The following is what I call a 'semantic design' for Nu's scripting system (as well as an unrelated replacement for micro-services called MetaFunctions). The concept of a semantic design is inspired by Conal Elliot's denotational design - https://www.youtube.com/watch?v=bmKYiUOEo2A. The difference is that semantic design does not connect back to an existing language such as mathematics but is instead built upon an orthogonal set of axiomatic definitions.

Whereas denotational design is a more thorough design treatment that is used in greenfield development to yield high-precision design artifacts, semantic design works well for projects that don't satisfy any simple denotational design, such as those that are already far into their implementation.

To specify semantic designs generally, I've created a meta-language called ADELA (for Axiomatic Design Language). First, we present the definition of ADELA, then the semantic design for Nu and MetaFunctions in terms of ADELA.

Adela Language Definition

Axiom :=	Axiom**[!]** str	*where ! denotes effectfulness and* str *is a string literal*
Prod :=	Prod = (**Type**, ..., **Type**)	*where* Prod *is an* **Identifier**
Sum :=	Sum = | A of **Type** ... | Z of **Type**	*where* Sum *is an* **Identifier**
Type :=	**Prod | Sum | Axiom**	
Alias :=	Alias = **Type**	*where* Alias *is an* **Identifier**
Semantic :=	fn (a : **Type**) ... (z : **Type**) : **Type** = **Semantic | Axiom**	*where* fn *is an* **Identifier**
fun *a b ... z -> expr* **:=**	\a (\b (... \z.expr))	
a **->** *b* **:=**	_ = (_ : *a*) : *b*	

Adela Language Prelude

```
Unit = Axiom "The empty value."
```

Nu Semantic Design

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

Address = Axiom "Indexes a global simulant or event."

Name = Axiom "Indexes a property of a simulant."


Stream<a> = Axiom "A stream of values."

eventStream<a> : Address -> Stream<a> = Axiom "Construct a stream of values from event data."

foldStream<a, b> : (b -> a -> b) -> Stream<a> -> b = Axiom "Fold over a stream."

productStream<a, b> : Stream<a> -> Stream<b> -> Stream<a, b> = Axiom "Combines two streams into a single product stream"

sumStream<a, b> : Stream<a> -> Stream<b> -> Stream<(a, b)> = Axiom "Combines two streams into a single sum stream."


get<a> : Name -> Relation -> a = Axiom "Retrieves a property of a simulant indexed by Relation."

getAsStream<a> : Name -> Relation -> Stream<a> = Axiom "Construct a stream of values from a simulant property."


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

setToStream<a> name relation stream = foldStream (fun _ -> set<a> name relation) stream
```

Semantic Design for MetaFunctions (a replacement for micro-services - unrelated to Nu)

```
Symbol = "Symbolic type such as defined by Prime."

Async<a> = "Asynchronous result monad where result is 'a' such as defined by F#."

IPAddress = String

Port = Int

MetaName = "Specifies which MetaFunction to call for a given Port for a given IPAddress."

MetaAddress = (IPAddress, Port, MetaName)

MetaContainer = Symbol -> Async<Symbol>

MetaFn = (MetaAddress | MetaContainer) -> Symbol -> Async<Symbol>
```