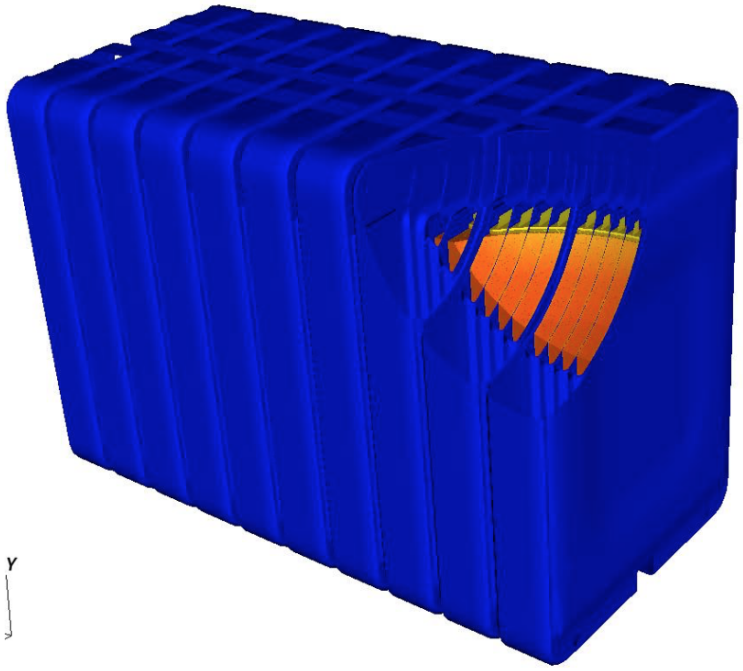
# CAEBAT OAS/VIBE




S. Allu, J.J. Billings, W. Elwasif, S. Kalnaus, A. Kumar, D. Lebrun-Grandie, A. McCaskey, S. Pannala, S. Simunovic, R.W. Smith, B. Turcksin, J. Turner

12/1/2016
Production Release v1.2

Document detailing the updated release to users of CAEBAT OAS and VIBE. Contains software description, installation, run instructions, and examples.

# Contents

# 1 New Features

The new release extends the previous version by implementing:

- Continuous execution mode of OAS which provides savings in computational time of up to 50%
- Capability to simulate dynamic discharge (variable potentiostatic/galvanostatic conditions)
- Improved Integrated Computational Environment (ICE)
- Deployment via Docker and addition to Virtual Machine

# 2 Introduction

As part of the CAEBAT (Computer Aided Engineering for Batteries) activities, ORNL developed a flexible, robust, and computationally scalable open-architecture framework that integrates multi-physics and multi- scale battery models. The physics phenomena of interest include charge and thermal transport, electrochemical reactions, and mechanical stresses. They operate and interact across the porous 3D structure of the electrodes (cathodes and anodes), the solid or liquid electrolyte system and the other battery components. The underlying lower-length processes are accounted for through closure equations and sub- models that are based on resolved quantities. The schematic of this framework is given in Fig. 1.

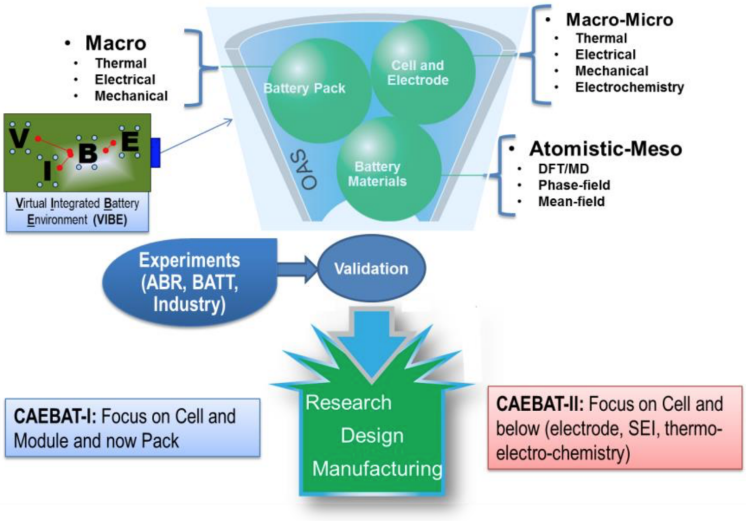



Figure 1: Schematic of the OAS modeling framework and interactions with other tasks within the CAEBAT program and external activities.

This framework enables seamless integration of the following physical phenomena that are necessary for development of realistic and predictive battery performance and safety models:

Mass Transport

- Lithium/electron transport through cathode, anode and electrolyte materials
- Spatiotemporal variations in material properties

Thermal Transport

- Thermal transport through various battery materials as a function of space and time

Electrochemistry

- Primary and secondary reactions
- Interfacial reactions

Mechanical behavior

- Linear and nonlinear mechanics
- Stress/strain relationships
- Fracture at primary and secondary particle levels

The objective of the project is to develop a mathematical and computational infrastructure, and modeling framework that will enable seamless multi-scale and multi-physics simulations of battery performance and safety. The modeling framework will transfer the information between models in a physically consistent and mathematically rigorous fashion for both spatial and temporal variations. The end result will be a verified, computationally scalable, portable, and flexible (extensible and easily-modified) framework that can integrate models from the other CAEBAT tasks and industrial partners. The framework will be used to validate models and modeling approaches against experiments and to support rapid prototyping of advanced battery concepts. Fig. 2 shows different parts of CAEBAT VIBE simulation environment that work together to provide user with flexibility in the problem setup, solution formulation and simulation launch. Each of the parts is discussed in subsequent sections with corresponding examples.

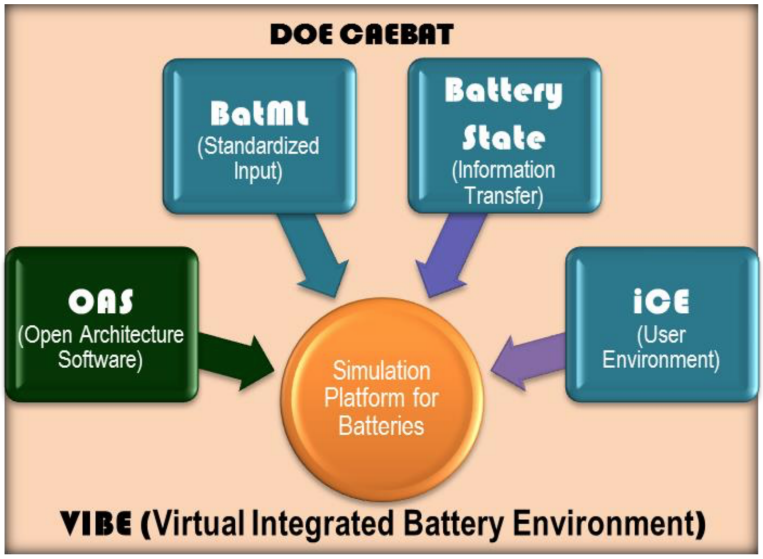



Figure 2: Parts of the CAEBAT VIBE environment

# 3 OAS

The goal is to create a modular and extensible software infrastructure that can support multiple modeling formulations and computer codes for simulation of battery performance and safety. The main guiding principles for the design of the framework are:

Flexibility

- Programing language-agnostic
- Supports multiple modeling approaches and codes
- Combines appropriate component models for problem at hand
- Supports integrated sensitivity analysis and uncertainty quantification

Extensibility

- Ability to add proprietary component models

Computational scalability from desktop to HPC platforms

- Portable and adaptable to various computer hardware architectures

The OAS infrastructure employs a modular design with strict interfaces, object-oriented data structures, and a lightweight backplane implemented in Python scripting language. This design is illustrated in Fig. 3. The framework services control the various software components through component adapters. The components update

the battery state through state adapters. The battery state is the minimal digital description of the battery in space and time such that each simulation component can apply their respective physics models and advance in time from each state point to the next. The OAS framework, along with physics and support components and the adapters constitute the Virtual Integrated Battery Environment (VIBE).

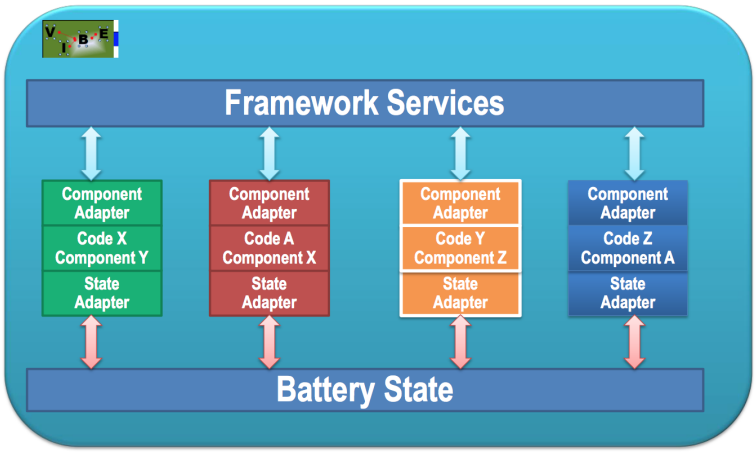



Figure 3: Schematic of the OAS modeling framework, which connects physics components through component adapters, with linkage to the battery state through state adaptors. A specific collection of components, adaptors, and the OAS framework defines one realization of VIBE (Virtual Integrated Battery Environment)

# 4 Battery Markup Language (BatML)

The objective of the BatML specification is to provide standardized format for definition of all the necessary information for battery performance and safety modeling. The overall design for the BatML is given in Fig. 4 below. The BatML Schema establishes the main structure for the BatML data files and enables data validation and consistency checking. BatML files can contain databases and models with default values or with company proprietary information. For e.g., Dow-Kokam or Johnson Controls can provide a database of their cell- sandwich properties that an OEM can directly use in their models. Several examples based on open literature for standard battery materials and components have been developed and made available to the project partners. The graphic workflow environment described later in this documentation (ICE) uses these Schemas and Databases along with any additional user input to create a BatML input file. This XML file can either be used directly by simulation packages or through translators that transform this input into native formats read by the different software components.

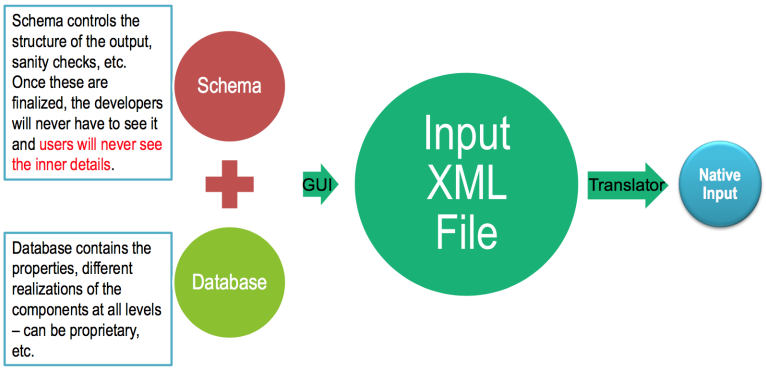



Figure 4: Overall structure of the battery markup language.

The top-level structure of the Battery ML Schema (available at the CAEBAT project website http://batterysim.org) is shown in Fig. 5. Here we define a battery component type that contains the base components such as anode, cathode, electrolyte, separator, current collectors. These base components are used to build higher-level components such as cell-sandwich, cell, module, pack, parts (e.g., busbar, cooling fins). Each of these components can contain additional sub-components, as their definition is dependent on the form of model used. For e.g., the cell-sandwich definition will depend on the model for electrochemical component. The Cell can be further specified as Cylindrical cell, Prismatic cell, etc. To enable this flexibility, we picked the relational data model (hierarchical data model will also be implemented in the language). The main considerations for selecting the relational versus hierarchical data model were:

- Batteries have very deep hierarchy and the hierarchical data model will lead to considerable duplication of the data
- Relational data model provides the flexibility to quickly modify the hierarchies of the models and add new components
- Relational approach requires that all the references to the data exist and that the model is self- contained. This does not impose a strong limitation because the input files will be primarily manipulated using GUI and not by user editing of the XML files.
- Cross referencing in relational data model drastically reduce data duplication and the corresponding risk for errors

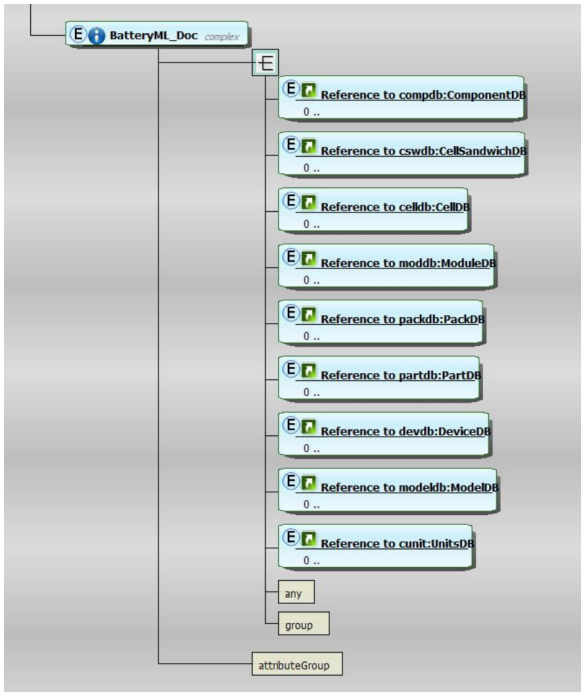



Figure 5: Battery markup language schema.

Once the DualFoil model has been selected, it can be further subdivided into different components of the cell sandwich. The Battery ML schema imports the cell-sandwich schema (whose structure is shown in Fig. 6 and can be downloaded from the project's web site) that further specifies the details of its components. Similar hierarchical expansion can be used to define cell, module, pack, etc. Current implementation contains the above battery hierarchy. Translators between BatML and other input formats of CAEBAT partners have been developed with the final goal of BatML becoming a standard. Fig. 7 compares the EC Power input and conversion to BatML. Similar translators have been developed for Text Battery Model (.tbm) files (BDS/CD Adapco), .svm files (NREL MatLab model), as well as ANSYS input. We keep the project's website up to date with the latest version of the schema, corresponding documentation and examples, schema validation tools, etc.

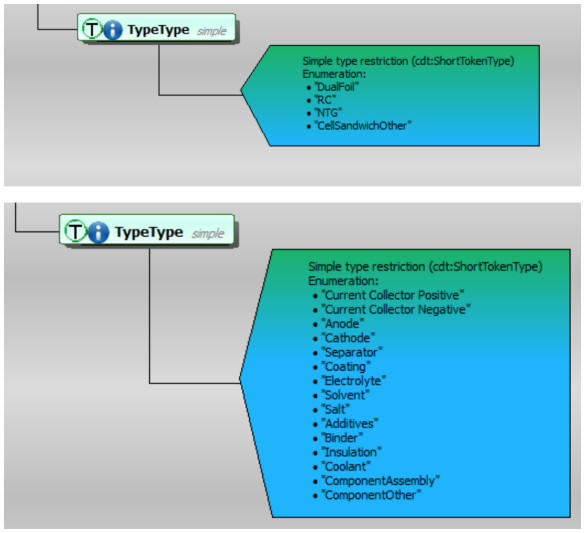



Figure 6: Cell sandwich and component markup language schema types.

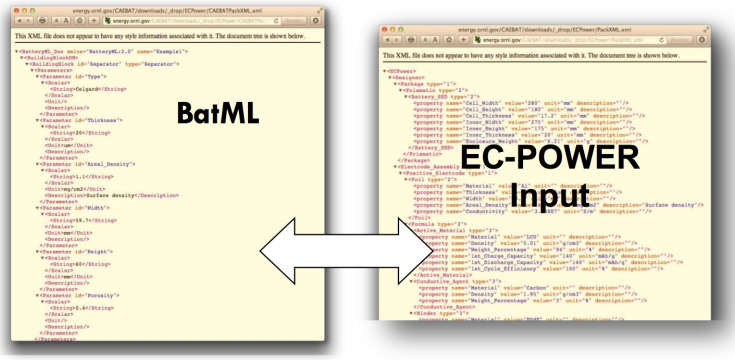



Figure 7: Translation between BatML and EC power input.

# 5 Battery State

The OAS framework integrates battery models using component and state adapters. The component adapters interact with the components by preparing the necessary inputs to run the components and by scheduling the component runs. The state adapters interact with the battery state file(s) by updating all the necessary information about the battery state and the methods for coupling the components. Fig. 8 shows a battery state file that transfers the information between the electrochemistry, thermal and electrical physics components. The device hierarchy is modeled by coarse-graining of the underlying sub-components. The top hierarchical level of the model is divided into zones. These zones then transfer information between the

components in case of loosely coupled multi-physics simulations. Further description and examples of the battery state are given in the sections that follow.

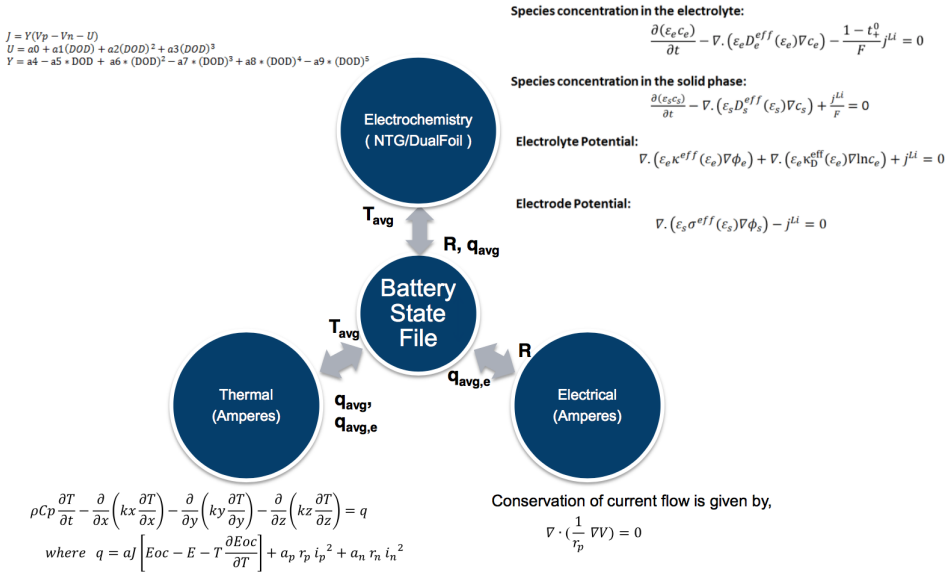



Figure 8: The battery state file is the core device for passing data between components.

# 6 Virtual Integrated Battery Environment (VIBE)

We have added several components for modeling electrochemistry and mass, electron, and heat transport in order to VIBE. The components that have been integrated so far are:

Electrochemistry

- Pseudo 2D Dual-Foil (Doyle, Fuller et al. 1993; Fuller, Doyle et al. 1994; Fuller, Doyle et al. 1994)
- 3D electrochemistry model known as AMPERES
- NTG (Seong Kim, Yi et al. 2011)
- NREL MSMD (Kim, Smith et al. 2011)
- AMPERES Single Particle
- AMPERES Pseudo-2D

Thermal and Electrical

- AMPERES

Cost Model

- ANL cost model

Mechanics

- EPIC, LS-Dyna, LIGGGHTS/LAMMPS

# 7 Integrated Computational Environment (ICE)

ICE is a graphical workflow editor for input, simulation setup, job launch and analysis. The driving force behind ICE is the development of software that provides an integrated set of capabilities for working with physics simulators for creating input files, managing and analyzing data, launching jobs and code coupling through data mapping. The framework was originally developed for the Nuclear Energy Advanced Modeling and Simulation program (NEAMS), - a DOE NE project. The capabilities of ICE for CAEBAT were designed specifically to aid in simulation setup and job launch. Within the graphical interface the selection of components (physics models), platforms, time-stepping schemes, meshes, etc becomes much easier task. Further updated information can be found on the ICE project webpage. Visualization of the simulation results is executed through a connection to the external visualization software application, VisIt. Detailed description of ICE usage to set up and run a simulation is provided in Appendix B.

# 8 Example Applications

The examples provided in the following sections are currently availabe as part of the VIBE package.

## 8.1 Example 1: Cylindrical Cell (Electrochemical-Electrical-Thermal)

This example (located in the VIBE repository at examples/case3/) represents the geometry of a rolled cylindrical cell. The main model properties are given in the table below. Fig. 9 shows the geometry and the finite element mesh used to resolve the geometry of the cylindrical cell and the current collectors. The top hierarchy model has 168 (56 each for the cell-sandwich and positive and negative current collectors) zones in 4 quadrants. The zones describe different current collector and cell sandwich regions. The simulation uses 56 concurrent Dualfoil simulations for different cell-sandwich zones. Typical results are shown in Fig. 10. The maximum temperature occurs at the cell core as expected.

* table here *

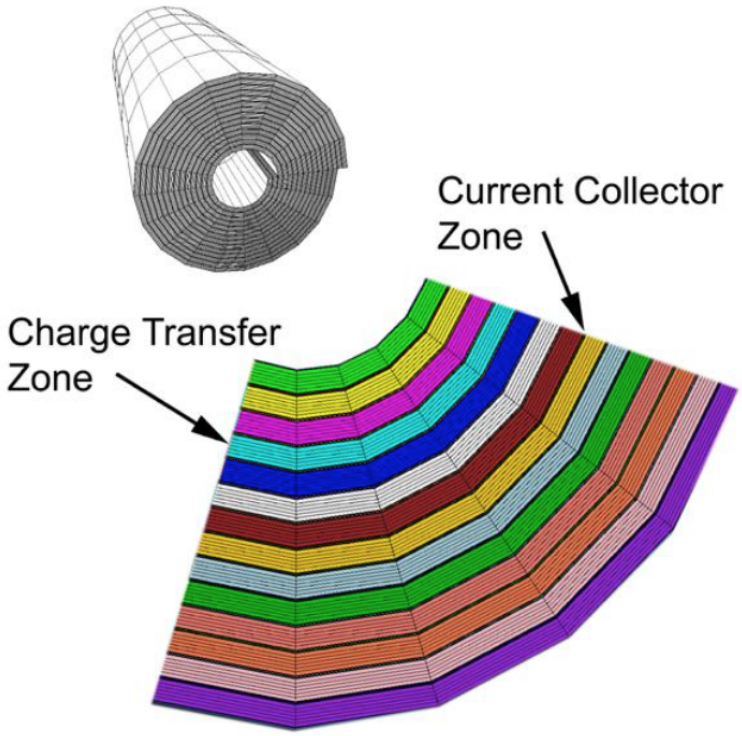



Figure 9: Geometry and mesh of the simulated cylindrical cell.

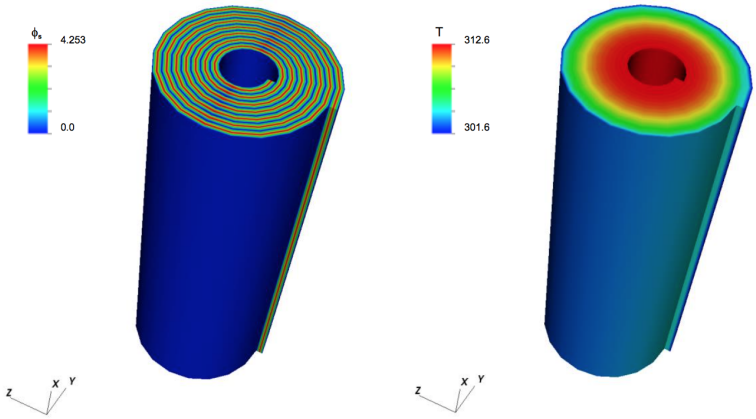



Figure 10: Sample results for example 3 (electrical potential on left and temperature on right).

## 8.2 Example 2: Pouch cell (Electrochemical-Electrical-Thermal)

This example (located in the VIBE repository at examples/case6/) represents the geometry of a prismatic pouch cell. The electrochemistry is modeled using the NTG model instead of the DualFoil model. The cell under consideration is a 70mm x 110mm x 10mm 4.3 Ah pouch cell manufactured by Farasis Energy, Inc with the properties given in the table below. The pouch cell in the current study contained 17 cathode and 17 anode layers and the finite element mesh was divided into 71 corresponding zones for cell sandwich, current collectors, and pouch (Fig. 11).

`* table here *`

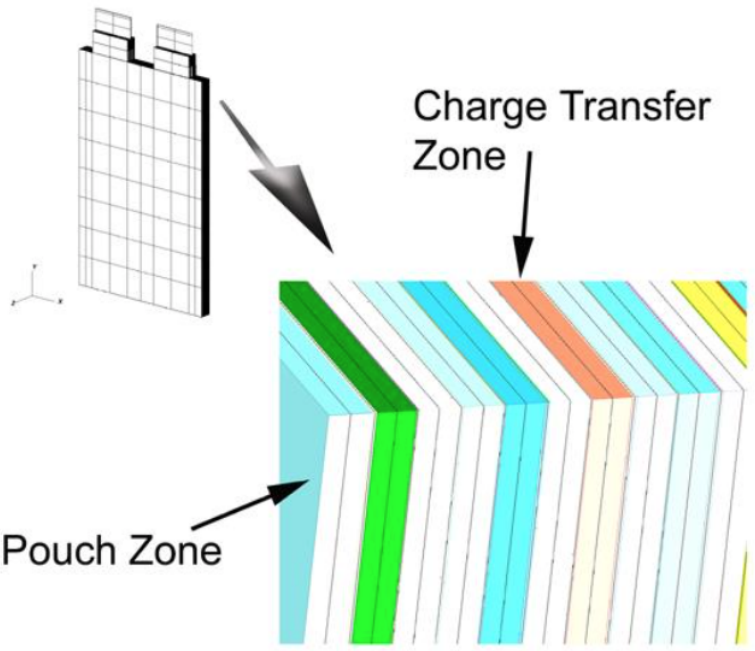



Figure 11: Geometry and mesh of pouch cell.

The example of the simulation results is shown in Fig. 12 and represents a temperature distribution in a pouch cell following a discharge at 5C rate of applied current. At such high applied current significant increase in temperature can be observed in the cell core. The simulation results have been validated with the experiments involving IR temperature measurement on the surface of the pouch cell (Fig. 12b). Experiments agree well with the predicted temperature profiles for all C-rates.

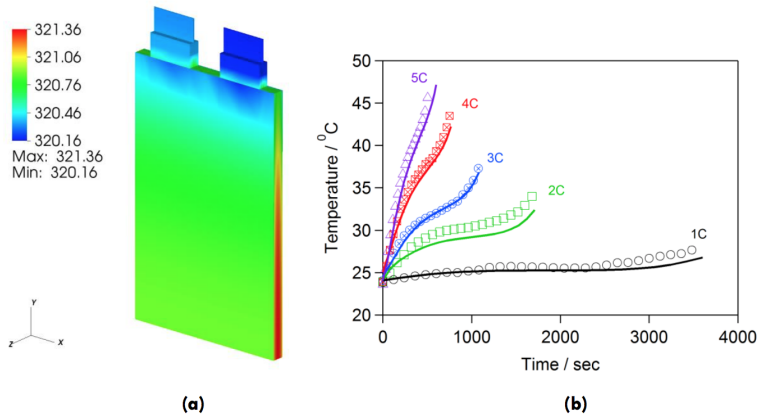



Figure 12: Simulation results (a) and validation with IR measurements (b). Source: S. Allu et al, J Power Sources 246, 2014, pp. 876-886.

## 8.3 Example 3: 4P and 4S battery module

In this example, the single pouch cell described in the previous section is used as a building block for a module, containing 4 cells in parallel or in series. The example is in the repository in examples/case7/. Meshes representing parallel (4P) and series (4S) module configurations are shown in Fig. 13. No cooling fins were placed between the cells in this model. The mesh consists of approximately 150,000 FE nodes and 308 zones in the whole module thus resolving each current collector. Concurrent electrochemical model runs (DualFoil was chosen in this case) were performed in 136 charge transfer zones within the module. The goal of this study was to estimate temperature variations across the cells connected in series and in parallel.

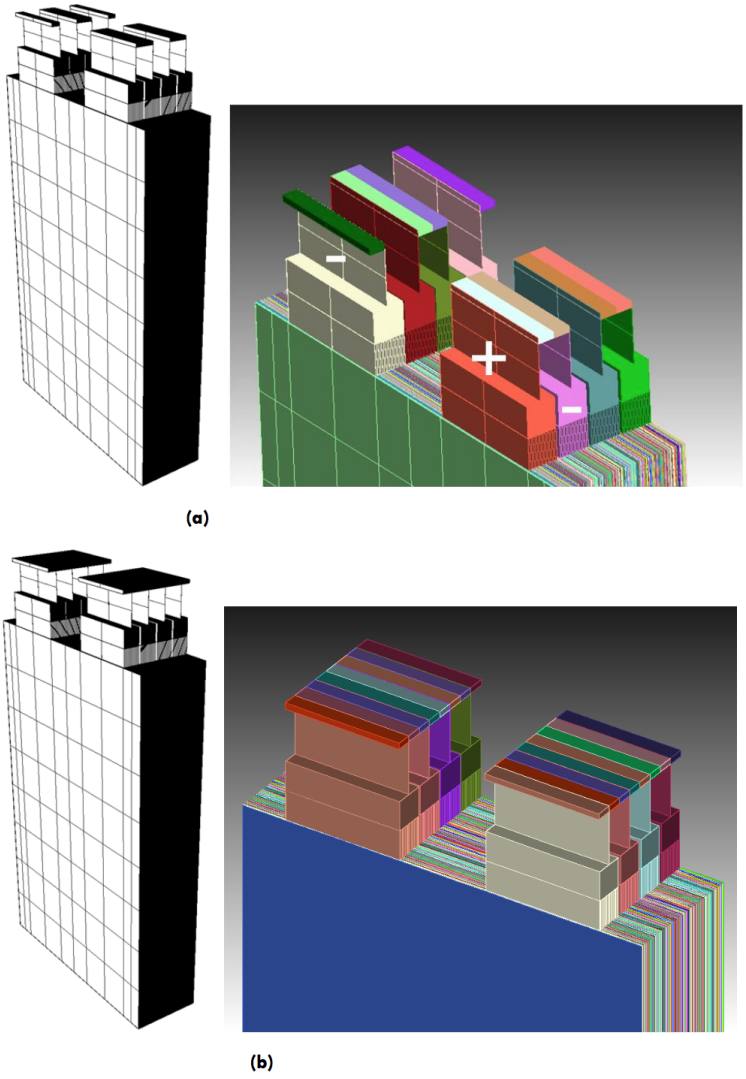



Figure 13: Module schematics with (a) series (4S) and (b) parallel (4P) cell arrangements.

The results of the simulations when symmetric cooling to the module surfaces is applied with a convective heat transfer coefficient of 35 W m-2 K-1 are shown in Fig. 14. As can be seen both parallel and series cases result in very similar distribution of temperature across the module. In both cases, a 5C discharge rate was applied.

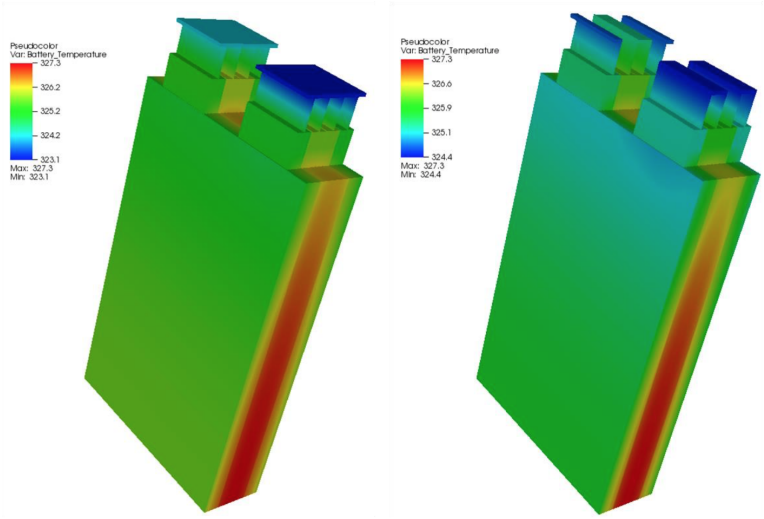



Figure 14: End of discharge temperature profiles in (a) 4P and (b) 4S modules.

## 8.4 Example 4: 4P module under dynamic discharge

In this example, the mesh corresponding to the 4P module is used to simulate the dynamic discharge under the user supplied variable potentiostatic/galvanostatic conditions. The example can be found in examples/case10/. The model uses DualFoil and AMPERES Thermal components. An option of driving simulation under varying current (dynamic discharge) has been added in the current release. The key-value pair file (for detailed description of sections of the input and configuration files please see Appendix A) contains several lines dedicated specifically for this new option. The following keywords are used to describe the cycling profile:

NUMSEG - Number of segments in the cycling profile.

CURRDEN - List of current density values corresponding to each segment.

MODESEG - List of segment modes. Most commonly used are 0 for potentiostatic and 1 for galvanostatic.

CUTOFFL - Lower cut-off potential.

CUTOFFH - Upper cut-off potential.

NOTE: If the NUMSEG keyword is missing in the key-value pair input file, the simulation will assume a default constant current discharge.

To utilize the dynamic discharge capability, the time stepping must be specified using the EXPLICIT option in the simulation config file with explicitly specified values of

time corresponding to the segments of the cycling profile. Please see Appendix A for detailed description of input and config files required to launch a simulation.

If zero current density is specified in CURRDEN, the simulation will be performed under potentiostatic condition using the OCP corresponding to the end of previous cycling segment.

# 9 Getting Started

The following sections provide instructions on running VIBE in a virtual machine or as a Docker container.

## 9.1 Running VIBE in a virtual machine

In order to enable a user to take VIBE for a 'test drive' we have the software packaged within a Virtual Machine (VM) which can be installed on user's machine of choice. This section describes the instructions on how to run the simulation in VM. The virtual machine is packaged into the open virtualization format archive `BatterySim-release<Version>.ova` which can be downloaded from the project website. Before using the VM the Virtual Box software needs to be installed on the user machine. The software can be downloaded from virtualbox.org together with the installation instructions and user manual. Once installed, start the Virtual Box and click on File > Import Appliance. This will open the dialogue box where you can select the BatterySim-release.ova as your virtual machine. After such selection, BatterySim will appear in the list of the virtual machines within the left panel of Virtual Box (Fig. 15).

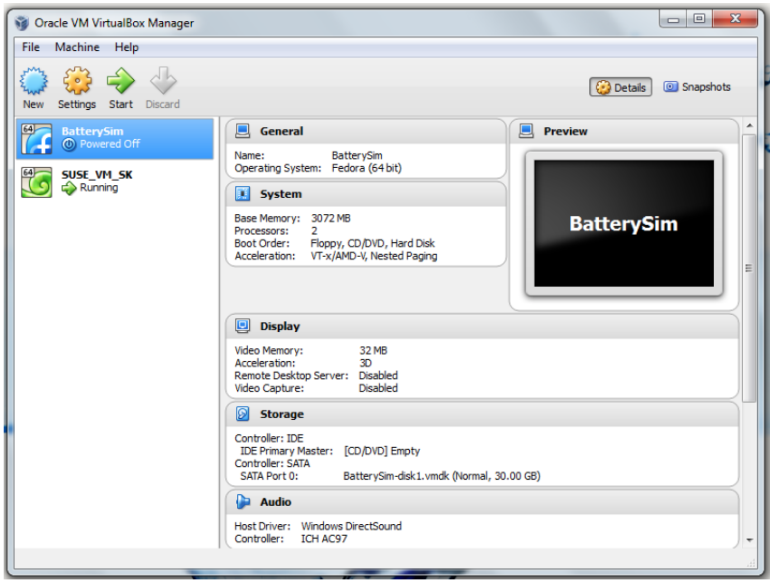



Figure 15: Virtual box with VM imported.

If you have several virtual machines select the BatterySim and start the VM. There is no password for the BatterySim, so simply press Return key. This will open the Fedora Linux environment as shown in Fig. 16.

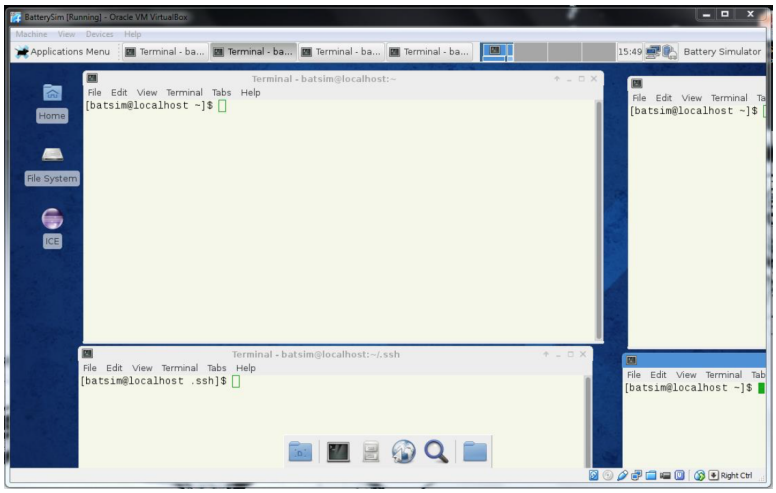



Figure 16: BatterySim virtual machine.

From here the user can navigate to the examples directory and run the simulations in command line as described in APPENDIX A. In Fig. 17a the terminal window navigating to case 2 is shown with corresponding simulation configuration file (thermal_.*conf) and the input directory which stores input files as well as mesh (exodus* .e

file). After the simulation is complete, the case directory will be populated with log files containing information on simulation run and possible errors as well as a new work directory which contains the results of simulation and the battery state CGNS file. More on the directory structure and command line launch instructions is given in Appendix A. Alternatively user may launch ICE by double-clicking on the ICE desktop icon (Fig. 17b). Instructions on running simulations with ICE can be found in Appendix B.

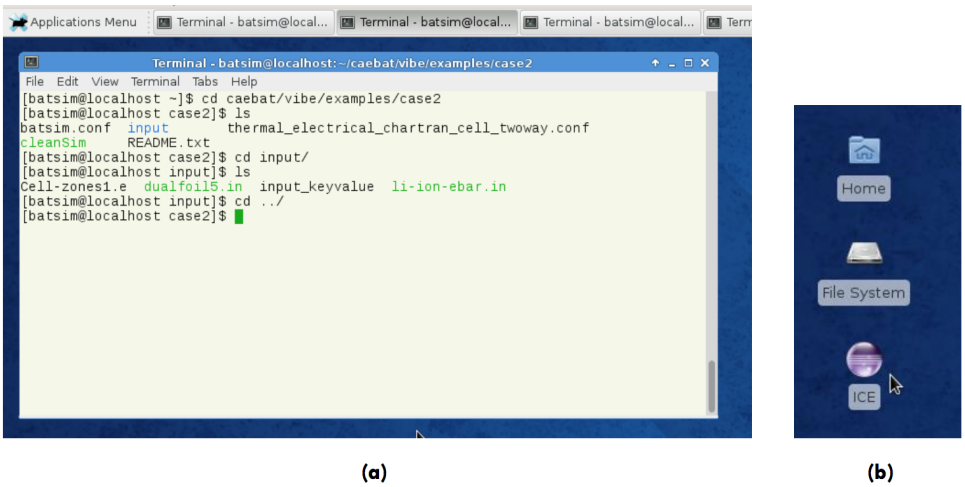



Figure 17: Using command line (a) or ICE (b) to setup and run a simulation in VIBE.

The home directory of the battery simulator is /home/batsim/ and it contains installation of OAS (/home/batsim/oas/) and VIBE with the simulation cases (/home/batsim/vibe/). Paths for components' drivers and executables are included in configuration file batsim.conf which is placed in a separate directory /PathTo/examples/config/ and is used by each simulation case. Simulation can thus be launched from the case directory by specifying the OAS directory and simulation configuration and pressing the Return key:

```
$ /home/batsim/caebat/oas/install/bin/ips.py \
--simulation=thermal_electrical_chartran_cell_twoway.conf \
--log=temp.log --platform=../config/batsim.conf -a
```

The BatSim virtual machine comes with four simulation cases packaged in examples directory. These involve different battery and module geometries and physical models as discussed in APPLICATION EXAMPLES section. For instance running the simulation of case2 as described above will provide a loosely coupled electrochemical-thermal-electrical solution for unrolled cell with DUALFOIL as electrochemical component. Details of the simulation cases are given in Appendix A and Appendix B.

## 9.2 Running VIBE in Docker

Using Docker container to run the software is an alternative way to using the virtual machine. The details and installation instructions depending on the OS can be found at docker.com. The following instructions explain how to:

- launch the VIBE container
- connect to the container from ICE
- stop and remove the container

### Launch the VIBE container

Pull the latest version of the VIBE container:

```
$ docker pull rombur/vibe-ssh
```

Download the private key id_rsa_vibe:

```
$ wget https://raw.githubusercontent.com/Rombur/VIBE/master/remote/id_rsa_vibe
```

Run the container:

```
$ docker run -d -p 2222:22 --name vibe_ssh rombur/vibe-ssh
```

This will run in detached mode a container named vibe_ssh using the image rombur/vibe-ssh. It will also map the port 2222 of your machine to the port 22 of the container. This port has been exposed in the container.

### Connect to the container from ICE

Use ICE like you would do in the VM with two differences.

Because the examples are in the container not in your local machine, you cannot browse them. If you want to use the input files from the examples, you will need to copy them from the container. You can connect to the container using the following command:

```
$ ssh -i id_rsa_vibe -p 2222 root@localhost
```

With this command, ssh will use the port 2222 of your machine and the private key id_rsa_vibe to connect to the container. The examples can be found in /opt/vibe/examples. If you want to copy case1 from the VIBE container into you working directory, you can do:

```
$ docker cp vibe_ssh:/opt/vibe/examples/case1 .
```

Inside the VibeLauncher window, in the Hosts pane, you need to change the hostname from localhost to docker and the Execution Path from /home/batsim/caebat to /opt. To do so, simply click on localhost and /home/batsim/caebat. Once you launch the job, a window should pop up to specify the properties of the new connection.

In the Host field, write localhost. In the User field, write root. Click on Network Connections, SSH2, then click on Add Private Key... and add the id_rsa_vibe file. Back on the New Connection window, under Advanced type 2222 in the Port field.

The container will still run even after you exit ICE. You will need to stop and remove it yourself using:

```
$ docker stop vibe_ssh
$ docker rm vibe_ssh
```

To check that the container has been stopped, type:

```
$ docker ps
```

There should not be any container named vibe_ssh. To check that the container has been removed, type:

```
$ docker ps -a
```

There should not be any container named vibe_ssh.

# 10 Appendix A: Command line OAS/VIBE launch instructions

This section describes the simulation launch procedure using command line as opposed to using integrated computation environment (ICE) which is detailed in the next section. All simulation scenarios (cases) are a part of VIBE and correspondingly are placed in PathTo/vibe/examples/ directory. When running in BatterySim virtual machine this directory will be located in /home/batsim/caebat/vibe/examples/. As an example the coupled electrochemical-electrical-thermal modeling of a prismatic cell (case 6) is considered here. The structure of the directory case6 is shown below.

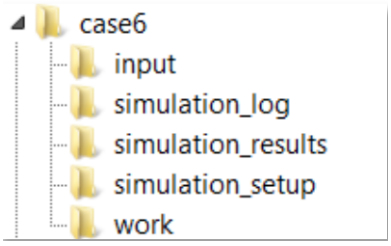



Figure 18: Structure of directory for Case 6.

The input directory contains mesh file (Exodus file) of the geometry as well as key-value pair input file to set up material constants in simulation models. This file needs

to be edited if different boundary conditions and/or model parameters are desired. In the present configuration of VIBE the input_keyvalue file has the following fields:

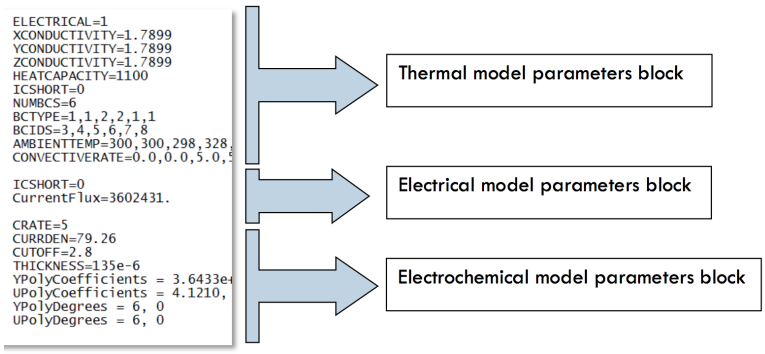



Figure 19: Example of input key-value file.

The names of the keys in the input file are to a great extent self-explanatory. ICSHORT parameter determines whether internal short circuit is modeled in either thermal or electrical components. NUMBCS sets up the number of boundary conditions with BCIDS representing IDs of the side sets in the corresponding mesh file where the boundary conditions are to be applied. Two types of boundary conditions can be set with BCTYPE=1 representing Robin boundary conditions and BCTYPE=2 representing the Dirichlet type of BC. In the above example, the block of electrochemical model parameters represents the NTG model setup. A slightly different setup is required when the Pseudo 2D model represented by DualFoil subroutine is used. Parameter CUTOFF sets the voltage at which discharge of the cell terminates. CURRDEN is the applied current density in A/m2 in the cell (current normalized by the total area of the cell). THICKNESS determines the thickness of the cell sandwich (in meters). The YPolyDegrees and UPolyDegrees vectors set the dimensions of the polynomial fits for Y and U functions in NTG model (please see formulation of models in the Battery State section of this release document). The first element of the vector represents the order of the polynomial that describes impedance (Y) or OCP (U) as a function of depth of discharge. If the discharge curves at of the cell different temperatures are available, the corresponding fits can be made bi-polynomial in which case the degree of polynomial in temperature is described by the second element of the vector. In case when the thermal behavior data are not available, this member is simply set to zero, as in the above example. YPolyCoefficients and YPolyCoefficients are the vectors containing the coefficients of the corresponding polynomial fits. Once the input file has been modified accordingly and saved the simulation parameters can be configured. This is done within the simulation configuration file SimulationName.conf. The simulation config file captures the components used in the simulation, total number of variables passed through the Battery State, input/output from the components and corresponding component drivers. If a different model is

desired this can be changed here. While the example case6 uses NTG model to describe electrochemistry, this can be changed to Pseudo 2D component (DualFoil) if needed and if all material constants required for DualFoil code to run are available. Within the configuration file [PORTS] change the [[CHARTRAN]] implementation to IMPLEMENTATION = DUALFOIL. The corresponding component specification can be added to the configuration file, for instance:

```
[DUALFOIL]
    CLASS = CHARTRAN
    SUB_CLASS =
    NAME = DualFoil
    NPROC = 1
    BIN_PATH = $CAEBAT_ROOT/bin
    INPUT_DIR = $SIM_ROOT/input
    INPUT_FILES = 'dualfoil5.in' , 'li-ion-ebar.in'
    OUTPUT_FILES = 'df_caebat.out'
    INPUT_VAR = 'lumped_temperature'
    OUTPUT_VAR = 'lumped_source' , 'lumped_resistance'
    SCRIPT = $BIN_PATH/dualfoil_chartran.py
```

In this case, the DualFoil Fortran code requires two input files that need to be placed in the input directory of the simulation case. More on the DualFoil code description and requirements can be found at cchem.berkeley.edu. The last set of parameters in the simulation config file defines the time marching.

```
[TIME_LOOP]
   MODE = REGULAR
   START = 0.0
   FINISH = 30.
   NSTEP = 2
   VALUES = 3.4 3.5 3.6 3.7
```

Two ways of setting up the time step can be implemented. REGULAR mode defines beginning, end and number of time steps to take during the simulation. EXPLICIT mode allows specification of a vector containing specific values of time at which simulation should be performed. In the latter case the software uses the variable named VALUES. Time is specified in minutes. EXPLICIT method allows using non-uniform size of time steps and is useful where cell potential changes abruptly compared to otherwise smooth profiles where a large time step is sufficient to progress the simulation. This mode must be used with dynamic discharge option.

Once the input files have been edited and simulation configuration has been set, the simulation (case6 in virtual machine BatterySim in this example) can be launched from the command line by running the following line from the simulation case directory:

```
$ /home/batsim/caebat/oas/install/bin/ips.py --simulation=thermal_electrical_chartran
--log=temp.log --platform=../config/batsim.conf -a
```

When simulation completes the results can be found in work directory where they are arranged by the simulation component.

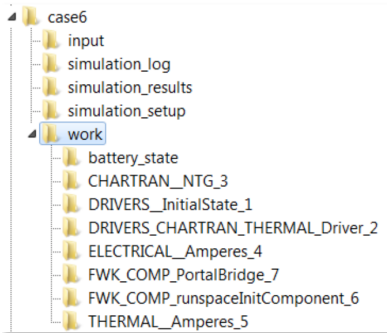



Figure 20: Example of a working directory.

The CFD General Notation System (CGNS) is used to store the variables in zones. The corresponding state file named cphit.cgns is stored in battery_state directory. The results of the simulation can be retrieved from THERMAL_Amperes_5 and ELECTRICAL_Amperes_4 directories with the integer in the directory name showing the number of the component in sequence. These directories contain the *.silo files that can be viewed and processed further with VisIt. Since the Virtual Machine comes with VisIt installed, the user can launch it from any directory by simply typing 'visit' in the command line and pressing Return key. This starts the visualization software where the simulation results can be loaded as a silo database (Fig. 21) or a single file.

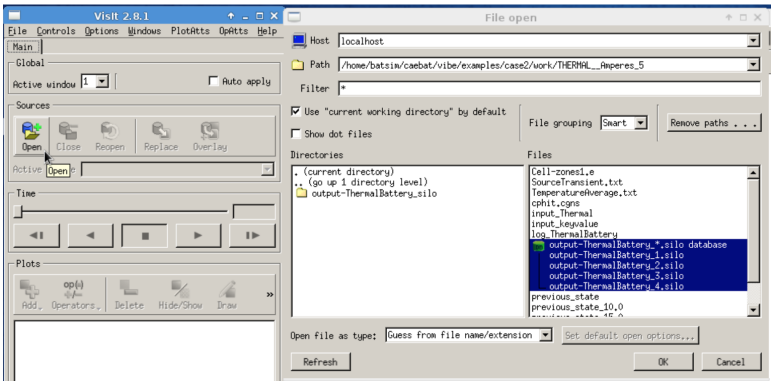
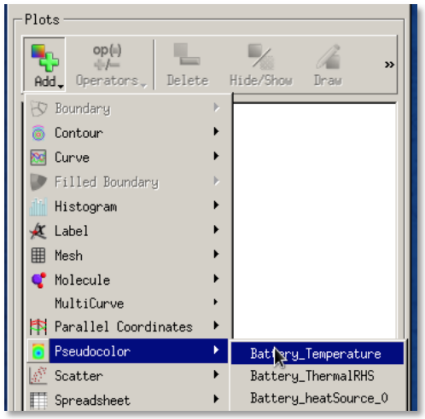



Figure 21: Selecting output files to visualize using VisIt in VM.

To view the temperature distribution in VisIt, add a Pseudocolor plot and select

Battery_Temperature from the list of variables (Fig. 22a). Clicking on the Draw button will create a plot in the output window (Fig. 22b).




(a)

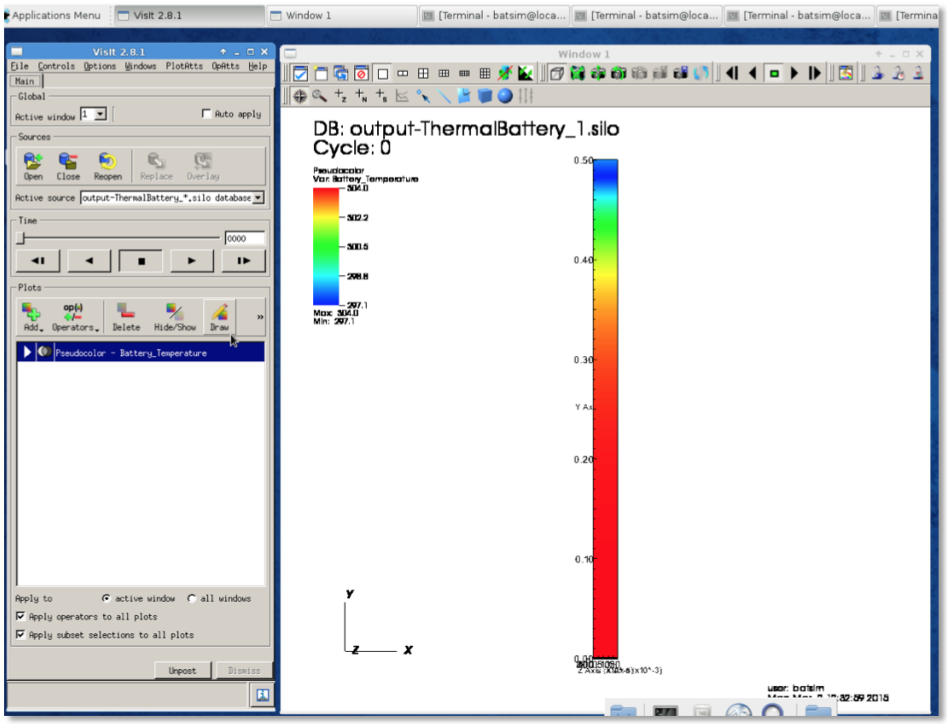



Figure 22:

Rotation, zoom, slicing, transforms, lighting, etc. can be performed in VisIt. More on this software can be found at wci.llnl.gov.

The plot in Fig. 22 shows the distribution of temperature in unrolled strip (once cell sandwich) when 2C discharge current is applied and one end of the cell is held at room

temperature. The solution is obtained with Pseudo2D model (DUALFOIL) used as an electrochemical component. As already mentioned, the model can be changed to NTG by replacing DUALFOIL with NTG in the PORTS section of the configuration tile. The BatterySim Virtual Machine comes with five different cell and module simulation setups, contained in /home/batsim/caebat/vibe/examples directory as:

- Case2: unrolled cell. Useful for testing new cell parameters (for example different materials or porosities) to get an idea about modeling on a cell-sandwich level.
- Case3: cylindrical Li-ion cell.
- Case6: pouch cell.
- Case7: 4P and 4S modules of four pouch cells from case 6 connected in series (4S) or in parallel (4P)
- Case10: 4P module of four pouch cells with dynamic discharge

Any of the above simulations can be launched either from command line as described in this section or using ICE as described in the Appendix B.

# 11 Appendix B: Launch instructions with ICE

This section provides the tutorial on setting up and running a battery simulation using ICE. It is encouraged that the user becomes familiar with the VIBE directory structure by studying Appendix A first and launching a simulation from command line. Overall the workflow in ICE consists of:

- Creating the model by working with simulation configuration file (Caebat Model)
- Creating the simulation input (Caebat Key-Value Pair Generator)
- Setting up the job launch and running a simulation (Caebat Launcher)
- Viewing the results

We created several predefined simulation cases dealing with different cell geometries: unrolled cell sandwich, rolled cylindrical cell, pouch cell and module of four pouch cells. All these cases come with the Virtual Machine VIBE release. The case of 4.3 Ah pouch cell (case6) is a default simulation setup in ICE and the tutorial below discusses the default case first.

## 11.1 Creating the model

To begin, launch ICE (if it isn't already running), and you should be presented with an empty workbench. Navigate to the ICE Perspective by choosing Window > Perspective > Other and scrolling to ICE in the pop-up view. In this Perspective, ICE provides three options for creating new items. The user may click on the green

plus icon (+) located near the top-right corner of the Item Viewer, click on the New Item button in the main ICE toolbar, or choose File > Create an Item. This will launch a dialog prompting you to select a task (or Item) to create (see Fig. 23). Find Caebat Model Builder in the Item Selector list and click Finish.

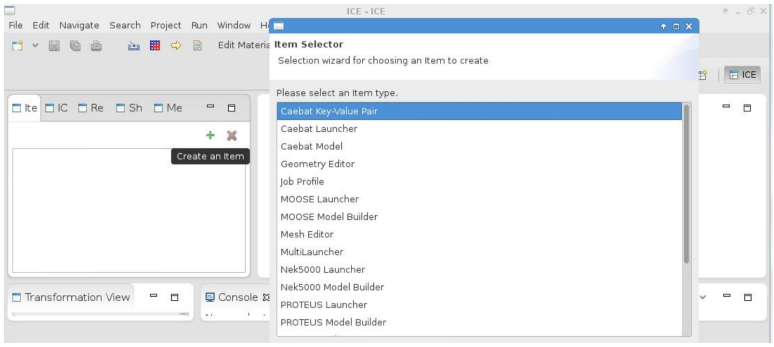
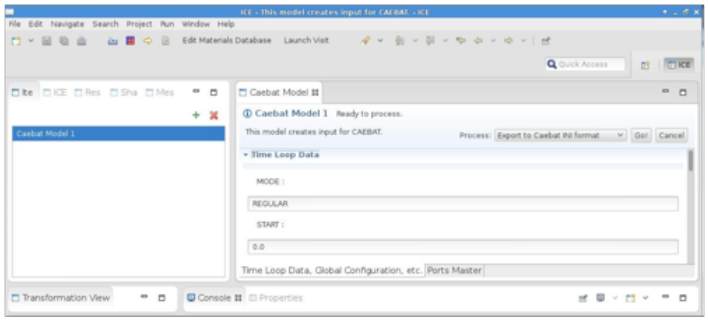



Figure 23: CAEBAT item selector in ICE.

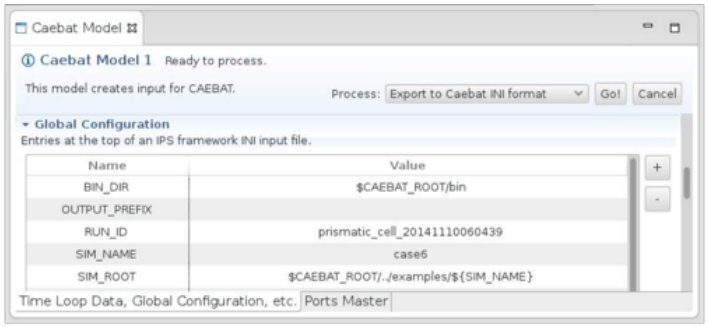
A CAEBAT Model Builder will appear in the main workspace with the default values corresponding to the pouch cell model. You can now edit the parameters if for instance a different number of time steps or different total time is desired. The CAEBAT Model window has two tabs (Fig. 21):

- Time Loop Data, Global Configuration, etc
- Ports Master

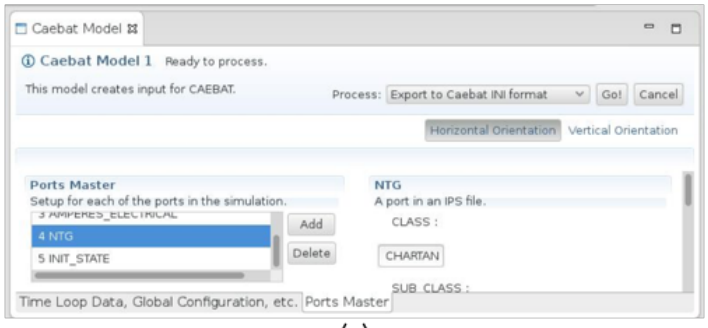
Time Loop Data window (Fig 21a, b) allows you to select the battery geometry, time stepping scheme, components taking part in the simulation and global configuration. The Ports Master window (Fig. 21c) shows the corresponding input directories, input/output variables that are passed through the battery state, and path to each component involved in the simulation. Input directories containing meshes are also specified here. For now leave all the parameters with their default values and click Go!.



(a)




(b)




(c)

Figure 24: Battery model setup in ICE.

## 11.2 Generating simulation input key-value pair file

The key-value pair input file contains numerical parameters necessary for simulation, such as material constants, boundary conditions and coefficients of polynomials when NTG model is used to represent electrochemical component. To pull up a default Key-Value file for edit, in Item Viewer click on the green plus icon again and select Caebat Key-Value Pair from the drop down menu (Fig. 23). The file with default values corresponding to the pouch cell simulation is displayed for edit (Fig. 25). The keys are explained in Appendix A. The default simulation represents a discharge of 4.3 Ah cell with gradient of temperature applied as boundary conditions (BCs) to the

cell surfaces. These settings can be edited here if different BC or different polynomials for NTG model are desired. For now accept the default values (case6) by clicking Go!.

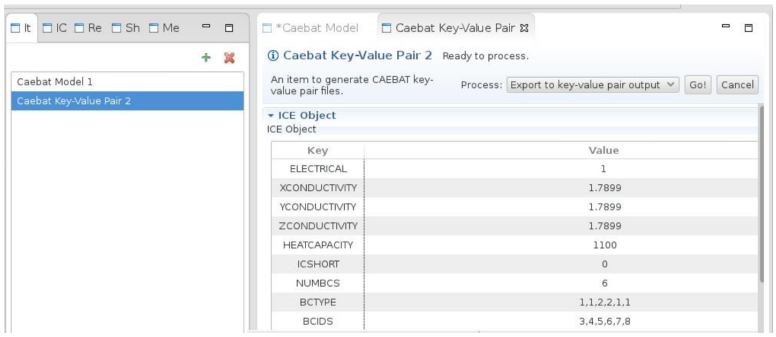



Figure 25: Key-value pair generator in ICE.

This completes the CAEBAT input generation task. The file generated will be used in the next step by the CAEBAT Launcher to run the CAEBAT problem. However, if you'd like to review your input file before launching, you can do so by opening the File > Open File... menu in ICE, and navigating to the file. Once opened, you will be able to review the input file generated.

## 11.3 Launching a CAEBAT job

Once the appropriate input files have been generated, launching a simulation is a relatively simple task. To get started, click the green "+" button once more to create a new ICE Item. Select Caebat Launcher (Fig. 23) from the menu and click OK. A form will appear in the main ICE workbench area (Fig. 26). This form contains the information necessary for launching a CAEBAT problem. The first piece of necessary information is to specify an input file. From the drop down menu choose the configuration file generated for the Caebat Model (in our case Caebat_Model_1.conf). If you created your own input file in the previous step using the CAEBAT Model Builder, this file should appear in the list of available files.

The next step is to specify on which machine CAEBAT will be run, either locally or remotely. A default is localhost, however, additional hosts can be added by clicking the "+" button to the right of the Hosts table. When adding hosts, set the Execution Path to the directory of the machine's CAEBAT installation. If you are launching on a remote machine, also be sure that you have appropriate privileges for the CAEBAT install directory.

Lastly, use the Process menu in the upper right-hand corner; select the Launch the Job task from the drop- down menu and click the Go! button. Depending on your host machine's configuration, you may be prompted for login credentials.

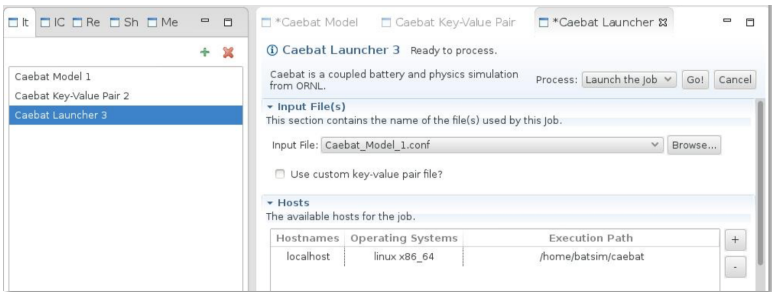



Figure 26: Battery simulation launcher in ICE.

As the simulation progresses the console window will display different information related to each component being executed in sequence. The simulation is finished when the Done! is displayed in Caebat Launcher:

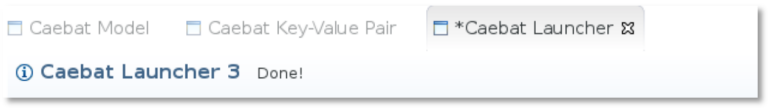



Figure 27: CAEBAT launcher in ICE.

## 11.4 Visualizing output

The output produced by a CAEBAT job can be visually analyzed in ICE by utilizing the VisIt plug-in. Click on Launch Visit and select the location for VisIt installation. If the simulation was run in Virtual Machine, select Launch Visit Locally and then click browse and select the path to VisIt binary (Fig. 28). Scroll down and give this connection a name (any characters) and then click Finish.

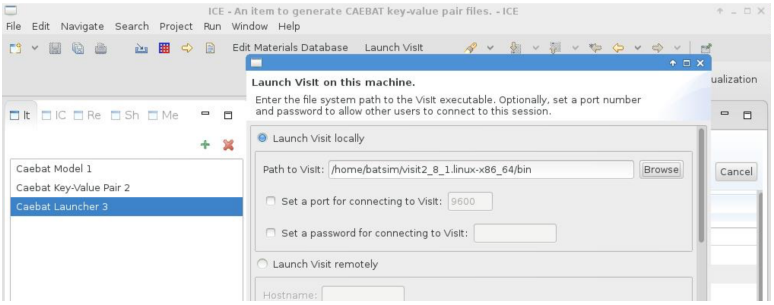



Figure 28: Launching VisIt within ICE.

Click on Open Perspective and select Visualization from the list (Fig. 29a). Switch from ICE to Visualization mode. In Visualization File Viewer selection can be made

for the files to view. Select the desired silo file(s) from the /home/batsim/ICEFiles/default/jobs/iceLaun directory. The work directory of the simulation contains the results as described in APPENDIX A. Select the silo file(s) corresponding to the thermal solution from the THERMAL_Amperes directory.

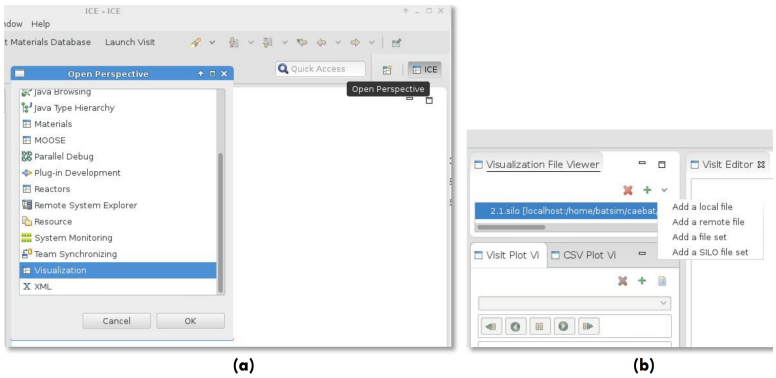



Figure 29: Switching to visualization mode and selecting files in ICE.

In the Visit Plot Viewer add a new plot by clicking the green "+" and selecting Scalars > Battery/Temperature to view the temperature distribution in the cell. Double click on the file name in the Visit Plot Viewer. Select pseudocolor from the drop down menu of plot options. The result shows a temperature distribution in the pouch cell under non-uniform cooling of the edges (Fig. 30). The visualization capabilities in ICE allow object rotation, translation, and zoom in/out. If several silo files were loaded with each file representing a time step, a play feature can be used to step through the solutions and see the progression in time. These capabilities provide a good tool to judge the goodness of solution. For extended visualization tools the user is advised to launch VisIt which comes as a part of VM (simply type visit in the command line and hit Return).

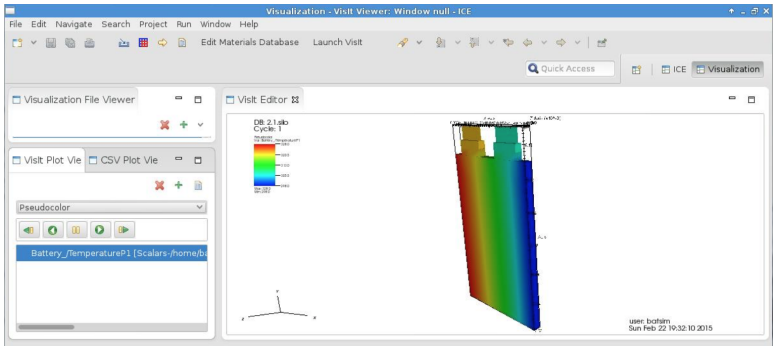



Figure 30: Temperature distribution in pouch cell visualized in ICE.

Simulation involving each of the cases located in examples directory can be performed

either using command line (Appendix A) or by using ICE. The BatterySim Virtual Machine comes with five different cell and module simulation setups, contained in /home/batsim/caebat/vibe/examples directory as shown in the table below. Each geometry, except unrolled cell, is discussed in details in APPLICATION EXAMPLES section. Any other meshes can be created by user to set up new simulation cases.

### 11.4.1 Case 2

Unrolled cell. Useful for testing new cell parameters (for example different materials or porosities) or testing new models on simple cell sandwich geometry.

Figure 31: Case 2 geometry.

### 11.4.2 Case 3

Cylindrical Li-ion cell.

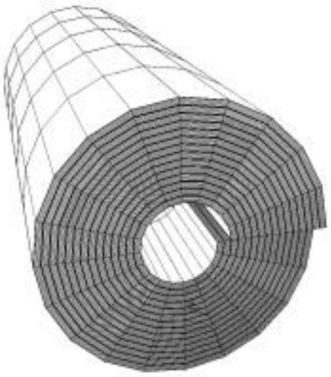


Figure 32: Case 3 geometry and mesh.

### 11.4.3 Case 6

Pouch cell. Default case in ICE with the NTG model coefficients based on NMC-Graphite cell discharge profiles.

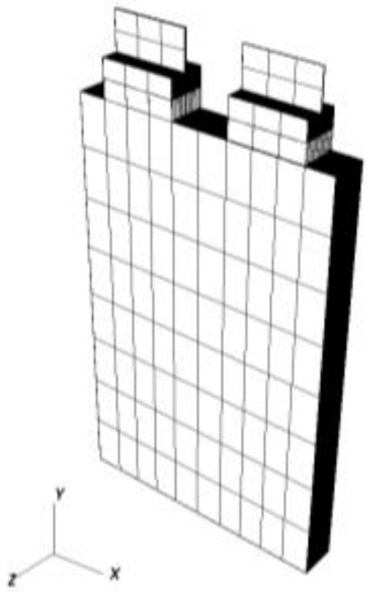



Figure 33: Case 6 geometry and mesh.

### 11.4.4 Case 7

4P and 4S modules of four pouch cells from case 6 connected in series (4S) or in parallel (4P).

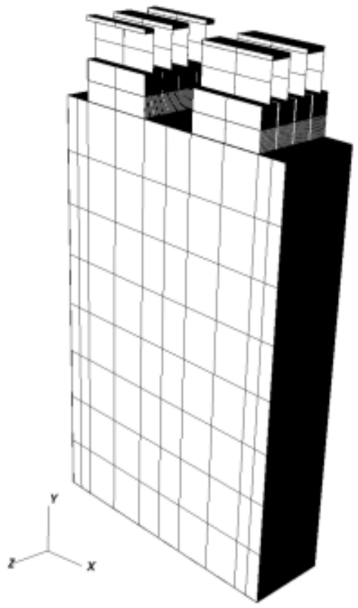



Figure 34: Case 7 geometry and mesh.

### 11.4.5 Case 10

4P module with demonstration of dynamic discharge.

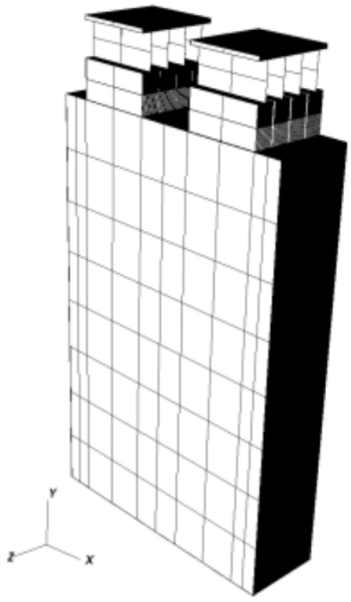



Figure 35: Case 10 geometry and mesh.

Simulation involving any of the above meshes can be prepared and launched using ICE. In the following example we will run the module simulation by importing the configuration files in ICE. The examples are based on Virtual Machine release of VIBE; with any other installation of VIBE and ICE the pathnames would be different.

Start with launching ICE by double clicking the Eclipse icon in VM. In order to import items (configuration files and key-value pair input files) in ICE click on the yellow arrow located in the top toolbar of the ICE window (Fig. 36).


Figure 36: Importing files in ICE.

This will create a dialog box where the user can browse to navigate to the files that should be imported. Let's start with the 4P module and first import the simulation configuration file into ICE. By clicking the Browse button navigate to the /home/batsim/caebat/vibe/examples/case7 directory and select the 4P.conf file. Select Caebat Model from the list and click Finish (Fig. 37a). This will create the Caebat Model form in ICE where the configuration parameters, components and time stepping are specified. Click Go! to create the corresponding ICE item. Similarly, import the key-value pair file from the case7/input directory (Fig. 30b) and click Go! to create the input file used within ICE. What is left is to create the job launcher by adding an item to the item viewer (click on the green "+" in the Item Viewer and select Caebat Launcher). Select the corresponding CaebatModel.conf file, check 'Use custom key-value pair file?' and select the corresponding CaebatKeyValuePair.dat file from the drop down list.

NOTE: Import of the key-value pair file into ICE is necessary only when this file will be modified by user. If no modifications are intended, ICE will use the file associated with the selected simulation case and the user can leave 'Use custom key-value pair file?' field unchecked.

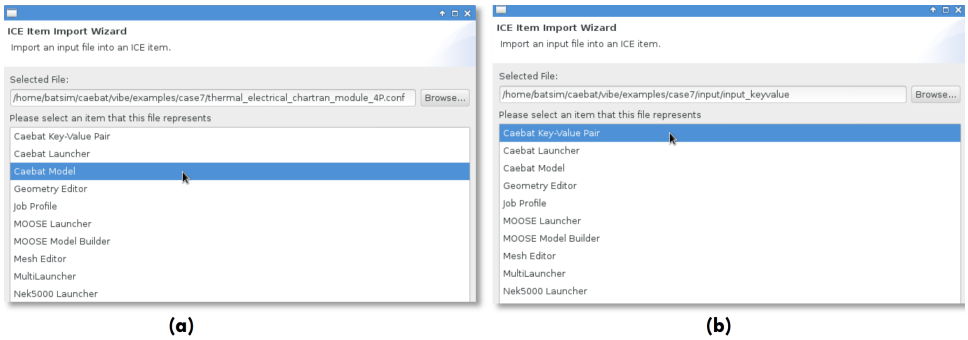



Figure 37: Importing items into ICE.

Launch the simulation by clicking Go!. When finished, display the result of the thermal solution as described in Visualizing Output section above (be sure to select the latest iceLaunch directory containing the results of the most recent simulation). The resulting window with the thermal solution is displayed in Fig. 38.

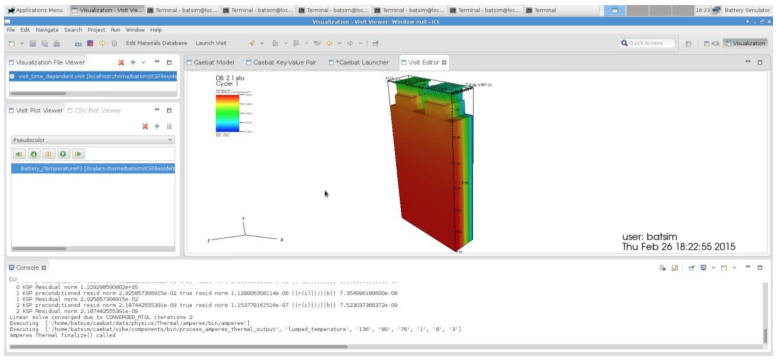



Figure 38: Temperature distribution in 4P Li-ion module.

Similarly, the simulation involving module with cells in series can be performed. Close the Visit Editor window, delete the files in Visualization Viewer and switch from Visualization to ICE mode which will open the Item Viewer. Close the current Caebat Model, Key-Value Pair and Launcher windows and delete the corresponding items from the Item Viewer. Using the procedure described for item import above, import the new Caebat Model using the configuration file for 4S simulation (_4S.conf) located in PathTo/examples/case7/ directory. Import the key-value pair input file from PathTo/examples/case7/input/ directory. Since the four cells are now connected in series, the total current flux should be four times less than the one used in the previous 4P simulation. Enter the corresponding number in the CurrentFlux field of the Key- Value Pair form as shown below (Fig. 39) and click Go!.


Figure 39: Changing values in the input file.

In the same manner as described for the 4P case, add the Caebat Launcher, select the appropriate model and key-value files and launch the simulation. When the simulation is done the solution can be checked by using the visualization viewer in ICE as previously described. This time let's check the electrical solution by viewing the potential distribution in 4S module. Launch Visit and select the silo file located in the ICE jobs directory where the recent launches are stored:

`/home/batsim/ICEFiles/default/jobs/iceLaunch_DateAndTime/work/ELECTRICAL_Amperes/outp`

Select the file 2.1.silo which corresponds to the final solution. In Visit Plot Viewer add an item (green "+") and select Battery/PotentialSolutionP1 in Scalars (Fig. 40). Click OK.

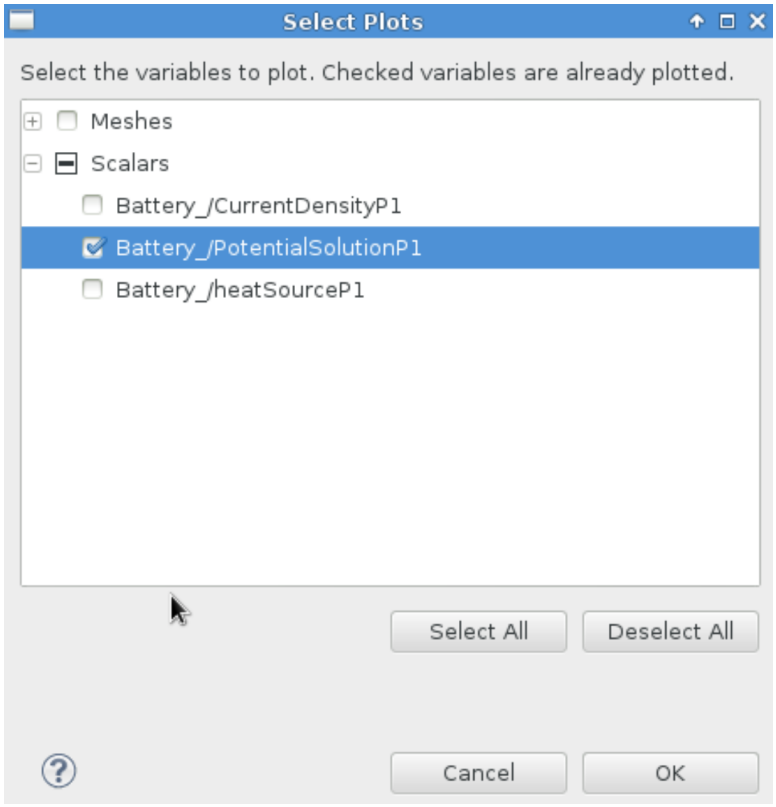



Figure 40: Selecting the potential as output variable from the solution.

After double clicking on the output variable name (Battery/PotentialSolutionP1) in the Visit Plot Viewer, the plot showing potential distribution in 4S module will appear (Fig. 41). Holding left mouse button down and moving the mouse will rotate the plot, holding Shift key down and dragging the mouse with left button pressed will translate the plot and using mouse scroll will zoom in and out.

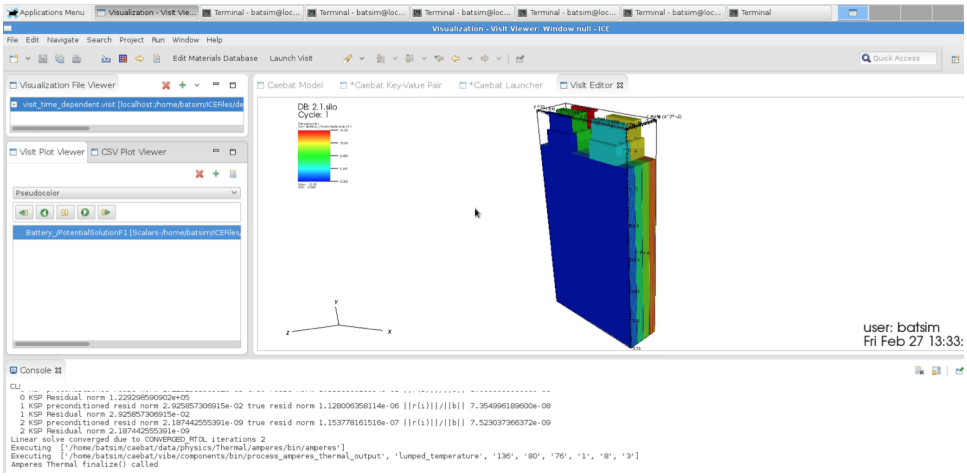



Figure 41: Output of electric potential in 4S module in ICE.

At this point user should be able to run any of the cases either from the corresponding case directory using command line or by using ICE to import the input files from the corresponding case directory and launching the simulation. Different discharge currents or time stepping can be applied. Pre-defined boundary conditions for thermal solution can also be changed. The default for the module case is uneven cooling of module sides with heat transfer coefficients of 15 W/m2K and 55 W/m2K which imitates failure of cooling system when air moves fast on one side and slow convective cooling is applied to the other side. These boundary conditions can be changed to investigate other cooling scenarios. Next the user can utilize the provided geometries and meshes to test other materials or models. If the discharge curves for other materials are available, the NTG coefficients can be determined and the user can input them into the key-value pair file as U and Y polynomials. The order of those polynomials can be changed as well (default is 6). Case2 and case3 are supplied with DUALFOIL as well as NTG pre-defined. The user can select either of the models by typing the corresponding name for CHARTRAN component in the Cebat Model form in ICE (as shown below). DUALFOIL model is based on porous electrode theory and requires significant amount of material parameters to be determined; if these are known for particular cell chemistry, user can set up DUALFOIL as an electrochemical component instead of NTG for case6 and case7 as well. Finally, the user can of course supply his mesh to set up a new simulation case in VIBE.

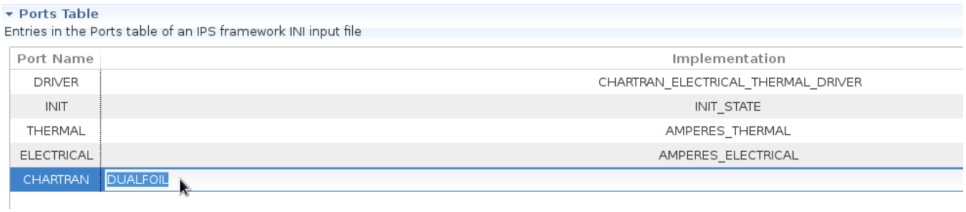



Figure 42: Port name and implementation in ICE.

# 12 Appendix C: Instructions for advanced installation

The following must be installed prior to installation of OAS and VIBE.

- CMake (version 2.8.6-rc3 recommended), available from cmake.org
- C, C++, and Fortran compilers (gcc4.7 or higher, g++, gfortran 4.3 and higher recommended)
- MPI available from mpich.org or open-mpi.org (open MPI 1.8.3 or higher)
- HDF5 (version 1.8.7 recommended), available from hdfgroup.org
- SILO (version 4.7.2 recommended), available from wci.llnl.gov
- CGNS is available at cgns.sourceforge.net When building cgns use ccmake to edit the flags before generating the Makefile. The flags can be edited to match the following:

```
BUILD_CGNSTOOLS          ON
CGNS_BUILD_SHARED        ON
CGNS_USE_SHARED          ON
CMAKE_BUILD_TYPE         Release
CMAKE_INSTALL_PREFIX     /PATH/TO/cgnsinstall_dir ENABLE_64BIT OFF
ENABLE_FORTRAN           ON
ENABLE_HDF5              ON
ENABLE_LEGACY            OFF
ENABLE_SCOPING           OFF
ENABLE_TESTS             OFF
FORTRAN_NAMING           LOWERCASE_
HDF5_INCLUDE_PATH        /PATH/TO/hdf5-1.8.9/include
HDF5_LIBRARY             /PATH/TO/hdf5-1.8.9/lib64/libhdf5.so
HDF5_NEED_MPI            OFF
HDF5_NEED_SZIP           OFF
HDF5_NEED_ZLIB           OFF
```

Type ‘c’ to configure and ‘g’ to generate the Makefile.

NOTE. Make sure the bashrc file contains the corresponding paths to hdf5 and cgns libraries:

```
Export LD_LIBRARY_PATH=$CGNS_ROOT/lib:$HDF5_ROOT/lib:$MPI_ROOT/lib:$LD_LIBRARY_PATH
```

### 12.1 ICE

ICE is a free and open source project available for download from the Eclipse download servers. To download, navigate to eclipse.org/ice and click the 'Downloads' button found in the site's top toolbar. ICE can be downloaded as a binary for Linux, Windows or Mac or as source code from the ICE GitHub repository github.com/eclipse/ice. ICE also provides a wiki and other applications that document the current capabilities of ICE as well as the bug tracker for the project. Detailed instructions on using CAEBAT with ICE can be found at wiki.eclipse.org and in an instruction video at youtube.com. ICE is released under the Eclipse Public License (EPL).

### 12.2 OAS/VIBE

Please contact the team to obtain access to repository.

## 13 Appendix D: Implementation of tight coupling

One of the major advantages of CAEBAT project is that the final product is an open source software. In other words the user can become a developer and implement his own simulation scenarios as well as integrate components into VIBE. In this example we discuss implementation of tight coupling (Picard iteration) between electrochemical (DualFoil) and Thermal components in VIBE.

In general, Picard iteration to a specified convergence criteria provides tight coupling of two physics components f0 and f1 exchanging variables via functions g. At each time step, a fixed-point iteration can be schematically represented as follows.

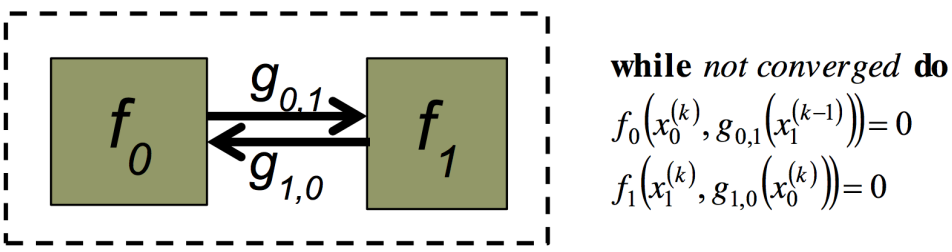



Figure 43: Example of convergence.

In terms of implementation in VIBE, such task is relatively simple and requires creating the new simulation driver which would call the corresponding components in the right sequence until the desired convergence is reached. Drivers can be found in VIBE/trunk/components (see figure below).

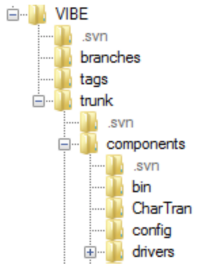



Figure 44: Components directory for VIBE.

Simulation drivers follow Python logic with the Driver class where the actual call of components is performed. The sequence and number of components are characteristic of the simulation setup. For instance the unit iteration sequence in this example can be represented as

```
class Driver(Component):

    def __init__(self, services, config):
        "code here"

    def step(self, timestamp=0):
        "code here"

    # Iterate throught the timeloop
    for t in tlist[1:len(timeloop)]:
        "code here"
        while abs(T_new_sum - T_old_sum) > tol :
            services.call(chartran_comp, 'init', t)
            services.call(chartran_comp, 'step', t)
            services.call(chartran_comp, 'finalize', t)

            services.call(electrical_comp, 'init', t)
            services.call(electrical_comp, 'step', t)
            services.call(electrical_comp, 'finalize', t)

            services.call(thermal_comp, 'init', t)
            services.call(thermal_comp, 'step', t)
            services.call(thermal_comp, 'finalize', t)
```

Where the Electrochemical ('chartran'), Electrical and thermal components are called in the above sequence at each time step. In this particular case the convergence is checked in terms of temperatures from the current and previous Picard iteration. The thermal component is coupled to DualFoil via temperature-dependent diffusivities and Buttler-Volmer kinetics in DualFoil component. To check the influence of tight coupling, simulations were run on an unrolled cell with properties of the polymer cell described in Doyle, Fuller 1993. The results are shown in Fig. 45. Weak dependence of sources on temperature results in very fast convergence of Picard iterations (typically within 4 iterations).

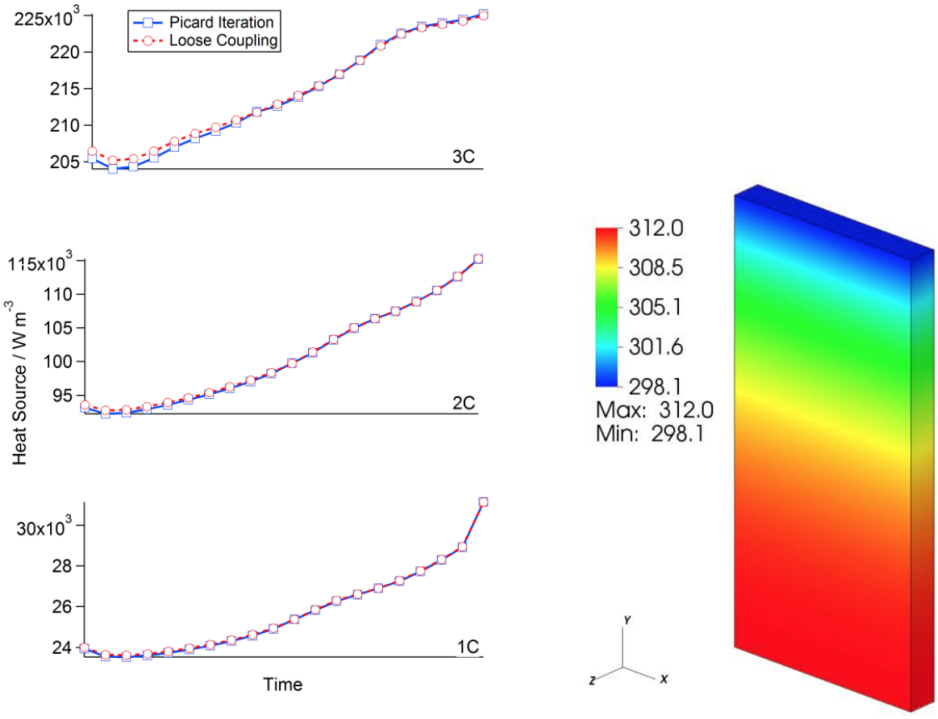



Figure 45: Influence of the coupling scheme on the heat source in Li-polymer cell.

# 14 References

Doyle, M., T. F. Fuller, et al. (1993). "Modeling of Galvanostatic Charge and Discharge of the Lithium/Polymer/Insertion Cell." Journal of the Electrochemical Society 140(6): 1526-1533.

Eldred, M. S. DAKOTA, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 4.1 reference manual.

Elwasif, W. R., D. E. Bernholdt, et al. (2012). Parameter Sweep and Optimization of Loosely Coupled Simulations Using the DAKOTA Toolkit. International Conferences on Computational Science and Engineering, Paphos, Cyprus, IEEE.

Fuller, T. F., M. Doyle, et al. (1994). "Relaxation Phenomena in Lithium-Ion-Insertion Cells." Journal of the Electrochemical Society 141(4): 982-990.

Fuller, T. F., M. Doyle, et al. (1994). "Simulation and Optimization of the Dual Lithium Ion Insertion Cell." Journal of the Electrochemical Society 141(1): 1-10.

Kim, G.-H., K. Smith, et al. (2011). "Multi-Domain Modeling of Lithium-Ion Batteries Encompassing Multi-Physics in Varied Length Scales." Journal of the Electrochemical Society 158(8): A955-A969.

Seong Kim, U., J. Yi, et al. (2011). "Modeling the Dependence of the Discharge Behavior of a Lithium-Ion Battery on the Environmental Temperature." Journal of the Electrochemical Society 158(5): A611-A618.

Srinivasan, V. and C. Y. Wang (2003). "Analysis of Electrochemical and Thermal Behavior of Li-Ion Cells." Journal of the Electrochemical Society 150(1): A98-A106.

Website, B. "BatPac." from cse.anl.gov/batpac.

Allu, S., Kalnaus, S., Elwasif, W., Simunovic, S., Turner, J.A., Pannala, S., A new open computational framework for highly resolved coupled three-dimensional multi-physics simulations of Li-ion cells, J Power Sources 246 (2014), 876-886.

# 15 Team

The CAEBAT ORNL team consists of multidisciplinary researchers working on various aspects of computational science related to batteries and we are working closely with the experimental groups at ORNL for validation. The team structure is given below. More information can be found at the project website batterysim.org.

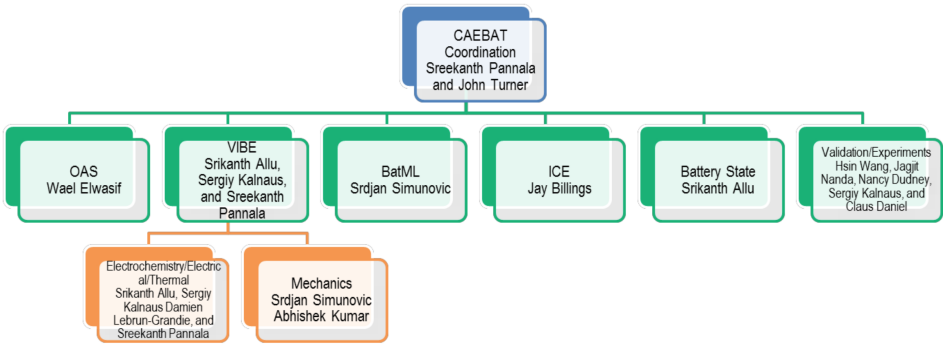



Figure 46: CAEBAT team members at ORNL.

# 16 Acknowledgment

Research was sponsored by the U.S. Department of Energy, Assistant Secretary for Energy Efficiency and Renewable Energy, Vehicle Technologies Program, Hybrid Electric Systems activity, under contract DE-AC05- 00OR22725 with UT-Battelle, LLC. The support of the CAEBAT (Computer Aided Engineering for Batteries) program with Brian Cunningham and David Howell as the managers is acknowledged. The support of the ORNL Sustainable Transportation program office (Ron Graves and Claus Daniels) is also acknowledged along with contributions from Damien Lebrun-Grandie, Abhishek Kumar, Jagjit Nanda, Hsin Wang, and Nancy Dudney.