Practical Session 2 –Dana Advanced Concepts

Assembly Process

The assembly process is a set of steps that leads to the composition of a fully functioning component-based Dana program. This process comprises the following stages i) loading components from memory, ii) wiring components into a fully functioning system, and iii) adapt the program by replacing one or a set of components with their variants. We now detail each stage of the assembly process, describing the code used to perform each one of its stages. As depicted in Fig 1., the assembly process is divided into LOAD, WIRE and ADAPT.

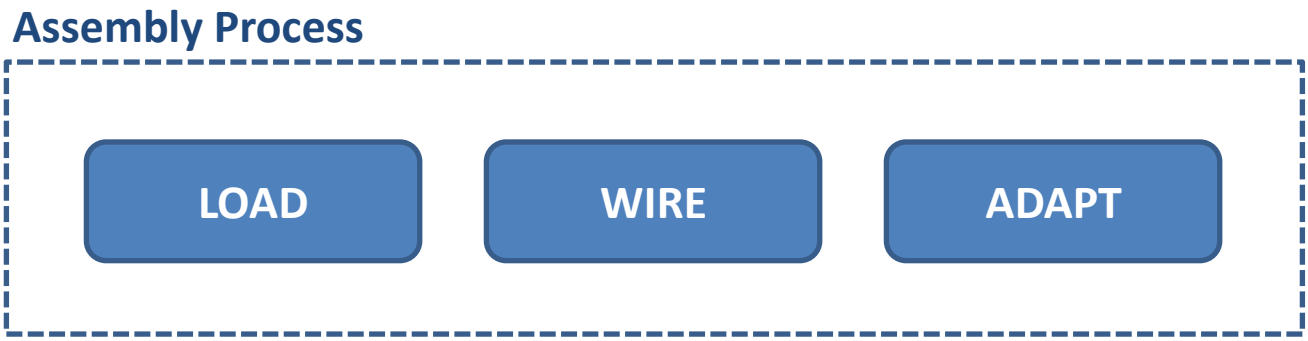


Fig 1. Assembly Process

The LOAD part of the assembly process is where components are loaded into memory from the file system. Only after loading a component to memory we are able to connect them into a functioning system. In practice, to load a component to memory we use a special component named Loader, and call the method *load()* passing as parameter the path to the component object file, as depicted below.

```
1 component provides App requires Loader loader {
2     int App:main(AppParam param[]) {
3         IDC component = loader.load("Component.o")
4     }
5 }
```

Note that the *load()* method returns an IDC. The IDC holds in memory every information of the loaded component necessary for the Dana runtime to connect them, create an object instance of the component, and perform adaptation. The LOAD stage must be performed for every component that is required to make a fully functioning version of the program. Once all components are loaded in memory the next stage of the assembly process begins.

WIRE is the next stage of the assembly process. It consists of connecting loaded components according to the provided-required interfaces. Note that if the component has one required interface with no component connected to it, the Dana runtime blocks the entire component execution stating that not all dependencies were satisfied. Therefore, it is important to make sure all components are properly wired.

The following code shows two components being wired through the interface *"Interface"*. We wire components by calling the Dana native method rewire() passing as parameters the component that requires the interface, and the component that provides the interface along with the interface name. Once the components are wired, an instance of the main component is created.

```
component provides App requires Loader loader,
 composition.Adapter adapter {
    int App:main(AppParam param[]) {
        IDC mainComponent = loader.load("MainComponent.o")
        IDC variantA = loader.load("VariantA.o")
        IDC variantB = loader.load("VariantB.o")

        dana.rewire(myComponent :> "Interface", variantA :< "Interface")

        MainComponent obj = new MainComponent() from mainComponent
        obj.method()

        adapter.adaptRequiredInterface(mainComponent,
            "Interface",
            variantB)
        obj.method()
    }
}
```

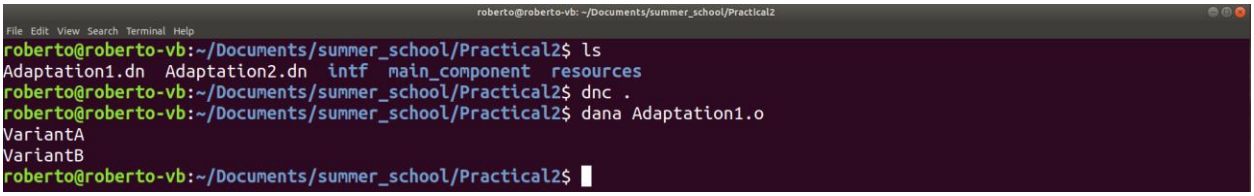
Finally, ADAPT is the last part of the assembly process. In case there are multiple components providing the same interface (namely component variants), we can replace components in the running program. In our example, we have two component variants that implement "Interface" and therefore we can perform adaptation between them. The adaptation process is encapsulated in the *composition.Adapter* component, and it is executed by invoking the method *adaptRequiredInterface()* passing as parameters the component that requires the interface, the interface, and the component that provides the interface, as illustrated in the code below.

```
component provides App requires Loader loader, composition.Adapter adapter {
    int App:main(AppParam param[]) {
        IDC mainComponent = loader.load("MainComponent.o")
        IDC variantA = loader.load("VariantA.o")
        IDC variantB = loader.load("VariantB.o")

        dana.rewire(mainComponent :> "Interface", variantA :< "Interface")

        MainComponent obj = new MainComponent() from mainComponent
        obj.method()

        adapter.adaptRequiredInterface(mainComponent, "Interface", variantB)
        obj.method()

        return 0
    }
}
```

Executing the Basic Adaptation Example

The code used to illustrate the adaptation process is available in the **GitHub** repository: https://github.com/barryfp/summer_school. Once you have access to the repository, you will find all the code we made available for this practice in the folder "*Practice 2*".

To execute the "assembly process" example we previously presented, compile all components in *Practical2/Adaptation* using the command "dnc .". After compiling all components, we run the adaptation example by executing "*dana Adaptation1.o*". The source code is in the file "Adaptation1.dn", and you should carefully study it to understand how to load components, wire them and adapt them.



Extracting Information and Searching Components

The code in *Adaptation1.dn* has some limitations. It clearly requires the path of **all components** and the name of **all interfaces** to be hardcoded for the example to work. In this section, we are going to learn how to extract a list of required interfaces from a component's object file (.o file) and to find a list of components that implements a specific interface, so we do not have to know beforehand all components and interfaces involved in the program.

The first step is to load components according to the code in *Adaptation1.dn*. Our first step is to search for the components that need to be loaded. However, in order to search for components, we need a starting point, by having someone to provide at least the name of the **root component**. The root component is the component from which all the other components are discovered. In other words, the root component implements the methods that get the program running.

Once the path to the root component is provided, we use the *ObjectWriter* to extract a list of required interfaces as illustrated in the following code.

```
char rootCompPath[] = "main_component/MainComponent.o"

// extracting info from mainComponent
ObjectWriter objWriter = new ObjectWriter(rootCompPath)
InfoSection infoSection = objWriter.getInfoSection("DNIL", "json")
Section sec = jsonEncoder.jsonToData(infoSection.content, typeof(Section), null)
```

All information inside a Dana object file is formatted in JSON. We use the *JSONEncoder* component to parse the information extracted from the object file into a Dana data structure named Section, which holds all provided and required interfaces of the component. In the code example below we show the Search component being used to find all component variants that implement the root component's required interface by calling the method *getComponents() and* passing as parameters the interface path.

```
// looking for components that provide requiredInterface
String compsPath[] = search.getComponents(sec.requiredInterfaces[0].package)
```

Once the interface is extracted from the root component object file, we can load the root component and the components that provide the root component required interface, and wire them and adapt them when necessary, just as we did in the *Adaptation1.dn*. The code that loads, wires and adapts is the same as in the Adaptation*1.dn* with some minor differences and it is illustrated below. Note that in the code below we know that there are only two component variants for the required interface, and that is why we hardcoded option variants[0] to be wired, and variants[1] to be adapted.

```
//load mainComponent.o
IDC mainComponent = loader.load(rootCompPath)

// loading components that provide requiredInterface
Variants variants[] = new Variants[compsPath.arrayLength]
for (int count = 0; count < compsPath.arrayLength; count++) {
    variants[count] = new Variants()
    variants[count].providedIntf = sec.requiredInterfaces[0].package
    variants[count].comp = loader.load(compsPath[count].string)
}
 // continuing with WIRE(ing)
 dana.rewire(mainComponent :> sec.requiredInterfaces[0].package,
     variants[0].comp :< sec.requiredInterfaces[0].package)

 MainComponent obj = new MainComponent() from mainComponent
 obj.method()

 // and ADAPT(ing)
 adapter.adaptRequiredInterface(mainComponent,
     sec.requiredInterfaces[0].package, variants[1].comp)
 obj.method()
```

Executing Advanced Adaptation Example

The execution of the advanced adaptation example follows from the execution of the basic adaptation. Since we have already compiled all components in *Practice2/Adaptation* using the command "*dnc .*", to execute the advanced adaptation example you simply execute *Adaptation2.o* with the command "*dana Adaptation2.o*". The complete source code of the advanced adaptation is in *Adaptation2.dn* file.

Assignment

The code for advanced adaptation only works if the root component requires only one interface, and the adaptation only occurs between the two fixed variants we know exist. For your assignment we expect you to expand the advanced adaptation example by adding more required interfaces to the root component, and more component variants for the *"Interface"*. Change the code that extract information from components to load all additional components and enable adaptation to be performed among multiple component variants.