AssignmentInstance

source — AssemblyConnector — target

instance — InstancePortRef — InstancePortRef — ComponentInstance

ComponentInstance

Concrete syntax for this:
connect clientComponent.clientcomp_serverInterface to gserverComponent.serverInterface

component — port — port — instance

AtomicComponent — RequiredPort — ProvidedPort — AtomicComponent

contents — contents

AssignmentExpr

left — right

GenericDotExpression — ReferenceExpr

expression — target — expression

GlobalVarRef — GenericMemberRef — GlobarRefExpr

var — member — var

GlobalVariableDeclaration (name / *A* __instance) — Member (name / *A* __ops) — GlobalVariableDeclaration (name / *A* __ops)

type

TypeDefType

typeDef — original — struct

TypeDef — StructType

members

StructDeclaration (name / *A* __cdata)

I've added the typing information between the instance and the struct declaration.

Concrete syntax for this:
ComponentsSample_instances_clientComponent__instance.clientcomp_serverInterface__ops = &ComponentsSample_gserverComponent_serverInterface__ops;

Background:

typedef struct ComponentsSample_ClientComponent__cdata ComponentsSample_ClientComponent__cdata_t;
...
struct ComponentsSample_ClientComponent__cdata {
  void *clientcomp_serverInterface__port;
  ComponentsSample_Server__idata_t *clientcomp_serverInterface__ops;
};
...
static ComponentsSample_ClientComponent__cdata_t ComponentsSample_instances_clientComponent__instance;
static ComponentsSample_Server__idata_t ComponentsSample_gserverComponent_serverInterface__ops;
...

This contract proves that any assignment made to the __ops member of the __inst variable is the address of the global var ops
But this leaves at least the following assumptions to be checked:
(1) we need to prove that the global var is assigned the correct operation in the wire function;
(2) the wire function is called at the initialization.
(3) the instance that requires the port gets assigned the correct instance (this in turns will generate 2 more contracts, at least)

GlobalVarGetsCorrectFunctionAddress at the initialization

contents — InstanceConfiguration

ComponentInstance — ClientServerInterface

component — intf — contents

AtomicComponent — ProvidedPort — Operation (name)

contents — providedPort

contents — Executable (name) — trigger — OperationTrigger — calledOperation

body — Function (name / *A* __init)

StatementList

statements

ExpressionStatement

We might need to perform the exact reconstruction of the name attribute of both the prototype and the function to ensure they are the same.

expr

FunctionCall

function

FunctionPrototype (name / *A* __wire)

Concrete syntax:
struct ComponentsSample_Server__idata {
  char* (*server_process)(char*,void*);
};
...
typedef struct ComponentsSample_Server__idata ComponentsSample_Server__idata_t; (not included, but to be included in the future)
...
static ComponentsSample_Server__idata_t ComponentsSample_gserverComponent_serverInterface__ops;
...
static inline void ComponentsSample_instances_gserverComponent__wire(void)
{
  ComponentsSample_gserverComponent_serverInterface__ops.server_process = &ComponentsSample_GoodServer_serverInterface_server_process;
}

Function (name / *A* __wire) — body — StatementList

statements

ExpressionStatement — expr — AssignmentExpr

left — right

GenericDotExpression — ReferenceExpression

expression — target — expression — function

GlobalVarRef — GenericMemberRef — FunctionRefExpr — FunctionPrototype (name)

var — member

GlobalVariableDeclaration (name / *A* __ops) — CFunctionPointerStructMember (name)

members

Maybe we can separate the part where the wire function gets called from the initialization function from the correct assignment

StructDeclaration (name / *A* __idata)