



EFDA

EUROPEAN FUSION DEVELOPMENT AGREEMENT

Task Force
INTEGRATED TOKAMAK MODELLING

4/05/2009

ISIP tools training

B. Guillerminet for the ISIP team

<https://portal.efda-itm.eu/portal/authsec/portal/itm/ISIP>
isip@mail.efda-itm.eu

ITM framework

- **What is it?**
 - Set of tools: design, running, visualization, post-processing ... tools
 - Set of resources: gateway, euforia, data
- **Status-Road map**
- **Contents of the training**
- **Introduction to KEPLER**
- **KEPLER in practice**
- **Advanced use of KEPLER**

ITM framework: what is it?

- **Set of tools:**
 - Data:
 - Standard naming: **CPO** (comprehensive description of a Tokamak/plasma through the CPOs)
 - Standard access: **UAL** (local, remote, GRID, HPC). Hidden data storage. jTraverser, Jscope
 - Experimental data: **exp2ITM**
 - Parameters setting of a simulation
 - Shot, time step, duration, Heating systems, ...: **ISE**
 - Workflow/orchestration: **KEPLER**
 - Integration of codes: **FC2K**, **WS2K**
 - Monitoring: **ISE** (UAL data), **Migrating desktop** (jobs), **Kepler actors** (wf data)
 - Visualization: **ISE**, **Numplot**, **Visit**, **Matlab**, **Scilab**
 - Post-processing: **Numpy**, **Matlab**, **Scilab**
- **Set of resources:**
 - Cluster at Portici: **Gateway**
 - GRID & HPC computers: **EUFORIA**
 - **HPC-FF** at Juelich

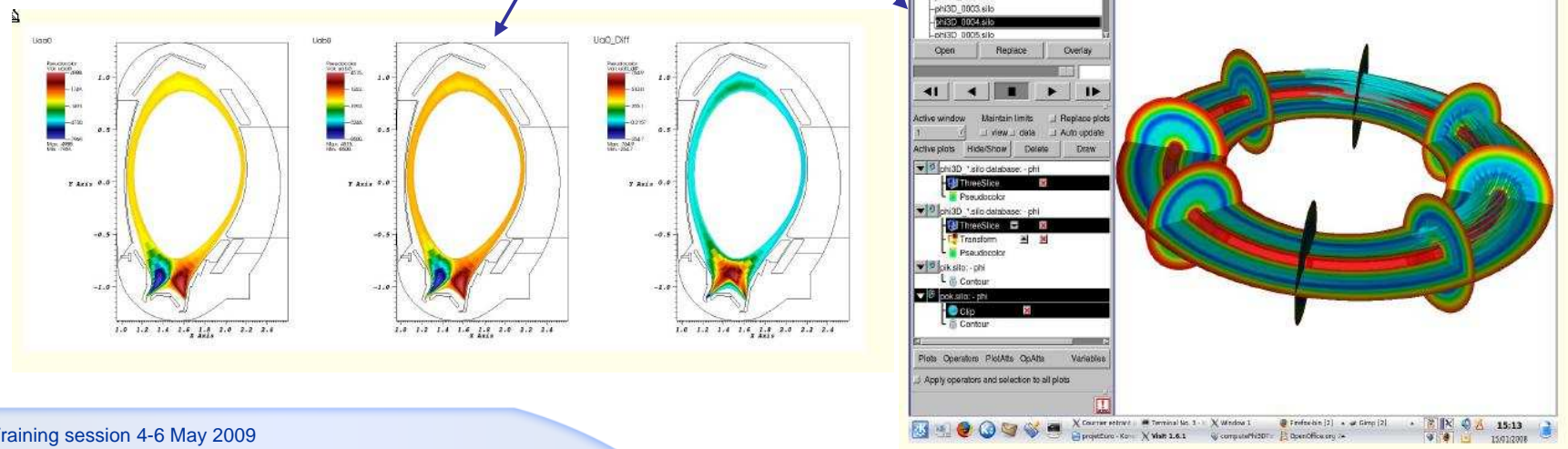


Status-Road map

- **Current version: v1.0** (released on December 08)
 - Data structure: 4.06d
 - UAL (CPO & time slice)
 - Local on the gateway: Fortran (g95 & pgi), C, C++, Java (1.5 or +), Matlab.
 - GRID: Fortran (g95) March 2009
 - HPC-FF: Fortran (g95 & pgi) 28 April 2009
 - Data:
 - Shot 3, run 1
 - Experimental data: exp2ITM for TS & JET
 - JET: ?
 - Tore Supra: 40000
 - Data storage:
 - MDSplus
 - HDF5
 - Memory (thread version => ok for multicores but not available for multi-nodes)
 - ISE (ITM Simulation Editor): released in June 2008 but due to change in the run number management, it must be reengineered
 - KEPLER: release 1.0, one UALinit & one UALcollector
 - FC2K: version 1.4 no time slice
 - Visualization: using KEPLER actors
 - Matlab
- **Additional tools:**
 - Numplot: standalone tool (available now)

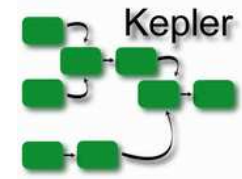
Status-Road map

- **version: v1.1** (released on May/June 09)
 - Data structure: 4.06d
 - KEPLER: release 1.0, new version of UALinit & UALcollector (occurrences + several UALinit or UALcollector)
 - FC2K: version 1.4b with time slice
- **version: v1.2** (released on July 09)
 - Data structure: 4.07
 - ISE (ITM Simulation Editor)
 - Visualization: using numpy actors + VISIT



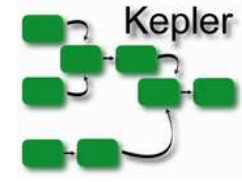
Training

- **Contents:**
 - Data (presented by F Imbeaux):
 - **CPO**
 - **UAL**
 - **exp2ITM**
 - Workflow/orchestration: **KEPLER**
 - Integration of codes: **FC2K**
 - **ISE** (not presented but ... slides available)
 - **Numpy, Visit, Matlab** (not presented but ...)



Introduction to KEPLER

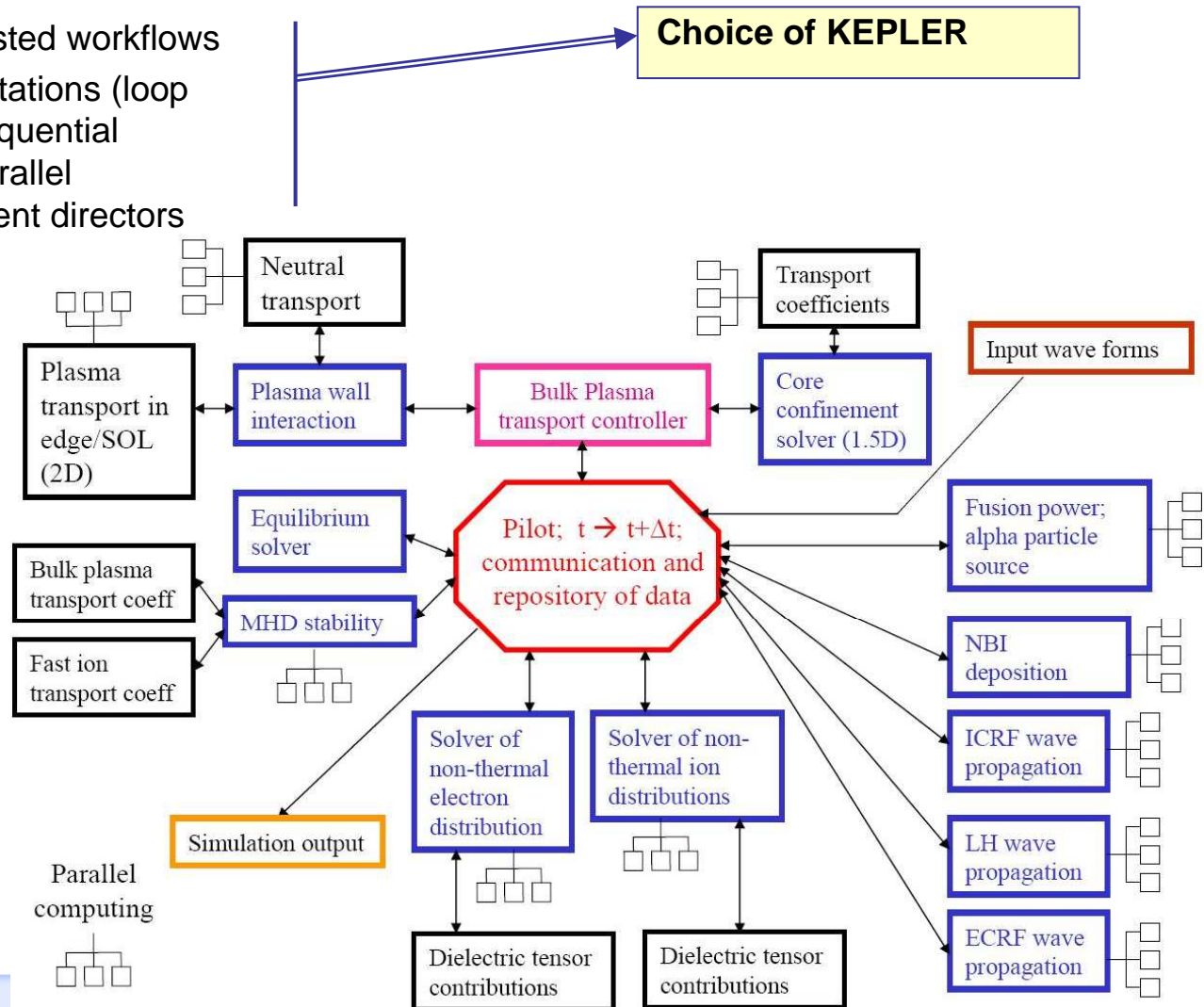
- **Workflow design:**
 - Why an orchestration tool
 - Introduction to KEPLER: terminology
 - A few actors
 - Computation model & directors
 - Fusion workflow:
 - Building simple workflows:
 - getting a CPO & plotting some data
 - Reading & updating a CPO

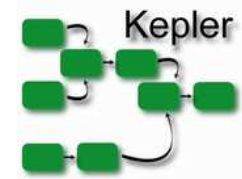


Orchestration tool

- Fusion simulation:**

- Complex workflow => nested workflows
- Different model of computations (loop on model time, solver, sequential processing, branches, parallel computation ...) => different directors



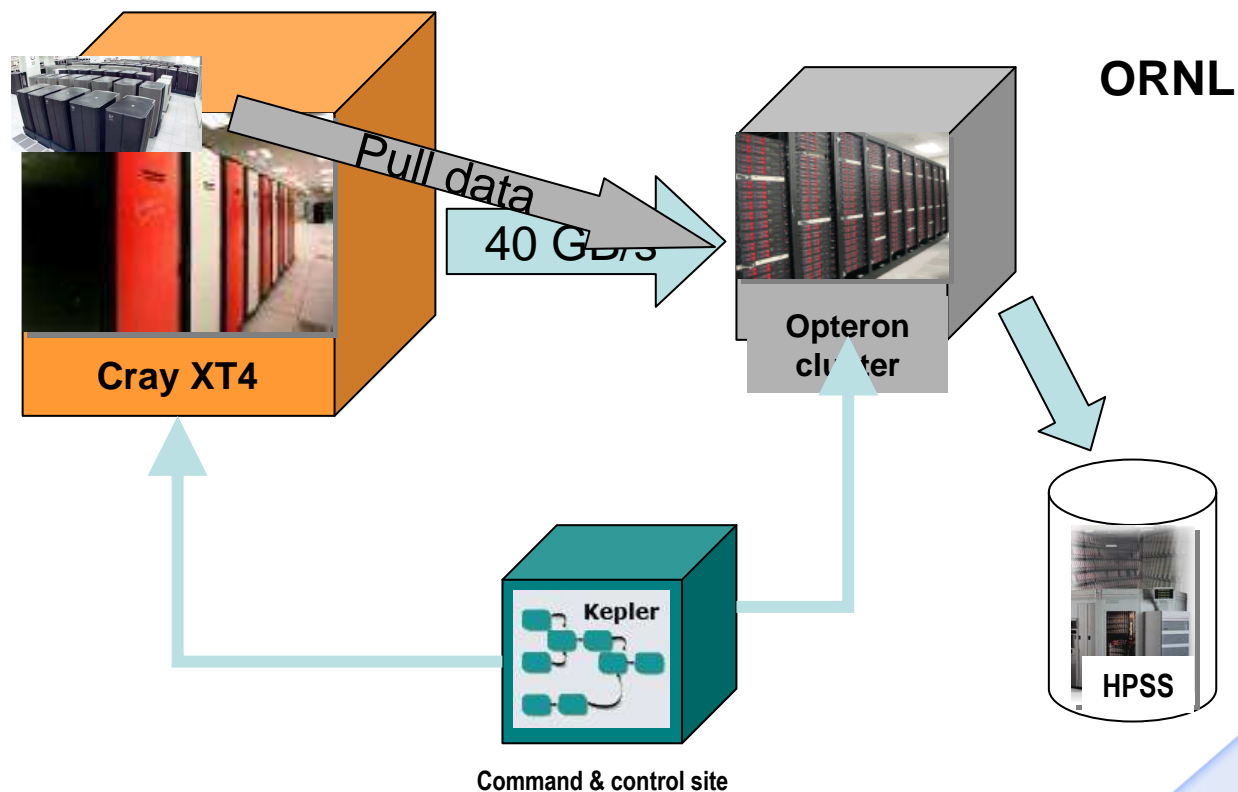


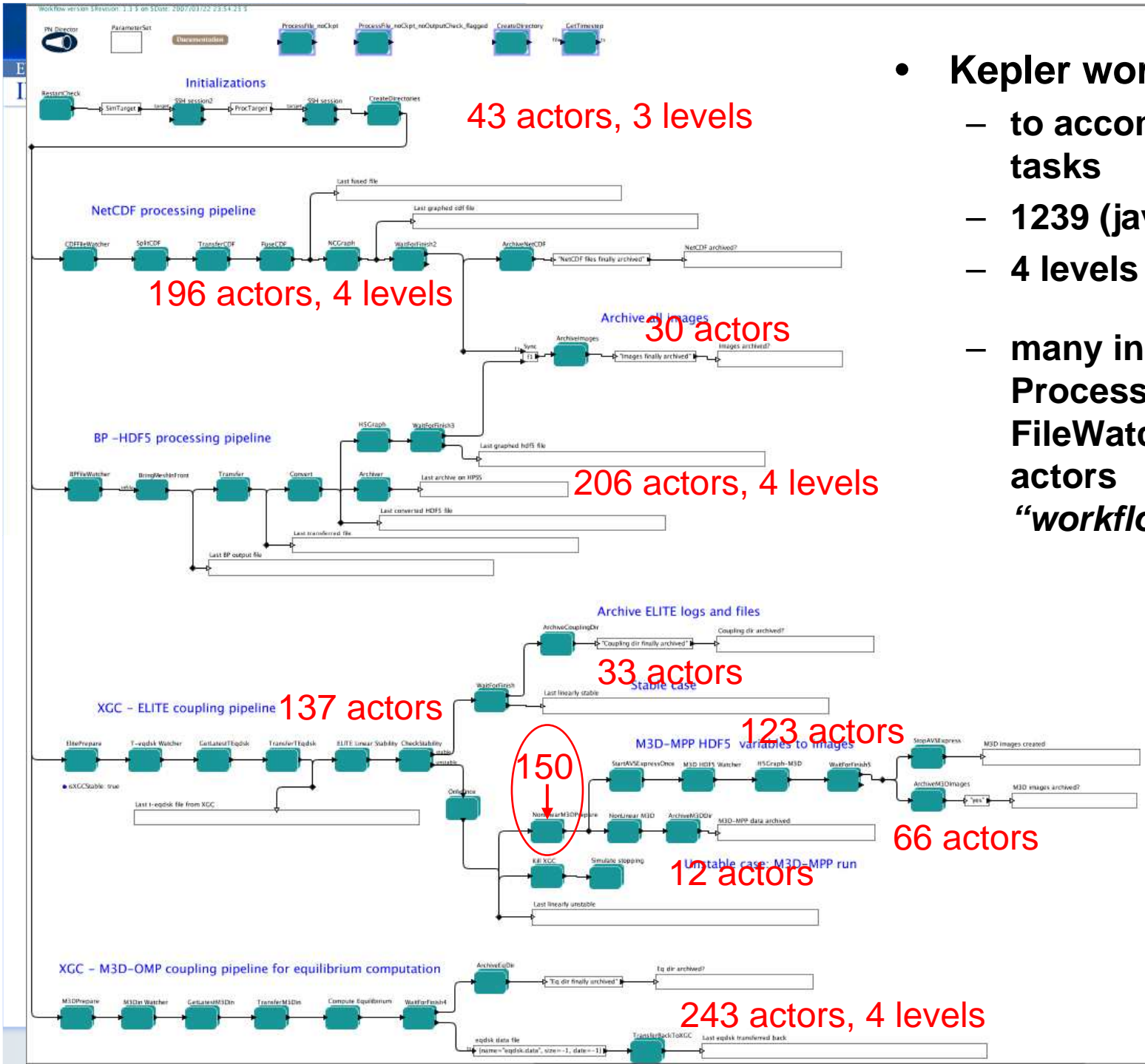
KEPLER

Based on Ptolemy II (Berkeley), San Diego, world widely used, friendly tool

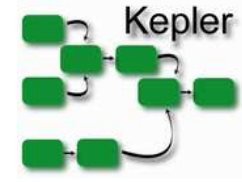
Used for:

- Simplify and automate the workflow (SDM, CPES ...)
- Coarse grained programming (one instruction is a big chunk): assembling components
- Graphical design
- Mixing complex models of computation





- **Kepler workflow**
 - to accomplish all these tasks
 - 1239 (java) actors
 - 4 levels of hierarchy
 - many instances of ProcessFile and FileWatcher composite actors
 - “workflow templates”



KEPLER

2 main usages

=>design of workflow

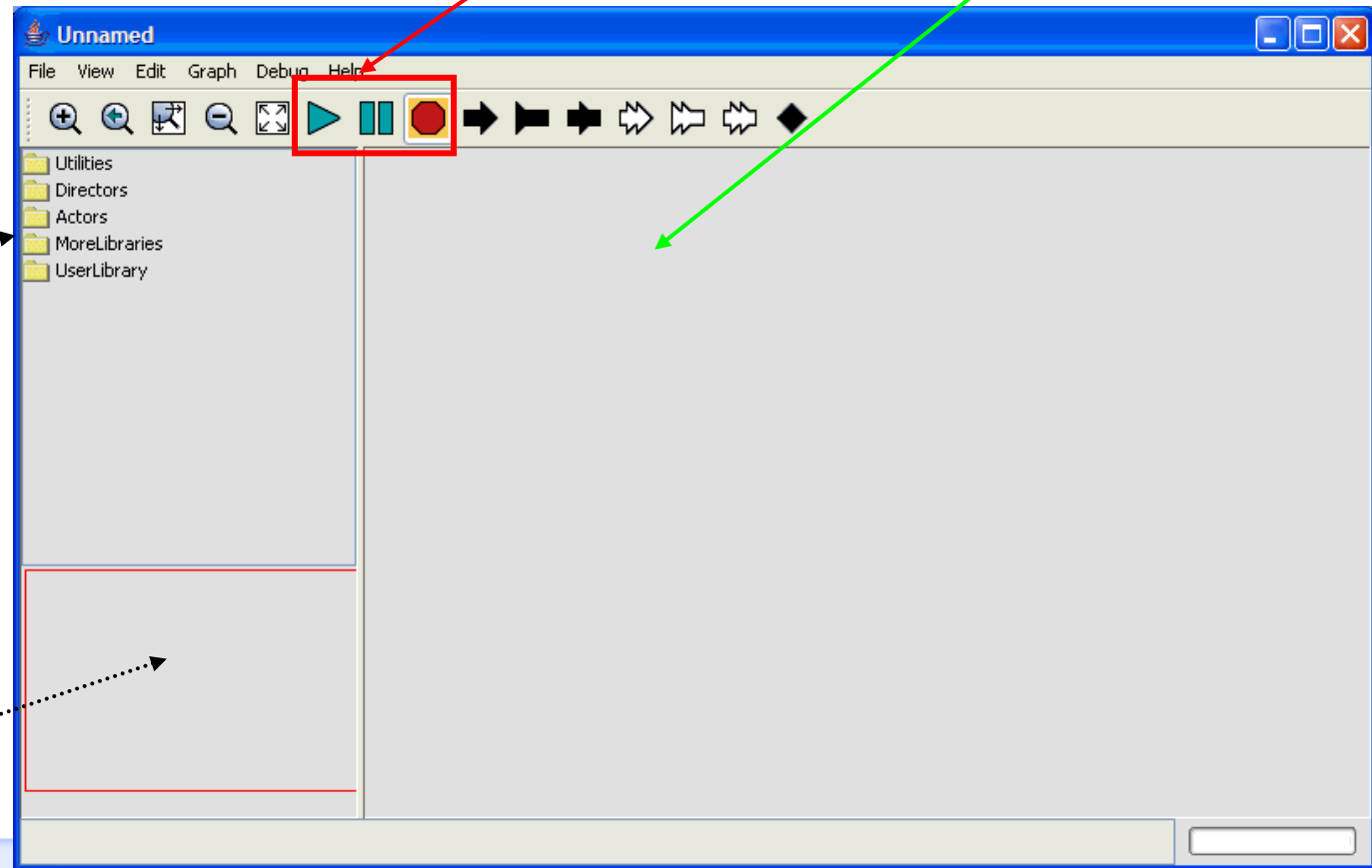
=>execution of the workflow

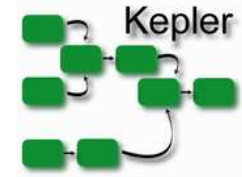
Run control buttons

Model-building area

Library of components

Navigation area

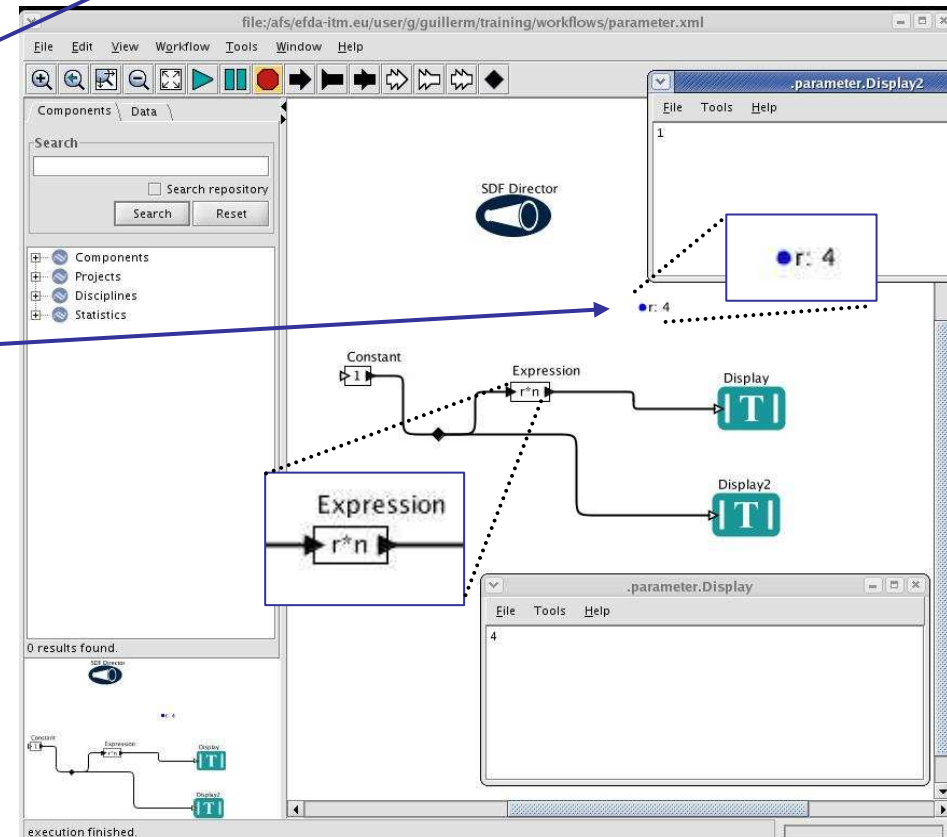
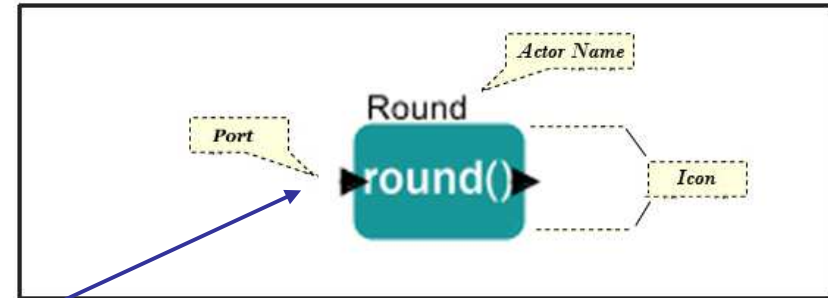


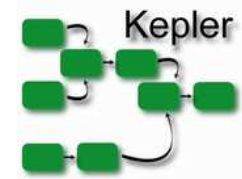


KEPLER

Terminology:

- **Manager**
 - **Single (suspend/resume...)**
- **Directors**
 - **Scheduler of the actors (components)**
- **Actors**
 - **Ports: Input & Output of an actor**
 - **Parameters: Shared by the actors**
 - **Will be detailed in the next session**

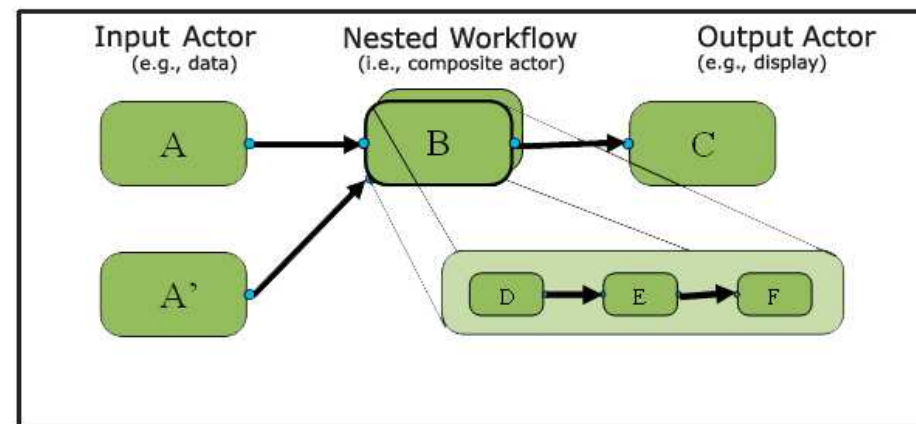




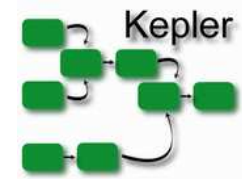
ACTORS

Components of the workflow:

- **Actors could be nested**
- **Fusion codes: Helena, Mishka, Orbit ...**
- **Categories**
 - **sources**
 - Const, string, clocks
 - Ramp, sinewave, shell, ...
 - **sinks**
 - Display, XYZplotter, timedplotter
 - Recorder, ...
 - **Array**
 - Arrayextract, arrayminimum, arraysort, ...
 - **Conversion**
 - Complextopolar, stringtoxml, ...
 - **Flow control**
 - Switch, Sampledelay, Exit, ...
 - **I/O**
 - Filereader, writer, ...
 - **Math**
 - Average, ...
 - **Matrix**
 - **Random**
 - **Signal processing**



- **Creation of actors will be detailed in the next session**
- **See “ActorReference.pdf”**



Computation models & directors

Directors

- Tell the actors when it has to produce its output = order in which they should execute
- 5 basic schedulers (directors):
- **DATA DRIVEN**
 - **SDF**
 - Synchronous data flow: fairly simple, sequential workflow
 - **DDF**
 - Dynamic data flow = SDF with loops
 - **PN**
 - Parallel processing on distributed computing systems
- **TIME DRIVEN**
 - **CT**
 - Continuous time driven
 - **DE**
 - Discrete event: modelling time
- **MIXING DIRECTORS**
- **Beware: workflows and actors could depend on the director**

SDF Director



PN Director

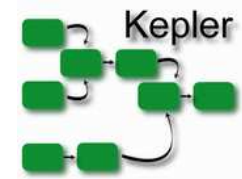


CT Director

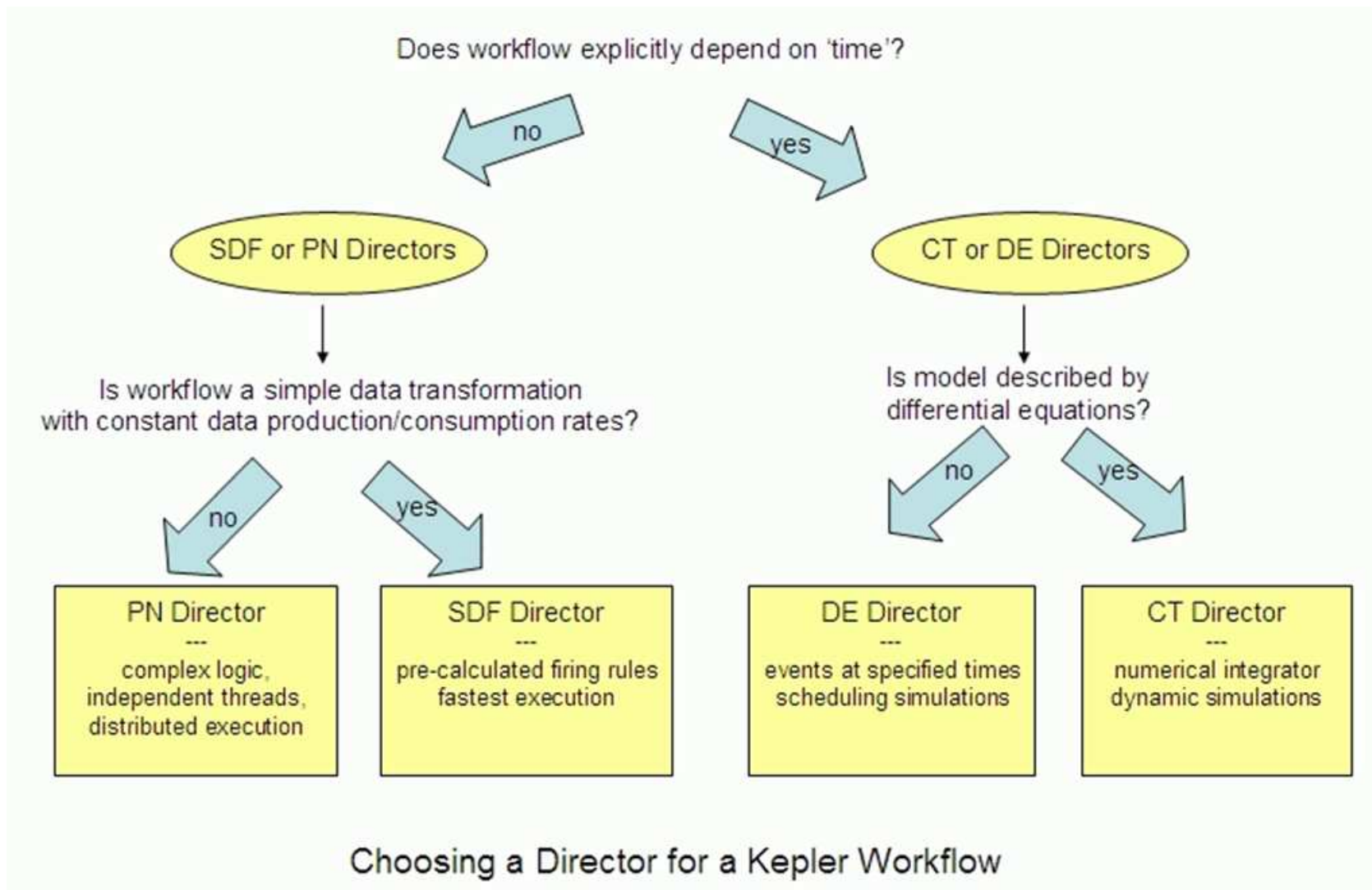


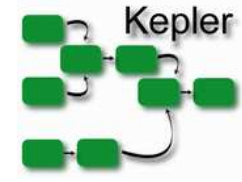
DE Director





How to choose a director



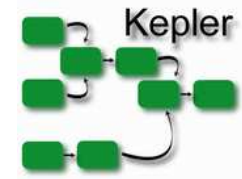


Design of a workflow (1)

Design

1. Choose your actors (for instance constant, addsubtract & display: use the search cmd)
2. Drag & drop them in the design area
3. Connect the actors => draw a link between input & output port
4. Define the director (SDF) and its parameters (number of iterations)

iterations:	2
vectorizationFactor:	1
allowDisconnectedGraphs:	<input type="checkbox"/>
allowRateChanges:	<input type="checkbox"/>
constrainBufferSizes:	<input checked="" type="checkbox"/>
period:	0.0
synchronizeToRealTime:	<input type="checkbox"/>
timeResolution:	1E-10
class:	ptolemy.domains.sdf.kernel.SDFDirector
semanticType00:	urn:lsid:localhost:onto:1:1#Director
semanticType11:	urn:lsid:localhost:onto:2:1#Director



Creating an actor in 5 steps

1. Identify your CPOs in & out
2. Turn code into a subroutine with CPOs as argument
3. To make sure the code handles the CPOs correctly, run it in a “testbed”
4. Make a library of your routine (compilation with the `-fPIC` option is essential).
5. Run FC2K

Prerequisite:

1. Install a private version of Kepler in your directory (available on `/afs/efda-itm.eu/project/switm/kepler/4.06d/kepler.tar`)

```
>cp /afs/efda-itm.eu/project/switm/kepler/4.06d/kepler.tar $HOME  
>tar xvf kepler.tar
```
2. Set the environment variables with ITMv1

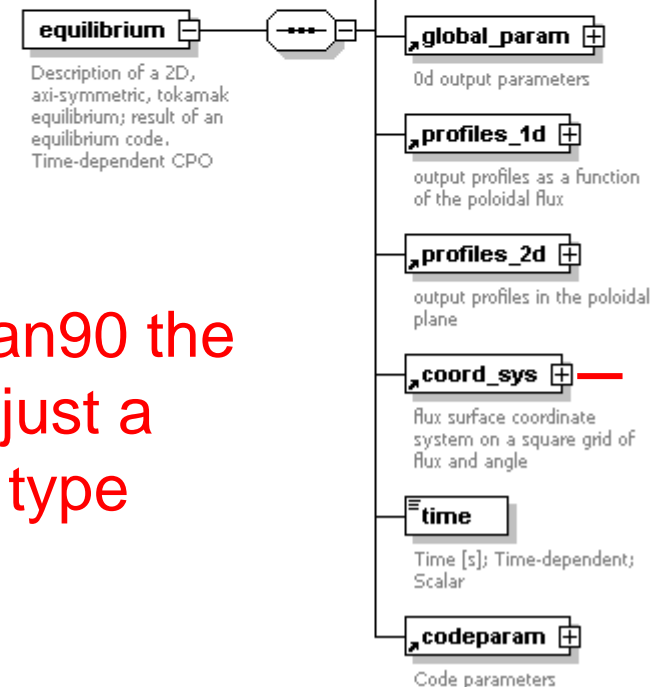
```
>source /afs/efda-itm.eu/project/switm/scripts/ITMv1 kepler test 4.06d
```

 (where kepler is the directory of your KEPLER version; use public for the standard KEPLER version)

Step 1

- A data structure describes how the data in a CPO are organised.
- The data structures are defined with XML schemas.
- The code developer only needs to know about the organisation itself.
- ITM tools automatically generate type declarations for inclusion in a code so that access to data is readily available.

Example: equilibrium CPO



In Fortran90 the CPO is just a derived type

Step 2

- **Turn code into a subroutine with CPOs as argument (example of a code which extract the pressure from the equilibrium CPO).**

```
subroutine cpo2ip(equi_in, ip)
  use euitm_schemas
  use eulTM_routines
  implicit none
  integer,parameter :: DP=kind(1.0D0)
```

Input arguments can be CPOs, integer, floating point, ...: single or array

All the ITM type declarations are included here

```
type (type_equilibrium),pointer :: equi_in(:)
integer :: ip(20)
```

Declaration of the equilibrium CPO

```
write(*,*) 'pressure: ',equi_in(1)%profiles_1d%pressure
do i=1,20
  ip(i) = int(equi_in(1)%profiles_1d%pressure(i))
enddo
write(*,*) 'ip:',ip
return
end subroutine cpo2ip
```

Get the pressure and fill a local array

Step 3

- To make sure the code handles the CPOs correctly run it in a “testbed”; F90 example:

```
program test_bed
use euITM_schemas
use euITM_routines

implicit none

type (type_equilibrium),pointer :: cptest(:)

character(len=5)::treename

integer :: idx, shot, run

interface
  subroutine mycode(cptest)
    use euITM_schemas
    type (type_equilibrium),pointer :: cptest(:)
  end subroutine mycode
end interface

shot = 180
run = 1
treename = 'euitm'

call euitm_open(treename,shot,run,idx)

call euitm_get(idx,"equilibrium",cptest)

call mycode(cptest)

stop
end
```

Example for
“mycode”

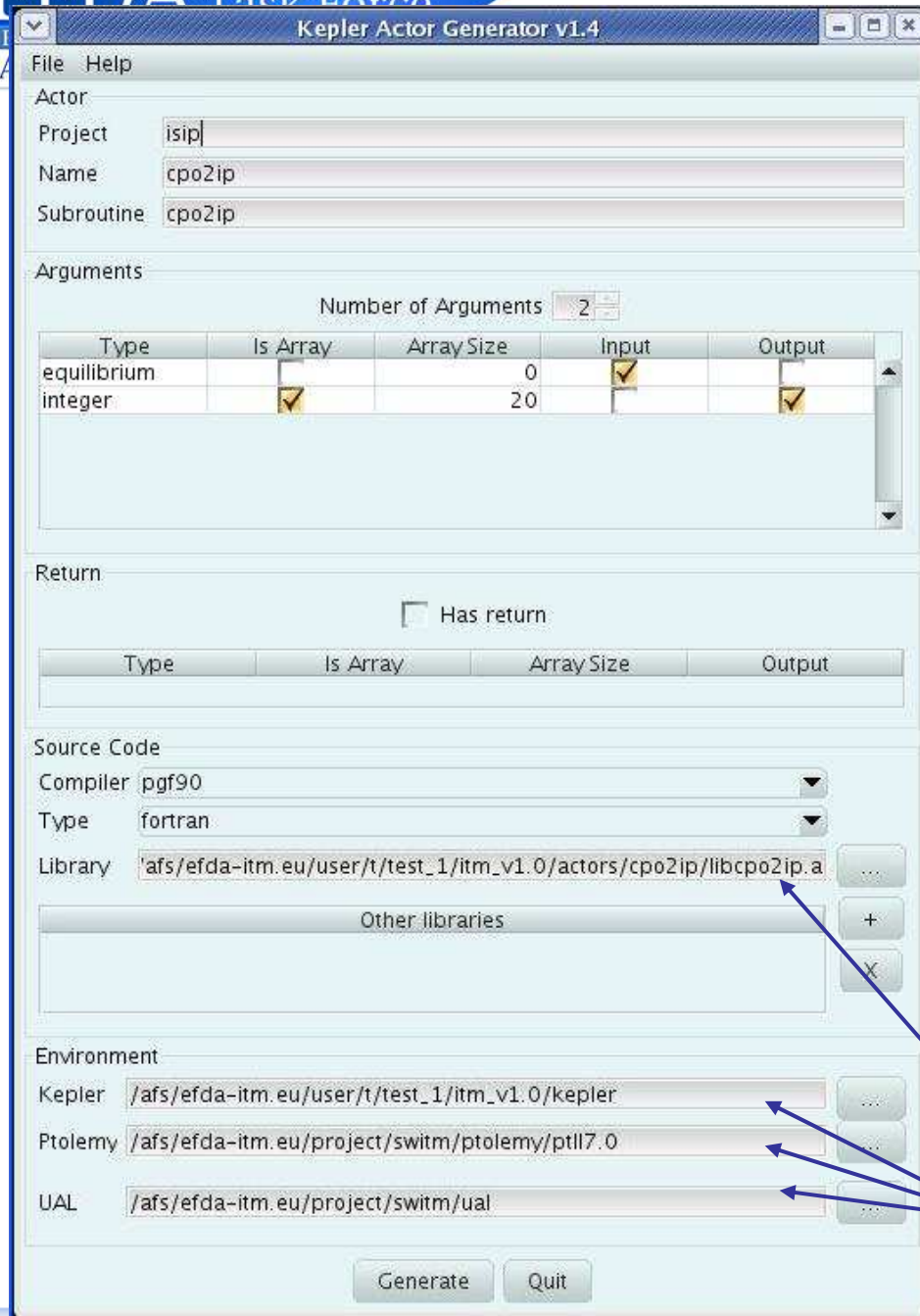
Specify
“mycode”

Run “mycode”

- One can start with a very simple test_bed program.
- Shown on the left is an example with only an equilibrium CPO as input
- Programs of this type with their Makefile are available on the Gateway for copying

Step 4

- Once the code runs correctly in the test bed it is time to make a Kepler actor of it.
- Make a library of your routine (either static, mylib.a, or dynamic, mylib.so; compilation with the `-fPIC` option is essential). Example for `cpo2ip.f90`:
 - `F90=pgf90`
 - `COPTS= -r8 -Mnosecond_underscore -fPIC`
 - `LIBS= -L/afs/efda-itm.eu/project/switm/ual/lib -IUALFORTRANInterface_pgi`
 - `INCLUDES= -I/afs/efda-itm.eu/project/switm/ual/include/amd64_pgi`
- `all: libcpo2ip.a`
- `libcpo2ip.a:cpo2ip.f90`
- `$(F90) $(COPTS) -c cpo2ip.f90 ${INCLUDES} $(LIBS)`
- `ar -rv libcpo2ip.a cpo2ip.o`
- `clean:`
- `rm *.a *.o`



Step 5

- **Run the KEPLER actor generator script: FC2K**
- isip = folder for description storage
- Name of the actor
- Name of the subroutine (no underscore allowed)
- Input & output arg.
- g95, pgf90, C, C++
- Your code
- Use \$KEPLER, \$PTII & \$UAL



• Add the libraries

Return Has return

Type	Is Array	Array Size	Output
------	----------	------------	--------

Source Code

Compiler `pgf90`

Type `fortran`

Library `/afs/efda-itm.eu/imp5/user/eriksson/orbit/orbib.a`

Other libraries

- `/afs/efda-itm.eu/project/switm/pspline/amd64_pgi/lib/portlib.a`
- `/afs/efda-itm.eu/project/switm/pspline/amd64_pgi/lib/libpspline.a`
- `/afs/efda-itm.eu/project/switm/pspline/amd64_pgi/lib/portlib.a`
- `/afs/efda-itm.eu/project/switm/pspline/amd64_pgi/lib/libcdf_dummy.a`
- `/afs/efda-itm.eu/project/switm/pspline/amd64_pgi/lib/libezcdf.a`
- `/afs/efda-itm.eu/imp5/user/eriksson/plr/plrlib.a`

Environment

Kepler `/afs/efda-itm.eu/imp5/user/eriksson/kepler2/kepler`

Ptolemy `/afs/efda-itm.eu/imp5/user/eriksson/ptolemy/ptll7.0`

UAL `/afs/efda-itm.eu/project/switm/ual`

Generate Quit



- **The actor generator creates a “wrapper” for the code, it manages the call to UAL etc.**
- Stored in (where xxx is the actor name) :
- \$KEPLER/src/cpp/itm/xxx (Makefile, FortranWrap.f90, libxxx.a, libxxx.so, ...). “>make” & “>kepler”
- \$KEPLER/src/eu/itm/xxx (xxx.java & JavaJniCall.java)
- \$KEPLER/kar/actors (xxx.kar)
- \$KEPLER/lib (libxxx.so)
- \$KEPLER/build/src/cpp/itm/xxx copy of \$KEPLER/src/cpp/itm/xxx
- \$KEPLER/build/src/eu/itm/xxx copy ...
- \$HOME/.kepler keeps a cache of your actors!!

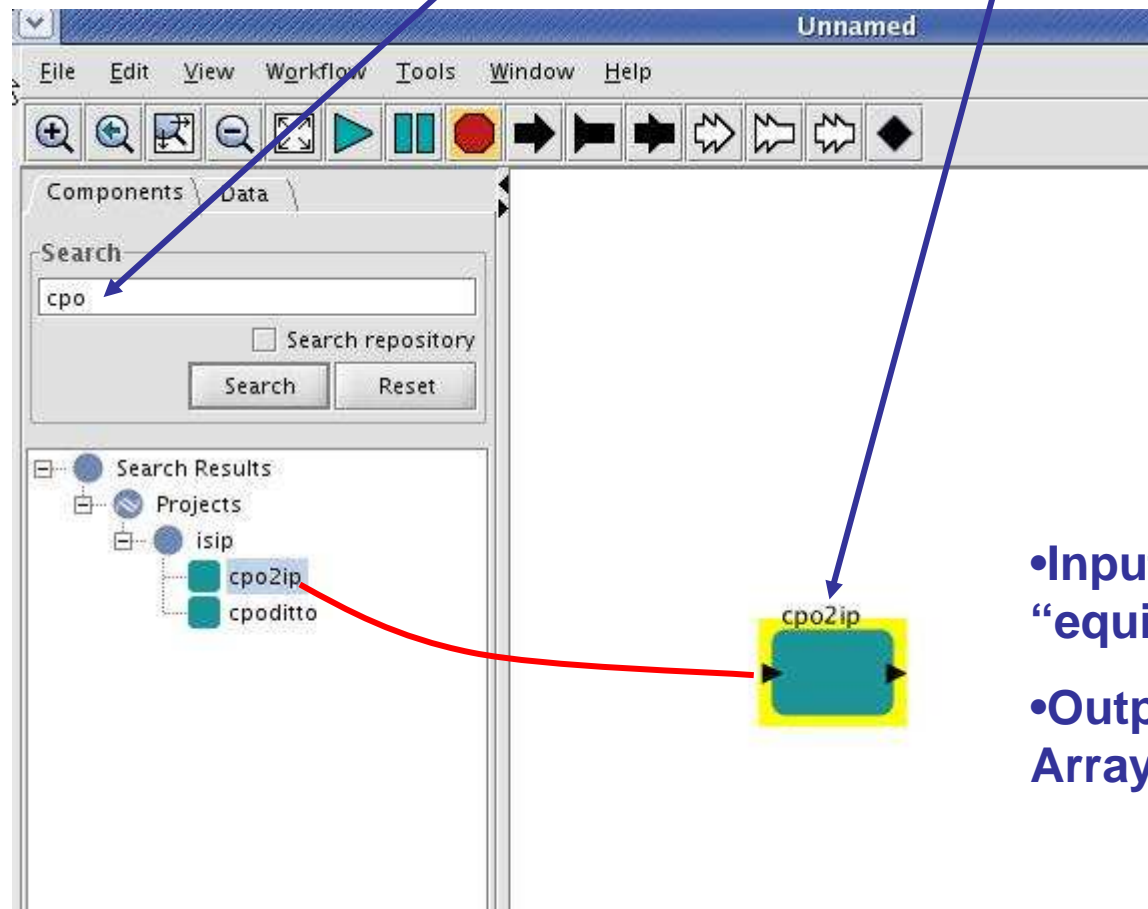
TIPS

- **A few tools to share actors:**
- **>rmactor xxx** (remove the actor xxx from your \$KEPLER version)
- **>getactor xxx** Get the actor xxx from \$KEPLER and build xxx.tar (tar tvf xxx.tar to look at its contents)
- **>putactor xxx** Put the actor contained in xxx.tar into \$KEPLER
 - Then update your KEPLER version by:
 - **cd \$KEPLER**
 - **ant buildkarlib**
- **These scripts are in /afs/efda-itm.eu/project/switm/scripts**

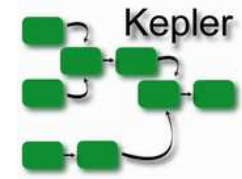


Building your workflow

Using KEPLER (“>kepler”), look for your actor and copy it to the design area



- Input port:
“equilibrium” CPO
- Output port:
Array of integer



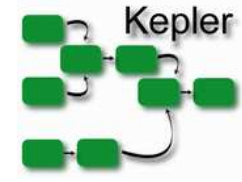
Fusion workflow (1)

Design

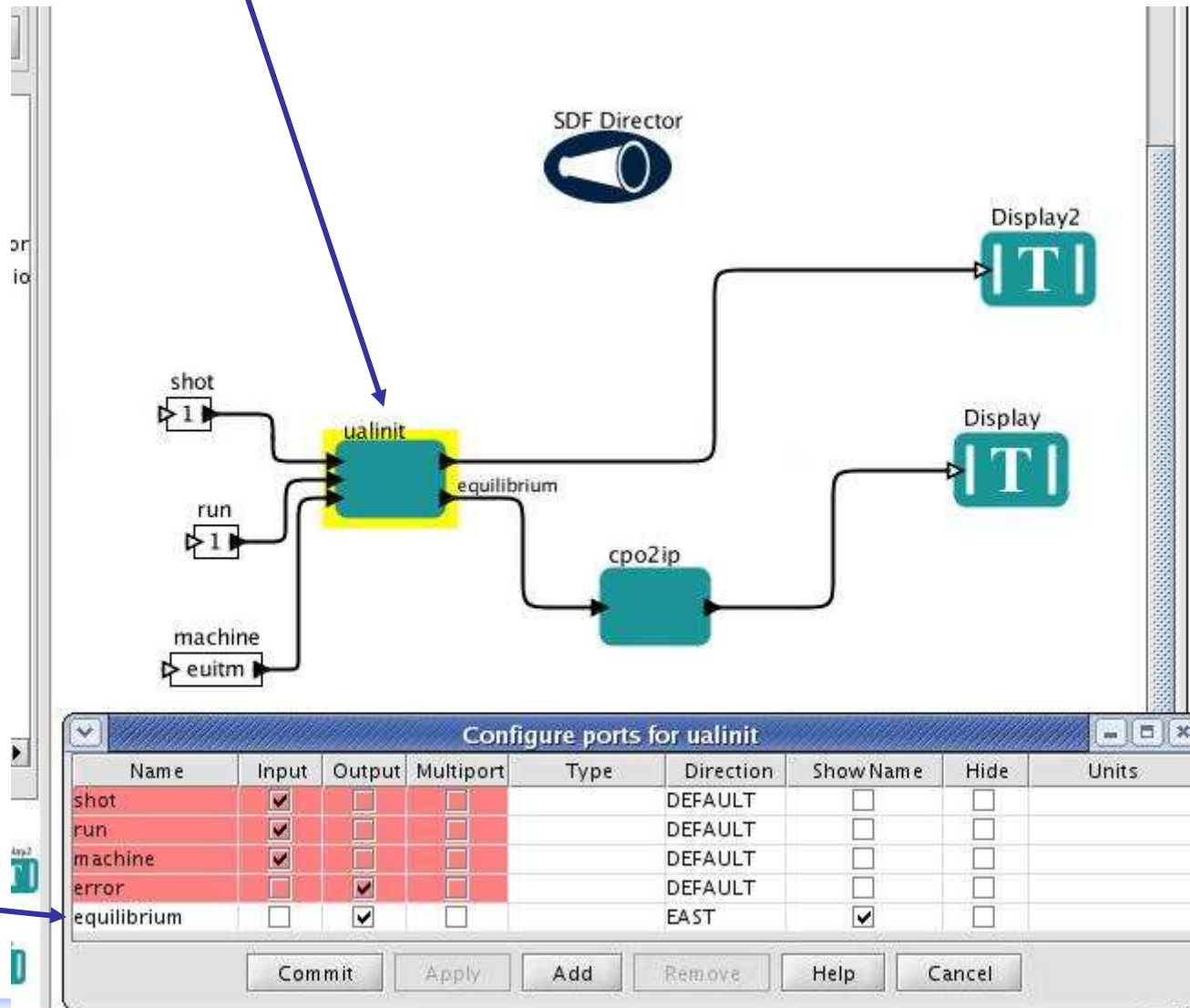
1. Connection with the UAL: reading the database and storing in memory UALinit
2. Insert your actor/workflow
3. Store the simulation in the database: UALcollector

The screenshot shows the Kepler workflow editor interface. On the left, the 'Components' panel displays search results for 'ual', including 'ualinit' and 'ualcollector'. In the main workspace, a 'test1 Actor' is shown with two components: 'ualinit' and 'ualcollector'. A context menu is open over the 'ualinit' component, with 'Configure Ports' selected. Below the workspace, the 'Configure ports for ualinit' dialog box is open, displaying a table of port configurations.

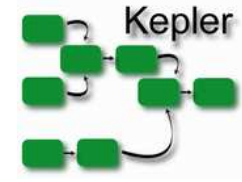
Name	Input	Output	Multiport	Type	Direction	Show Name	Hide	Units
shot	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		DEFAULT	<input type="checkbox"/>	<input type="checkbox"/>	
run	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		DEFAULT	<input type="checkbox"/>	<input type="checkbox"/>	
machine	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		DEFAULT	<input type="checkbox"/>	<input type="checkbox"/>	
error	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>		DEFAULT	<input type="checkbox"/>	<input type="checkbox"/>	
equilibrium	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>		DEFAULT	<input type="checkbox"/>	<input type="checkbox"/>	



Add the actor which connect to the ITM database



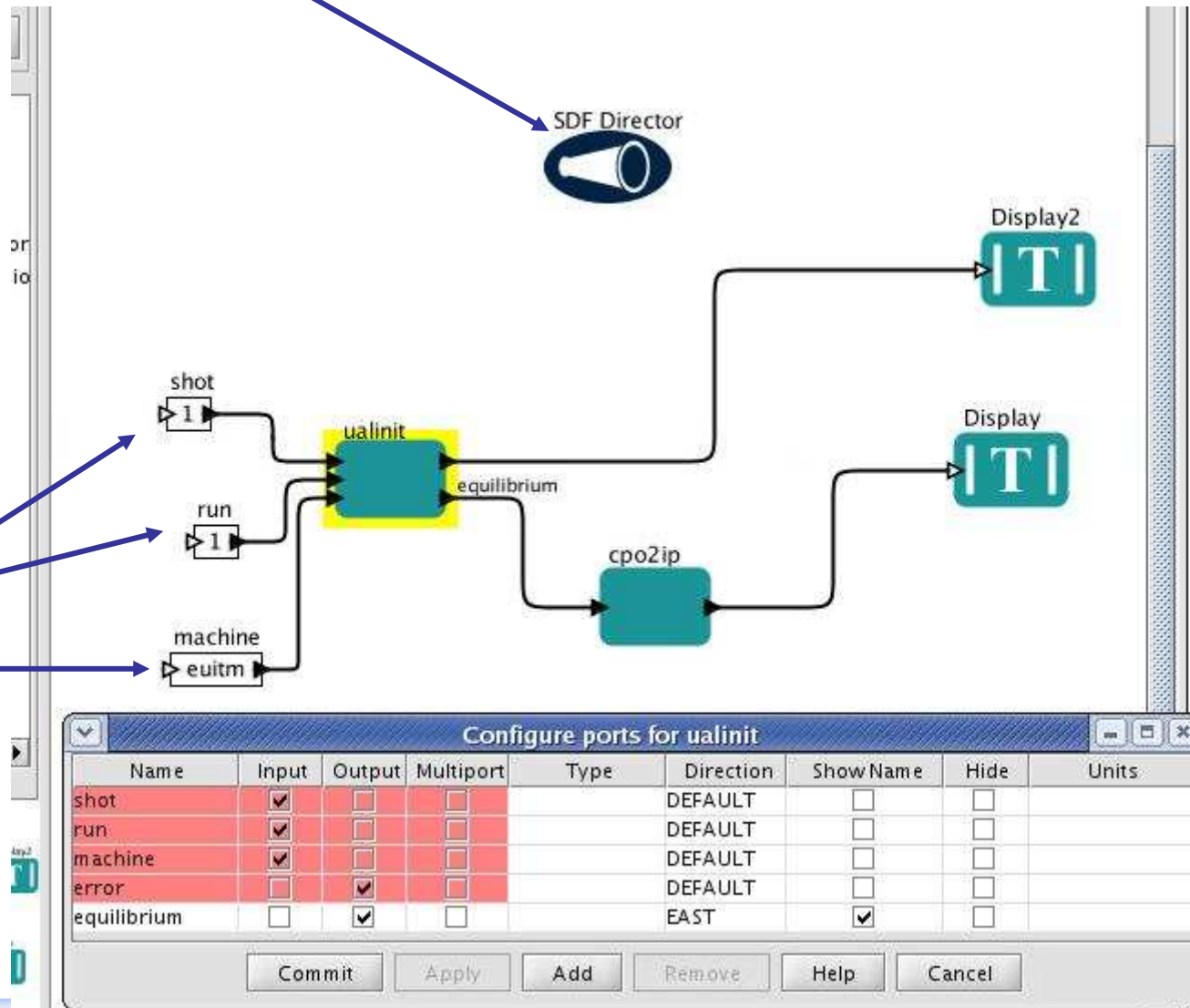
Add the output CPOs



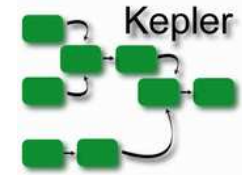
Add the Director (Synchronous Data Flow)

Add the inputs:

- Shot
- Run
- machine



Configure ports for ualinit									
Name	Input	Output	Multiport	Type	Direction	Show Name	Hide	Units	
shot	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		DEFAULT	<input type="checkbox"/>	<input type="checkbox"/>		
run	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		DEFAULT	<input type="checkbox"/>	<input type="checkbox"/>		
machine	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		DEFAULT	<input type="checkbox"/>	<input type="checkbox"/>		
error	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>		DEFAULT	<input type="checkbox"/>	<input type="checkbox"/>		
equilibrium	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>		EAST	<input checked="" type="checkbox"/>	<input type="checkbox"/>		



Run the workflow

Kepler Actor Generator v1.4b
 Actor: isip
 Project: cpo2ip
 Name: cpo2ip
 Subroutine: cpo2ip

workflow_1.xml

```

<test_1@enea144 ~>create_user_itm_dir test 4.06d
<test_1@enea144 ~>source set_itm_data_env test_1 test 4.06d
set_itm_data_env: No such file or directory
    
```

file:/afs/efda-itm.eu/isip/user/test_1/itm_v1.0/actors/cpo2ip/workflow_1.xml
 Components: Data
 Search: plot
 Search repository:
 Search: Search Reset

Search Results
 Components
 Data Output
 Workflow Output
 Graphical Output
 Array Plotter
 Barplot
 Boxplot
 Scatterplot
 Sequence Plot
 Timed Plotter
 XY Plotter

Statistics
 Statistical Analysis
 Graphical Analysis
 Timed Plotter
 Bar Chart
 Box And Whisker
 Boxplot

18 results found.

SDF Director
 EU ITM
 equilibrium
 cpo2ip
 EU ITM
 Display
 Array Plotter

Array Plotter
 x10⁴
 4.8
4.7
4.6
4.5
4.4
4.3
 0 2 4 6 8 10 12 14 16 18

```

3813      6014.8309242
5492      5497.1758712
5336      4977.7892293
7044      4456.4188813
4253      3932.8119465
5283      3406.7139525
8031      2877.8680135
4276      2346.0140056
3921      1810.8877338
7050      1272.2200834
4012      729.73614860
    
```

```

ip=23
do i=1,20
  ip(i) = int(equ
enddo

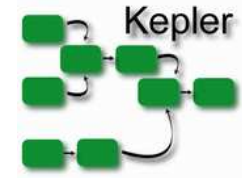
write(*,*) 'ip:',
return
    
```

execution finished.

Ln 29, Col 52 INS Wed 29 Apr 2009 05:00:33 PM MES

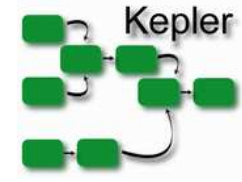
A reminder: Code to KEPLER

- Port your code to the Gateway
- Identify relevant CPOs
- Make a subroutine of your code that has CPOs as input/output or integer, ...
- Run it in a test bed to check that the CPOs are correctly implemented.
- Make a library of the routine: mylib.a or mylib.so
- Use FC2K to add your code in \$KEPLER
- Include your new actor in a workflow and press run
- And Bob's your uncle (hopefully).

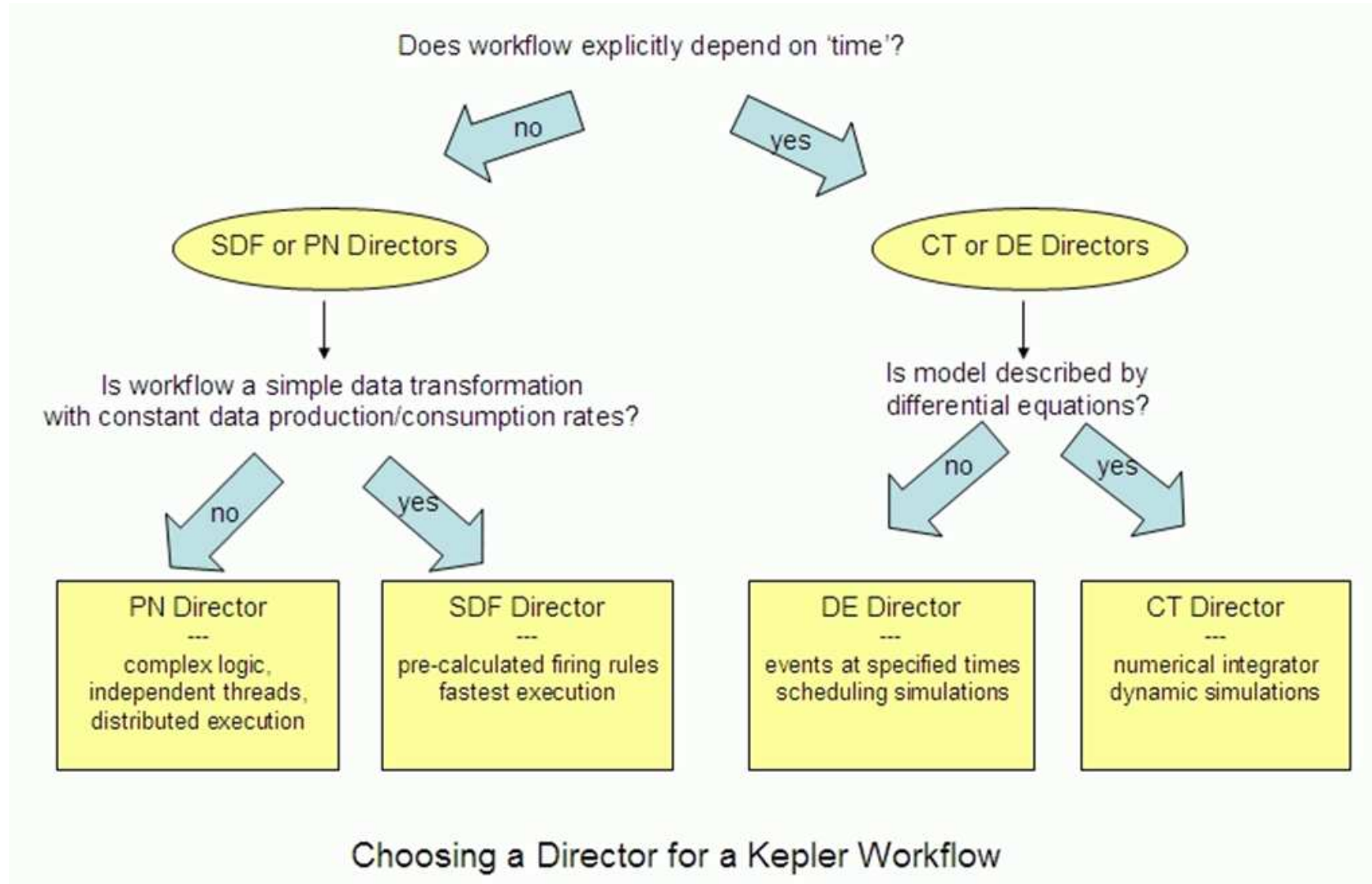


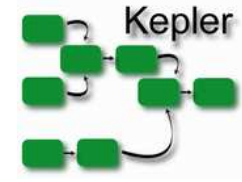
Advanced use of KEPLER

- Directors
- Iterations
- Debug



How to choose a director

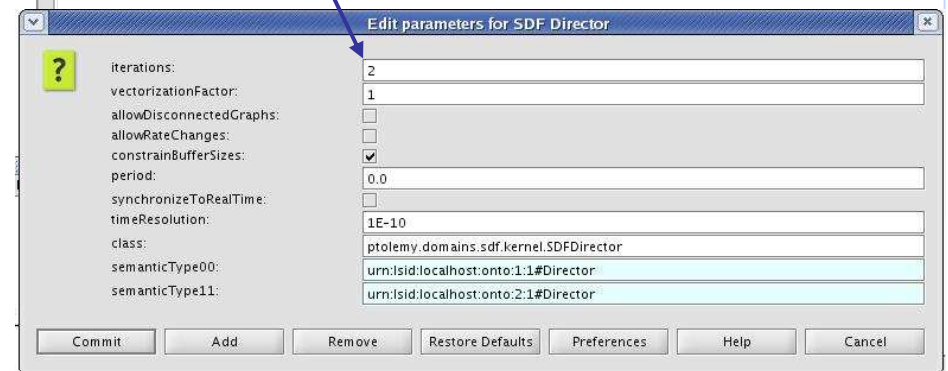
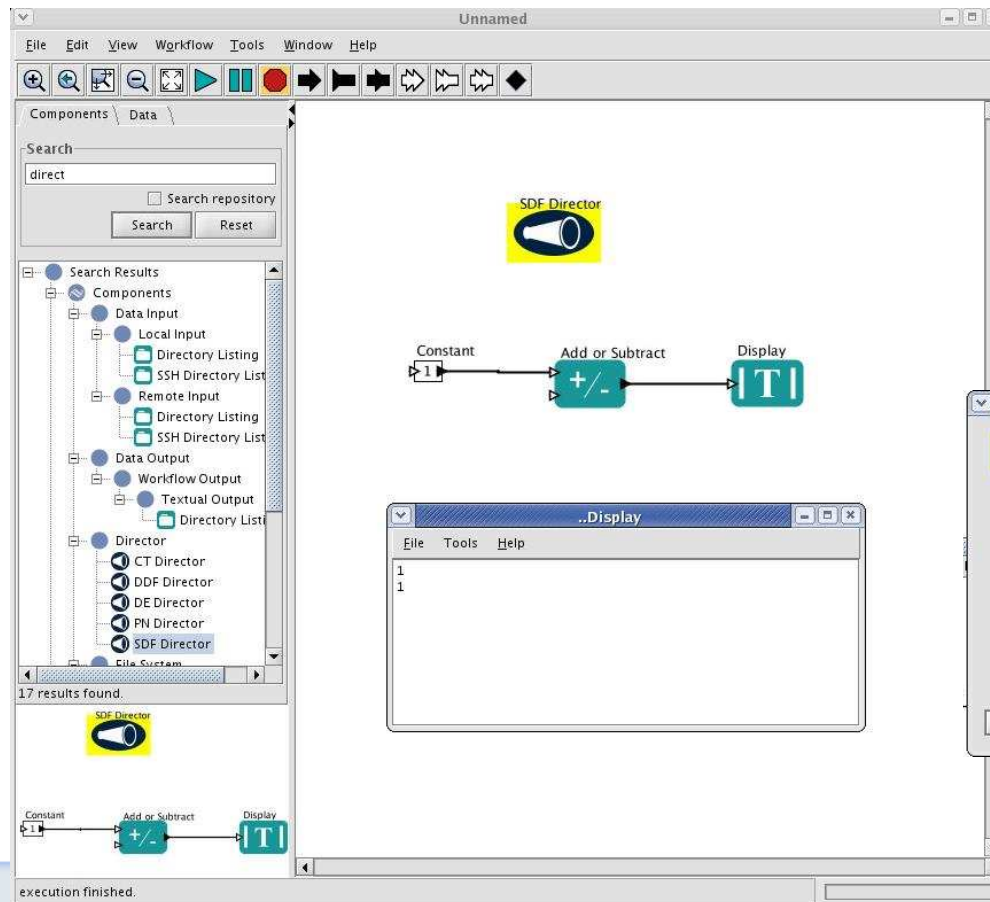




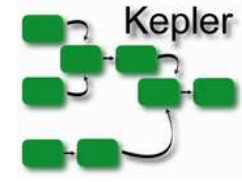
Design of a workflow (1)

Design

1. Choose your actors (for instance constant, addsubtract & display: use the search cmd)
2. Drag & drop them in the design area
3. Connect the actors => draw a link between input & output port
4. Define the director (SDF) and its parameters (number of iterations)



[~guillerm/public/training/add_with_SDF.xml](https://github.com/guillerm/public/training/add_with_SDF.xml)



Design of a workflow (2)

Outcome with PN director

1. With PN, scheduling is done by each thread (actor) => no global scheduling

The screenshot shows a Kepler workflow with the following components:

- Constant**: Provides a value of 1.
- Add or Subtract**: Receives the value from the Constant actor.
- Display**: Receives the output from the Add or Subtract actor.
- PN Director**: A yellow megaphone icon that monitors the workflow for deadlocks.

 The console window titled '..PN Director' displays the following log:


```

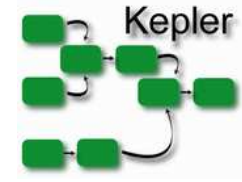
    Checking for deadlock:
    There are 2 Blocked actors, 0 Stopped actors, and 3 active threads.
    Waiting for actors to stop.
    Checking for deadlock:
    There are 1 Blocked actors, 0 Stopped actors, and 3 active threads.
    Waiting for actors to stop.
    Checking for deadlock:
    There are 1 Blocked actors, 0 Stopped actors, and 3 active threads.
    Waiting for actors to stop.
    Checking for deadlock:
    There are 1 Blocked actors, 0 Stopped actors, and 3 active threads.
    Waiting for actors to stop.
    Checking for deadlock:
    There are 1 Blocked actors, 0 Stopped actors, and 3 active threads.
    Waiting for actors to stop.
    Checking for deadlock:
    There are 0 Blocked actors, 0 Stopped actors, and 3 active threads.
    Waiting for actors to stop.
    Checking for deadlock:
    There are 1 Blocked actors, 0 Stopped actors, and 3 active threads.
    Waiting for actors to stop.
    Checking for deadlock:
    There are 1 Blocked actors, 0 Stopped actors, and 3 active threads.
    Waiting for actors to stop.
    Checking for deadlock:
    
```

 The status bar at the bottom indicates 'execution finished.'

Endless

Trick

1. Constant provide a value at each step
2. PN will stop if no more datatoken source available
3. Use "SingleFireConstant" instead



Design of a workflow (3)

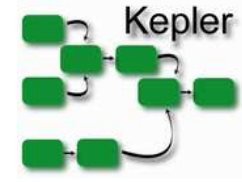
Outcome with PN director

1. With PN & one single fire => OK

The screenshot shows the Kepler workflow editor interface. The main workspace contains a workflow with the following components: a Constant actor (value 1), an Add or Subtract actor (+/-), and a Display actor (T). A yellow callout box with the text "End after one iteration" has an arrow pointing to the Display actor. Below the main window is a dialog box titled "Edit parameters for Constant" with the following fields:

Parameter	Value
firingCountLimit:	1
value:	1
class:	ptolemy.actor.lib.Const
semanticType00:	urn:lsid:localhost:onto:1:1#ConstantActor
semanticType11:	urn:lsid:localhost:onto:2:1#Constant
kar:	urn:lsid:kepler-project.org:kar:57:1

Buttons at the bottom of the dialog include: Commit, Add, Remove, Restore Defaults, Preferences, Help, and Cancel. The status bar at the bottom of the editor shows "execution finished."



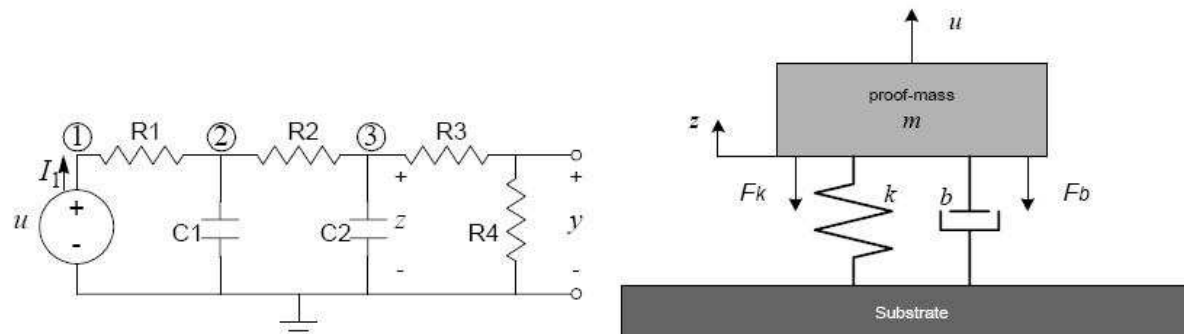
Differential Equations

$$m\ddot{z}(t) + b\dot{z}(t) + kz(t) = u(t)$$

$$y(t) = c \cdot z(t)$$

$$z(0) = 10, \dot{z}(0) = 0.$$

The equations could be a model for an analog circuit as shown in figure 2.1(a), where z is the voltage



(a) A circuit implementation.

(b) A mechanical implementation.

Could be written as

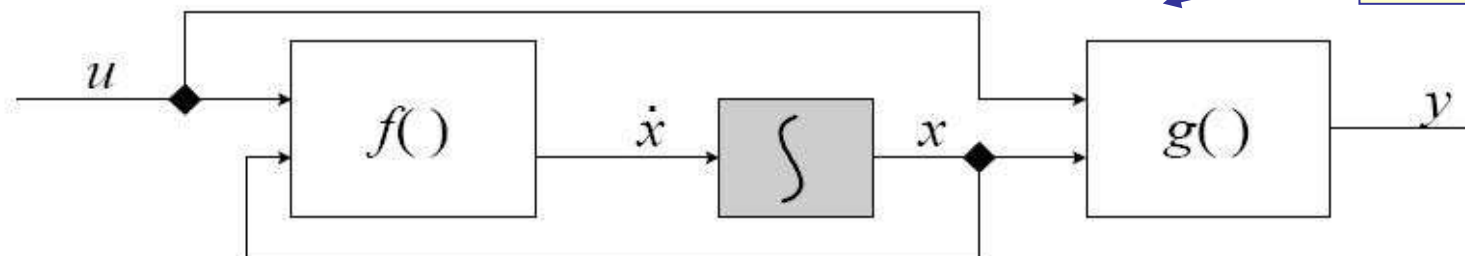
an ODE-based continuous-time system has the following form:

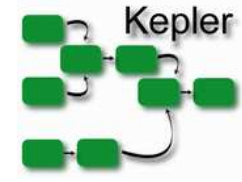
$$\dot{x} = f(x, u, t)$$

$$y = g(x, u, t)$$

$$x(t_0) = x_0.$$

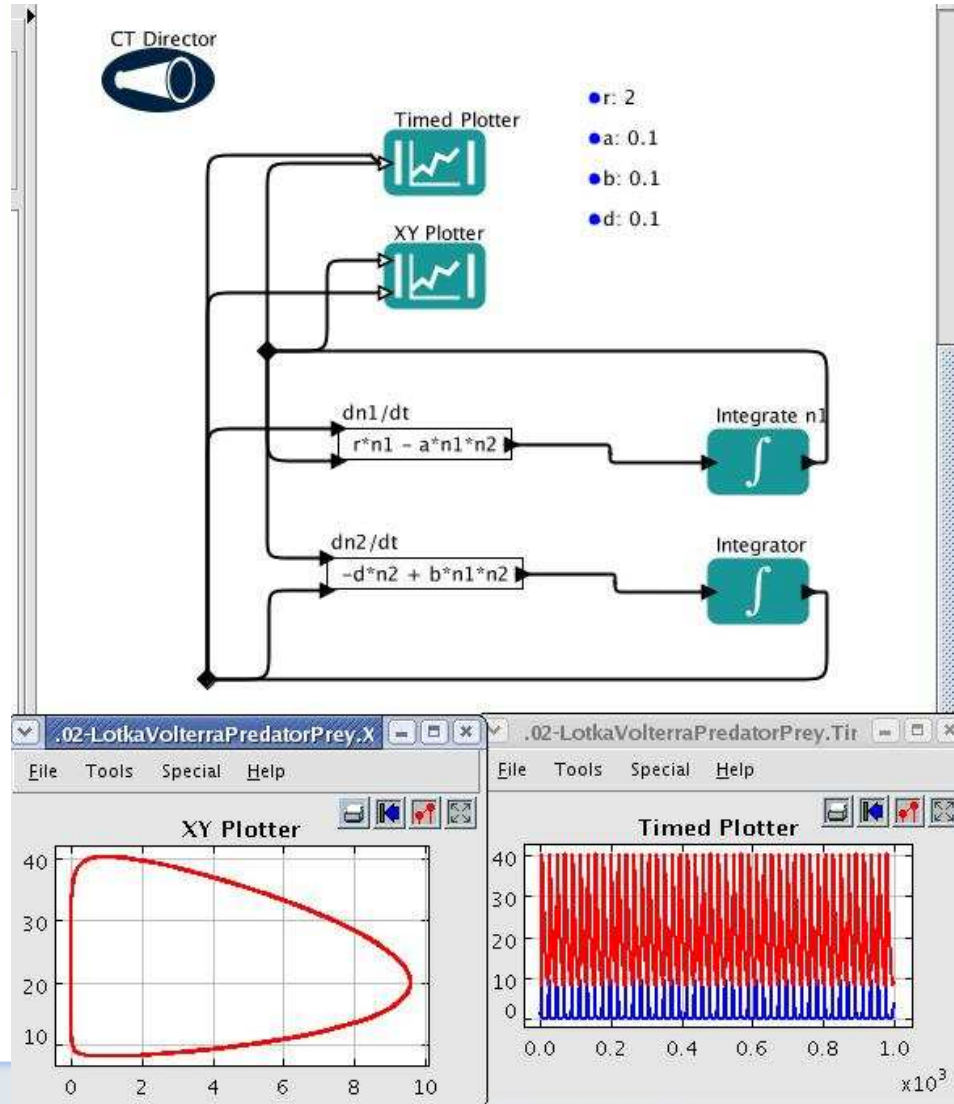
Could be implemented as



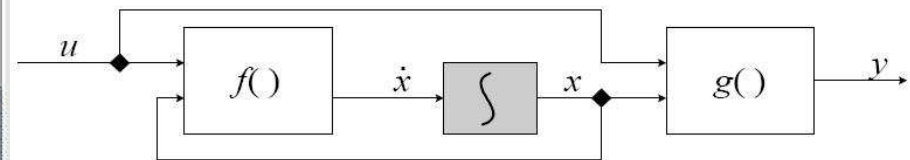


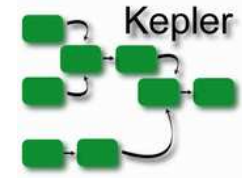
Differential Equations (2)

Using the CT director



Example of 2 coupled differential equations

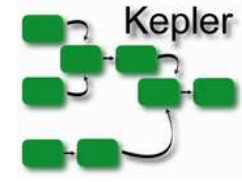




How to iterate?(1)

Different ways

1. Iterations in the Director (seen previously)
2. Ramp or repeat actors
 - Ramp = “for (i=initial;i=i+step;i<final)”
 - Repeat = repeat the same input some specified number of times
3. Using arrays & Rexpression
4. Using composite actors (encapsulate the iterations in a single actor)
5. Feedback connections
 - Beware: how to start with some directors (SDF for instance)
6. Using map (Java)



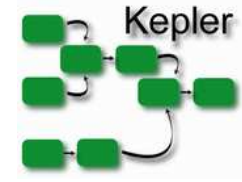
How to iterate?(2)

Using the RAMP actor (for $i=0;i++;i<10$)

The screenshot shows the SDF Director interface. The main workspace contains a workflow with three actors: 'Ramp', 'Add or Subtract', and 'Display'. The 'Ramp' actor is highlighted with a blue arrow pointing to its 'Edit parameters for Ramp' dialog box. The dialog box shows the following parameters:

firingCountLimit:	10
init:	0
step:	1
class:	ptolemy.actor.lib.Ramp
semanticType00:	urn:lsid:localhost:onto:1:1#iterativeMathOperationActor
semanticType11:	urn:lsid:localhost:onto:2:1#iterativeOperation
semanticType22:	urn:lsid:localhost:onto:2:1#WorkflowInput
firingsPerIteration:	1

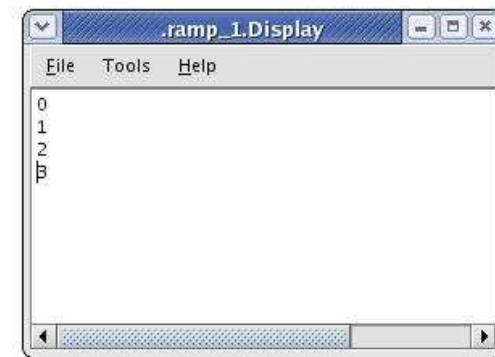
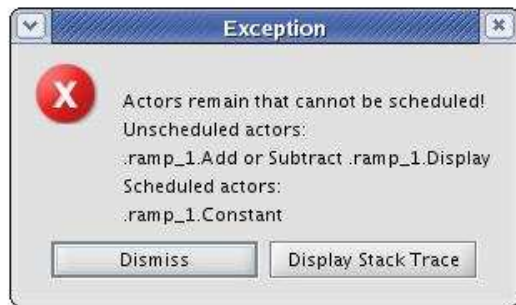
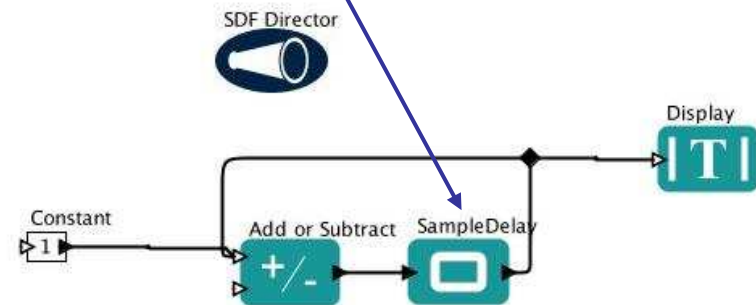
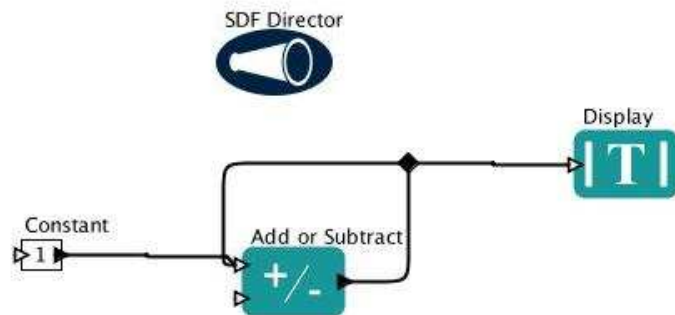
At the bottom right of the dialog box, there is a URL: ~guillerm/public/training/ramp.xml

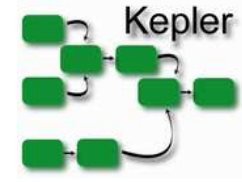


How to iterate?(3)

Using a Feedback connection (SDF)
 Don't know how to start => error

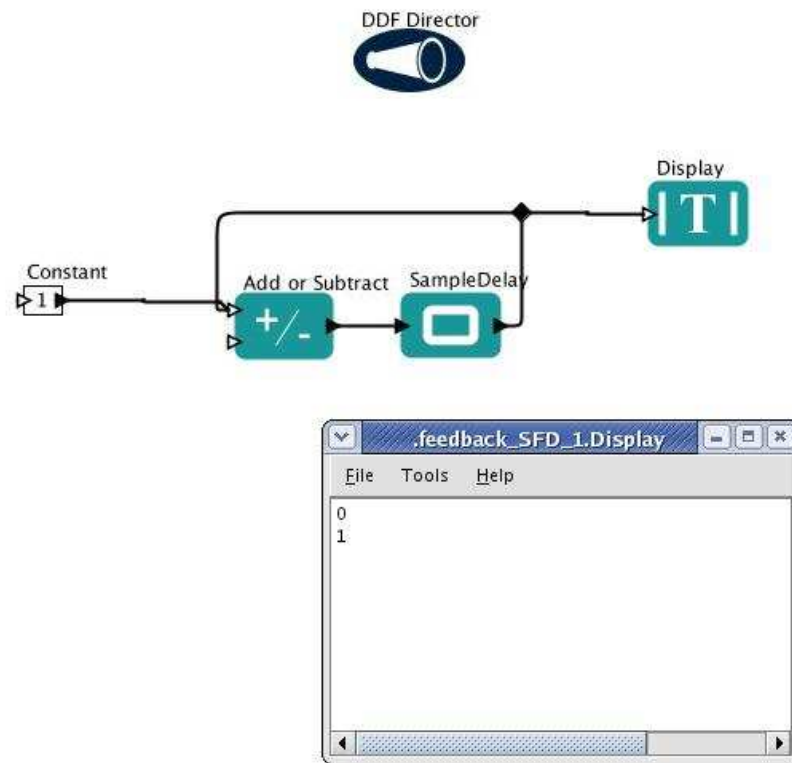
Trick:
 Add a null time delay => start from
 sampledelay output (0..3)



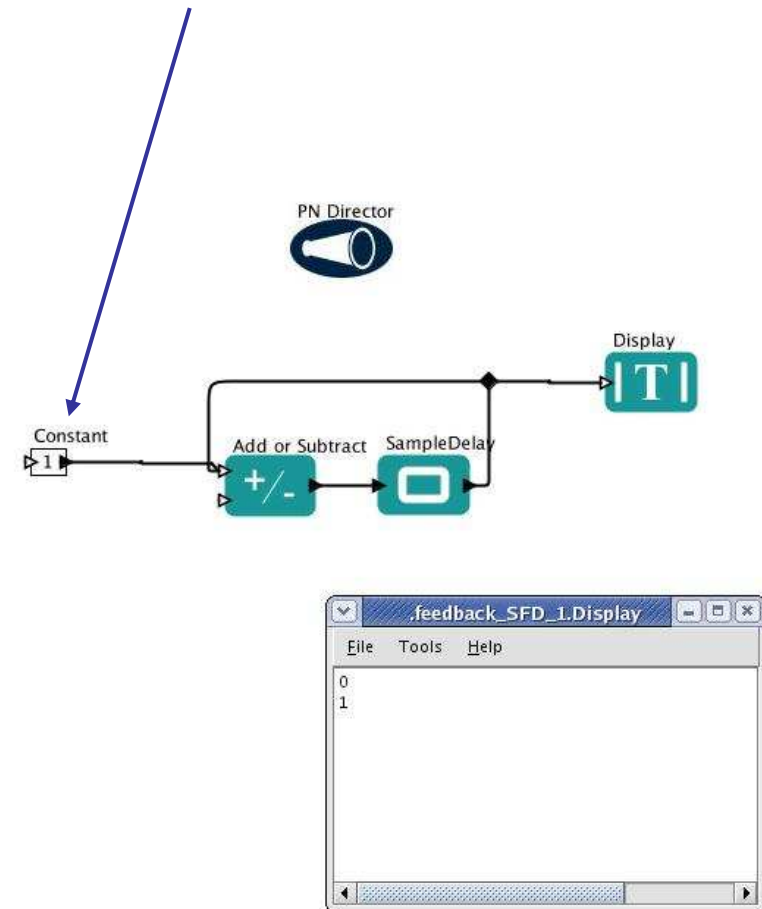


How to iterate?(4)

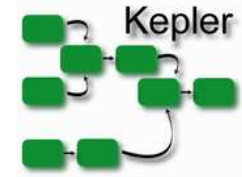
Using a Feedback connection (DDF)
Iterations=4



Using PN:
with single firing constant



~guillerm/public/training/loop_DDF.xml



How to iterate?(5)

Using a Feedback connection (CT)
 Duration=100s, step=0.1s

Edit parameters for CT Director

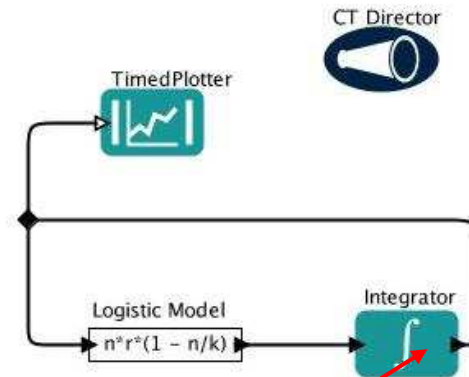
timeResolution:	1E-10
startTime:	0.0
stopTime:	100.0
initStepSize:	0.1
minStepSize:	1e-5
maxStepSize:	1.0
maxIterations:	20
errorTolerance:	1e-4
valueResolution:	1e-6
synchronizeToRealTime:	<input type="checkbox"/>
ODESolver:	"ExplicitRK23Solver"
breakpointODESolver:	"DerivativeResolver"
runAheadLength:	0.1
class:	ptolemy.domains.ct.kernel.CTMixedSignalDirect
semanticType000:	urn:lsid:localhost:onto:1:1#Director
semanticType111:	urn:lsid:localhost:onto:2:1#Director

Buttons: Commit, Add, Remove, Restore Defaults, Preferences

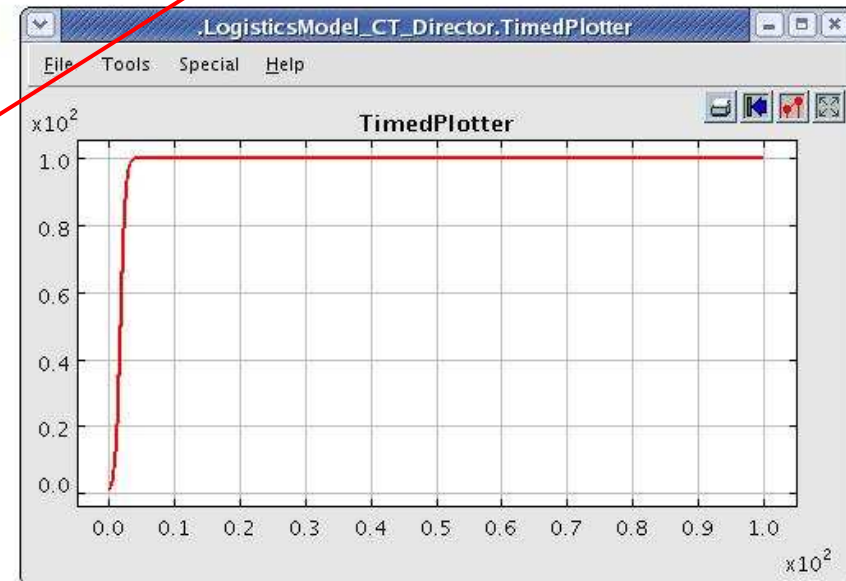
Edit parameters for

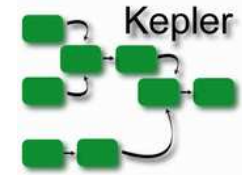
initialState:	1.0
class:	ptolemy.domains.ct.lib.Integri
semanticType000:	urn:lsid:localhost:onto:1:1#C
semanticType111:	urn:lsid:localhost:onto:2:1#C

Buttons: Commit, Add, Remove, Restore Default



- initial population • initPop: 1.0
- growth factor • r: 2.6
- carrying capacity • k: 100

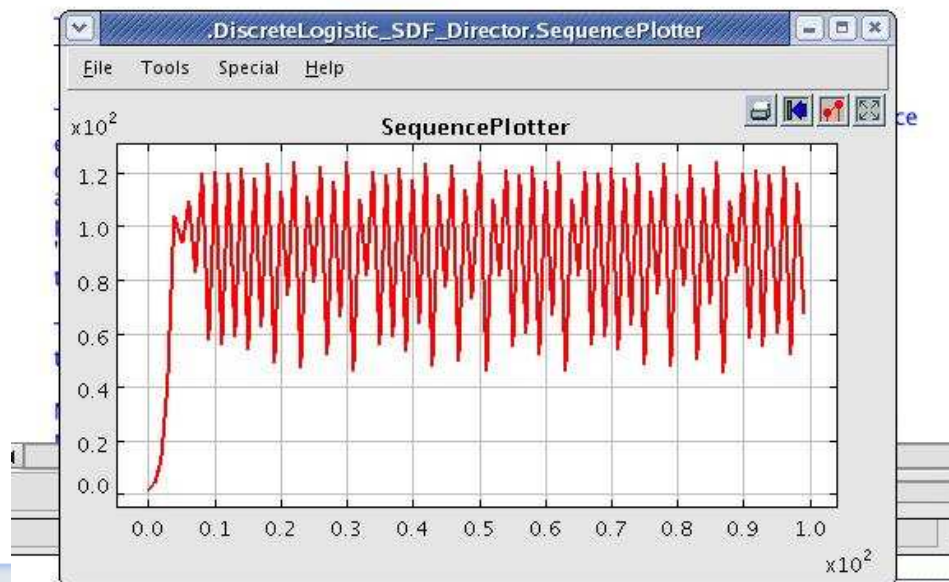
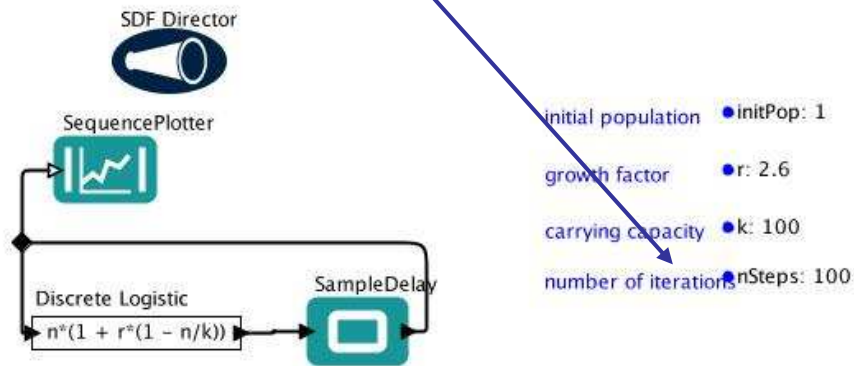




How to iterate?(6)

Using a Feedback connection (SDF)

Iterations=100

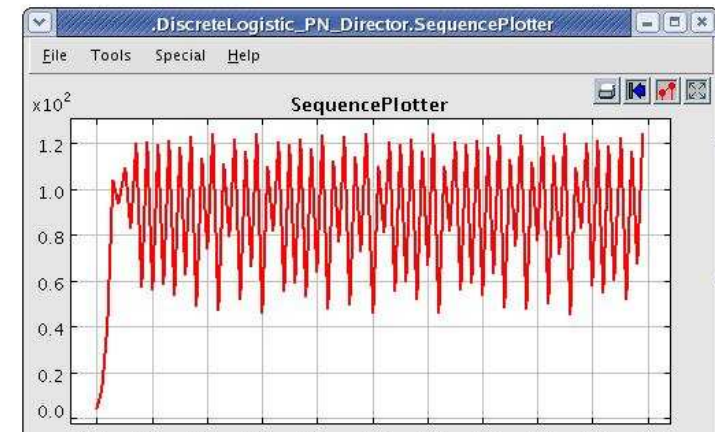
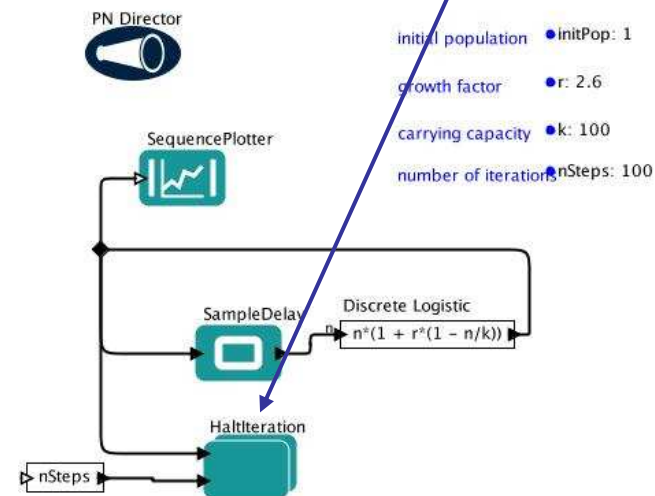


Training session 4-6 May 2009

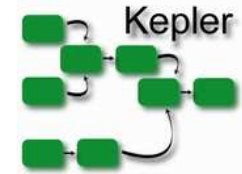
Using a Feedback connection (PN)

Iterations=100 need a stop

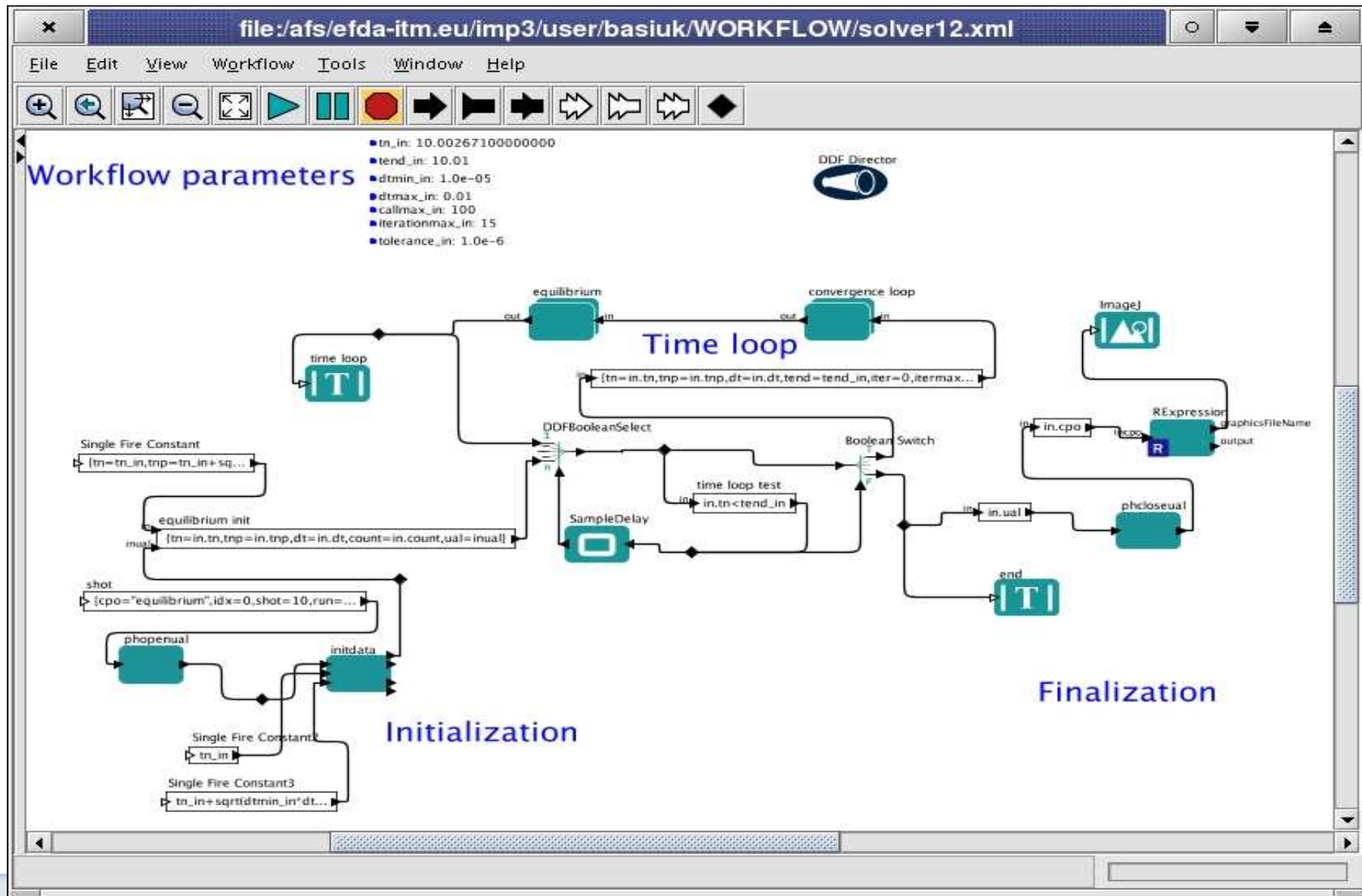
See \$Kepler/demos/SEEK/DiscreteLogistics_PN_Director.xml

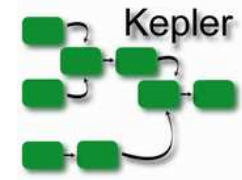


~guillerm/public/training/loop_expression_PN.xml



A more complex fusion workflow

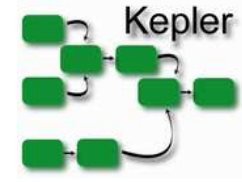




How to debug?

Different ways

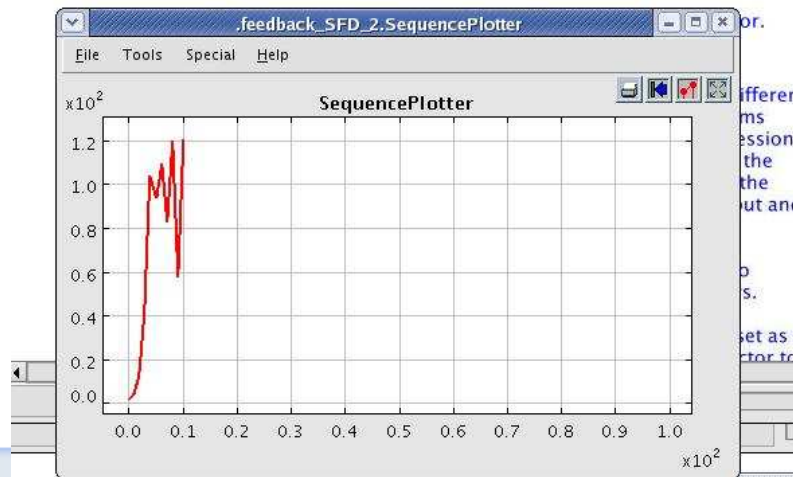
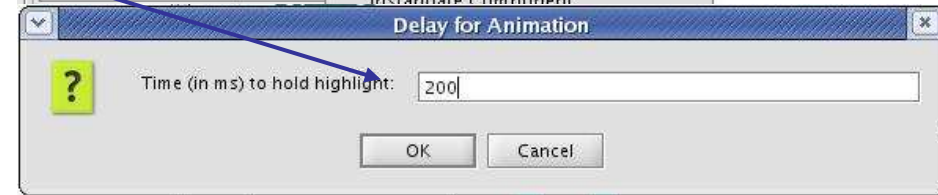
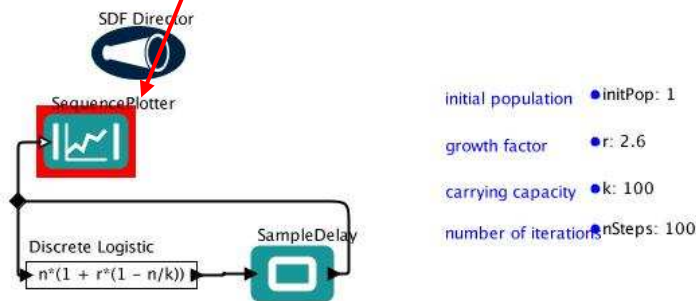
1. **Animate the workflow**
2. **Listen to:**
 - **Actor**
 - **Port**
 - **Director**
3. **Using the “Provenance” actor**

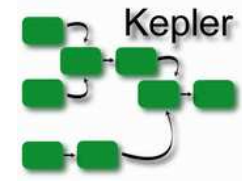


How to debug?(2)

Animate the workflow

- Specify the time in ms
- Run=>active actor is in Red





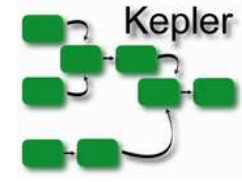
How to debug?(3)

Listen to

- Director
- Notify the actions

The screenshot displays the Kepler IDE interface. At the top, a menu bar includes File, Edit, View, Workflow, Tools, Window, and Help. A 'Tools' dropdown menu is open, showing options like 'Animate at Runtime...', 'Listen to Director', 'Create Composite Actor', 'Expression Evaluator', and 'Instantiate Component'. A red arrow points from the 'Listen to Director' option to the 'SDF Director' component in the workflow diagram. The workflow diagram shows a 'Discrete Logistic' component with the formula $n \cdot (1 + r \cdot (1 - n/k))$ connected to a 'SampleDelay' component, which is then connected to a 'SequencePlotter' component. A blue arrow points from the 'Notify the actions' bullet point to the 'SequencePlotter' component. Below the workflow, a console window titled '.feedback_SFD_2.SDF Director' shows a log of messages:

```
File Tools Help
The actor .feedback_SFD_2.SampleDelay was iterated.
Director: Called postfire().
Director: Called prefire().
Director prefire returns true.
The actor .feedback_SFD_2.SequencePlotter will be iterated.
The actor .feedback_SFD_2.SequencePlotter was iterated.
The actor .feedback_SFD_2.Discrete Logistic will be iterated.
The actor .feedback_SFD_2.Discrete Logistic was iterated.
The actor .feedback_SFD_2.SampleDelay will be iterated.
The actor .feedback_SFD_2.SampleDelay was iterated.
Director: Called postfire().
Director: Called prefire().
Director prefire returns true.
The actor .feedback_SFD_2.SequencePlotter will be iterated.
The actor .feedback_SFD_2.SequencePlotter was iterated.
The actor .feedback_SFD_2.Discrete Logistic will be iterated.
The actor .feedback_SFD_2.Discrete Logistic was iterated.
The actor .feedback_SFD_2.SampleDelay will be iterated.
The actor .feedback_SFD_2.SampleDelay was iterated.
```

How to debug?(4)

Listen to

- **Actors**
- **Notify the actions**

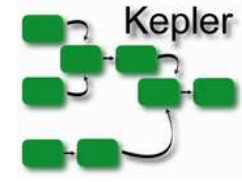
The screenshot displays a Kepler workflow with the following components and settings:

- SDF Director**: Controls the workflow.
- SequencePlotter**: Monitors the workflow's execution.
- Discrete Logistic**: Performs the calculation $n^*(1 + r*(1 - n/k))$. Parameters:
 - initial population: \bullet initPop: 1
 - growth factor: \bullet r: 2.6
 - carrying capacity: \bullet k: 100
 - number of iterations: \bullet nSteps: 100
- SampleDelay**: Delays the signal. A context menu is open over this actor, with "Listen to Actor" selected.

The log window at the bottom shows the following execution trace for the `.feedback_SFD_2.SampleDelay` actor:

```

Called iterate(1)
Called prefire()
Called fire()
Called postfire()
Called iterate(1)
Called prefire()
Called fire()
Called postfire()
Called iterate(1)
Called prefire()
Called fire()
Called postfire()
Called iterate(1)
Called prefire()
Called fire()
Called postfire()
Called iterate(1)
Called prefire()
Called fire()
Called postfire()
  
```

How to debug?(5)

Listen to

- **Ports**
- **Show the data**

SDF Director

SequencePlotter

Discrete Logistic

SampleDelay

initial p

growth

carryin

numbe

.feedback_SFD_2.Discrete Logistic.output

File Tools Help

```

send to channel 0: 12.554209824
send to channel 0: 41.0386238205638
send to channel 0: 103.9506609818003
send to channel 0: 93.2731416520984
send to channel 0: 109.5864571525915
send to channel 0: 82.2722643766533
send to channel 0: 120.1932891287415
send to channel 0: 57.0887453222528
send to channel 0: 120.782237255915
send to channel 0: 55.5189843711463
send to channel 0: 119.7270454704334
send to channel 0: 58.3186628495003
send to channel 0: 121.5194589078052
  
```

sequencePlotter

Discrete Logistic

SampleDelay

$n*(1 + r*(1 - n/k))$

This is the Discrete Logistic equation. The number of iterations is controlled by the SampleDelay block.

- Configure Port Ctrl-E
- Documentation
- Listen to Port