<Key, A> : Key -> Maybe<A> -> Simulation<Key> -> Simulation<Key> =

Axiom "Set a simulation property associated with the given key if it exists."

let handlePropertyChange<Key> : Key -> PropertyChangeHandler<Key> -> (PropertyChangeUnhandler<Key>, Simulation<Key>) =

Axiom "Invoke the given handler when a property with the given key is changed."