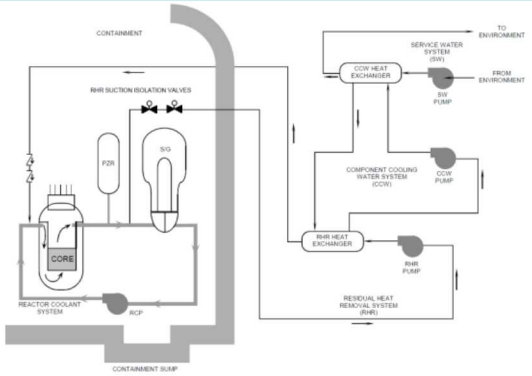
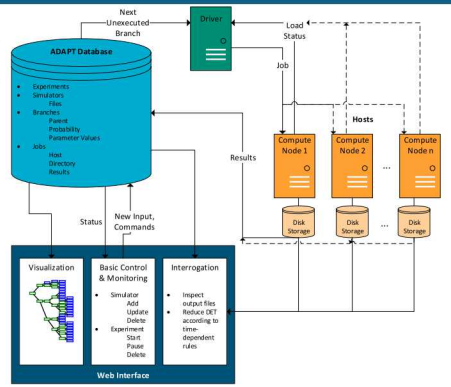
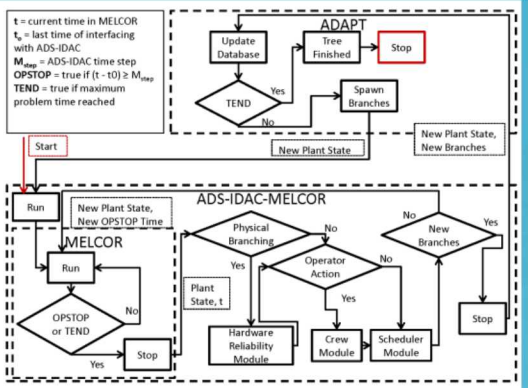
SAND2019-4755C

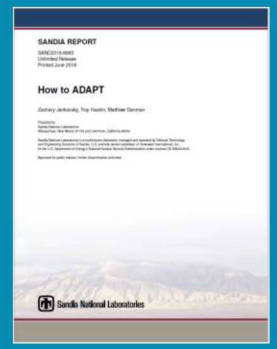
Sandia National Laboratories

ADAPT









PRESENTED BY

Zac Jankovsky

U.S. DEPARTMENT OF ENERGY NNSA

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Outline

- Motivation
  - **To address risk-related uncertainties relating to event timing, human interactions, and physical parameter values in a computationally efficient manner**
- Dynamic PSA and its Status
  - I will forget the name of the conference and say PRA/DPRA
- ADAPT
  - Approach
  - Applications
  - Project
  - Future
- Example ADAPT Analysis
  - Input Generation
  - Running
  - Interpreting

Dynamic PSA

Dynamic Probabilistic Safety Analysis (DPSA)

- Traditional PSA requires analysts to assume order of events
  - Does not explicitly account for timing of events
    - Will an event have different effects on incident progression based on its timing?
  - Uncertainties in event ordering increase with incident complexity and time
    - Especially if modeling human interactions
- Dynamic PSA is driven by time-resolving models of the relevant phenomena
  - Events occur according to physically-meaningful rules
    - E.g., hydrogen igniter success is queried only when a combustible mixture has accumulated
    - Does not venture into space that would never be reached physically
  - Events may re-occur as appropriate (e.g., valve cycles to failure)
  - Dynamic event trees (DETs) may be incorporated into a traditional PSA with relative ease

DPSA has been a long time coming…

Where are we on the spectrum of academic curiosity/paper generator to full-blown industrial application?

- Anonymous, 2015

DPSA has been a long time coming… (ML19066A390)

Technical Opinion Paper[1]

Dynamic PRA for Nuclear Power Plants: Not If But When?

N. Siu

Despite the considerable interest and effort, the tools developed by ongoing activities are not, to my knowledge, being used to support current U.S. NPP decisions.[7] This paper provides a brief summary of the reasons for continuing interest in dynamic PRA, potential pitfalls associated with its practice and use, and the status of the field. It then provides some cautions for reviewers of dynamic PRAs, and concludes with some personal opinions regarding where the field is headed and why the NRC should consider preparing for a future that includes dynamic PRA.

The incipient age of DPSA? (ML19038A473)

DG-1353 Pilot

Submittal to Support NRC Development and Implementation of DG-1353, A Guidance to Risk-Inform Application Development and Contents Including Event Selection and SSC Classification

OKLO

Final Safety Analysis Report

[redacted]

(xi)}{eci} **The DPRA tool used to drive the entire process is ADAPT, developed by Sandia National Laboratory [23]. The frequencies for the four analyzed event sequences are shown in Table 5-2 and Table 5-3.**

5.1.5.2.3 Consequence Analysis

[redacted]

Well, maybe. (ML19079A387)

5	Section 4.1, Dynamic PRA Page 116	General Observations concerning the use of dynamic PRA (DPRA): • DPRA has not been used to support any previous risk-informed NRC regulatory decision. The "learning curve" for the NRC staff and peer reviewers of the Oklo DPRA may be steep. • The non-LWR PRA standard is oriented toward conventional event-tree/fault-tree PRA methodology, although many of the high-level requirements and supporting requirements would also apply to DPRA. Has Oklo considered engaging the Joint Committee on Nuclear Risk management (JCNRM) concerning the applicability of the non-LWR PRA standard to DPRA? • The LMP guidance is oriented toward conventional event-tree/fault-tree PRA methodology. Has Oklo considered engaging NEI and developers of the LMP guidance concerning the use of DPRA?
17	PRA Acceptability	Can a DPRA meet the Non-LWR PRA Standard "ASME/ANS RA-S-1.4"? The time dependent PRA modeling may not fit the stationary sequence approach envisioned in the PRA Standard, which was written as a product of the IE frequencies and BE probabilities. If Oklo DPRA cannot meet the Non-LWR PRA Standard, how would Oklo justify its PRA acceptability in support of the DC application?
18	Other Uses of PRA	DPRA, does not provide the traditional risk important measures, would there be sufficient risk information to support other PRA applications during DC and COL stages, i.e., physical security (vital equipment, target sets), Tech Spec, environmental review (costs and benefits of SAMDAs), etc.

Stay Tuned.

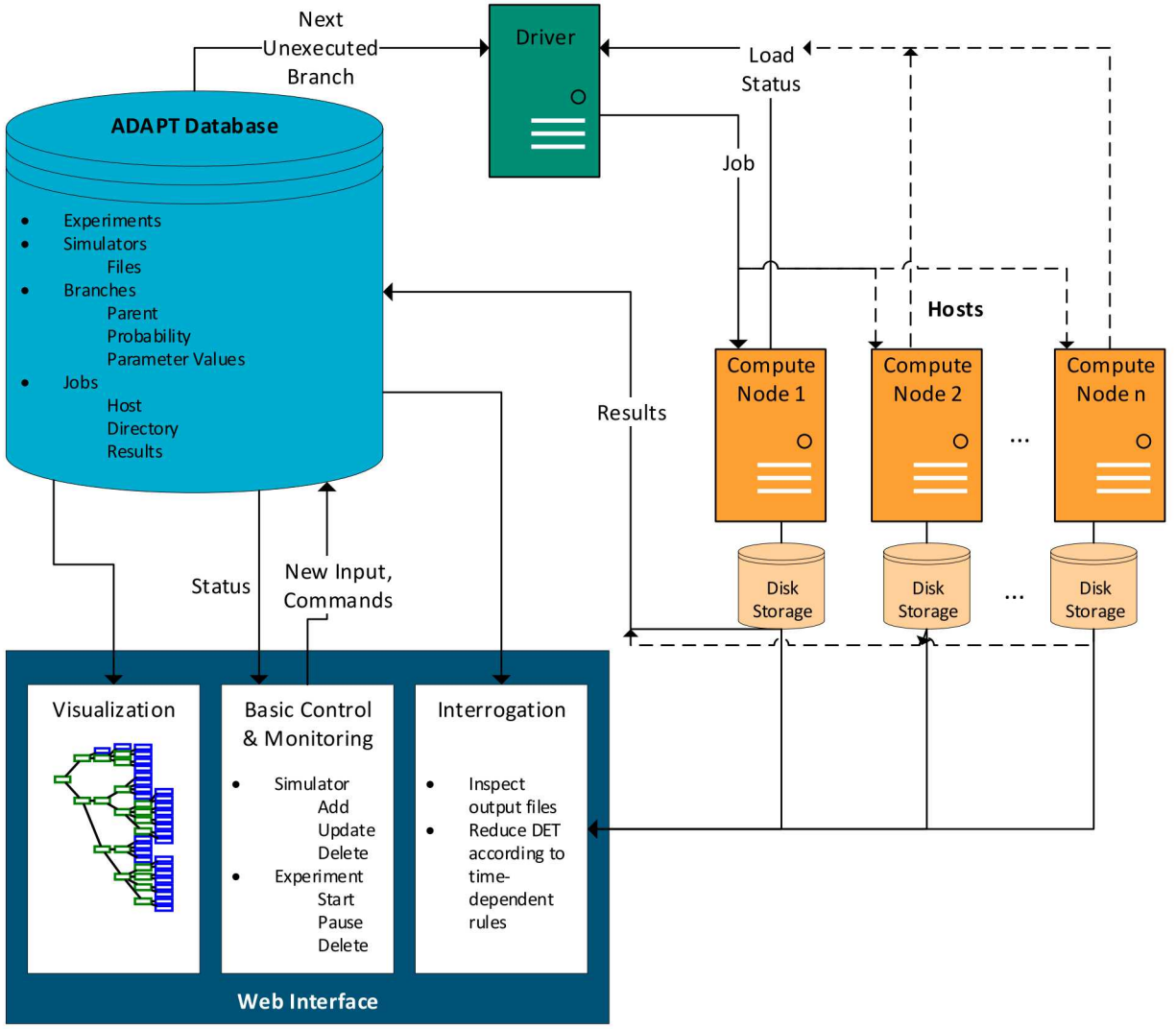
5	Section 4.1, Dynamic PRA Page 116	General Observations concerning the use of dynamic PRA (DPRA): • DPRA has not been used to support any previous risk-informed NRC regulatory decision. The "learning curve" for the NRC staff and peer reviewers of the Oklo DPRA may be steep. • The non-LWR PRA standard is oriented toward conventional event-tree/fault-tree PRA methodology, although many of the high-level requirements and supporting requirements would also apply to DPRA. Has Oklo considered engaging the Joint Committee on Nuclear Risk management (JCNRM) concerning the applicability of the non-LWR PRA standard to DPRA? • The LMP guidance is oriented toward conventional event-tree/fault-tree PRA methodology. Has Oklo considered engaging NEI and developers of the LMP guidance concerning the use of DPRA?
17	PRA Acceptability	Can a DPRA meet the Non-LWR PRA Standard "ASME/ANS RA-S-1.4"? The time dependent PRA modeling may not fit the stationary sequence approach envisioned in the PRA Standard, which was written as a product of the IE frequencies and BE probabilities. If Oklo DPRA cannot meet the Non-LWR PRA Standard, how would Oklo justify its PRA acceptability in support of the DC application?

- Dynamic PSA Standard Development Initiation – panel session
- Friday, 8:00-9:30, Emerald Salon One

ADAPT

Approach

- DET driver developed by SNL (2006-present)
  - Originally python 2.4, now targeting 3.6
    - Ships with RHEL8 and Ubuntu 18.04LTS
  - Generates DET database, launches jobs, and presents results
  - Jobs may be run on local machines up to HPCs
  - Supports linking multiple simulator codes
  - Calculates figures of merit using time-dependent output data
    - Dynamic sensitivity measures
- Simulator- and domain-agnostic
  - Simulators must meet a short list of requirements
    - Capable of restarting from saved state with new input
  - Simulator interactions performed via pre-negotiated files rather than shared memory
    - Traceability
    - Portability over diverse computational hosts



Applications

Years	System	Incident	Simulator(s)
2006-2011	PWR	SBO	MELCOR
2009	SFR	Aircraft Crash	RELAP5
2013	PWR	SBO	MELCOR
2013-2014	PWR	SBO	MELCOR
2014	HTGR	LOFC	MELCOR
2015-2017	PWR	SBO	MAAP4
2015-2017	SFR	TOP	SAS4A/SASSYS-1
2015-2018	PWR	ISLOCA	MELCOR, RADTRAD
2015-2018	BWR	SBO	MELCOR
2016-2018	SNF Cask	Derailment	STAGE, RADTRAN

PWR: Pressurized Water Reactor
SFR: Sodium-cooled Fast Reactor
HTGR: High Temperature Gas-cooled Reactor
BWR: Boiling Water Reactor
SNF: Spent Nuclear Fuel

SBO: Station Blackout
LOFC: Loss of Forced Cooling
TOP: Transient Overpower
ISLOCA: Interfacing System Loss of Coolant Accident

Select Applications

- Transient overpower
  - Pool-type metallic-fueled sodium-cooled fast reactor
  - SAS4A/SASSYS-1
  - Investigated epistemic uncertainties in reactivity coefficients
  - Other uncertain parameters:
    - TOP severity
    - Scram and primary pump trip success
    - Operator attempts to enhance heat removal
    - Operator override of primary pump thermal protection shutdown
- Residual heat removal interfacing system LOCA
  - Pressurized water reactor
  - MELCOR and RADTRAD
  - Epistemic uncertainties in material pressure capacity
  - Other uncertain parameters:
    - Operator mitigating action success and timing
    - Flooding of auxiliary building with primary water, may impede operator mitigation actions
      - Status of doors

Annals of Nuclear Energy 115 (2018) 55–72

Contents lists available at ScienceDirect

Annals of Nuclear Energy

journal homepage: www.elsevier.com/locate/anucene

Dynamic event tree analysis with the SAS4A/SASSYS-1 safety analysis code

Zachary K. Jankovsky [a,b,*], Matthew R. Denman [b], Tunc Aldemir [a]

[a] Nuclear Engineering Program, The Ohio State University, 201 West 19th Avenue, Columbus, OH, USA
[b] Risk & Reliability Analysis, Sandia National Laboratories, 1515 Eubank Boulevard SE, Albuquerque, NM, USA

ARTICLE INFO

Article history:
Received 6 September 2017
Received in revised form 2 January 2018
Accepted 5 January 2018
Available online 2 February 2018

Keywords:
ADAPT
SAS4A/SASSYS-1
SFR
DET
Dynamic PRA
Transient overpower

ABSTRACT

The consequences of a transient in an advanced sodium-cooled fast reactor are difficult to capture with the traditional approach to probabilistic risk assessment (PRA). Numerous safety-relevant systems are passive and may have operational states that cannot be represented by binary success or failure. In addition, the specific order and timing of events may be crucial which necessitates the use of dynamic PRA tools such as ADAPT. The modifications to the SAS4A/SASSYS-1 sodium-cooled fast reactor safety analysis code for linking it to ADAPT to perform a dynamic PRA are described. A test case is used to demonstrate the linking process and to illustrate the type of insights that may be gained with this process. Newly-developed dynamic importance measures are used to assess the significance of reactor parameters/constituents on calculated consequences of initiating events.

© 2018 Elsevier Ltd. All rights reserved.

1. Introduction

Computer models (simulators) are used in Probabilistic Risk Assessment (PRA) to inform the likelihood of possible outcomes given an initiating event and set of uncertain parameters. Simulators are regarded as repositories of knowledge of physical phenomena and can provide best-estimate values based on current knowledge (Schmidt et al., 2011). SAS4A/SASSYS-1 is one such simulator for Sodium-cooled Fast Reactors (SFRs) developed by Argonne National Laboratory (Fanning et al., 2017) (an earlier version (Fanning, 2012) has been used for this work). SAS4A/SASSYS-1 deterministically models coupled phenomena in a plant including point kinetics, thermal hydraulics, and mechanical changes such as expansion or failure of structural components. A SAS4A/SASSYS-1 model typically begins with steady-state operation, introduces a transient, and follows the progression of the transient through potential fuel damage and relocation within the core region.

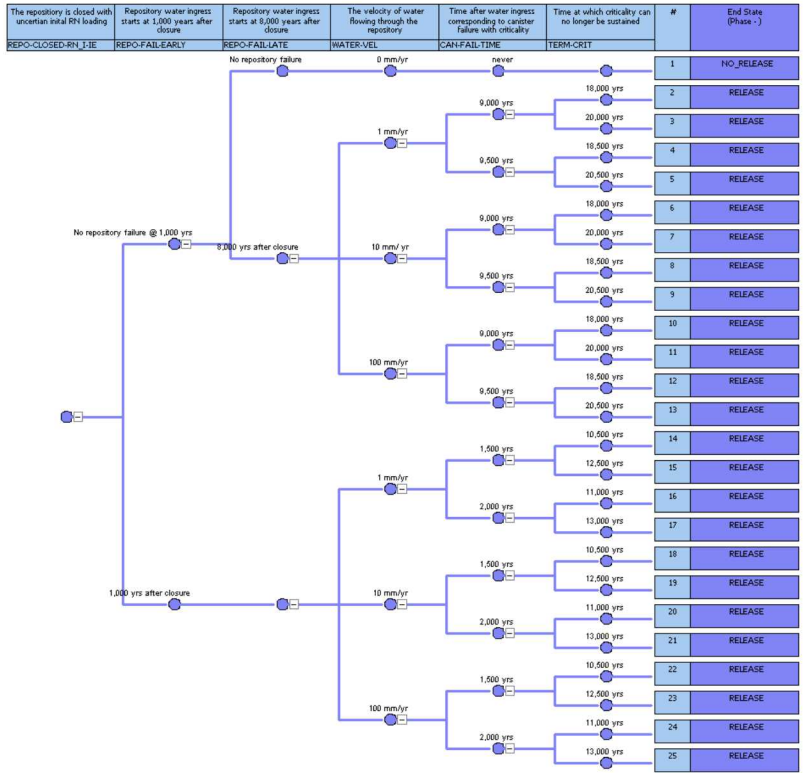
SFRs have traditionally used PRAs to support their licensing process. Clinch River Breeder Reactor (CRBR) conducted a multiple part PRA to evaluate its safety characteristics (Brasie et al., 1983). Power Reactor Innovative Small Module (PRISM) included a Level 3 PRA in its Preliminary Safety Information Document (PSID) submittal to the United States Nuclear Regulatory Commission (NRC) (PRISM Preliminary Safety Information Document, 1987). The general flow of a PRA for an SFR is shown in Fig. 1 (PRISM Preliminary Safety Information Document, 1987). Recently, work has been performed with the goal of developing licensing processes for advanced reactors in the United States (Grabaskas et al., 2016). The work in Grabaskas et al. (2016) includes an attempt to model the potential release and transport of radionuclides from SFR fuel beyond containment in the case of an accident.

The PRA representation shown in Fig. 1 potentially has a number of limitations including possibly subjective ordering of events and difficulty in capturing the effects of event timing, representing logic loops, and accounting for epistemic uncertainties (Aldemir, 2013). These limitations make the process challenging to apply for accident sequences where significant uncertainties exist regarding the behavior of inherent safety systems and/or operator actions (Jankovsky et al., 2015). The objective of this paper is to describe how such a traditional PRA flow can be augmented through a Dynamic Probabilistic Risk Assessment (DPRA) framework using SAS4A/SASSYS-1 simulations.

DPRA is a terminology used to describe methods that explicitly consider the time element in system evolution to account for complex hardware/process/software/human interactions. Among the DPRA methods proposed to date (Aldemir, 2013; Bucci et al., 2008), the Dynamic Event Tree (DET) approach is perhaps the most

* Corresponding author at: Nuclear Engineering Program, The Ohio State University, 201 West 19th Avenue, Columbus, OH, USA.
E-mail address: Zachary.Jankovsky@sandia.gov (Z.K. Jankovsky).

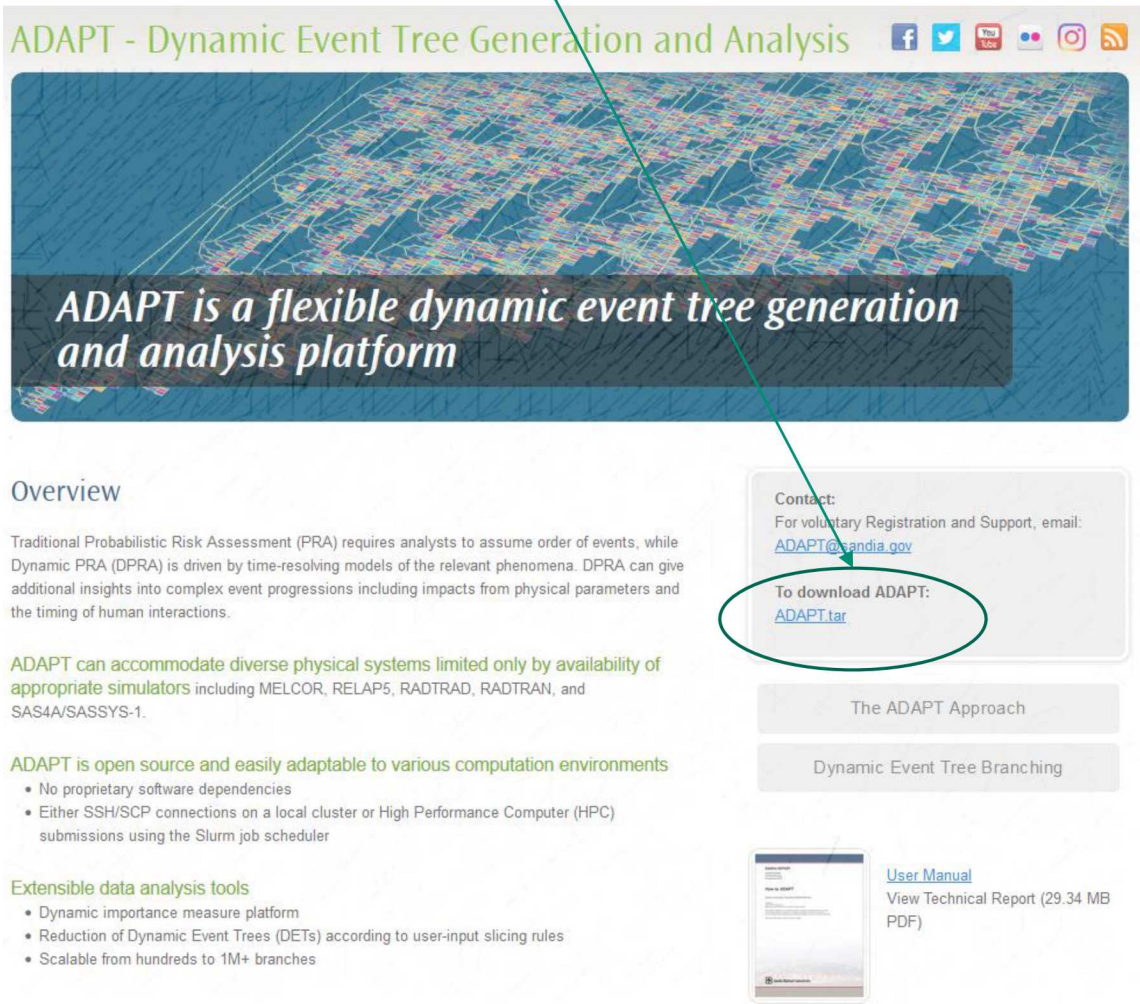
https://doi.org/10.1016/j.anucene.2018.01.001
0306-4549/© 2018 Elsevier Ltd. All rights reserved.



ADAPT as a Project

- Controlled under internal SNL GitLab repository
  - Two developers: all commits are checked before merging with master
  - Recently open-sourced with LGPL license
    - Users are free to modify their copy and integrate into larger projects
    - Read-only external access
- http://www.sandia.gov/adapt/
- Users
  - SNL: severe accident analysis and risk assessment
  - Universities: Ohio State and Maryland
  - Industry: Oklo
  - Others?
    - 108 unique website visitors
    - 12 code downloads

Version used for this demonstration



Future

- Project to link with PFLOTRAN has driven upgrades
  - PFLOTRAN models reactive flows, used for repository science
    - Mesh-based, natively parallel
  - Evaluating risk of criticality after breach of repository and waste container
- ADAPT has historically used single-threaded simulators
  - E.g., MELCOR, RELAP5, SAS4A/SASSYS-1
  - Job scheduling based around single-thread assumption
- "New" ADAPT release in summer 2019
  - Allow for multi-threaded simulators
  - Better scaling to large numbers of simultaneous jobs (10k+)
  - Updated coding techniques
    - Python has matured significantly since 2006
  - Reduced Windows incompatibilities
    - Not yet a Windows version of ADAPT
    - Tested on RHEL7 and Ubuntu 18.04

Example ADAPT Analysis

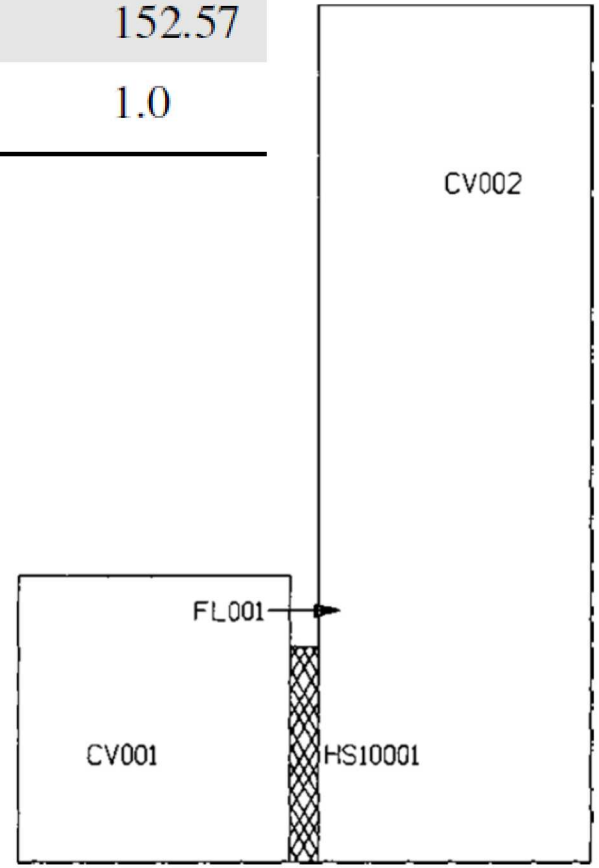
Example ADAPT Analysis

- Example case
- Required files
- Launching and tracking
- Processing outputs
- Generating plots and measures

Example Case

- Adapted from non-proprietary MELCOR assessment case
  - Small volume of water blowing down into larger volume of steam through valve, maximum time 300s
  - *SAND2015-6693R, 2.1 "Saturated Liquid Depressurization"*
- Uncertainties added:
  - Valve opening occurrence
  - Valve open fraction
  - CV1 water addition occurrence
    - If pressure falls below a set point
  - CV1 water addition rate

Parameter	CV001	CV002
Pressure (MPa)	7.999	0.01
Temperature (K)	568.23	568.23
Water Mass (kg)	72240	0.0
Steam Mass (kg)	0.0	152.57
Void Fraction	0.0	1.0



Required Files

Required Files

- **Template Simulator Input File (TSIF)**
  - A MELCOR input file with uncertain parameters replaced by ADAPT variables
    - When the BRF is applied to the TSIF, a valid MELCOR input file is created
- **Branching Rules File (BRF)**
  - Defines actions to take when a branching condition is reached
    - Replacements to be made in the TSIF to generate a unique branch input file
    - Conditional probabilities applied to parent branch total probability to calculate child branch total probabilities
- **Restart File**
  - Represents the state of the system before the uncertain phenomena become relevant
  - i.e., run MELCOR until just before the valve would be opened
- **Wrapper**
  - A script to be run for each branch
    - Generate input file using BRF and TSIF
    - Run MELCOR and extract plot variables
    - Interpret branch outcome and update ADAPT database

Template Simulator Input File (TSIF)

- Contains what is necessary to restart MELCOR with uncertain parameters changed
  - Does not require MELGEN input
  - Currently limited to a single file per simulator
  - Mostly control function (CF) value redefinitions
    - Depends on parameter database constraints of simulator
- Significant care to define input that has no physical impact if not triggered
    - …and has appropriate impact if triggered
    - …and does not trigger more or less often than appropriate
  - Test individual branching conditions separately and confirm intended effects
  - Requires a knowledgeable analyst for each simulator to be used

Template Simulator Input File (TSIF)

- Relevant CFs defined in MELGEN

```
! CF to be changed by ADAPT for open fraction of flow path.
cf_id 'FlowFrac' 10000 read
cf_sai 1.0 0.0 0.0

! Set time to stop for whether valve will open or not.
cf_id 'Stp-Alea-Vlv' 20001 read
cf_sai 1.0 0.0 1.e20

! Trigger for stop for whether valve will open or not.
cf_id 'Stp-Alea-Trig' 20002 l-gt
cf_liv FALSE
cf_arg 2
       1 exec-time               1.0 0.0
       2 cf-valu('Stp-Alea-Vlv') 1.0 0.0

! Set whether valve opening BC is armed.
cf_id 'Stp-Alea-Arm' 20003 l-read
cf_liv FALSE

! Stop and branch.
cf_id 'Stp-Alea-Go' 20004 l-and
cf_liv FALSE
cf_msg 2 'LIZARDKING 20004 determine whether valve opens or not'
cf_arg 2
       1 cf-valu('Stp-Alea-Trig')
       2 cf-valu('Stp-Alea-Arm')
```

Flow path open fraction refers to the value of CF 10000

MELCOR will stop and write a restart file if these CFs are both true

Upon stopping, MELCOR will signify ADAPT branching condition 20004 in the message file

Template Simulator Input File (TSIF)

- ADAPT variables set off by "{" and "}"
  - ADAPT finds and replaces variable names with values according to these opening and closing separators
  - {} not commonly used in MELCOR input, low chance of picking up an incorrect match
    - Separators are user-defined in BRF for each simulator
    - Commit to not using the chosen separator characters in TSIF comments

```
! CF to be changed by ADAPT for open fraction of flow path.
cf_id 'FlowFrac' 10000 read
cf_sai 1.0 0.0 {V10000}

! Set time to stop for whether valve will open or not.
cf_id 'Stp-Alea-Vlv' 20001 read
cf_sai 1.0 0.0 {V20001}

! Set whether valve opening BC is armed.
cf_id 'Stp-Alea-Arm' 20003 l-read
cf_liv {V20003}

! Stop and branch.
cf_id 'Stp-Alea-Go' 20004 l-and
cf_liv {V20004}
```

Change to alter the valve open fraction

Change to FALSE to disable branching condition even if the check time has been reached

Template Simulator Input File (TSIF)

- TSIF vs generated input file for a branch

```
! CF to be changed by ADAPT for open fraction of flow path.
cf_id 'FlowFrac' 10000 read
cf_sai 1.0 0.0 {V10000}

! Set time to stop for whether valve will open or not.
cf_id 'Stp-Alea-Vlv' 20001 read
cf_sai 1.0 0.0 {V20001}

! Set whether valve opening BC is armed.
cf_id 'Stp-Alea-Arm' 20003 l-read
cf_liv {V20003}

! Stop and branch.
cf_id 'Stp-Alea-Go' 20004 l-and
cf_liv {V20004}
```

```
! CF to be changed by ADAPT for open fraction of flow path.
cf_id 'FlowFrac' 10000 read
cf_sai 1.0 0.0 0.75

! Set time to stop for whether valve will open or not.
cf_id 'Stp-Alea-Vlv' 20001 read
cf_sai 1.0 0.0 0.0

! Set whether valve opening BC is armed.
cf_id 'Stp-Alea-Arm' 20003 l-read
cf_liv FALSE

! Stop and branch.
cf_id 'Stp-Alea-Go' 20004 l-and
cf_liv FALSE
```

- Valve is open to 75% of its full area
- Branching condition 20004 is disarmed

Template Simulator Input File (TSIF)

- Managing ADAPT stopping for MELCOR (simulator-dependent)
  - MELCOR 1.8.5 added ability to stop and write restart file on a CF becoming true using "EXEC_STOPCF"

```
EXEC_INPUT !(
EXEC_TITLE  'Depressurization (Basecase)'  ! Title of the calculation
EXEC_TEND 300.0            ! End of calculation time
EXEC_STOPCF 'ADAPTCF'
```

  - The stopping CF becomes true if any of the branching condition evaluators become true
    - Ensure that multiple branching conditions will not become true in the same time step

```
cf_id 'ADAPTCF' 99999 L-OR
cf_liv FALSE
cf_arg 4
       1 cf-valu('Stp-Alea-Go')
       2 cf-valu('Stp-Time-Go')
       3 cf-valu('Stp-Frac-Go')
       4 cf-valu('Stp-Source')
```

- SNL modification of SAS4A/SASSYS-1 5.0.3 implemented a similar function
  - Code is not released but process documented in SAND2017-4764
  - May be simpler to implement in SAS4A/SASSYS-1 5.1+

Branching Rules File (BRF)

- Provides information necessary to branch
  - TSIF to modify
  - Initial ADAPT variable values to produce input for first branch
  - Message file (and location within) to check for branching condition
  - Branch conditional probabilities
  - ADAPT variables to change for each branch
- One entry per line
  - Each line starts with a type signifier
  - Sections may be re-ordered from this example
  - If input is duplicated, a warning is given and the latest definition is used

Branching Rules File (BRF)

- Housekeeping definitions (1/2)

```
// Name of the simulator template input files with variables to be replaced by ADAPT.
InputFile 1 analytic1.cor.inp

// The files used to determine the branching code: Stoppingword <simulator> <filename> <magic word> <word on line that contains magic word>
StoppingWord 1 analytic1.mes LIZARDKING 2

// The characters used to designate ADAPT variables in the simulator template input files.
VarSeparator 1 "{" "}"

// The name of the simulators for the database.
SimulatorExecutable 1 melcor

// The simulator to run for the root branch.
InitialSimulator 1
```

  - TSIF for simulator 1 is “analytic1.cor.inp”
  - Opening and closing separators in the TSIF for simulator 1 are “{” and “}”
  - Simulator 1 runs “melcor”
    - Used in the wrapper to determine executable syntax
  - Simulator 1 is run for the first branch of the analysis

Branching Rules File (BRF)

- Housekeeping definitions (2/2)

```
// Name of the simulator template input files with variables to be replaced by ADAPT.
InputFile 1 analytic1.cor.inp

// The files used to determine the branching code: Stoppingword <simulator> <filename> <magic word> <word on line that contains magic word>
StoppingWord 1 analytic1.mes LIZARDKING 2

// The characters used to designate ADAPT variables in the simulator template input files.
VarSeparator 1 "{" "}"

// The name of the simulators for the database.
SimulatorExecutable 1 melcor

// The simulator to run for the root branch.
InitialSimulator 1
```

- When running simulator 1, search "analytic1.mes" for the branching condition that was reached
  - Use the "2"nd word on the **last** line that contains the string "LIZARDKING"
  - 23030 is the branching condition that was reached

```
Listing written TIME=  5.04385E+00 CYCLE=      143
Listing written TIME=  6.02351E+00 CYCLE=      162
Listing written TIME=  1.00776E+01 CYCLE=      203
Restart written   TIME =   1.007756E+01 CYCLE=      203
/SMESSAGE/  TIME= 1.24265E+02 CYCLE= 327
 MESSAGE FROM CONTROL FUNCTION PACKAGE
 CONTROL FUNCTION Stp-Source       BECAME .TRUE.
 LIZARDKING 23030 determine water addition
Listing written TIME=  1.25265E+02 CYCLE=      328
Restart written   TIME =   1.252649E+02 CYCLE=      328
Calculation terminated by: STOPCF CONTROL FUNCTION ADAPTCF
TIME= 1.25265E+02 CYCLE= 328 CPU =  1.47754E-01
```

Branching Rules File (BRF)

- Defining a branching condition (1/3)
  - EXEC_STOPCF starts as FALSE and is reset to FALSE after every stop
    - Branching condition evaluators should also be reset to FALSE after every stop
    - Necessity depends on order of operations in simulator
  - Initial values usually represent pre-transient conditions
  - The branching condition evaluator may be initially armed
    - May be dependent on another branching condition having occurred
    - May be disabled to generate a pared-down analysis

```
// Initial values not tied to a particular branching condition.
INIT V99999 FALSE // ADAPT stop CF

// The valve is initially closed.
INIT V10000 0.0

// The stop for whether to open the valve or not is initially armed but not triggered.
INIT V20001 0.0
INIT V20003 TRUE
INIT V20004 FALSE

BranchingSimulator 20004 1
BranchProbability  20004 1 0.5
BranchProbability  20004 2 0.5
BranchingConditionName 20004 Aleatory valve opening

// Open or do not open, there is no try.
// Do not open.
20004 1 V20003 FALSE
20004 1 V20004 FALSE
20004 1 V20011 1.e20
20004 1 V20021 1.e20
20004 1 V99999 FALSE
// Do open.
20004 2 V20003 FALSE
20004 2 V20004 FALSE
20004 2 V99999 FALSE
```

Branching Rules File (BRF)

- Defining a branching condition (2/3)
  - Define simulator to be run for child branches generated by this branching condition
  - Define conditional probabilities for each child branch
  - Define a branching condition name for visualization
    - …and whoever may be reading the input in 6 months

```
// Initial values not tied to a particular branching condition.
INIT V99999 FALSE // ADAPT stop CF

// The valve is initially closed.
INIT V10000 0.0

// The stop for whether to open the valve or not is initially armed but not triggered.
INIT V20001 0.0
INIT V20003 TRUE
INIT V20004 FALSE

BranchingSimulator 20004 1
BranchProbability  20004 1 0.5
BranchProbability  20004 2 0.5
BranchingConditionName 20004 Aleatory valve opening

// Open or do not open, there is no try.
// Do not open.
20004 1 V20003 FALSE
20004 1 V20004 FALSE
20004 1 V20011 1.e20
20004 1 V20021 1.e20
20004 1 V99999 FALSE
// Do open.
20004 2 V20003 FALSE
20004 2 V20004 FALSE
20004 2 V99999 FALSE
```

Branching Rules File (BRF)

- Defining a branching condition (3/3)
  - What to do for each branch?
  - If no input is given for an ADAPT variable, the parent branch value is used
    - May be INIT or another branching condition may have modified
  - Branch 1:
    - De-trigger the branching condition (V20004, V99999)
    - Disarm the branching condition (V20003)
    - Disarm dependent branching conditions (V20011, V20021)
  - Branch 2:
    - De-trigger the branching condition (V20004, V99999)
    - Disarm the branching condition (V20003)

```
// Initial values not tied to a particular branching condition.
INIT V99999 FALSE // ADAPT stop CF

// The valve is initially closed.
INIT V10000 0.0

// The stop for whether to open the valve or not is initially armed but not triggered.
INIT V20001 0.0
INIT V20003 TRUE
INIT V20004 FALSE

BranchingSimulator 20004 1
BranchProbability  20004 1 0.5
BranchProbability  20004 2 0.5
BranchingConditionName 20004 Aleatory valve opening

// Open or do not open, there is no try.
// Do not open.
20004 1 V20003 FALSE
20004 1 V20004 FALSE
20004 1 V20011 1.e20
20004 1 V20021 1.e20
20004 1 V99999 FALSE
// Do open.
20004 2 V20003 FALSE
20004 2 V20004 FALSE
20004 2 V99999 FALSE
```

Branching Rules File (BRF)

- Another branching condition
  - Tables may be used to define values for ADAPT variables
    - All child branches generated simultaneously
    - Conditional probability math:

$$CP_{23030,Branch\ 1,Table\ entry\ 3} = 0.1 * (0.50 - 0.25) = 0.025$$

    - The exact values (and CDF) entered will be used as input

```
// The stop for water addition is initially armed but not triggered.
// Triggering when CV1 falls below 3MPa.
INIT V23000 3.e6
INIT V23010 FALSE
INIT V23020 TRUE
INIT V23030 FALSE
INIT V23040 0.0

BranchingSimulator 23030 1
BranchProbability  23030 1 0.1
BranchProbability  23030 2 0.9
BranchingConditionName 23030 Water addition

23030 1 V23040 t2
23030 1 V23000 0.0
23030 1 V23010 FALSE
23030 1 V23020 FALSE
23030 1 V23030 FALSE
23030 1 V99999 FALSE

23030 2 V23000 0.0
23030 2 V23010 FALSE
23030 2 V23020 FALSE
23030 2 V23030 FALSE
23030 2 V99999 FALSE

t2 5  0.0 100.0 500.0 2000.0 5000.0
t2p 5 0.0 0.25  0.50  0.75   0.999
TableProbabilityType t2 CDF
```

Restart File

- Initial state is defined in MELGEN input
  - MELGEN generates a restart file
    - This restart could be used by ADAPT
- Depending on the situation, it may be advantageous to use a later restart state
  - As long as the phenomena examined using ADAPT have not yet come into play
  - E.g., extending an existing Level 1 analysis to investigate core degradation and release uncertainties
- Restart file is updated by MELCOR during each branch
  - Stopping for a branching condition triggers a dump to the file
  - File is passed from parent to each child branch

Wrapper

- Defines the steps to follow for each branch
  - Will demonstrate required and common elements
  - Significant flexibility in pre- and post-processing
- Must be executable on every computation host
  - Original ADAPT cases used bash
  - All new wrappers written in python
    - Easier to implement commonly-used commands and control sequences
    - Step toward Windows ADAPT

Wrapper

- Generate input file for the initial branch

```
# Check if this is the root branch.
if (str(os.getenv('NCENGINE_ROOT')) == '1') and (str(os.getenv('NCENGINE_RESUMING_CHECKPOINT')) == '0'):
    print('initializing edit rules for root job at %s' % (time.asctime()))
    shutil.copy2(os.path.join(MELCOR_ROOT, RST), this_dir)
    shutil.copy2(os.path.join(MELCOR_ROOT, SIM1TEMPLATE), this_dir)
    shutil.copy2(os.path.join(MELCOR_ROOT, EDITRULES), this_dir)

    editrule_cmd = 'adapt-editrule-apply --init %s %s 0 melcor %s' % (EDITRULES, BRANCHESF, OTHERTEMP)
    print(editrule_cmd)
    f = subprocess.Popen(editrule_cmd, shell=True, stdout=subprocess.PIPE, stderr=subprocess.STDOUT)
    f_out = f.stdout.readlines()

    # No loop here, because there is only one branch for the initial simulator for the root branch.
    branchesf_content = open(BRANCHESF, 'r').readlines()[0]
    shutil.copy2(branchesf_content.split()[0], os.path.join(this_dir, SIM1INPUT))
    os.remove(branchesf_content.split()[0])
```

- Run MELCOR

```
if THIS_EXECUTABLE == 'melcor':
    # Execute MELCOR.
    print('started executing melcor at %s' % (time.asctime()))
    print('ls is:')
    this_dir_list = sorted(os.listdir(this_dir), key=str.lower)
    for listed_file in this_dir_list:
        (mode, ino, dev, nlink, uid, gid, size, atime, mtime, ctime) = os.stat(str(listed_file))
        mod_date = datetime.datetime.fromtimestamp(float(mtime)).strftime('%B %d %Y %X')
        print('Name: %s, Size: %s, Modified: %s' % (listed_file, str(size), mod_date))

    sim1_exec_command = 'echo E | nice %s %s' % (os.path.join(MELCOR_ROOT, SIM1EXE), SIM1INPUT)
    f = subprocess.Popen(sim1_exec_command, shell=True, stdout=subprocess.PIPE, stderr=subprocess.STDOUT)
```

Wrapper

- Handle checkpoint and early termination

```
# If we were checkpointed, stop here.
if os.path.isfile('adapt.cp'):
    print('we were checkpointed at %s on %s, removing analytic1.stp' % (time.asctime(), os.uname()[1].split('.')[0]))
    os.remove('adapt.cp')
    f = subprocess.call('adapt-checkpoint-note-taken', shell=True)
    sys.exit(0)

# Early termination
if os.path.isfile('term-early'):
    print('jobid %s terminated early at %s on %s, recording result to database' % (jobid, time.asctime(), os.uname()[1].split('.')[0]))
    f = subprocess.call(('adapt-job-record jobid %s term_early 1' % (jobid)), shell=True)
    sys.exit(0)
```

- Some MELCOR-specific post-processing – truncate restart file

```
if THIS_EXECUTABLE == 'melcor':
    # Truncate restart file to save space.
    print('truncating at %s' % time.asctime())
    SIM1INPUT_2 = 'analytic1.cor2'
    os.rename('analytic1.mes', 'analytic1.mes.tmp')

    SIM1INPUT_contents = open(SIM1INPUT, 'r').read()
    SIM1INPUT_contents = SIM1INPUT_contents.replace('''!MEL_RFMOD       'newrestart.rst' -1''', '''MEL_RFMOD       'newrestart.rst' -1''')
    f = open(SIM1INPUT_2, 'w')
    f.write(SIM1INPUT_contents)
    f.close()
    subprocess.call('echo E | %s %s' % (os.path.join(MELCOR_ROOT, SIM1EXE), SIM1INPUT_2), shell=True)
    os.rename('analytic1.rst', 'analytic1-nontruncated.rst')
    os.remove('analytic1-nontruncated.rst')
    os.rename('newrestart.rst', 'analytic1.rst')
    os.rename('analytic1.mes.tmp', 'analytic1.mes')
    print('done truncating at %s' % time.asctime())
```

Wrapper

- Some MELCOR-specific post-processing – extracting plot files

```
# Extracting MELCOR data.
extracting_plots = True
if (THIS_EXECUTABLE == 'melcor') and (extracting_plots):
    plot_command = '''\
        melcor_plot analytic1.ptf analytic1_ separate CVH-P_1 CVH-P_2 CFVALU_10000 CFVALU_20001 CFVALU_20002 CFVALU_20011 CFVALU_20012 CFVALU_20021 CFVALU_20022 CFVALU_99999 \
        '''
    f = subprocess.call(plot_command, shell=True)
```

- Read the message file for information on why MELCOR stopped

```
if THIS_EXECUTABLE == 'melcor':
    stop_word = 'LIZARDKING'
    # Gather some attributes about the completed branch.
    melcor_message_file = 'analytic1.mes'
    melcor_message_file_contents = open(melcor_message_file, 'r').readlines()
    for line in reversed(melcor_message_file_contents):
        if stop_word in line:
            mystopping_code = line.split(stop_word)[1].split()[0]
            break
    for line in reversed(melcor_message_file_contents):
        if line.startswith(' TIME='):
            sim_elapsed = float(line.split(' TIME=')[1].split()[0])
            break
    for line in reversed(melcor_message_file_contents):
        if line.startswith(' Normal termination  TIME='):
            normal_term = float(line.split(' Normal termination  TIME=')[1].split()[0])
            sim_elapsed = normal_term
            break
```

Wrapper

- Interpret the reason for stopping
  - Try to record a branching condition being reached
  - If that fails, try to record maximum time being reached
    - Exit if maximum time is reached
  - If that fails, the simulator is assumed to have failed somehow
    - Exit if simulator failed

```
# At this point normal_term, sim_elapsed, and mystopping_code all MAY exist.
# See if we have a branching condition.
submitted_bc = False
submitted_terminal = False
try:
    f = subprocess.call(('adapt-job-record code %s simtime %s' % (mystopping_code, str(sim_elapsed))), shell=True)
    print('Submitted result of job %s: branching condition %s at time %s.' % (jobid, mystopping_code, str(sim_elapsed)))
    submitted_bc = True
except:
    pass

# See if we have normal termination of melcor for max time.
try:
    f = subprocess.call(('adapt-job-record max_time %s simtime %s' % (str(normal_term), str(sim_elapsed))), shell=True)
    print('Submitted result of job %s: maximum problem time %s.' % (jobid, str(normal_term)))
    print('Exiting job %s, as maximum problem time has been reached.' % (jobid))
    submitted_terminal = True
except:
    pass

if submitted_terminal is True:
    sys.exit(0)

# If nothing else has been submitted, we've failed somehow.
if (submitted_terminal is False) and (submitted_bc is False):
    try:
        f = subprocess.call(('adapt-job-record logical_fail 1 simtime %s' % (str(sim_elapsed))), shell=True)
        print('Submitted result of job %s: simulator (or wrapper) failure at time %s.' % (jobid, str(sim_elapsed)))
        print('Exiting job %s, as a failure occurred in either the simulator or wrapper. Marking the branch as finished.' % (jobid))
        submitted_terminal = True
    except:
        f = subprocess.call(('adapt-job-record logical_fail 1'), shell=True)
        print('Submitted result of job %s: simulator (or wrapper) failure. No time available.' % (jobid))
        print('Exiting job %s, as a failure occurred in either the simulator or wrapper. Marking the branch as finished.' % (jobid))
        submitted_terminal = True

if submitted_terminal is True:
    sys.exit(0)
```

Wrapper

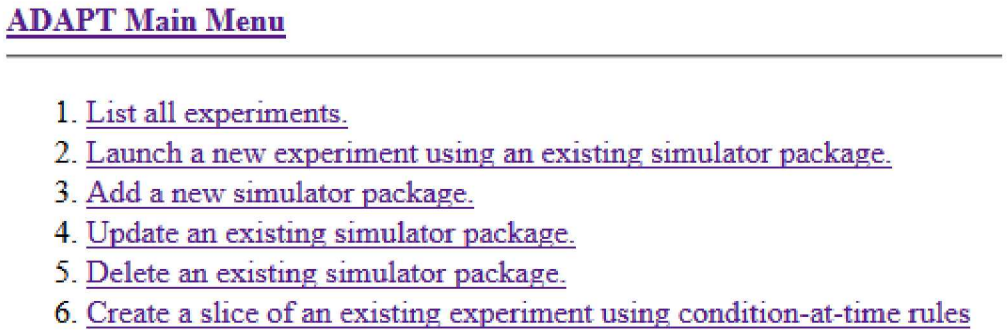
- If a branching condition was reached, submit new branches and their input files

```
# If we reach this line, there is a branching condition to be handled.

# Create child branches and submit them to the database. Check $THIS_EXECUTABLE to send correct simulator just run to the database.
editrule_cmd = 'adapt-editrule-apply %s %s %s %s %s' % (EDITRULES, BRANCHESF, str(sim_elapsed), THIS_EXECUTABLE, OTHERTEMP)
print(editrule_cmd)
f = subprocess.Popen(editrule_cmd, shell=True, stdout=subprocess.PIPE, stderr=subprocess.STDOUT)
f_out = f.stdout.readlines()

# Identify the new simulator.
branchesf_content = open(BRANCHESF, 'r').readlines()[0]
NEWSIM = branchesf_content.split()[7]

# Submit new branches.
branchesf_content = open(BRANCHESF, 'r').readlines()
for line in branchesf_content:
    (config, newstate, branchnum, stopcode, branchhit, probability, terminate_early, newsim) = line.split()
    submit_cmd = ''
    submit_cmd += 'adapt-submit                '
    submit_cmd += '--terminate_early %s        ' % (terminate_early)
    submit_cmd += '--executable      %s        ' % (newsim)
    submit_cmd += '--handoff         %s %s     ' % (EDITRULES, EDITRULES)
    submit_cmd += '--handoff    %s %s.state    ' % (newstate, EDITRULES)
    submit_cmd += '--handoffref %s %s          ' % (RST, RST)
    submit_cmd += '--handoffref %s %s          ' % (SIM1TEMPLATE, SIM1TEMPLATE)
    submit_cmd += '--handoff    %s %s          ' % (config, SIM1INPUT)
    if os.path.isfile('analytic1.mes'):
        submit_cmd += '--handoffref analytic1.mes analytic1.mes '
    submit_cmd += '--probability %s ' % (probability)
    submit_cmd += 'analytic1                   '
    submit_cmd += '"sb %s"   ' % (branchnum)
    submit_cmd += '%s' % (THISSCRIPT)
    print(submit_cmd)
    f = subprocess.Popen(submit_cmd, shell=True, stdout=subprocess.PIPE, stderr=subprocess.STDOUT)
    f_out = f.stdout.readlines()
    os.remove(newstate)
    os.remove(config)
    print(f_out)
```

Launching and Tracking

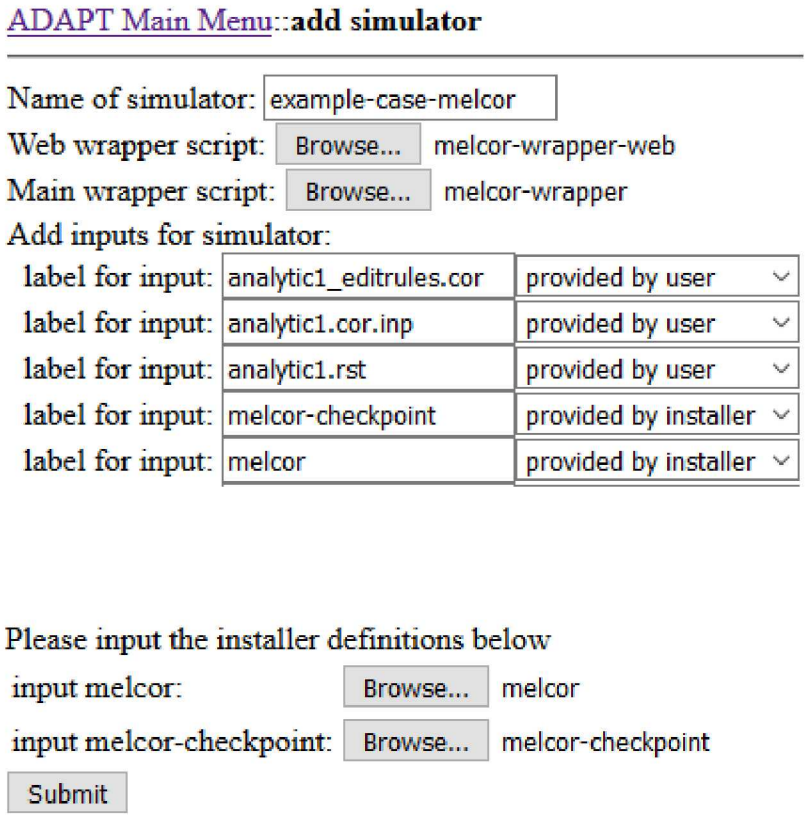
Launching and Tracking

- Through the web interface
  - Add simulator package
  - Add experiment
  - Check experiment listing
  - Visualize an experiment
  - View branch details



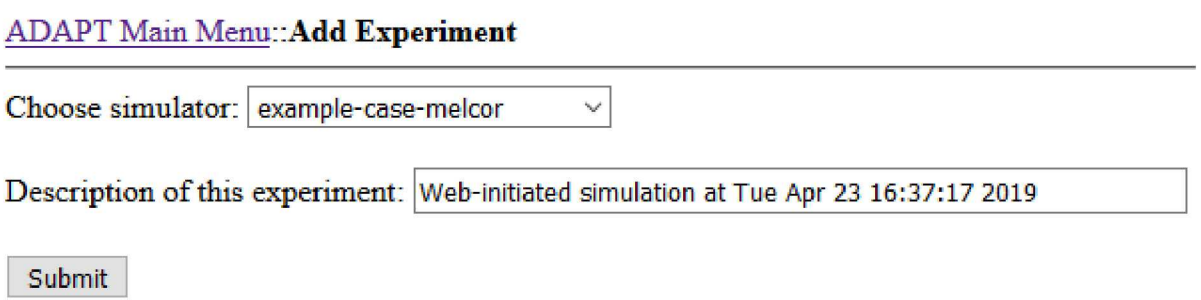
Add Simulator Package

- Add all required files, even if not populating now
  - Provided by user: populate at time of experiment launch
    - What files will change with each experiment?
  - Provided by installer: populate now
- Web wrapper and wrapper required
  - Web wrapper will be dropped in new version of ADAPT
- Prompt for installer-provided files
  - Consider max_allowed_packet for the SQL server



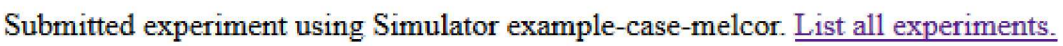
Add Experiment

- Choose the simulator package to use



- Prompt for user-provided files

- Confirmation of successful submission of the experiment and first branch



Check Experiment Listing

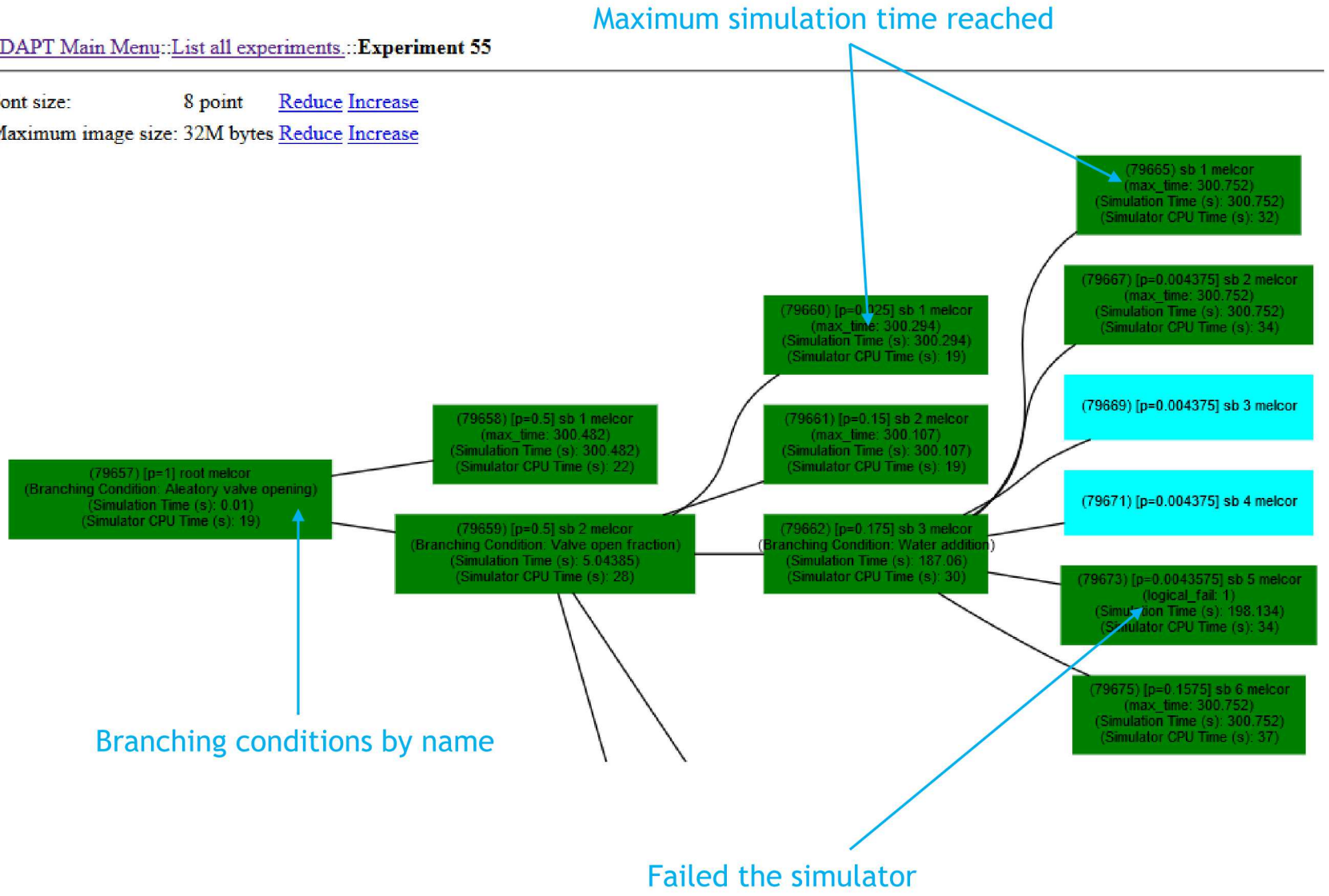
ADAPT Main Menu::**List Experiments** (refresh)

Experiment#	Name	Description	State	Branches (Total)	Branches (Completed)	Branches (Running)	End States			
1	jelly	Web-initiated simulation at Fri Jul 13 18:36:06 2018	*Checkpointed*	2915	1980	0	1859	Runtime	Restart	Delete
4	jelly	focus on upper head temperature reduced copy of Web-initiated simulation at Fri Jul 13 18:36:06 2018	*Checkpointed*	2483	1175	0	1594	Runtime	Restart	Delete
28	jelly	SAMG Dynamic Timestep Thu Jul 19 21:28:12 2018	*Checkpointed*	1236	666	0	744	Runtime	Restart	Delete
29	jelly	Web-initiated simulation at Tue Jul 24 21:09:26 2018	*Checkpointed*	8259	7935	0	5278	Runtime	Restart	Delete
30	jelly	4hr reduced copy of Web-initiated simulation at Tue Jul 24 21:09:26 2018	*Checkpointed*	8181	7793	0	5321	Runtime	Restart	Delete
33	vienna	Web-initiated simulation at Tue Dec 18 15:19:59 2018	*Finished*	38006	38006	0	13698	Runtime		Delete
34	melcor	Web-initiated simulation at Tue Dec 18 15:20:23 2018	*Finished*	1526	1526	0	950	Runtime		Delete
49	analytic1	Web-initiated simulation at Fri Apr 19 13:12:37 2019	*Finished*	26	26	0	21	Runtime		Delete
52	analytic1	Web-initiated simulation at Tue Apr 23 16:37:17 2019	*Active*	1	0	1	1	Runtime	Checkpoint	Delete

- Branch counts give an idea of progress
  - Although they may grow quickly if interesting phenomena are reached simultaneously in many branches

Visualize an Experiment

- Information for each branch
  - End simulation time
  - Reason for ending
  - Time required to run
  - Total probability
- May quickly become too large to feasibly visualize



Processing Outputs

Processing Outputs

- Many simulators generate a binary plot file by default
  - Need to be extracted to a simple text format for ADAPT to use

```
# Extracting MELCOR data.
extracting_plots = True
if (THIS_EXECUTABLE == 'melcor') and (extracting_plots):
    plot_command = '''\
        melcor_plot analytic1.ptf analytic1_ separate CVH-P_1 CVH-P_2 CFVALU_10000 CFVALU_20001 CFVALU_20002 CFVALU_20011 CFVALU_20012 CFVALU_20021 CFVALU_20022 CFVALU_99999 \
        '''
    f = subprocess.call(plot_command, shell=True)
```

```
[zkjanko@tree1 b079660-j074865]$ cat analytic1_CVH-P_1
5.0438499 7999044.0
5.1346221 7999023.0
5.2976999 7998985.0
5.3976998 7998961.0
5.4977002 7998937.0
5.5977001 7998912.5
5.6977000 7998888.5
5.7976999 7998864.5
5.8976998 7998840.0
5.9977002 7998816.0
6.0977001 7998791.5
6.1977000 7998767.5
```

```
[zkjanko@tree ~]$ adapt-json-outputs 55 analytic1_CVH-P_1 /home/zkjanko False 12
Checking for plot files under directories associated with expid 55. Total 26 branches.
Checking jobdir /scratch-local/zkjanko/e55/b079657-j074862 on host tree01 for plotfile analytic1_CVH-P_1.
Checking jobdir /scratch-local/zkjanko/e55/b079658-j074863 on host tree02 for plotfile analytic1_CVH-P_1.
Checking jobdir /scratch-local/zkjanko/e55/b079659-j074864 on host tree03 for plotfile analytic1_CVH-P_1.
...
Per availability and/or input argument, get_exp_heritage_plot_data is pulling new plot data. This may take some time.
Total end states: 21
Processing plot data for host:file tree02:/scratch-local/zkjanko/e55/b079658-j074863/analytic1_CVH-P_1.
Processing plot data for host:file tree01:/scratch-local/zkjanko/e55/b079657-j074862/analytic1_CVH-P_1.
Processing plot data for host:file tree03:/scratch-local/zkjanko/e55/b079659-j074864/analytic1_CVH-P_1.
...
```

- ADAPT can concatenate properly-formatted plot data across branches
  - Save to JSON format
    - May be GB scale

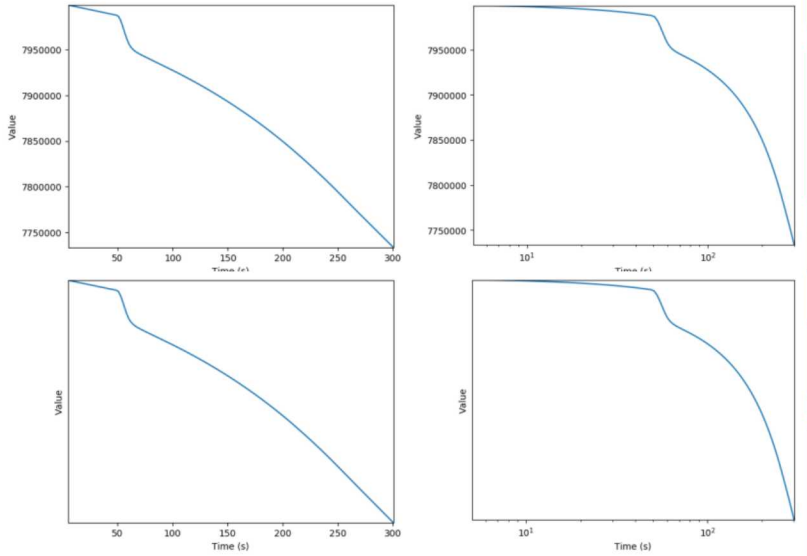
Processing Outputs

- *parallel-job-runner*: Executes a user-specified script in each job directory for an experiment
  - E.g., wish to extract an additional plot variable from the binary plot files
  - Splits list of directories into n lists where n is a user-input number of independent processes to open
- *adapt-fixprob:* Allows a different set of conditional probabilities to be applied to a branching condition across an experiment
  - E.g., updated belief in distribution for an uncertainty
  - May choose to commit to the database

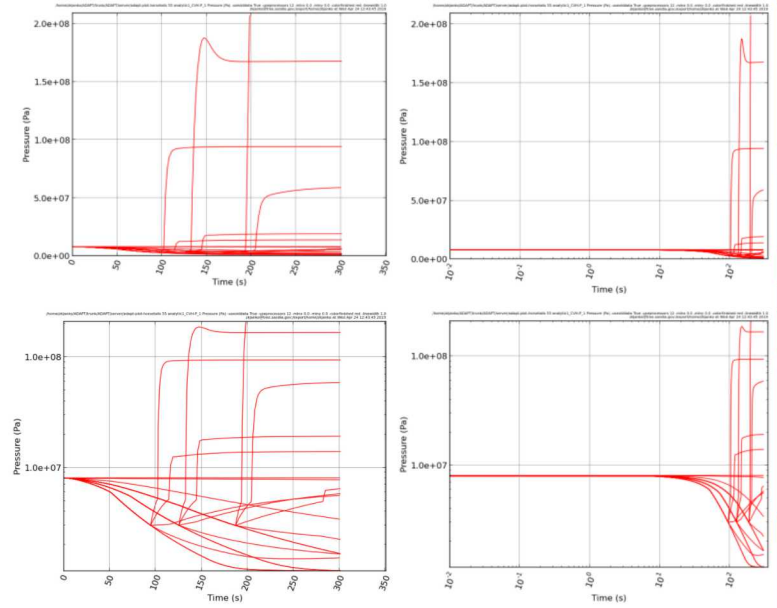
Generating Plots and Measures

Generating Plots and Measures

- *plot_pdf*: Generates simple images from a single properly-formatted plot file
  - Meant for confirming basic behavior in a single branch



- *adapt-plot-horsetails*: Generates images from a single plot file concatenated across an experiment
  - Apply different colors to finished/unfinished scenarios
  - Calculate and display median/mean values
  - Use cached copy of plot data
    - Allows a snapshot to be taken mid-way through an experiment



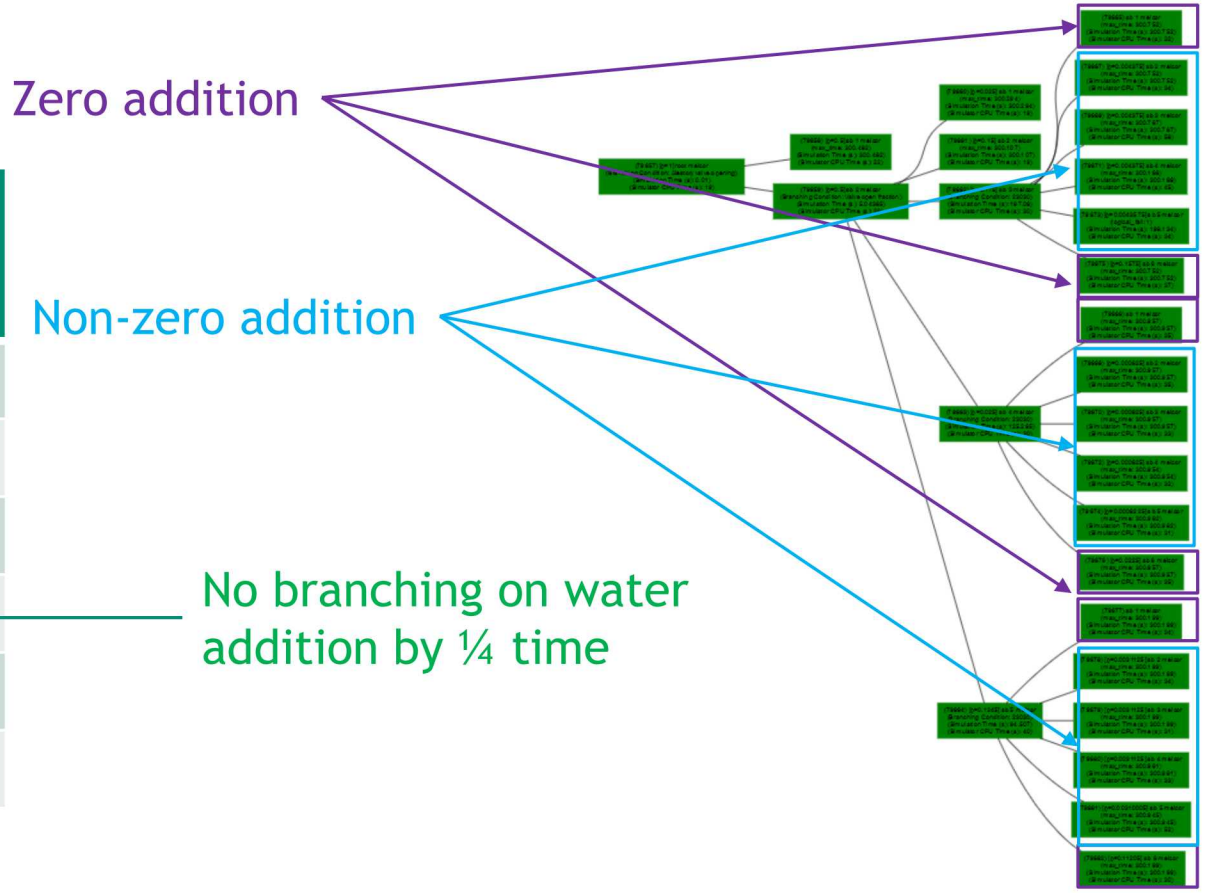
Generating Plots and Measures

- *dynamic_importance_measure_calculation*
- Divides scenarios into sets by parameter value
  - Event occurs
  - Event does not occur
  - Event occurs at some time or with some severity
    - Includes continuous physical parameters with no concept of "occurrence"
- Uses representative value of plot variable for each scenario
  - Final value
  - Peak/trough value
  - Value at ¼ or ½ or ¾ of simulation time
  - All plot time steps
    - Involves matching plot time steps, e.g., "300.752" vs "300.787" vs "300.166"
- Compare expected value in each set
  - Currently includes ratio measures
  - Differences may also yield insight

Generating Plots and Measures

- *dynamic_importance_measure_calculation*
- Examine effect of occurrence of water addition to CV1 on CV1 pressure
  - Effectively comparing all scenarios with CF 23040 value "0.0" to all scenarios with different values

Type	Ratio Pres.(occurrence)/ Pres.(non-occurrence)
Peak Value	6.8
Lowest Value	1.9
Last Value	36.3
¼ Time Step	1.0
½ Time Step	6.0
¾ Time Step	12.6

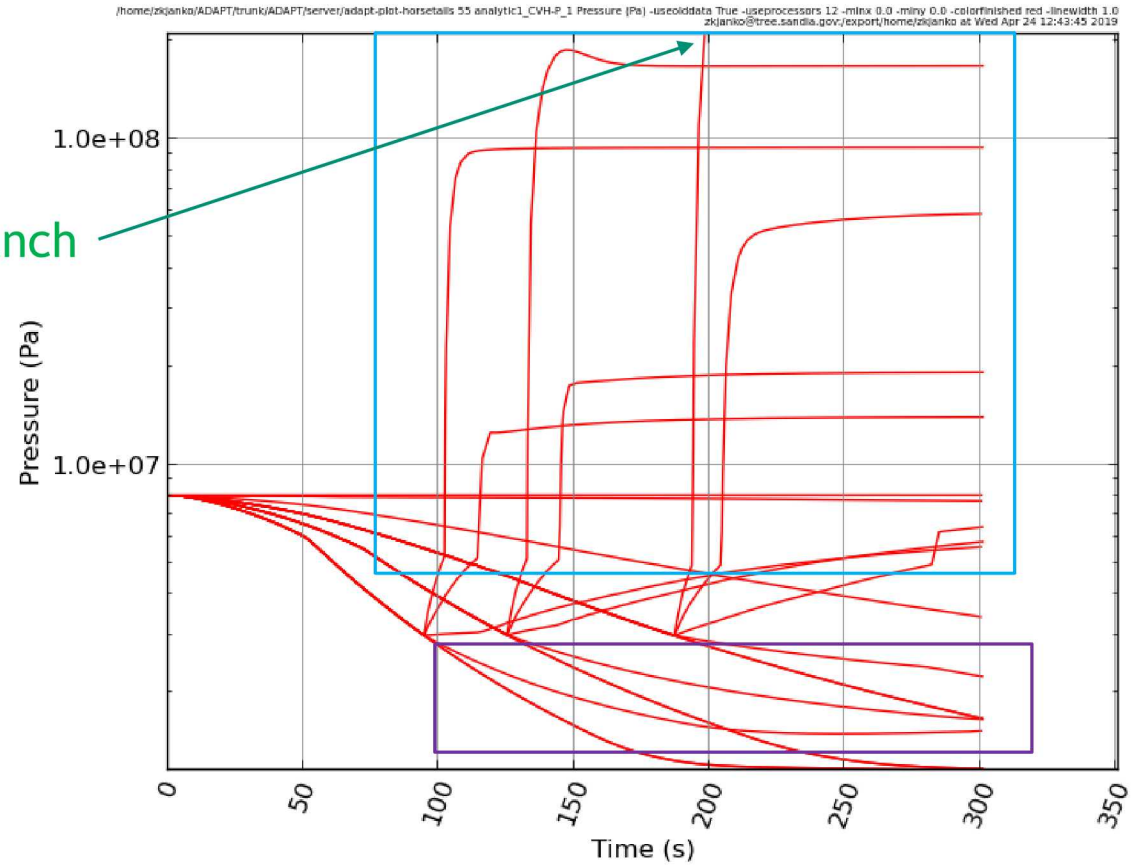


Generating Plots and Measures

- *dynamic_importance_measure_calculation*
- Examine effect of occurrence of water addition to CV1 on CV1 pressure
  - Effectively comparing all scenarios with CF 23040 value "0.0" to all scenarios with different values

Type	Ratio Pres.(occurrence)/ Pres.(non-occurrence)
Peak Value	6.8
Lowest Value	1.9
Last Value	36.3
¼ Time Step	1.0
½ Time Step	6.0
¾ Time Step	12.6

Failed branch



Wrapping Up

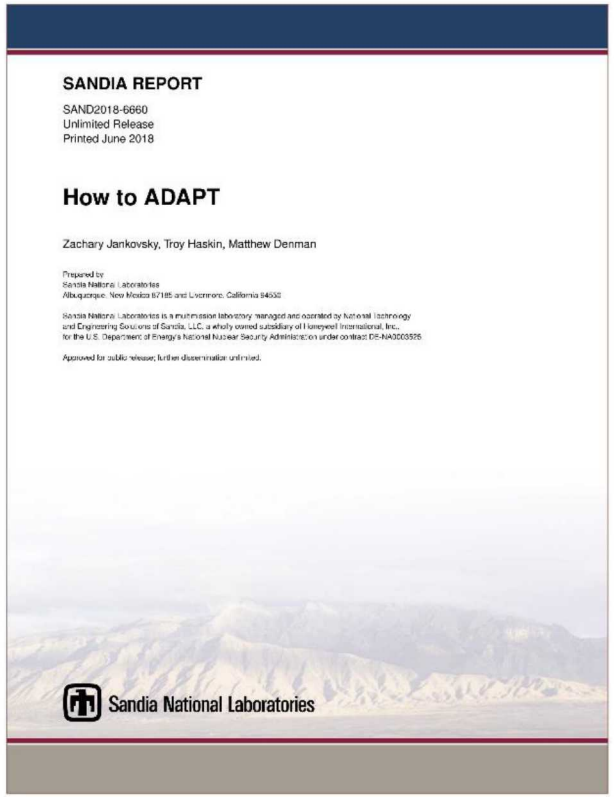
ADAPT Summary

- DPRA can give additional insight to complex event progressions
  - What physical parameters are impactful?
  - How do timing and human interactions affect the outcome?
- ADAPT is a flexible DET generation and analysis platform
  - Diverse physical systems
    - Nuclear power plant safety
    - Shipping cask safety/security/safeguards
    - Limited only by availability of appropriate simulators
  - Easily adaptable to various computational environments
    - No proprietary software dependencies
    - SSH/SCP connections or HPC submission using the Slurm job scheduler
  - Extensible data analysis tools
    - Dynamic importance measure platform
    - Reduction of DETs
    - Scalable from hundreds to 1M+ branches

New ADAPT coming soon

Getting ADAPT

- http://www.sandia.gov/adapt/
- Current manual: SAND2018-6660
- SFR TOP analysis with SAS4A/SASSYS-1



SANDIA REPORT

SAND2018-6660
Unlimited Release
Printed June 2018

How to ADAPT

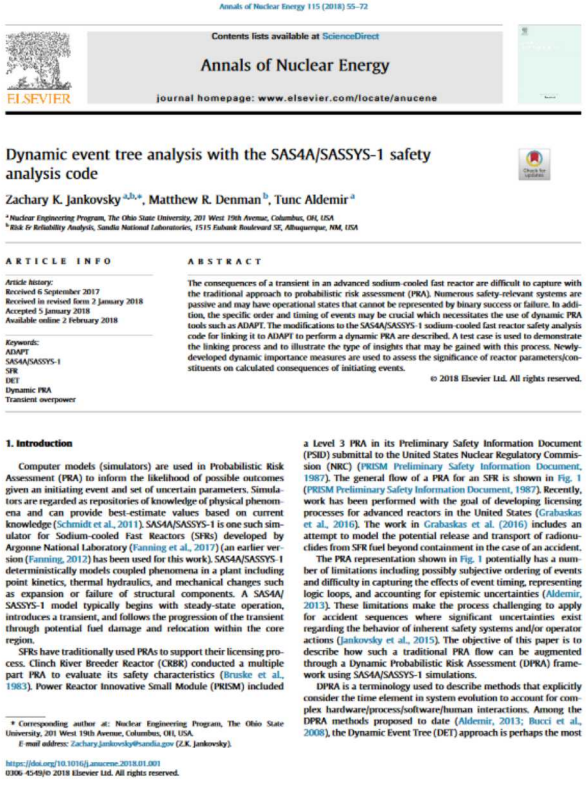
Zachary Jankovsky, Troy Haskin, Matthew Denman

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories



Annals of Nuclear Energy 115 (2018) 55–72

Contents lists available at ScienceDirect

Annals of Nuclear Energy

journal homepage: www.elsevier.com/locate/anucene

Dynamic event tree analysis with the SAS4A/SASSYS-1 safety analysis code

Zachary K. Jankovsky [a,b,*], Matthew R. Denman [b], Tunc Aldemir [a]

[a] Nuclear Engineering Program, The Ohio State University, 201 West 19th Avenue, Columbus, OH, USA
[b] Risk & Reliability Analysis, Sandia National Laboratories, 1515 Eubank Boulevard SE, Albuquerque, NM, USA

ARTICLE INFO

Article history:
Received 6 September 2017
Received in revised form 2 January 2018
Accepted 5 January 2018
Available online 2 February 2018

Keywords:
ADAPT
SAS4A/SASSYS-1
SFR
DET
Dynamic PRA
Transient overpower

ABSTRACT

The consequences of a transient in an advanced sodium-cooled fast reactor are difficult to capture with the traditional approach to probabilistic risk assessment (PRA). Numerous safety-relevant systems are passive and may have operational states that cannot be represented by binary success or failure. In addition, the specific order and timing of events may be crucial which necessitates the use of dynamic PRA tools such as ADAPT. The modifications to the SAS4A/SASSYS-1 sodium-cooled fast reactor safety analysis code for linking it to ADAPT to perform a dynamic PRA are described. A test case is used to demonstrate the linking process and to illustrate the type of insights that may be gained with this process. Newly-developed dynamic importance measures are used to assess the significance of reactor parameters/constituents on calculated consequences of initiating events.

© 2018 Elsevier Ltd. All rights reserved.

1. Introduction

Computer models (simulators) are used in Probabilistic Risk Assessment (PRA) to inform the likelihood of possible outcomes given an initiating event and set of uncertain parameters. Simulators are regarded as repositories of knowledge of physical phenomena and can provide best-estimate values based on current knowledge (Schmidt et al., 2011). SAS4A/SASSYS-1 is one such simulator for Sodium-cooled Fast Reactors (SFRs) developed by Argonne National Laboratory (Fanning et al., 2017) (an earlier version (Fanning, 2012) has been used for this work). SAS4A/SASSYS-1 deterministically models coupled phenomena in a plant including point kinetics, thermal hydraulics, and mechanical changes such as expansion or failure of structural components. A SAS4A/SASSYS-1 model typically begins with steady-state operation, introduces a transient, and follows the progression of the transient through potential fuel damage and relocation within the core region.

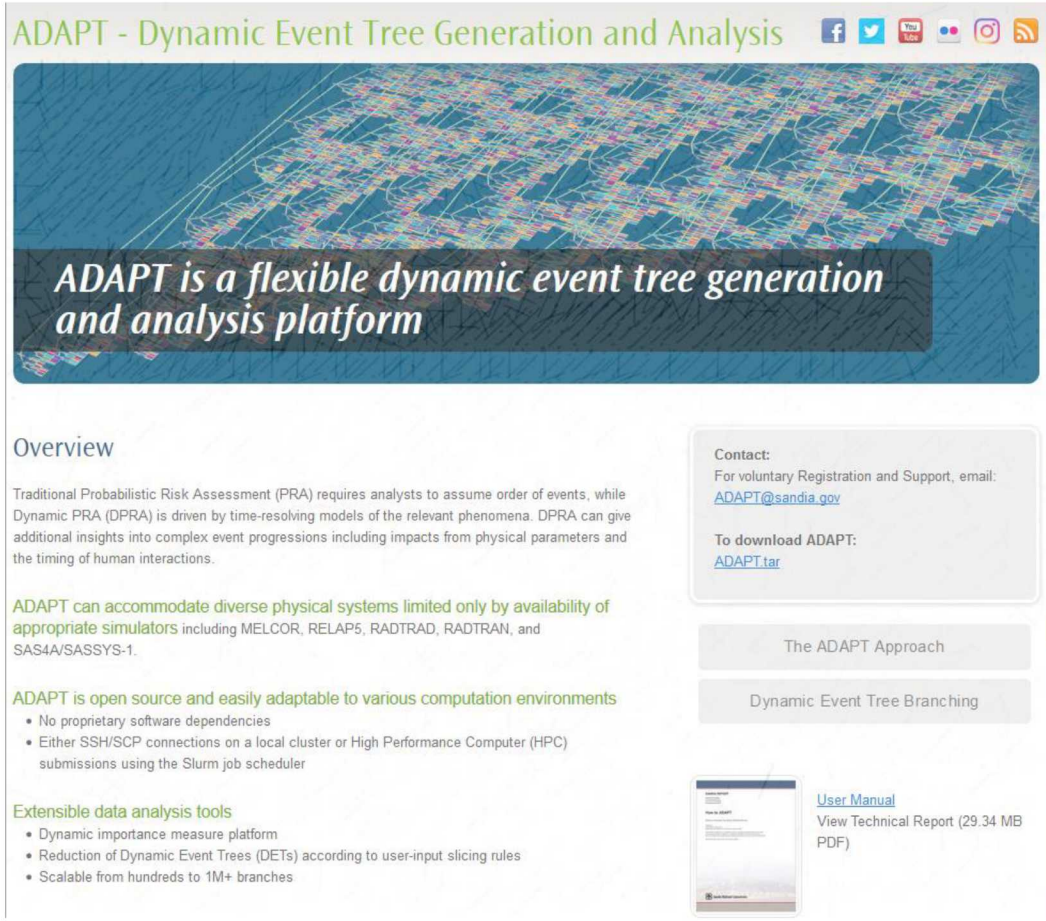
SFRs have traditionally used PRAs to support their licensing process. Clinch River Breeder Reactor (CRBR) conducted a multiple part PRA to evaluate its safety characteristics (Brunske et al., 1983). Power Reactor Innovative Small Module (PRISM) included a Level 3 PRA in its Preliminary Safety Information Document (PSID) submittal to the United States Nuclear Regulatory Commission (NRC) (PRISM Preliminary Safety Information Document, 1987). The general flow of a PRA for an SFR is shown in Fig. 1 (PRISM Preliminary Safety Information Document, 1987). Recently, work has been performed with the goal of developing licensing processes for advanced reactors in the United States (Grabaskas et al., 2016). The work in Grabaskas et al. (2016) includes an attempt to model the potential release and transport of radionuclides from SFR fuel beyond containment in the case of an accident.

The PRA representation shown in Fig. 1 potentially has a number of limitations including possibly subjective ordering of events and difficulty in capturing the effects of event timing, representing logic loops, and accounting for epistemic uncertainties (Aldemir, 2013). These limitations make the process challenging to apply for accident sequences where significant uncertainties exist regarding the behavior of inherent safety systems and/or operator actions (Jankovsky et al., 2015). The objective of this paper is to describe how such a traditional PRA flow can be augmented through a Dynamic Probabilistic Risk Assessment (DPRA) framework using SAS4A/SASSYS-1 simulations.

DPRA is a terminology used to describe methods that explicitly consider the time element in system evolution to account for complex hardware/process/software/human interactions. Among the DPRA methods proposed to date (Aldemir, 2013; Bucci et al., 2008), the Dynamic Event Tree (DET) approach is perhaps the most

* Corresponding author at: Nuclear Engineering Program, The Ohio State University, 201 West 19th Avenue, Columbus, OH, USA.
E-mail address: Zachary.Jankovsky@sandia.gov (Z.K. Jankovsky).

https://doi.org/10.1016/j.anucene.2018.01.001
0306-4549/© 2018 Elsevier Ltd. All rights reserved.



ADAPT - Dynamic Event Tree Generation and Analysis

ADAPT is a flexible dynamic event tree generation and analysis platform

Overview

Traditional Probabilistic Risk Assessment (PRA) requires analysts to assume order of events, while Dynamic PRA (DPRA) is driven by time-resolving models of the relevant phenomena. DPRA can give additional insights into complex event progressions including impacts from physical parameters and the timing of human interactions.

ADAPT can accommodate diverse physical systems limited only by availability of appropriate simulators including MELCOR, RELAP5, RADTRAD, RADTRAN, and SAS4A/SASSYS-1.

ADAPT is open source and easily adaptable to various computation environments
- No proprietary software dependencies
- Either SSH/SCP connections on a local cluster or High Performance Computer (HPC) submissions using the Slurm job scheduler

Extensible data analysis tools
- Dynamic importance measure platform
- Reduction of Dynamic Event Trees (DETs) according to user-input slicing rules
- Scalable from hundreds to 1M+ branches

Contact:
For voluntary Registration and Support, email:
ADAPT@sandia.gov

To download ADAPT:
ADAPT.tar

The ADAPT Approach

Dynamic Event Tree Branching

User Manual
View Technical Report (29.34 MB PDF)