Definition

The framework is a library that controls the flow of events and data through well-defined interface points defined by user-written algorithm components. The user creates algorithm components using framework tools. These components can be arranged into an execution or data-flow sequence that is managed by the framework library.

Library Organization

<release_dir>/lib/IRIX6_XXX/

Main program object files. Batch, interactive versions.	framework.o iframework.o cframework.o	libframework.a libio_packages.a	Framework related libraries
Framework registration object files. One per framework package.	RegEMReco.o RegCPSReco.o RegCftExamine.o	libEMreco.a libCPSReco.a libCftExamine.a	Reconstruction/ Analysis package libraries

Must include one main program object file and a set of Reg object files when linking your executable. One Reg file must be linked for each package that you want available in the executable. The framework RCP controls which user package objects actually get created at run time.

Example User Package Class Definition

```
Class UserPackage : public fwk::Package, fwk::Process, fwk::Tag,
        fwk::RunInit
{
public:
        UserPackage(Context*);
        ~UserPackage();

        fwk::Result processEvent(edm::Event&);
        fwk::Result tagEvent(edm::Event&);
        fwk::Result runInit(const RunNumber&);

        void reinitialize(edm::RCP);
        void statusReport();
        void flush();
}
```

Framework provided classes

Interfaces available to user:

Generate:	generateEvent(WorkQueue&)
Decide:	makeDecision(const edm::Event&, WorkQueue&)
Filter:	filterEvent(const edm::Event&)
Process:	processEvent(edm::Event&)
Analyze:	analyzeEvent(const edm::Event&)
Dump:	dumpEvent(const edm::Event&)
Tag:	tagEvent(const edm::Event&)
Output:	outputEvent(const edm::Event&)
JobSummary:	jobSummary()
RunInit:	runInit(const RunNumber&)
RunEnd:	runEnd(const RunNumber&)

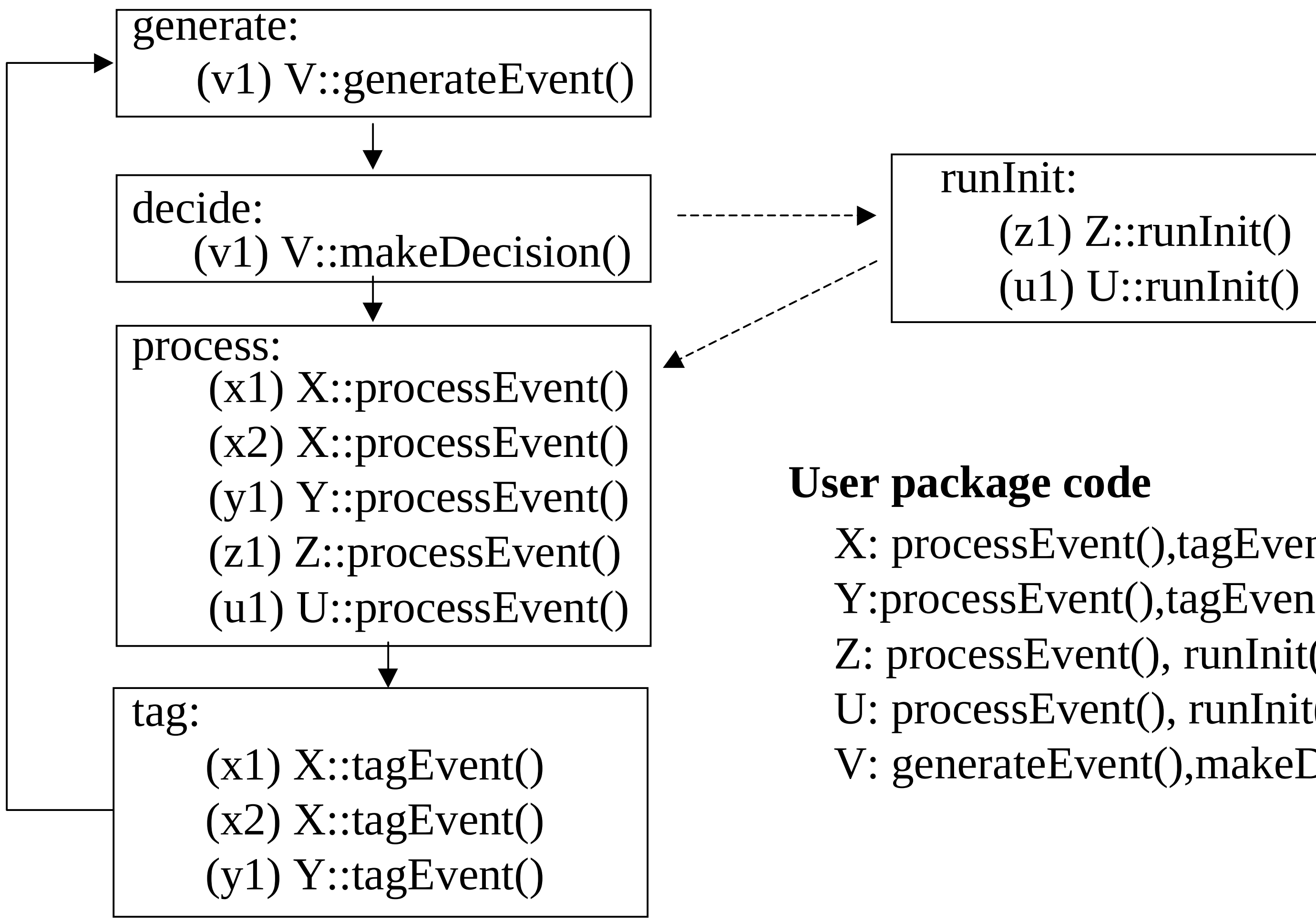
Framework provided package available for use in RCP files.

Controller: Groups together interfaces by name.
Controls event flow through the interface groups.
Is a package and implements a generic interface.
Can masquerade as any other available interface.
Interface name assigned by used in RCP file.
Can contain Controllers.

Inside the Framework

InterfaceNames="generate decide process tag runInit"
Packages="v1,x1, x2, y1, z1, u1"
Flow="generate decide process tag"

Event loop



User package code

X: processEvent(),tagEvent()
Y:processEvent(),tagEvent()
Z: processEvent(), runInit()
U: processEvent(), runInit()
V: generateEvent(),makeDecision()

Important Abstract Framework Components

Package:

Base class for user created, plug-in components.
Reconstruction/Analysis algorithms implemented by derived classes
Instance creation controlled by framework RCP file.
Instances exist for entire run of program
Automatically registers all the interfaces it implements

Interface:

Base class for framework call-out points.
Instances identified by string name.
Implemented by inheritance (similar to Package)
Specifies when and where data will be passed into a users code.
Basic component that framework uses to pass event into user packages
Event data flows through Interfaces.
A package typically implements several of these interfaces.

Framework RCP Configuration

***Defaults.rcp*:**

```
string InterfaceName = "process"
string Interfaces = "generator decide filter analyze process dump tag output
              jobSummary runEnd runInit"
string Flow = "generator decide filter process analyze dump tag output"
```

InterfaceName: We are configuring a controller. This is the interface name that the controller will be assigned. It will appear to the system as an “Process” interface.

Interfaces: A list of all the interfaces that that will be used in this run. The interface named are assigned dynamically when the program starts. The user can add new interface classes and assign them a name without changing the framework library.

Flow: An event will be directed through interfaces in the *order* specified by the flow. Any interface not in the flow is considered a free standing interface. This flow indicates that events will be passed through all the packages that implement processEvent() before being passed through all the packages that implement analyzeEvent().

High-level Framework RCP file example

User.rcp:

RCP FrameworkInfo = <defaults.rcp>
string Packages = "generate_events first_processor second_processor”

RCP generate_events = <generate.rcp>
RCP first_processor = <processor1.rcp>
RCP second_processor = <processor2.rcp>

Packages: Specifies the instance names of all the user package instances that should be created. The order is significant; interfaces will be distributed and executed in the order they appear in this parameter. The package type or class name is specified in the packages RCP file.

There must be one RCP line and RCP file present for each of the strings in the Packages parameter. The FrameworkInfo parameter tells where to get the framework RCP information from. The information in defaults.rcp can just be included in this RCP.

Nesting

Controllers can be placed into controllers

User.rcp

string InterfaceName = "process"
string Interfaces = "generator decide process jobSummary runEnd runInit"
string Flow = "generator decide process"
string Packages = "gen group1 group2"
RCP gen = <gen.rcp>
RCP group1 = <group_even.rcp>
RCP group2 = <group_odd.rcp>

group_even.rcp

string PackageName = "Controller"
string InterfaceName = "process"
string Interfaces = "filter analyze"
string Flow = "filter analyze"
string Packages = "filter"
RCP filter = <filter_even.rcp>

group_odd.rcp

string PackageName = "Controller"
string InterfaceName = "process"
string Interfaces = "filter analyze"
string Flow = "filter analyze"
string Packages = "filter"
RCP filter = <filter_odd.rcp>

filter_even.rcp

string PackageName = "Filter"
string type = "even"

filter_odd.rcp

string PackageName = "Filter"
string type = ”odd"

Batch Framework

```
<d02ka> D0reco.x -show
CPSDigiPkg      $Name: v00-05-03 $
CPSReco $Name: v00-05-03 $
CalClusterReco  $Name: v00-05-03 $
CalWeight       $Name: v00-05-03 $
CftClusterPkg   $Name: v00-05-03 $
Controller      $Name: v00-05-03 $
DropChunks      $Name: v00-12-04 $
DumpEvent       $Name: v00-12-04 $
EMReco  $Name: v00-05-03 $
FPSClusterPackage       $Name: v00-05-03 $
FpsDigiPackage  $Name: v00-05-03 $
GtrFindPkg      $Name: v00-05-03 $
JetReco $Name: v00-05-03 $
MissingETReco   $Name: v00-05-03 $
MuoHitReco      $Name: v00-05-03 $
MuoSegmentReco  $Name: v00-05-03 $
ReadEvent       $Name: v00-12-04 $
SMT1DPosPack    $Name: v00-05-03 $
SMTClusterPack  $Name: v00-05-03 $
SMTInterfacePack        $Name: v00-05-03 $
SMTLocaltoGlobalTransPack       $Name: v00-05-03 $
SmtClusterPkg   $Name: v00-05-03 $
TauReco $Name: v00-05-03 $
VertexReco      $Name: v00-05-01 $
VertexSelect    $Name: v00-05-01 $
WriteEvent      $Name: v00-12-04 $
geometry_management     $Name: v00-05-03 $
<d02ka>
```

Interactive Framework Features

Link using iframework.o
Execute with "-i" option
Still quite a primitive terminal interface
Framework internals allow for many different UIs to exist

Commands

	quit
	start
	help
Wait for user input after current interface completes →	pause
	resume
One interface at a time →	step
One event at a time →	next
	flush
Print all the available packages →	show
	config
	time
Print the package instance within controller →	ls
	cd
Dump the RCP for a package →	rcp
	report (status)
Deactivate/Activate a package instance →	state
Print the current package/interface →	where

Future Plans