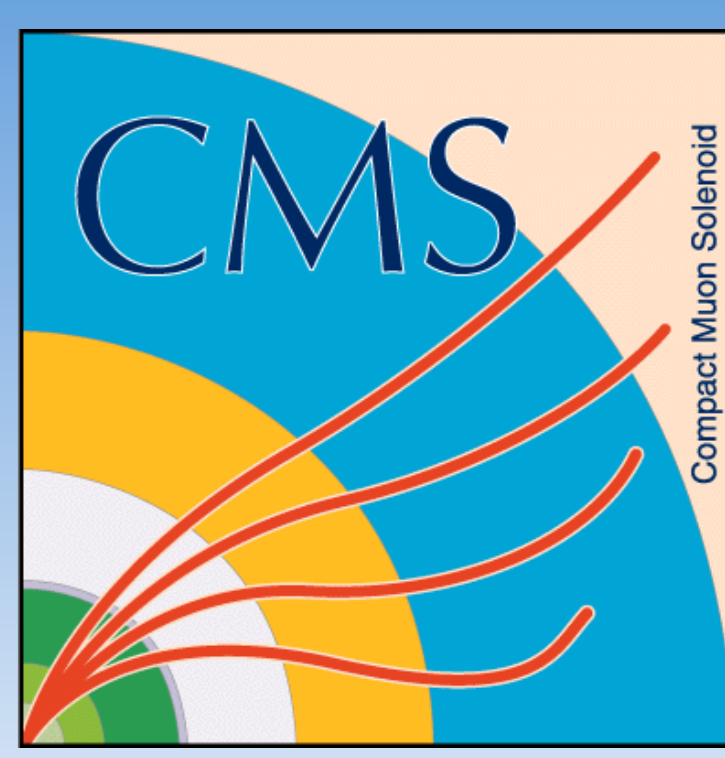




Request for All - A Generalized Request Framework for PhEDEx



Chih-Hao Huang¹, Tony Wildish², Natalia Ratnikova¹, Alberto Sanchez-Hernandez³, Xiaomei Zhang⁴, Nicolo Magini⁵
¹Fermi National Accelerator Laboratory, ²Princeton University, ³Centro Invest. Estudios Avanz, ⁴Institute of High Energy Physics, ⁵CERN

Since its deployment in 2004, PhEDEx (Physics Experiment Data Export) has been the central data management system for CMS (Compact Muon Solenoid Experiment) community. Today, it manages 60 Petabytes of data in 23 million files, in 1.6 million blocks, in 127 thousand datasets and handles all data movements among 130 sites and thousands of users all over the globe. Every PhEDEx operation is initiated by a request, such as request to move data, request to delete data, and so on. A request has its own life cycle, including creation, approval, notification, and book keeping and the details depend on its type. Currently, only two kinds of requests, transfer and deletion, are fully integrated in PhEDEx. They are tailored specifically to the operations' workflows. To be able to serve a new type of request it generally means a fair amount of development work.

After several years of operation, we have gathered enough experience to rethink the request handling in PhEDEx. Generalized Request Project is set to abstract such experience and come up with a request system which is not tied into current workflow yet it is general enough to accommodate current and future requests. The challenges are dealing with different stages in a request's life cycle, complexity of approval process and complexity of the ability and authority associated with each role in the context of the request.

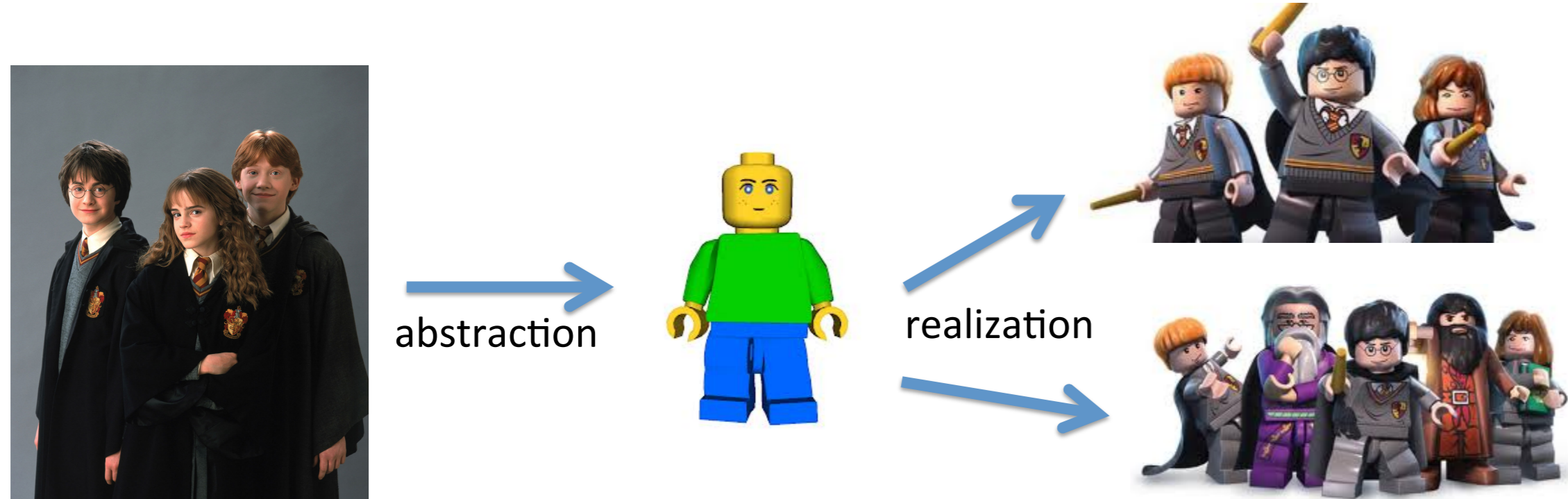
We start with a high level abstraction driven by a deterministic finite automata, followed by a formal description and handling of approval process, followed by a set of tools that make such system friendly to the users. As long as we have a formal way to describe the life of a request and a mechanism to systematically handle it, to server a new kind of request is merely a configuration issue, adding the description of the new request type, rather than development effort.

Why generalized request?

Over the years, there are other kinds of requests that are currently handled through e-mails and human interactions. It is highly desirable to incorporate them into PhEDEx. In current implementation, to serve a new kind of request means significant development effort. Generalized Request project is set to provide a framework so that future request types could be easily implemented, including the complex approval process, in which, multiple approval parties are involved in certain logical relationship.

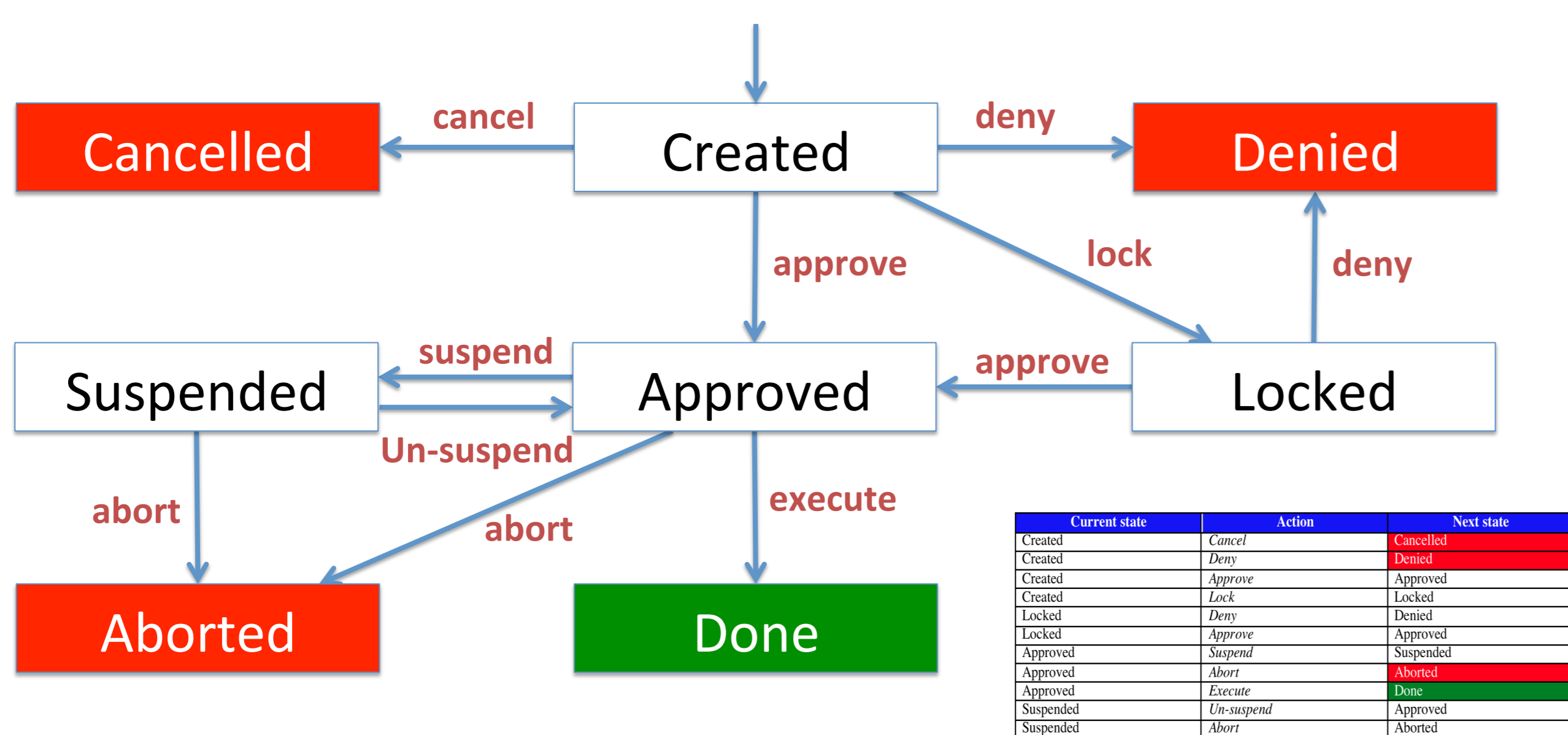
Generalization is abstraction and realization

The concept of generalization is to abstract current processes to their conceptual level which does not depend on their essences. That is, a generic "request" that is not associated with any current "types", and, a set of generic operations that deal with it. This generic request is general enough to accommodate current and future requests. Then, attach type specific information to individualize it. In the end, to create service to a new type of request is to "define" it to the system. It will be more an incremental configuration change rather than a code update.



States rather than steps

For a generalized request, we have to think not from the procedures that need to be done for a particular request but from the stages of its general life cycle and the actions that move it from one stage to another. We define the states of a general request and it is realized by a deterministic finite state automaton, DFA. The next state is a function of current state and action taking place. Therefore, the control flow, the business logic, is implied by this DFA. Not all requests go through this DFA the same way, but the mechanism is the same.



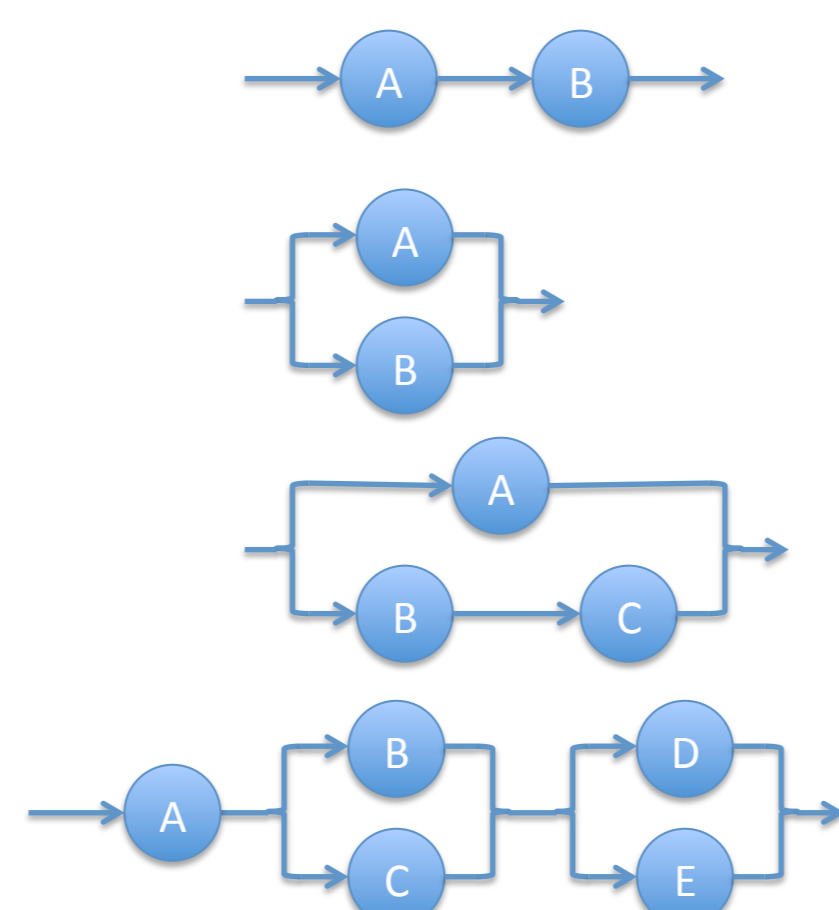
Roles are the players

A "role" is an entity that acts upon the requests. In general, it is, but not necessarily, a person with certain privilege, such as site data manager, group manager, global admin and so on. In some cases, it could be a process, acting like a person, that executes predefined actions based on predefined conditions.

Complex approval process

An approval process could be as simple as a single decision by a role or as complex as involving many decisions from many roles, some of whom are required to make decisions and some of whom only need to do it if none of their peers made one. For each role, the decision could be in one of three values: **approved**, **denied**, or **undecided**. If we relax the dependency among the decisions, all approvals can be represented by an **approval plan**, consisting of logical expressions. The current status of an approval can be determined by evaluating its corresponding logical expression using current values of its elements.

- A and B – need both roles A and B to approve
- A or B – either A or B can approve
- A or (B and C) – If A approves, it is approved. If A denies, it would take both B and C to approve it.
- A and (B or C) and (D or E)



Approval plan and template

The Approval Plan is evaluated whenever a decision is made, including the timeout and default decisions. At any given time, the evaluation returns one of the three values: approved, denied, and undecided. Approval plan is instantiated with supplied information at run time from a template which is predefined according to request type. A role may be given a timer and default decision. We use the following syntax to define approval plan template:

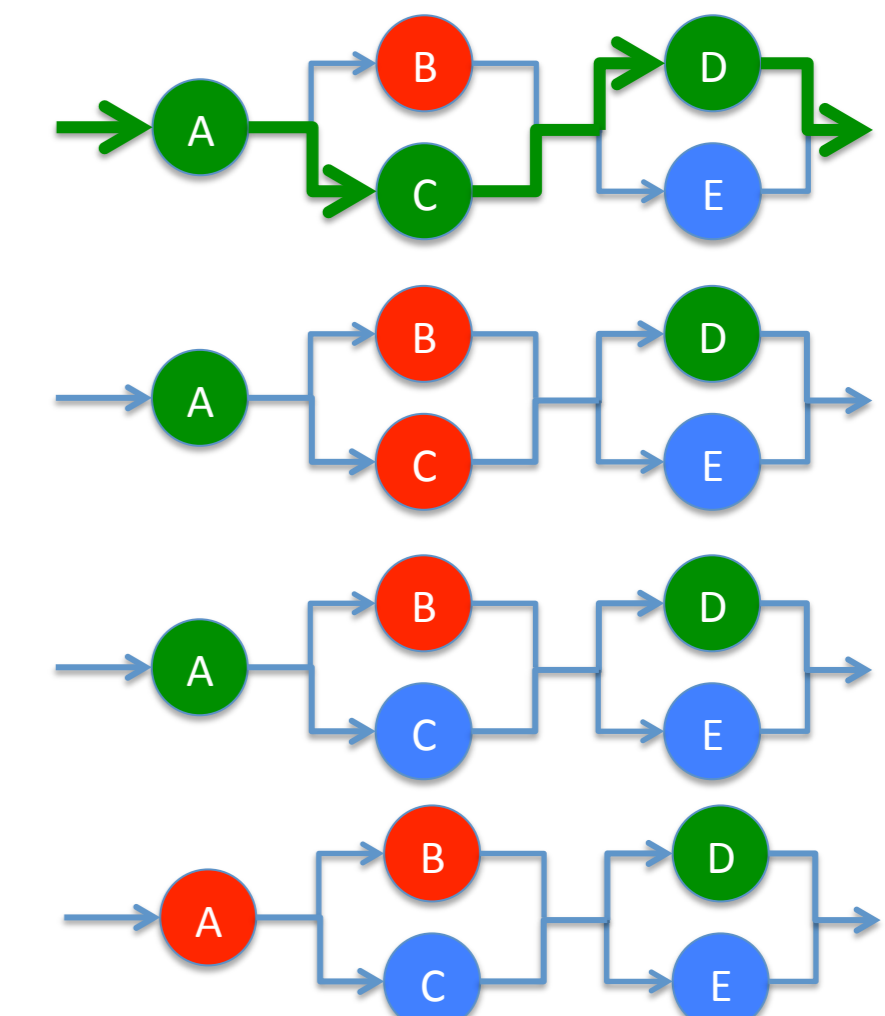
```
approval ::= role_element | op '[' approval_list ']'
approval_list ::= approval | approval approval_list
op ::= all | any
role_element ::= role | role ':' timeout ':' default
role ::= 'ManagerDataSite' | 'ManageDataGroup' | 'AdminSite' | 'ManagerGroup' |
        'AdminGlobal' | 'OperatorGlobal' | 'UnprivilegedUser' | 'AgentGlobal' |
        'PhEDExContactSite' | 'PhEDExContactGroup'
timeout ::= integer
default ::= 'y' | 'n'
```

Example: `all [ManagerGroup any [ManagerDataSite AdminSite:86400:y]]` – group manager and, site data manager or site admin, need to approve. If site admin does not act within a day, approval is automatically entered on behalf of him/her.

Evaluation of an approval plan

Consider approval plan as an electrical network and each node as a switch. To conclude a request is approved is to prove a circuit exists from the beginning to the end. To conclude a request is denied is to prove no such circuit can exist. Otherwise, it is undecided.

- A, C, and D approve, B denies, and E is still undecided. The request is approved since we can prove a circuit A-C-D.
- A and D approve, B and C deny, and E is undecided. The request is denied since we can prove no circuit exists.
- A and D approve, B denies, C and D are undecided. The request is undecided since we can not either prove a circuit exist nor prove it doesn't exist.
- Of course, if A denies, the request is denied regardless of other's decisions.



Putting it together

- When a new request type is defined, an approval plan template is defined
- When a request is created
 - An approval plan is instantiated from the template with the specific information supplied by the request
 - From the approval plan, all parties that match the roles will be resolved and notification is sent to each one
 - The request enters created state
- Each role shall make a decision regarding that request
 - If there are more than one party that match a role, whoever takes the action first sets the value on behalf of that role and others can not change.
 - Once a decision is made, the approval plan is evaluated
 - If the evaluation results in approved, the request is moved into approved state
 - If the evaluation results in denied, it enters denied state, which is final
 - Otherwise, the request remains in current state waiting for next decision
- In created state, certain roles may move the request to cancelled or locked state regardless of its approval status.
- When the request reaches one of the final states, it is finished, all transactions are archived.

Architecture

- TMDB – Oracle RDBM as persistent store
- DataService API – web based data transport system to access TMDB
- User interface – web pages for users to interact through DataService

User interfaces

There are two views of this request system for the users. Both take user's role and privilege into consideration.

- Detailed request view* – everything about a request, including state, approval status and list of actions
- Summary view* – one summary per line, including action list for each and global submit button

References

- Egeland R, Wildish T and Metson S 2008 Data transfer infrastructure for CMS data taking XII Advanced Computing and Analysis Techniques in Physics Research (Erice, Italy: Proceedings of Science)
- Egeland, R., Wildish, T. and Huang, C-H. PhEDEx Data Service. Journal of Physics: Conference Series, 219(062010), 2010.