

PantaRhei – An Execution Engine for Data Flow Networks II

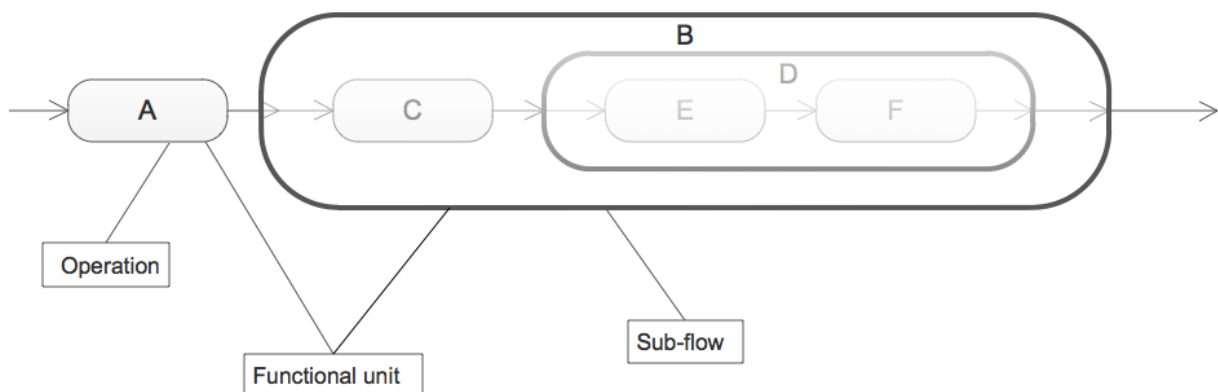
So far data flow networks consist of only a single level. Operations are connected to form a flow, and operations are atomic functional units. With regard to the data flow network they do not have a structure.

This of course does not scale well. Once the number of operations exceeds maybe 10 or 20 it's hard to understand how the network actually works.

A hierarchy of flows

To help this, data flow networks should allow for **sub-flows**. Once nesting is possible, scalability to hundreds or thousands of operations is no problem; nobody needs to see them all the time. Working with the data flow network can focus on a certain level of abstraction or a sub-flow.

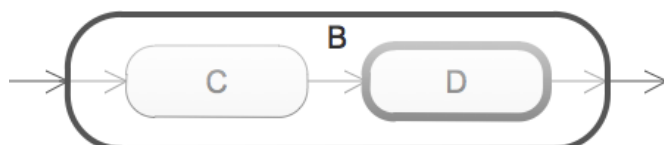
Data flow networks thus consist not only of operations but also of sub-flows. Both are **functional units**.



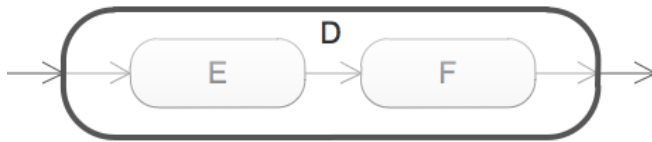
On the highest level of abstraction the above data flow network looks like this:



Since B is a sub-flow, you can zoom into it to reveal its details:



And then again, you can zoom into D:



On the outside sub-flows look and behave like operations. They have named input and output ports and transform data streaming in into data streaming out.

Inside, though, they look different from operations. They don't contain any program logic, they are not implemented using a 3GL. Their only purpose is to connect functional units into data flow networks, sub-flows.

Thus sub-flows can be defined by a table like data flow networks so far. Here for example the description of sub-flow B:

.In	C.In
C.Out	D.In
D.Out	.Out

As you can see, D – which is a sub-flow itself – is treated like any operation; when describing a (sub-)flow, all functional units are the same.

A multi-level data flow network most easily can be considered a collection of such tables. The above flow would be defined by three tables:

Top or root flow

.In	A.In
A.Out	B.In
B.Out	.Out

Sub-flow B

.In	C.In
C.Out	D.In
D.Out	.Out

Sub-flow D

.In	E.In
E.Out	F.In
F.Out	.Out

Single table definition of a flow hierarchy

Such multi-table definition might be easy to derive from a data flow network diagram. For execution purposes, though, it is less than optimal. Fortunately a single table representation of a multi-level flow can be generated from such a set of tables. Here it is for the above flow:

.In	A.In
A.Out	B/C.In
B/C.Out	B/D/E.In
B/D/E.Out	B/D/F.In
B/D/F.Out	.Out

Notice how sub-flows have been removed as functional units. Only operations are connected. But since sub-flows provide context they could not be discarded; instead they morphed into paths for operations and their ports.

The operation C nested in sub-flow B is addressed by the name B/C, and the output port of operation F nested in sub-flow D nested in sub-flow B is referred to by the name B/D/F.Out. That way functional units used in several sub-flows can be uniquely identified.

Changes to the execution engine

To accommodate hierarchical flows only small changes seem necessary to the API:

```
interface IMessage {
    IPort ToPort {get;}
    object Data {get;}
}

interface IStream {
    IPort FromPort {get;}
    IPort ToPort {get;}
}

interface IPort {
    string Path {get;}
    string OperationName {get;}
    string Name {get;}
    string Fullname {get;}
}
```

The execution engine needs to understand port descriptions with paths. A message flowing into B/D/F.In addresses the operation F with port In nested two levels deep in sub-flows.

Any message flowing out of the operation as a consequence of the incoming message needs to be assigned to the operation's sub-flow context B/D.