

MCRunjob:

An HEP Workflow Planner for Grid Production Processing

Greg Graham

CD/CMS Fermilab

Iain Bertram and Dave Evans

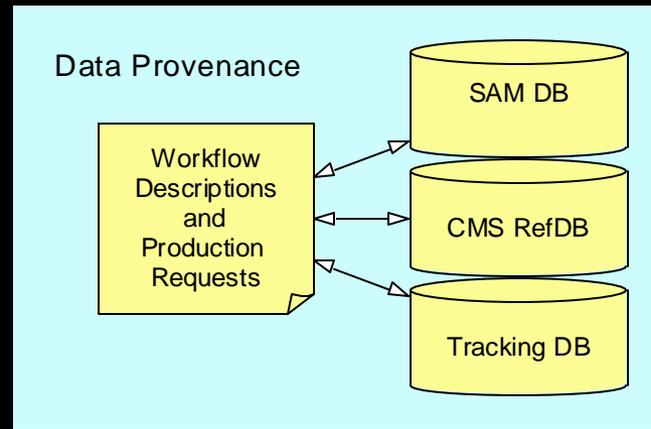
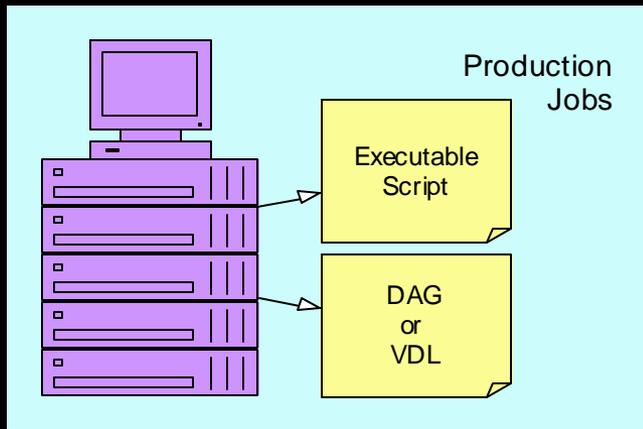
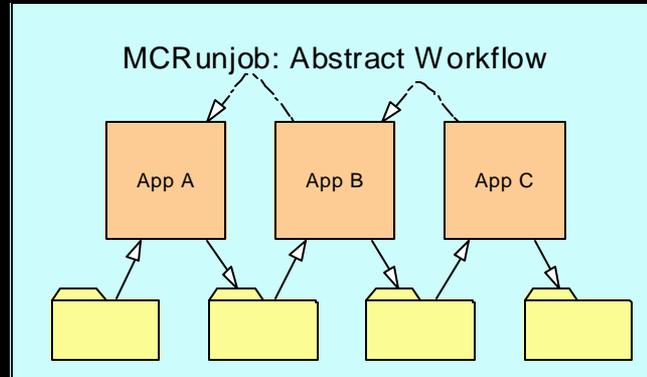
Lancaster university

CHEP 2003

Ethos of MCRunjob

- Applications in complex production processing environments often need to be tamed
 - Hundreds of input parameters during MC Production
 - Heterogeneous runtime environments
 - Complex multi-application workflows
 - Dependencies and relationships among the metadata often modeled inside of obscure shell scripts
- MCRunjob captures such specialized knowledge and makes it available to non-expert users
 - Metadata and schema oriented descriptions
 - Tracks dependencies among metadata
 - Tracks synonyms between groups of metadata, versions
 - Organization of user registered functions that do the actual work
 - Framework driven organization of tasks

Ethos of MCRunjob



MCRunjob uses a modular component based architecture.

Schema Modeling: Each application or task has its own schema representation.

ScriptGeneration: Each target script environment has its own generator.

External Services: Each service or interface is described internally by its schema.

MCRunjob Project

- In use at DZero since 1999 and at CMS since 2002.
 - Supported by respective programs.
 - For MC production only so far.
 - DZero Monte Carlo Challenges (CHEP 2001)
 - CMS Integration Grid Testbed (CHEP 2003)
- We are considering a joint DZero/CMS project to address common issues at Fermilab soon
 - The actual code bases have diverged somewhat, but there is a common repository that was started in 2003.
 - Joint project name: Shahkar
 - (which is Urdu for “Great Job”)
- Exploring ways to integrate with experiment frameworks.
 - There is some integration with DZero framework already going on
 - Need to explore ORCA interactions

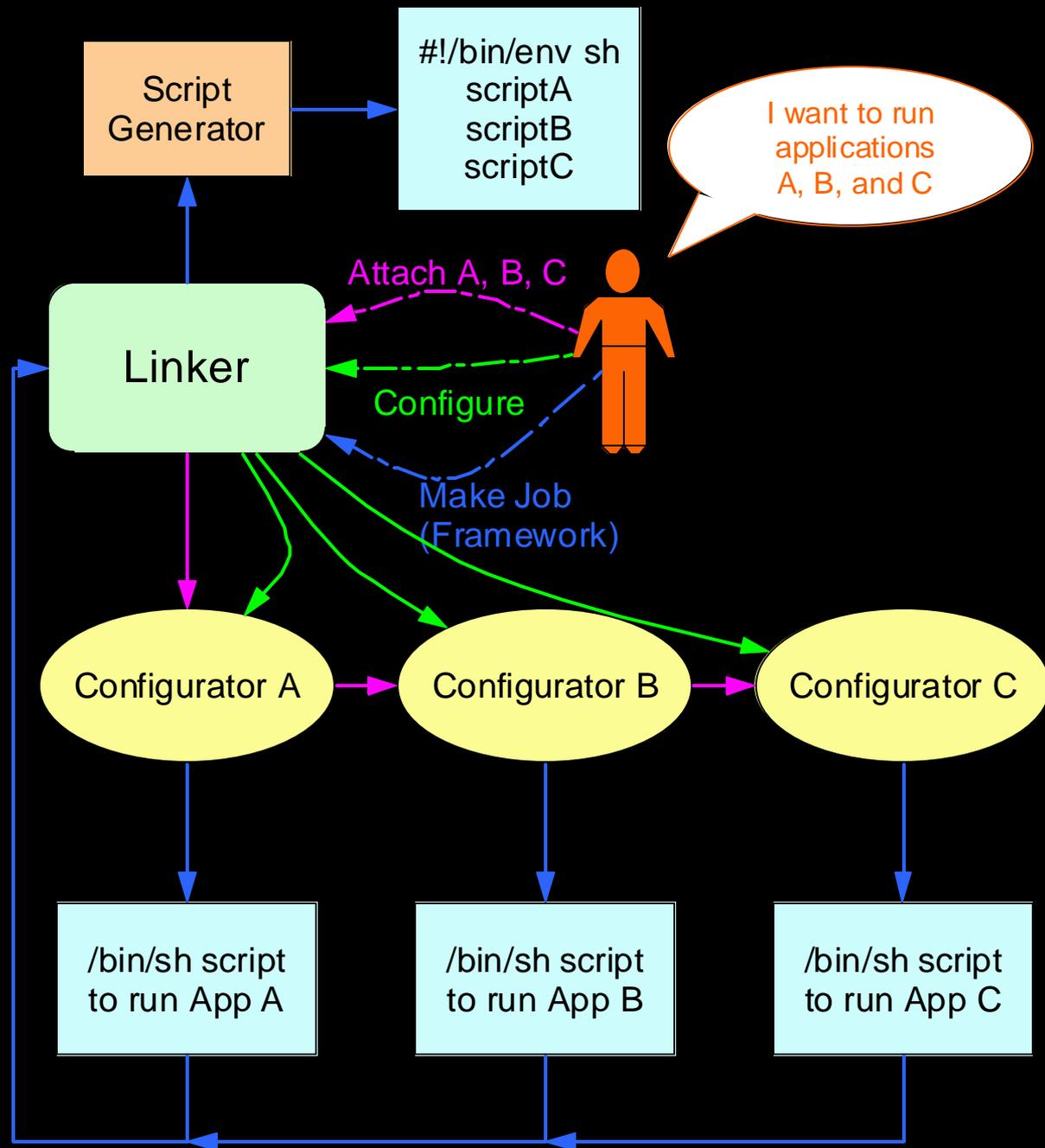
Architecture of MCRUnjob

- There are three major components of MCRUnjob
 - **Configurator:**
 - A container for schema describing some well defined application input, task, or external interfaces to DB or grid services
 - Implements framework interfaces
 - Register functions to handle framework calls, extend own interface, extend schema, define rules and dependencies to construct values for parameters.
 - **Linker**
 - a container for Configurators, checks dependencies, enables inter-configurator communication.
 - a container for “script objects” generated by Configurators
 - Runs the framework
 - **ScriptGen**
 - Mixin class for Configurators that adds methods for script object generation and framework method delegation
- All components are implemented in Python

A user who wants to run applications A,B, and C attaches corresponding Configurators to a Linker. The Linker verifies that dependencies are satisfied.

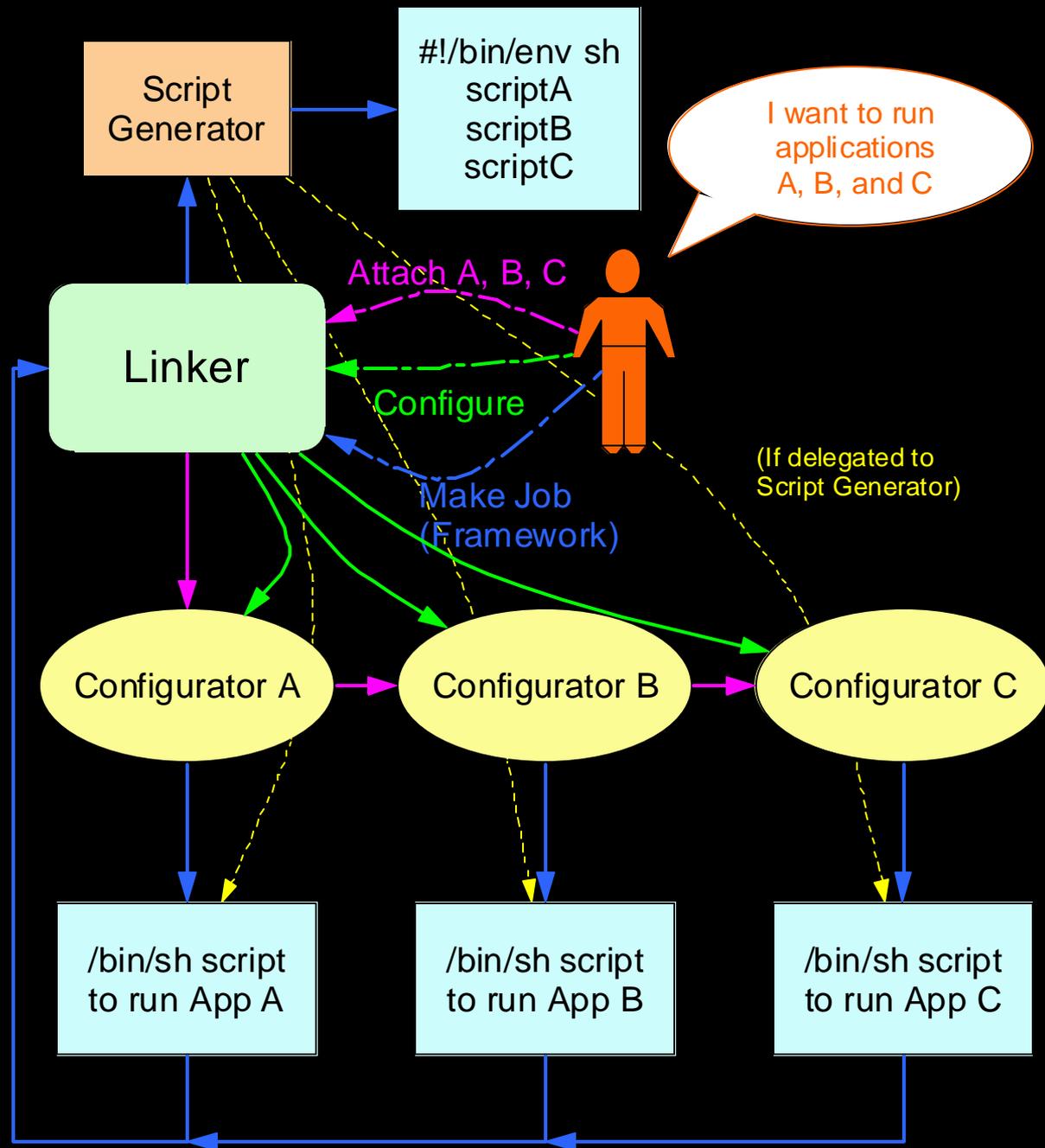
Once attached, the user sets values for the various schema elements defined in each configurator, and defines filename rules, random seed rules, etc.

The user then executes the framework. Each Configurator may generate scripts used to run the corresponding application. The scripts are collected by a ScriptGen object.



The ScriptGen object is obviously a very specialized component. Therefore, Configurators are able to delegate framework handlers to ScriptGen objects. This allows script generating code that targets specific environments to be collected in a single ScriptGen module.

Multiple ScriptGen objects can be attached at once, allowing two different environments to be targeted by the same workflow description.



Configurator Functionality - 1

- Configurators re-implement the UserDict() interface with built-in triggers on reads and writes
 - Schema elements must be declared before use
 - Triggers are typically defined in the constructor, but can be added dynamically also.
- Interfaces exist to extend the schema of a Configurator
 - Also possible to “lock” the schema to prevent further alterations.
- Parameter values can reference other Configurators
 - Inter-Configurator parameter lookup is implemented using a global read trigger on all of the schema elements.
 - Different lookup behaviors (ie: CMS vs. DZero) possible by defining different triggers.
- Dependencies on other Configurators are explicitly declared
 - This is required for parameter lookup in the CMS implementation

Configurator Functionality - 2

- Construction of parameter values
 - User can register functions, keyed by schema element, that will be called by the read trigger
 - eg- Filename rules, RunNumber rules, Random Seed rules, etc.
- Framework handling
 - Handling of framework calls can be done in specific methods organized by inheritance
 - This is considered deprecated, but it is not disallowed.
 - User can register functions to handle specified framework calls.
 - User can also specify delegation of framework calls to other Configurators that satisfy the ScriptGen interface
 - Useful, for example, when trying to gather all related script generating functions into a single module
- Inheritance
 - Configurators are grouped by inheritance according to experiment or function
 - Eg- Dzero vs. CMS, Generator vs. Processor, InputPlugins, etc.

Configurator Descriptions and Namespaces

- Configurators themselves are also described by an extensible list of key-value pairs.
 - Class ConfiguratorDescription(key1=val1, key2=val2, etc)
 - This is useful to assign “meta-metadata” to the grouping of application metadata
 - But also: parameters are specified globally in a Linker space by name and ConfiguratorDescription.
 - eg- ::ConfigDesc:ParamName
 - And: The ConfiguratorDescriptions also function as namespaces
 - To keep namespaces distinct, one can also give them arbitrary aliases. This mechanism is also used to distinguish Configurators of a common type inside of the Linker space.
 - Studying whether Configurators should contain themselves.
 - A pilot project at DZero is studying this.

Linker Functionality

- Container for Configurators and “script objects”
 - Linker guarantees that dependencies are satisfied by adding Configurators in serialized order.
 - Exception thrown when this is not satisfied.
 - A script object may be a bash script, a derivation in Virtual Data Language, a DAG node, etc.
- Also runs the framework methods. Examples:
 - PreJob: runs before each script object
 - MakeJob: creates each script object
 - Reset: runs between script objects
 - RunJob: Submits a suitable “script object” to some Grid interface or batch queue
- Framework methods are also user definable and user callable.

ScriptGen Interface

- Implemented as a mixin class for Configurators
- Specifies methods for generating scripts
 - HandleFrameworkCall: a callback for Configurators to delegate handling of framework calls; mainly those that generate script objects
 - Used so that all related “script generating” code can be gathered into a single module.
 - MakeScript: a method called by the linker to collect related script objects into a composite script object.
- Example ScriptGen objects:
 - Impala scripts (CMS MC production scripts)
 - RCP based scripts at DZero
 - Virtual Data Language of Chimera
 - MOP DAGs for Condor-G/DAGMan
- A Linker can support multiple ScriptGen objects simultaneously

Macro Script Language

- The Linker has a facility to read “macro” scripts and parse lines one by one
 - Functions available include:
 - Attaching and naming Configurators, setting parameter values, adding schema values, defining synonyms, executing the framework or selected framework calls, executing selected methods, exception handling, executing other scripts
 - Procedural constructs supported for handling multiple jobs.
- Parsing is done by Configurators themselves
 - Users (experienced ;-) can extend the “macro” script interface by registering their own parser functions to the Configurators.
 - Multiple Parsers can be attached; first Parser to handle the line “wins”
- Many things are missing:
 - Full functionality is not yet available in the “macro” language
 - Needs parser that supports both expressions and conditionals
 - Syntax needs to be reviewed as a whole.

Synonyms and Ontology

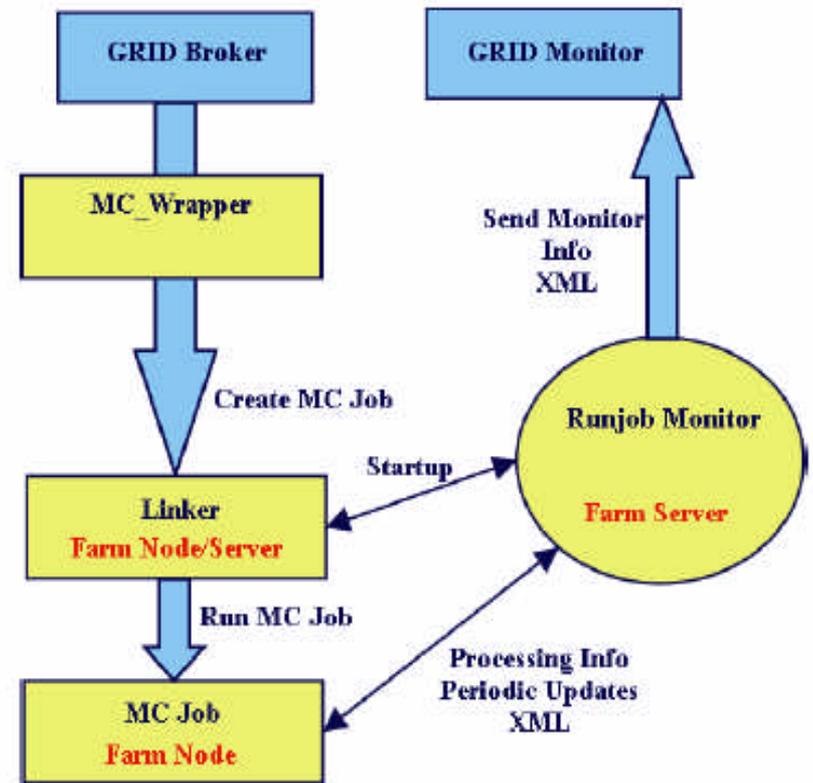
- Configurators also contain an internal synonym table to automatically keep track of translations between schema elements of different Configurators
 - Example:
 - `cfg CMSIM synonym RanSeed1 ::generator:CMKIN:RunNumber`
 - `cfg CMSIM print`
 - Causes resolution of `RanSeed1` by synonym lookup when parameter is not given
 - implicit synonyms- when schema elements have the same name
 - eg- I didn't have to say
“`synonym RunNumber ::generator:CMKIN:RunNumber`”
 - These ontological definitions can be stored in files or database tables.
 - These can be used to “connect” Configurators across different versions or interface definitions on the same Configurator.

Stored Commands

- Configurators can also have a user specified list of stored commands to execute during framework operation
 - These commands are in the macro script language
 - Eg- `cfg CMSIM addcommand on reset inc RunNumber`
 - When “reset” framework method is invoked, the command “inc RunNumber” is invoked on the CMSIM Configurator.
 - The CMSIM Configurator has to have a Parser registered to it that can interpret “inc RunNumber”
- Together with synonyms and parameter lookup, stored commands can allow Configurators to track dynamically changing values in other Configurators.

MCRunjob at Runtime: SAM/JIM

- Innovations from SAM/JIM grid execution service for Dzero
 - XML based monitoring wrappers
 - XML database backend for persistent storage of jobs
 - Linker running as a server with GSI authentication
 - Macro preprocessor for abstract planning
 - Interface with SAM database production request system

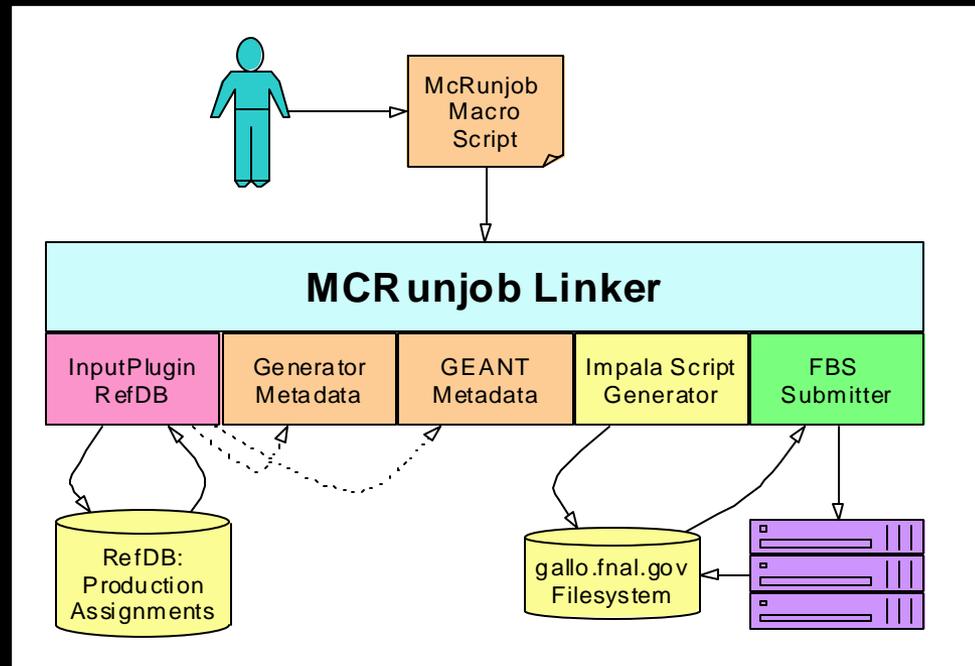


Production Job Configuration Management

- Application designer:
 - writes down schema needed to configure desired application or task
 - writes ontological configuration files to define dependencies, synonyms, and chain behaviors
 - writes down ontological configuration files to handle possible version changes in schema
- Application deployment:
 - writes configuration files to define physics parameters, number of events, production assignments.
- Regional Center Contact:
 - writes configuration file for his/her regional center (if applicable)

Fun with Configurators

- LNameStreamConfigurator
 - Can register a function to this Configurator that will fill a LogicalNameList with names (eg- LFNs, PFNs)
 - During framework operation, this Configurator will iterate over the list, setting the schema element “OutputSpec” to the current value.
- InputPluginConfigurator
 - InputPluginBashFile will parse environment variable definitions in a sh script and expose these by including the symbols as schema elements with the corresponding values
 - InputPluginRefDB will obtain schema elements and values from a web server with database backend



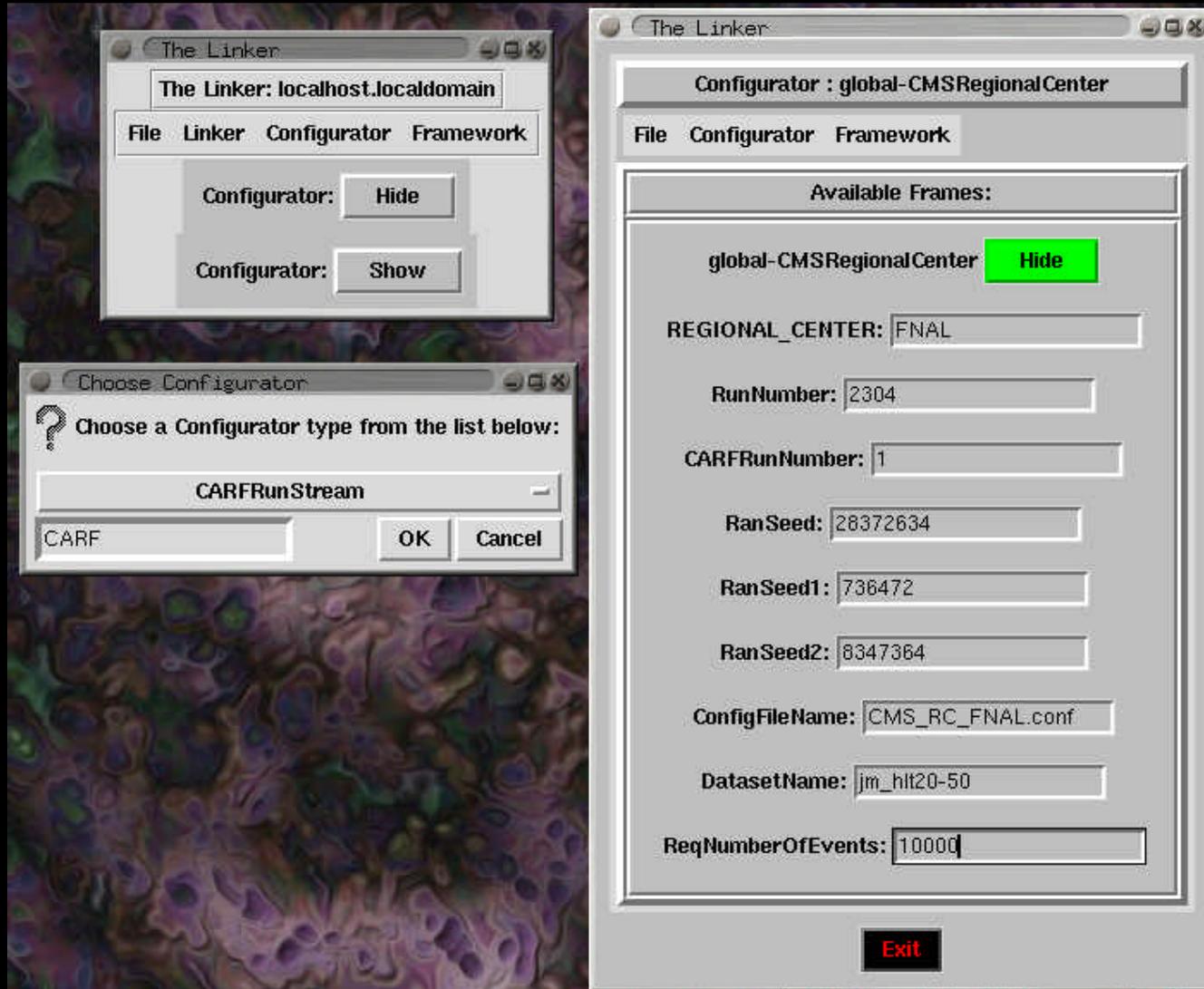
Fun with Configurators

- RogueConfigurator
 - No schema whatsoever- user defines it all at runtime!
- TableConfigurator
 - Derives from LNameStream, but has multiple schema elements. Can read from a table file or a database table and iterate over the rows
- ParamSweepConfigurator
 - Similar to a TableConfigurator, but has added logic to generate its own table internally according to some rules.
- MOPDagGen
 - A ScriptGen Configurator that takes scripts generated by other ScriptGens, turns them into DAG nodes, and creates a master DAG.
- RunJobConfigurator
 - Takes specified script object, submits it to batch interface or grid portal.

No Tool is Complete Without a GUI

- MCRunjob has a Tk based GUI that functions as a wrapper around existing MCRunjob classes
 - No special classes needed: there is a single GUILinker wrapper class and a single GUIConfigurator wrapper class.
- Configurators have a “hot swappable” internal dictionary implementation
 - TK GUI Linkage is accomplished using the “HotSwap” method to replace the default internal UserDict with one that has Tk linkage.
 - Other GUI packages are possible by writing new wrapper classes and, perhaps, new dictionaries with different Linkage.
- Screenshot of GUI is next:

No Tool is Complete Without a GUI



Relationship to Other Projects

- SAM
 - One of the first great applications of MCRUnjob was to automatically generate the metadata needed by the SAM system in order to store MC production results.
 - Closer integration with SAM is proceeding apace in the context of automatic generation of MC jobs from request metadata stored in SAM
- CHIMERA
 - MCRUnjob has a ScriptGen which produces Virtual Data Language
 - Conceptually, Configurator schemas are like transformations, Configurators with values are like derivations, and ConfiguratorDescriptions and dependencies define “types” on the data appearing at the endpoints of a transformation.
 - MCRUnjob can either generate VDL, VDL+wrapper scripts (custom transformations), or function as an abstract planner.

Relationship to Other Projects

- SAM/JIM
 - In the JIM grid execution environment, MCRunjob scripts are sent as the job instead of shell scripts or conventional executables.
 - MCRunjob macro scripts are GSI-authenticated and re-parallelized by a remote MCRunjob Linker process.
 - Delayed abstract planning!
- Data Provenance
 - MCRunjob is already capable of a fully declarative specification of workflow, and can communicate with external databases and servers.
 - Besides a bare specification of parameters, MCRunjob keeps track of the dependencies that existed among parameters when they were created.

Future Plans

- Formalize the DZero/CMS joint project
 - Shahkar
 - Language Definition
- Continue close cooperation with GriPhyN and better integrate MCRunjob into the Chimera environment.
 - Add tools to define wrapper scripts with transformations
 - Delayed abstract planning
- Add features:
 - Configurators for generic application monitors (BOSS)
 - Grid portals (currently MOPDagGen and VDLScriptGen)
 - XML representations and XML database
- Explore runtime functionality
 - Provenance, external parameter lookup services, and application monitoring services in experiment frameworks? (ala GANGA?)

Conclusions/Questions

- MCRunjob provides functionality to model complex workflows found in MC Production.
 - Is it possible/desirable to bring this to a finer granularity needed in analysis?
- MCRunjob is a powerful workflow planner with modular component based interfaces to external services.
- Preparation for Analysis
 - Take it from a former Kaon physicist: Sharpening our understanding of MC production processing still has much to teach us about the more complex environments expected in physics analysis.
 - Understanding the behavior of the underlying Grid services and the coming challenges of knowledge management in the face of clean predictable input and measurable results still has value.

References

- USCMS MCRunjob page :
 - <http://www.uscms.org/scpages/subsystems/DPE/Projects/MCRunjob/>
- DZero MCRunjob page :
 - <http://www-clued0.fnal.gov/runjob/>
- Previous Talks and Papers:
 - Tools and Infrastructure for CMS Distributed Production (4-033), G.E. Graham, et al. Proceedings of Computers in High Energy Physics 2001 (CHEP 2001), Beijing, China
 - Dzero Monte Carlo Production Tools (8-027), G.E. Graham, et al.. Proceedings of Computers in High Energy Physics 2001 (CHEP 2001), Beijing, China
 - Dzero Monte Carlo, G.E. Graham. Proceeding of Advanced Computing and Analysis Techniques 2000 (ACAT 2000), Fermilab, Batavia, IL
- ggraham@fnal.gov evansde@fnal.gov