MOCASSIN: User Manual

Authors: Sebastian Eisele, Steffen Neitzel-Grieshammer
Forschungszentrum Jülich & RWTH Aachen University

1 Contents

2 Introduction

MOCASSIN is a Markov chain Metropolis (MMC) and Kinetic Monte Carlo (KMC) model building, model visualization, and simulation system developed for crystalline solid electrolytes. It is based on fixed position lattices, with symmetry processing using space groups, and supports numeric simulation of many defect transport and distribution problems in arbitrary crystals with interacting defects. Example properties of interest obtained by KMC/MMC simulations are:

- Defect distribution in thermal equilibrium
- Diffusion coefficients & correlation factors
- Conductivity and mobility
- Average activation energy for ionic transport
- Migration barrier distributions
- …

The model system supports multiple complex components in mostly arbitrary combinations within a single model to ensure broad applicability of the system, including:

- Up to 63 mobile species
- Vacancy, interstitial, interstitialcy, and vehicle mechanisms
- Small polar hopping or similar charge transports
- Combined charge transport and physical movement
- Automatic detection and symmetry reduction of pair interaction
- Custom multi-Body interactions containing up to 9 positions
- Local site energies
- Custom event attempt rates for each reference migration
- …

This user guide gives an overview of available components and provides basic usage instructions for model building, simulation, and result evaluation using simple examples. Examples are provided through example boxes:

Example: I am an example box

i. Do something …

Additionally, the guide annotates some passages with several importance tags to inform about a topic more deeply or provide supporting information:

i. **Note:** Provide additional information that is “nice to know” but not critical for the correct functionality of MOCASSIN
ii. **Important:** Provides information about problems or unexpected behavior of the program If these passages are written in red, there are especially critical and might corrupt the validity of simulation results if not considered during usage of the program
iii. **Tips:** Provides usage tips/additional information at the end of certain sections. They are primarily based on experience and development insights

2.1 Prerequisites & Recommendations

Usage of MOCASSIN, and partially this guide, requires users to have at least a basic understanding of the following topics:

- Using a Linux shell (Better: using a high-performance supercomputing cluster)
- Basic SQL queries
- Python/Shell/… scripting for data collection/evaluation

Programmatic access to MOCASSIN and/or binary simulation state dumps for advanced result evaluation with the MOCASSIN API additionally require:

- Basic C# programming with .NET Core
- Using NuGet packages

It is generally recommended to have the following tools available when using the MOCASSIN system:

- An SQLite database browser

2.2 Available Components

MOCASSIN is divided into components for model building, the simulator/solver, and helper scripts. The following components are available:

- Graphical User Interface for model building (WPF App with Setup, Windows 8/10 64-bit)
- Solver binaries for x86-64 systems (Win64/Linux)
- NuGet packages with documentation for the processing assemblies (.NET Standard 2.0)
- Python3 concurrency startup wrapper for the solver (shared/distributed/hybrid)
- Python3 submit wrapper for the SLURM workload system
- Python3 result collection script

2.3 Platform Compatibility

The user interface is a WPF based graphical application available for 64-bit versions of Windows 8/10 only. The solver binaries are available for x86-64 CPU architecture running Linux and Windows 64-bit operating systems only. The processing API complies to the .NET Standard 2.0 and thus runs on x86-64 Win64/Linux/MacOS supporting the .NET Core runtime. Other target platforms, especially non x86-64 architectures, are not officially supported.

2.4 License & Legal Disclaimer

MOCASSIN is a closed source project, copyrighted by Forschungszentrum Jülich GmbH/DE and RWTH Aachen University/DE. Provided end user components can be used free of charge for research purposes. Publishing results generated with MOCASSIN requires the work of the original program authors to be acknowledged. As soon as an official publication of MOCASSIN exists, an appropriate citation is required.

End user components of MOCASSIN are provided “as is” without any warranties under the legal disclaimer of the MIT license:

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

3 Setup

3.1 GUI installation

The graphical model builder is a WPF application shipped as a Click-Once Windows App installer. The installation process can be triggered using “ModelBuilderGUI/setup.exe”. Depending on your system, the setup might request to install the following mandatory components:

I. .Net Framework 4.8

If you are upgrading from a previous version, it is recommended to use “CleanInstall.cmd” to start the installation. This will delete the old space group database created by MOCASSIN causing MOCASSIN to redeploy the newest version.

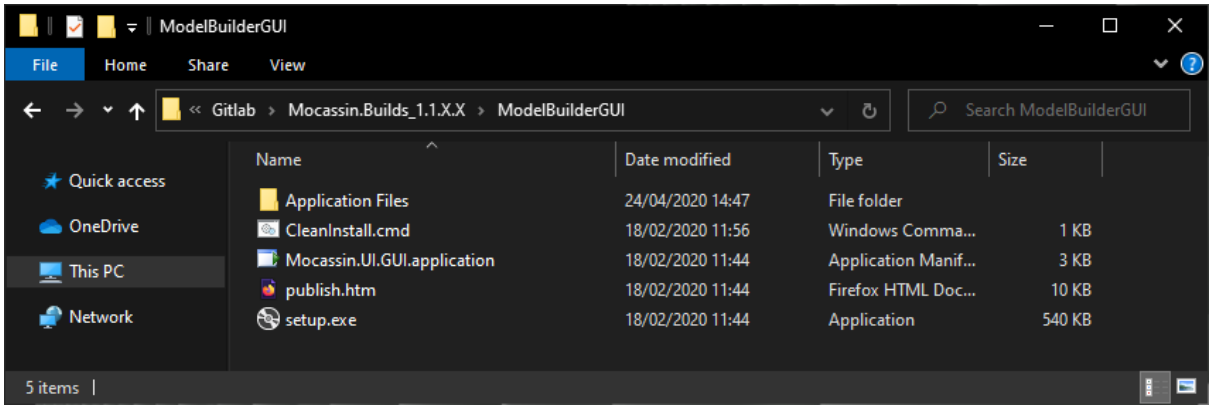

Figure 1: Contents of the UI setup folder.

The installation registers the following file types to be associated with the MOCASSIN App only if the extension is not already associated otherwise:

I. Mocassin Project File “*.mocprj”

3.2 Simulator installation

The simulator does not require installation, copy either “simulator_win64” or “simulator_linux64” to your Win64 or Linux target machine, respectively.

3.3 Concurrent startup configuration (Wrapper for HPC, Linux)

The MOCASSIN simulator is a single-core application that does not support parallelism out of the box. It is meant to be executed in a multiprocessing approach using either shared, distributed, or hybrid memory models on a Linux HPC cluster. To use the provided startup script “moccasin_mt.py”, a python3.6+ installation is required. If you want to use MPI over multiple computing nodes, the package “mpi4py” is required, which can be installed for the user using the following shell command:

```
python3 -m pip install --user --upgrade pip
python3 -m pip install --user mpi4py
```

To setup the execution wrapper, open the “mocassin_mt.cfg” file and edit the “AutoSearchPath” to target either “bin_icc” (Compiled with Intel ICC) or “bin_gcc” (Compiled with GNU GCC) directory in the “simulator_linux64” directory on your executing machine.

Note: It is not difficult to write your own concurrent execution system if you do not intend to use the included python solution.

4 The WPF User Interface

4.1 Introduction

The WPF graphical interface is built to assist in the model building process through a validation service, a limited amount of 3D visualization for geometry and a deploy system for databases containing simulation series. Almost all features of MOCASSIN model building can be accessed through the interface. It should be noted that creating a modern UI is an extremely work intensive task. Thus, some functionality of the MOCASSIN UI lacks the convenience commonly found in high-standard industrial products, as the effort far outweighs potential benefits.

Figure 2: The empty main window for Windows 10 that is displayed when no project is loaded.

4.2 Main Window

The main window of the MOCASSIN app gives access to all controls and tools of the model builder system. Unless a ".mocprj" file is directly opened or dragged into the shortcut, the window should be mostly empty and look like shown in Figure 2. The actual layout might differ slightly if the software is run on Windows 8 instead of 10. MOCASSIN provides most controls as dynamic work tabs that will spawn in the main tab-control and can be relocated between available tab-hosts by drag & drop.

4.2.1 Menus

File Menu

The file menu provides access to all operations concerning the project files. The menu is structured as follows with quick access via "Alt" key:

I. **New**
   a. **Project** — Opens a file dialog to create a new project file
   b. **Model** — Creates a new model in an opened project file
   c. **Window** — Opens a new work window with a tab-host

II. **Open Project** — Opens a file dialog to load a project file

III. **Close Project** — Closes the currently opened project

IV. **Save Project** Saves the currently opened project
V. **Exit** Close the program

Project Menu
The project menu provides access to all content controls that manipulated the models in the current project. The following menu entries are available:

I. **Particle Model** Opens a new control tab for particle model data
II. **Structure Model** Opens a new control tab for structure model data
III. **Transition Model** Opens a new control tab for transition model data
IV. **Energy Model** Opens a new control tab for energy model data
V. **Simulation Model** Opens a new control tab for simulation model data
VI. **Lattice Model** Opens a new control tab for lattice model data
VII. **Customization Templates** Opens a new control tab for customization templates
VIII. **Job Templates** Opens a new control tab for job templates

Viewers Menu
The viewers menu provides access to the 3D visualization controls of MOCASSIN. The following options are available:

I. **DX9 3D Viewer** Opens a new DX9 based viewer tab (deprecated)
II. **DX10+ 3D Viewer** Opens a new DX10/11 based viewer tab
III. **DX10+ 3D Window** Opens a new DX10/11 based viewer in a new window

Tools Menu
The tools menu provides access to tools assisting with building and deploying models. The following controls are provided:

I. **Model Validator** Creates a new model validator tab for live validation of a model
II. **Simulation Builder** Creates a new simulation builder tab to deploy simulation databases
III. **Msl Tools** Creates a new tab to access "*.msl" file tools

Help Menu
The help menu supplies access to help and update functionality. The following controls are available:

I. **Check for updates** Triggers an application update (Only for network deployed versions)

4.2.2 Solution Explorer

The solution explorer tab is located at the left tab-host of the main window and cannot be relocated by drag & drop. It displays a simple tree version of the currently opened project file and allows to rename the major project components. The following additional functionality is available:

I. **Context Menu (right mouse click)**
   a. **Delete <project>:** Deletes the project named <project>. This action requires confirmation by the user
   b. **Duplicate <project>:** Duplicates the project named <project>. Option to select all content or model only (user prompt)

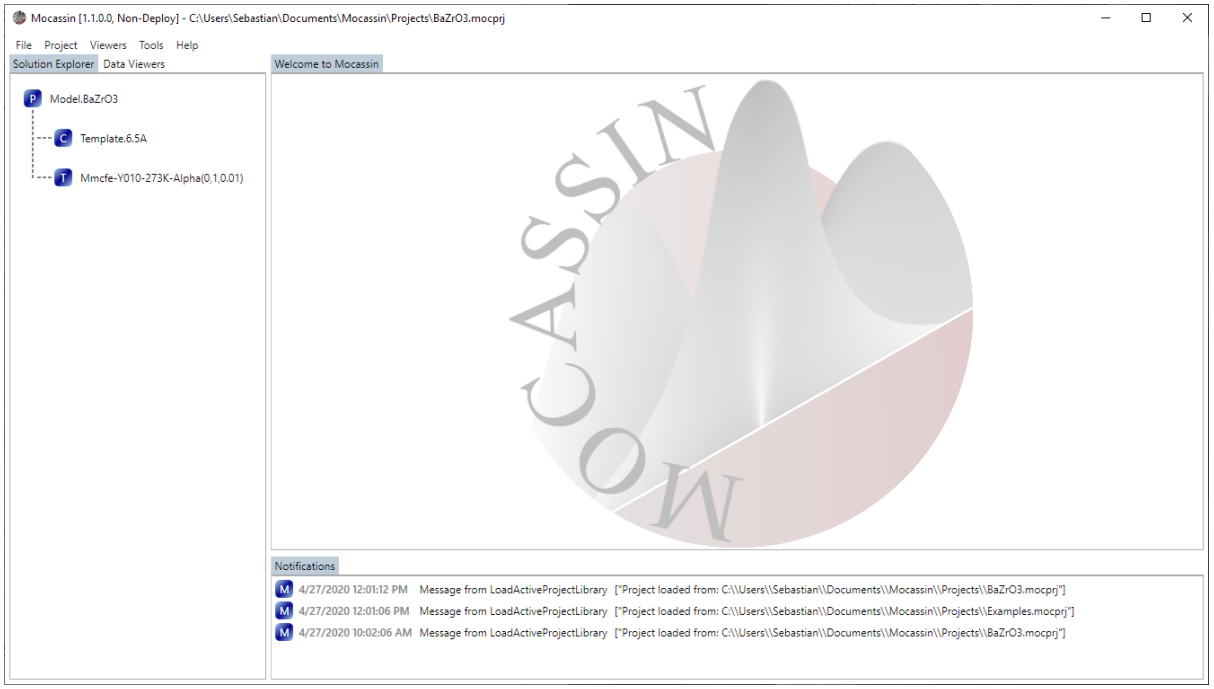

Figure 3: Solution explorer with a model, customization template and job translation template.

II. **Project Selection:** Selecting a project by left mouse clicking will cause it to be the default target project to be set for new work tabs

4.2.3 Data Viewer

The data viewers tab is located at the left tab-host of the main windows and cannot be relocated by drag & drop. If selected, the tab shows sub-tabs at the left corner of the control, providing multiple ways of displaying project data. The context menu of each viewer can be opened by right mouse click on the tab header. The following view options are available:

I. **Tree Viewer:** Displays objects / projects as an expandable tree
II. **XML Viewer:** Displays objects / projects as their XML representation in a text editor
III. **JSON Viewer:** Displays objects / projects as their JSON representation in a text editor

The viewers support the following actions and functionality:

I. **Context Menu (right mouse click)**
   a. **Load project <project>:** Loads a complete project into the viewer.
   b. **Generate folding markers:** Generates the folding markers for the data in the viewer. This option is currently supported for XML only

II. **Drag & Drop:** Dropping objects onto the header of one of the viewers will cause the affiliated representation to be displayed in the viewer.

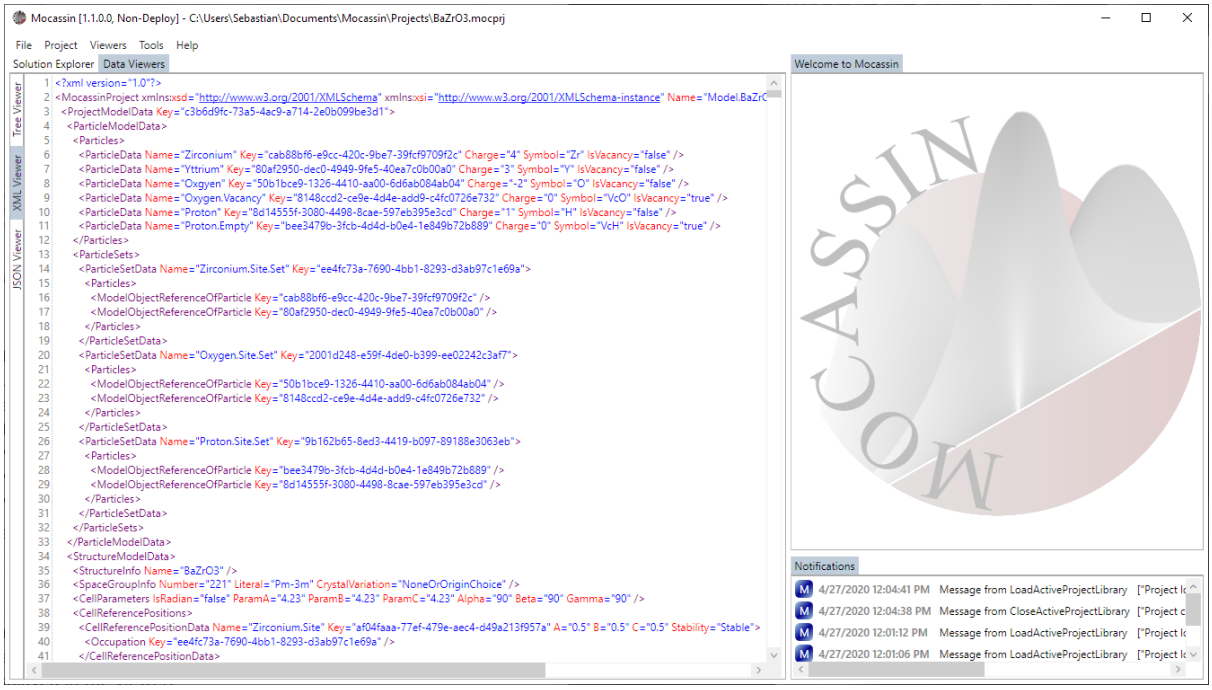

Figure 4: Data viewer with XML representation of a project.

4.2.4 Exceptions & Error Logging

MOCASSIN generally notifies about caught exceptions in the “Notifications” tab. Most exceptions displayed there are uncritical and/or expected, e.g. failed expression parsing, and will not compromise the stability of the program. When a critical exception slips through all exception handling, the program will display a notification before terminating and write a log to “%USERPROFILE%/Mocassin/”.

Note: If the program terminates without a notification, it is most likely caused by a native library and the windows event log will contain an affiliated entry about the failed module. A known issue is “d3d9.dll” crashing if a display overlay tool like “RivaTuner Statistics Server” is running. In these cases, the overlay must be deactivated or “Mocassin.UI.GUI.exe” must be added to the exclusion list.

4.2.5 Custom Settings

The default settings of MOCASSIN are not very restrictive, allowing most models to pass validation without hitting soft limitations. If a soft limitation is encountered and results in a rejected model instead of a simple warning, warning and error limits can be manually set in “%USERPROFILE%/Mocassin/Mocassin.Project.Settings.Global.xml”.

Note: It is not recommended to change the validation limitations and geometry comparison tolerances unless you know what you are doing. It may cause the geometry processing pipeline to produce weird results and/or cause out of memory exceptions/endless loops if the limits are set out of useful value ranges.

4.3 Model Building

The following sections explain how to use the model building system by creating a very simple model of both MMC and KMC based for doped ceria with oxygen vacancy migration.

4.3.1 Principles & theoretical background

Model building in MOCASSIN evolves around describing the three possible states (initial, transition and final state) energetically. This is done using a set of assumptions and default calculation principles that must be kept in mind when building a model. The energies of initial $E(S_0)$ and final state $E(S_2)$ are directly calculated by summation of local defect energies of all *stable* positions and the *stable to stable* interactions (pairs + clusters) directly involved in an MMC or KMC event. For describing the transition state energy $E(S_1)$, required for KMC only, MOCASSIN uses a simple interpolation method with a surrounding dependent base value $E(S_1)_{\text{base}}$, which is calculated as the summation of interactions, and the other two state energies as shown in eq. (1).

$$E(S_1) = \frac{E(S_2) - E(S_0)}{2} + E(S_1)_{\text{base}} \quad (1)$$

This means that the interactions for the transition state must be defined in such a way that they define an environment dependent base value for the migration barriers as a summation of *unstable to stable* interactions and local defect energies on the transition sites. The way of how the base barriers and the other state energies are translated into an absolute $E(S_1)$ can be replaced through a custom C function, however this is an advanced topic and requires compilation of a small C library.

Important: Eq. (1) is currently dangerous for charge transport KMC (e.g. polaron hopping), as the stable sites of the migration path are not set to "empty" (non-interacting) by MOCASSIN if the transport involves no physical movement of ions. This allows a user to define *stable to stable* site interactions that affect $E(S_1)_{\text{base}}$. It is thus highly recommended to never model interactions between "default" occupations of lattice components, instead use the occupation of the host matrix as the reference point (as commonly done in DFT). For example, in ceria there are no interactions for the Ce^{4+}, O^{2-} species if all interaction partners are located on *stable* sites.

If an electric field is defined for KMC, the difference in potential energy between the two stable states $\Delta E_{\text{el}}(S_0 \rightarrow S_2)$ affects the migration barrier as defined in eq. (2). The equation is a simplified version of the scalar projection of the shift in charge focal point between the two states onto the electric field vector $\vec{E}_f$.

$$\Delta E_{\text{el}}(S_0 \rightarrow S_2) = \frac{1}{2} \cdot \vec{E}_f \cdot \int (\vec{r_2} - \vec{r_1}) \mathrm{dq}$$

4.3.2 Creating projects/models

MOCASSIN stores data in solution files with the extension ".mocprj" that can contain multiple models. Projects can be created using the file menu "File→New→Project" and models with "File→New→Model". Creating new models is only possible if a project file is open. MOCASSIN does not support auto saving, thus it is recommended to save regularly. If you attempt to close an open project with unsaved changes in any fashion, the program is going to prompt if unsaved changes should be accepted or discarded.

Example: Create your first project file & model

i. Bring up the project creation dialog with "File→New→Project"
ii. Choose a file location for your project
iii. Create a new model with "File→New→Model"
iv. Left click the model name in the "Solution Explorer" tab to rename it
v. Left click the model to select it as default work project

4.3.3 Model validation

One of the convenience features of MOCASSIN is a live validation system designed to prevent numerous obvious input mistakes, which might lead to model inconsistencies that cause the simulation build system to fail during creation of P1 model extensions or the low level data structures for the solver. When building or manipulating models, it is thus highly recommended to hook a

validator to your current model. A validator tab targeting an "empty" project can be created by "Tools→Model Validator" and will be spawned in the "Notifications" tab-host. After creation, the drop-down menu at the top can be used to select to which model should be validated. The validation observes the model and will update the shown report list each time a change is detected. The following additional functionality exists:

I. **Context Menu (Right click)**
   a. **Deactivate auto updates:** On/Off switch auto updating of the report list
   b. **Show errors only:** On/Off switch for displaying only error reports
   c. **Show target selection:** On/Off switch for displaying the target selection drop-down menu

Each report displays a text popup of its contents on mouse-over, providing details about the performed operation and potential warnings/errors. There are two reasons for object rejection: (i) "*Invalid*", signaling that the affiliated object failed a consistency check or is outside of parameter limitations; and (ii), "*Exception*", which signals that the model processing system has caught a program exception. The latter is usually caused by an empty entry in one of the data grids and can then be solved by removing the entry from the affiliated data grid.

Example: Attach a model validator to your new model

i. Create a validation tab "Tools→Model Validator"
ii. Select "<YourModelName>" from the drop-down menu
iii. As shown in Figure 5, four valid reports should appear
iv. Bring up the context menu (right mouse click) and enable "Show errors only"

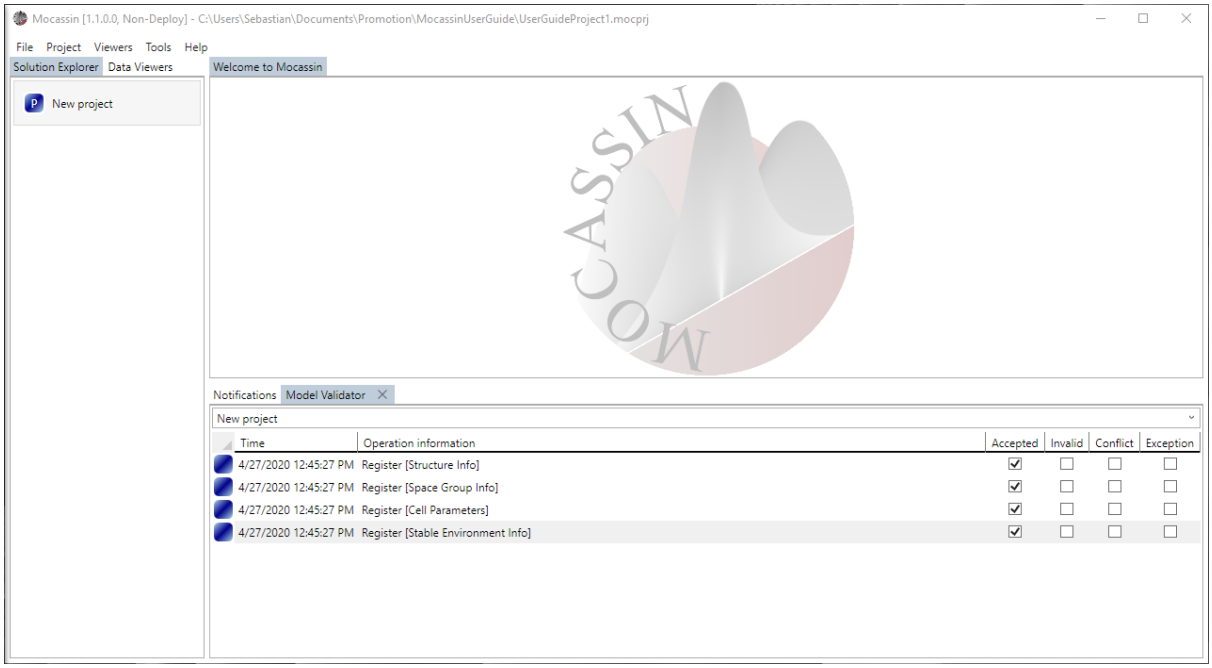

Figure 5: A model validator tab targeting the newly created project.

4.3.4 Work Tabs, Model Objects & Object References

All controls are provided in work tabs that require the user to select which model or template the work tab should target. The targets can be selected by either using the drop-down menu at the top of each work tab, or by starting to type the name of the target into the selection box and use the autocompletion. Each work tab defines model parameters (unique) or multiple model objects through data grids, building your Monte Carlo model.

Note: Model objects are identified using an ID in the background. A name can be set optionally to display references in the UI in a readable fashion. It is recommended to use descriptive naming to improve readability of a model, e.g. “Occupation.Cation” to describe an occupation set.

Important: Many objects are referenced at multiple points within a model later. This is realized through either data grids with drop-down selection or drag & drop onto lists. Any data grid entry where the row header cursor changes to a “Hand Symbol” on mouse over supports to start a drag operation by pulling the header symbol (Colored rectangle).

4.3.5 Particle model

The first step in any model is to define which species exist and in which combinations the species are required to describe occupations of cell sites. For this, MOCASSIN provides the “Particle Model Control”, a new work tab can be created using “Project→Particle Model Control”. In the control, multiple instances of the following objects and properties can be defined through the data grids:

I. **Particle:** Defines a species/pseudo species
   a. ***Name:*** Set the UI display name
   b. ***Symbol:*** Set an element/pseudo element symbol. The symbol identifies the underlying species, causing mass conservation checks to recognize multiple charge states of the same species, e.g. “Ce^{3+}, Ce^{4+}”
   c. ***Charge:*** Assigns an arbitrary charge state to a species. It is recommended to use relative charge states when simulating polaron hopping
   d. *Vacancy Flag:* On/Off switch for marking the species as a vacancy pseudo particle that supports physical movement

II. **Particle Set:** Defines occupation options as a set of particles
   a. ***Name:*** Set the UI display name
   b. ***Particles:*** Sets the particle references belonging to the set. Particles can be added by drag & drop of particles from the neighboring data grid

Note: There is an implicit particle object called “Void” which serves as the “nothing” state. It cannot be referenced in occupation sets manually and is automatically included if an occupation set is assigned to an unstable site.

Important: The first particle of a particle set is currently the “default” occupation value for a site, it is therefore recommended to set the occupation of the host matrix as the first particle for each set. On migration/unstable sites, the “Void” particle serves as the default state in all cases.

Example: Define the particles & particle sets for yttrium-doped ceria

i. Open a new particle control tab “Project→Particle Model Control”
ii. Select “<YourModelName>” from the drop-down menu if not selected
iii. Define cerium, yttrium, oxygen, and vacancy particle
iv. Define cation occupation, anion occupation and migration site occupation and add the affiliated particles by drag & drop
v. Your model control should look like Figure 6

Note that cerium and oxygen will be the default particles and <Migration.Site.Occupation> will be used for the unstable transition position, having “Void” as implicit default particle

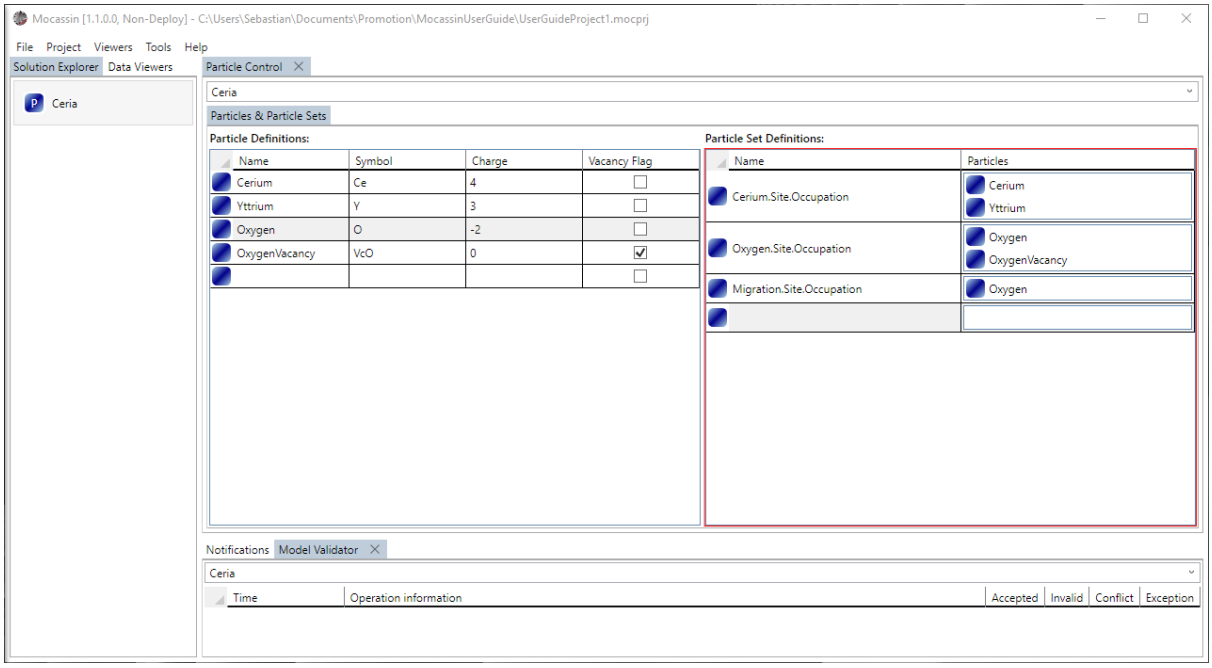

Figure 6: The particle control tab with the model objects to model yttrium-doped ceria.

Additional information & tips:

- You cannot define more than 63 custom species
- You can use the same vacancy for multiple migrating species, this will however prevent individual tracking of affiliated vacancies in the simulation
- It is recommended to define pseudo species as migration site occupation for charge transport and vehicle movement cases, e.g. “Electron” and “OH-Ion”

4.3.6 Structure model

The structure model defines the symmetry and unit cell required for your model and the affiliated control can be created using “Project→Structure Model Control”. The following unique model properties must be selected in the “Geometry & Space Group” sub tab:

I. **Space Group:** Select a space group from the provided list of options
II. **Cell Settings:** Set the cell geometry
   a. ***Name:*** Name your structure (optional)
   b. ***Parameters:*** Set the unit cell parameters in [Å]
   c. **Angles**: Set the unit cell angles in [°]

The geometry of the unit cell will be limited by your space group selection and affiliated text boxes may switch to read-only mode. The list located right next to the selection grid will display the symmetry operations included in the selected group. The following model objects can be defined in the “Cell Positions” sub tab:

I. **Cell Position:** Defines a reference site (e.g. Wyckoff Position)
   a. ***Name:*** The UI display name
   b. ***A:*** The coordinate in ‘A’ direction of the cell $[0 \dots 1)$
   c. ***B:*** The coordinate in ‘B’ direction of the cell $[0 \dots 1)$
   d. ***C:*** The coordinate in ‘C’ direction of the cell $[0 \dots 1)$
   e. ***Stability:*** Select either *stable* or *unstable* site behavior
   f. ***Occupation:*** Select the affiliated occupation *particle set*

Each position will be symmetry extended and the result is displayed in the list located to the right of the position data grid.

Important: It is not supported to place atoms unrealistically close to each other, the default geometry comparison tolerance is equivalent to around 0.1 – 1 pm tolerance in most unit cells.

Example: Define the ceria unit cell

i. Open a new "Project→Structure Model Control" tab and target your model
ii. Select space group 225 (Fm-3m) from the list (192 operations)
iii. Name your structure ceria (optional)
iv. Set the cell parameter 'a' to 5.411 (Other fields are grayed out due to cubic symmetry)
v. Switch to the "Cell Positions" sub tab
vi. Add the stable sites cerium at (0 0 0) and oxygen at (¼ ¼ ¼)
vii. Add the migration site at (½ ¼ ¼), set the stability to *unstable*
viii. Select the affiliated occupations from the drop-down menus

You should verify that you have 4 cerium sites, 8 oxygen sites and 24 oxygen migration sites using the list on the right, and that your model looks like shown in Figure 7 and Figure 8. You can optionally open the DX10+ 3D viewer and configurate the atom colors and sizes. If you use the common color choices for cerium and oxygen, your unit cell should look like shown in Figure 9.

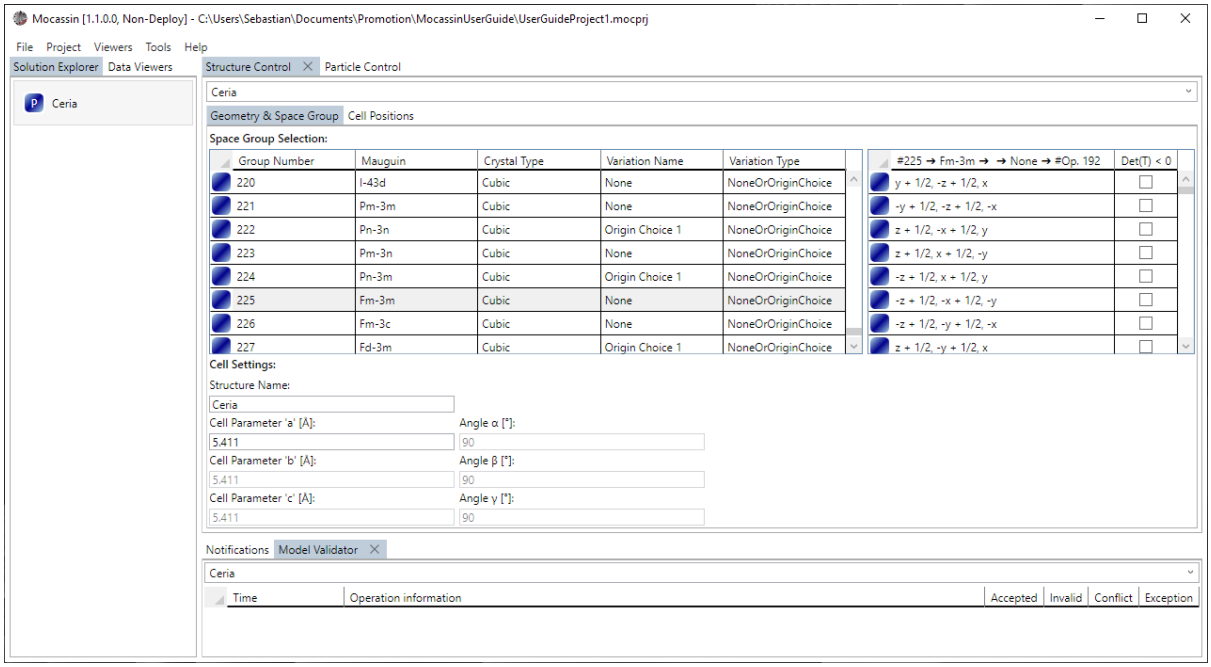

Figure 7: Structure control "Geometry & Space Group" sub tab with the settings for ceria.

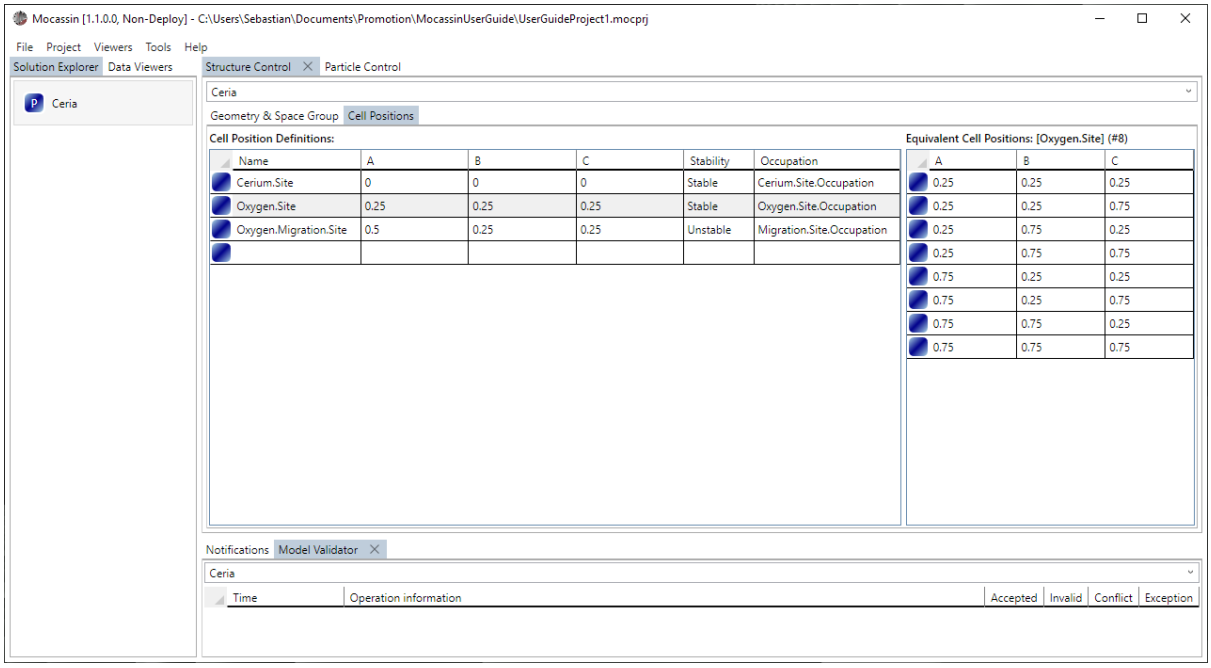

Figure 8: Structure control "Cell Positions" sub tab with the reference sites for an oxygen migration in ceria.

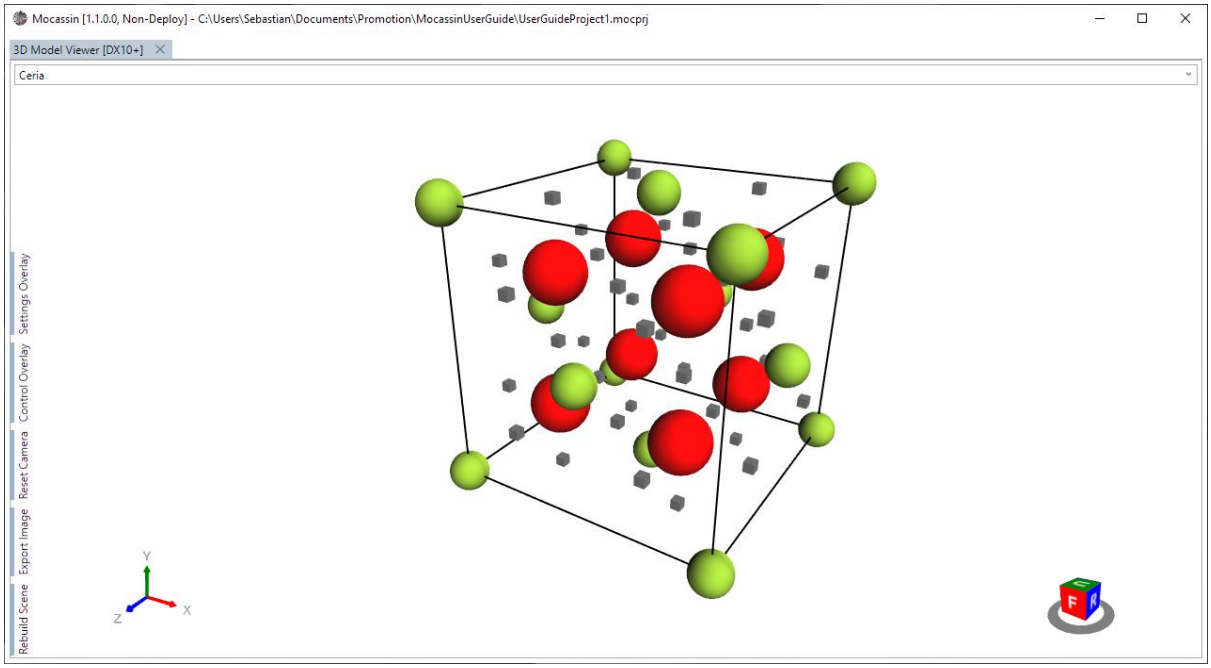

Figure 9: The DirectX10+ viewer relocated to a separate work window displaying the unit cell of ceria: oxygen (red), cerium (yellow-green) and oxygen migration sites (gray boxes) are visible.

Additional information & tips:

- Avoid defining sites that are not required to fully define your model, e.g. unstable sites for MMC simulations, they just slow down your model
- Use the space group with the highest symmetry that matches your problem. Deliberately using lower symmetry will just slow down your model and increase the input effort

4.3.7 Transition model

The transition model control "Project→Transition Model Control" is used to define event processes and bind them to geometries to create KMC and MMC events. Transition defined this way can be attached and quantified for each simulation definition later and thus an arbitrary number of unused

ones can be defined without compromising simulation efficiency. The "State Changes & Transition Abstraction" sub tab gives access to geometry-independent definition of state change processes through the following set of model objects:

I. **State Change:** Defines a state change from one particle to another
   a. ***Name:*** The UI display name
   b. ***Acceptor State:*** Select the first particle state (commonly the host/acceptor state)
   c. ***Donor State:*** Select the second particle state (commonly the guest/donor state)

II. **State Change Group:** Defines a group of local state changes sharing common properties
   a. ***Name:*** The UI display name
   b. ***State Changes:*** Add the affiliated state changes by drag & drop

III. **State Change Chain:** Defines a chain of state changes with sequential interdependency
   a. ***Name:*** The UI display name
   b. ***Connection:*** Define the connection pattern between the change groups by a N-1 sequence of the words "Dynamic" (interdependent) or "Static" (not dependent).
   c. ***Association:*** On/Off switch for creating association/dissociation like behavior over directional movement. This flag has no effect unless the process describes a proper vehicle mechanism
   d. ***Change Groups:*** Add the affiliated state change groups of your process chain in the correct order by drag & drop

Note: In many cases, each *state change group* will contain just one *state change.* It is a convenience feature intendent for processing similar ions within a single process chain, e.g. all the halide ions.

Important: The set of valid connection patterns is limited by existing, physically meaningful transport mechanisms. Due to the limited number of useful options, manual connection pattern definition has been replaced by a drop-down menu for mechanism selection in newer versions of MOCASSIN.

Useful pattern strings, affiliated position stability requirements for the binding geometry, and modelled mechanisms are defined below and can be selected using their affiliated name:

a. "Dynamic" (**2-Site Metropolis**)
   ➔ (Stable-Stable)
   ➔ MMC exchange
b. "Dynamic Dynamic" (**3-Site Migration**)
   ➔ (Stable-Unstable-Stable)
   ➔ three position KMC, e.g. vacancy mechanisms
c. "Dynamic Dynamic Dynamic Dynamic" (**5-Site Intersticialcy-Like**)
   ➔ (Stable-Unstable-Stable-Unstable-Stable)
   ➔ interstitialcy mechanism
d. "Dynamic Dynamic Dynamic Dynamic Dynamic Dynamic" (**7-Site Intersticialcy-Like**)
   ➔ (Stable-Unstable-Stable-Unstable-Stable-Unstable-Stable)
   ➔ Extended interstitialcy mechanism
e. "Dynamic Dynamic Static Dynamic Dynamic" (**6-Site 2-Species Vehicle**)
   ➔ (Stable-Unstable-Stable-Stable-Unstable-Stable)
   ➔ vehicle mechanism or association/dissociation with distinct unstable transition sites
f. "Dynamic Static Static Dynamic" (**5-Site 2-Species Vehicle**)
   ➔ (Stable-Stable-Unstable-Stable-Stable)
   ➔ vehicle mechanism or association/dissociation with shared unstable transition site

After definition of the geometry independent process information, the second sub tab "Metropolis & Kinetic Transitions" allows to defined geometry bindings for the processes. This is done through the following model objects:

I. **MMC Transition:** Defines an MMC event type by process chain and geometry
   a. ***Name:*** The UI display name
   b. ***State Change Chain:*** Select the process chain you want to use in the event

c. ***Reference Pos.* 1**: Select the first reference position
d. ***Reference Pos.* 2**: Select the second reference position

II. **KMC Transition:** Defines a KMC event type by process chain and geometry
a. ***Name:*** The UI display name
b. ***State Change Chain:*** Select the process chain for the event
c. ***Binding Geometry:*** Define a reference sequence of points that describes the geometry of the event

Example Part I: Define the base process items for oxygen vacancy migration in ceria

i. Open a new "Project→Transition Model Control" and target your model
ii. Create two *state changes* to describe the change on stable and unstable sites:
a. Acceptor: *OxygenVacancy*, Donor: *Oxygen* ➔ Stable site change pair
b. Acceptor: *Void*, Donor: *Oxygen* ➔ Unstable site change pair
iii. Create two *state change group* entries, one for each of your created *state change* entries and link the affiliated *state change* entries by drag & drop
iv. Create one *state change chain* entry that describes the geometry independent process:
a. Select "3-Site Migration" as mechanism
b. Leave the association flag as false
c. Drag & drop the process *state change group* entries into the filed in the correct order: (VcO,O) ➔ (Void, O) ➔ (VcO, O)
v. Switch to the "Metropolis & Kinetic Transitions" sub tab
vi. Create one *Kinetic Transition* entry to model the 1NN migration process
a. Select your *state change chain* from the selection box
b. Enter points (0.25 0.25 0.25), (0.50, 0.25 0.25), (0.75 0.25 0.25) as geometry

Verify that your migration input is what you expect by using the 3D viewer. When setting symmetry extension to "Full" mode your unit cell should look like Figure 10. Additionally, the validation entry mouse over popup for your transition should contain an info that 1 rule was added to the model.

Example Part II: Add the base process items for cerium & oxygen site MMC in ceria

i. Open a "Project→Transition Model Control" and target your model
ii. Create an additional *state change* to describe the change on stable cerium site:
a. Acceptor: *Yttrium*, Donor: *Cerium* (or inverted)
iii. Create another *state change group* entry and link the affiliated *state change* entry
iv. Create two *state change chain* entries that describes the exchange process:
a. Select "2-Site Metropolis" as the mechanism (default selection)
b. Leave the association flag as false
c. Drop the *state change group* sequence (Vc,O)➔(Vc,O) for the oxygen MMC
d. Drop the *state change group* sequence (Y,Ce)➔(Y,Ce) for the cerium MMC
v. Switch to the "Metropolis & Kinetic Transitions" sub tab
vi. Add two *Metropolis Transition* entries to model the site exchanges
a. Set both reference positions to O-Site or Ce-Site
b. Select the affiliated *state change chain* from the drop-down menu

Verify that your entire MMC and KMC transition set is defined. It should look like shown in Figure 10 and Figure 11.

Additional information & tips:

- KMC event paths can only define 2-8 positions, single position state changes are not supported
- KMC event paths containing a ring definition are not supported

- KMC event paths may be chiral to each other, allowing right and left orientation to be modelled independently if both are defined by the user. In most cases, it is highly recommended to define just one orientation. MOCASSIN will auto-generate the missing orientation using the same settings, effectively removing the chirality.

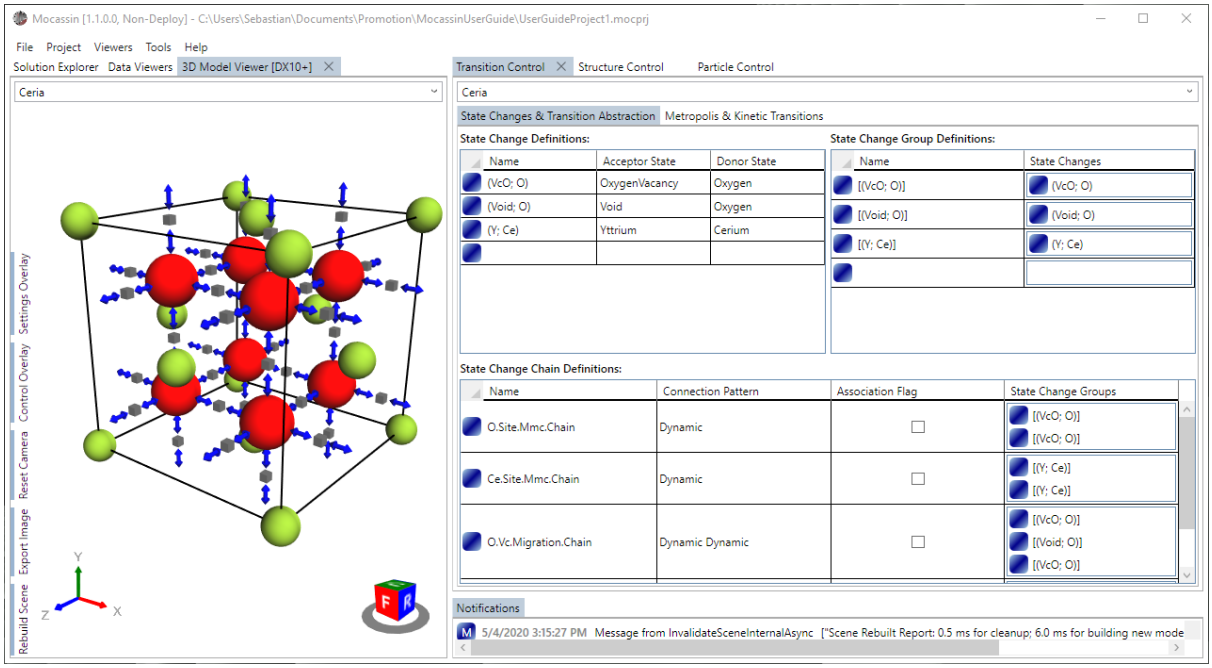

Figure 10: The MMC & KMC process chain definitions in ceria. The resulting cell with the 1NN vacancy migration of oxygen (left) with symmetry extension set to "Full" mode. MMC transition visualization is not supported by the viewer.

4.3.8 Energy model

The energy model control "Project→Energy Model Control" provides controls input related to defining boundary conditions for *energy parameterization templates* that are later generated and used to

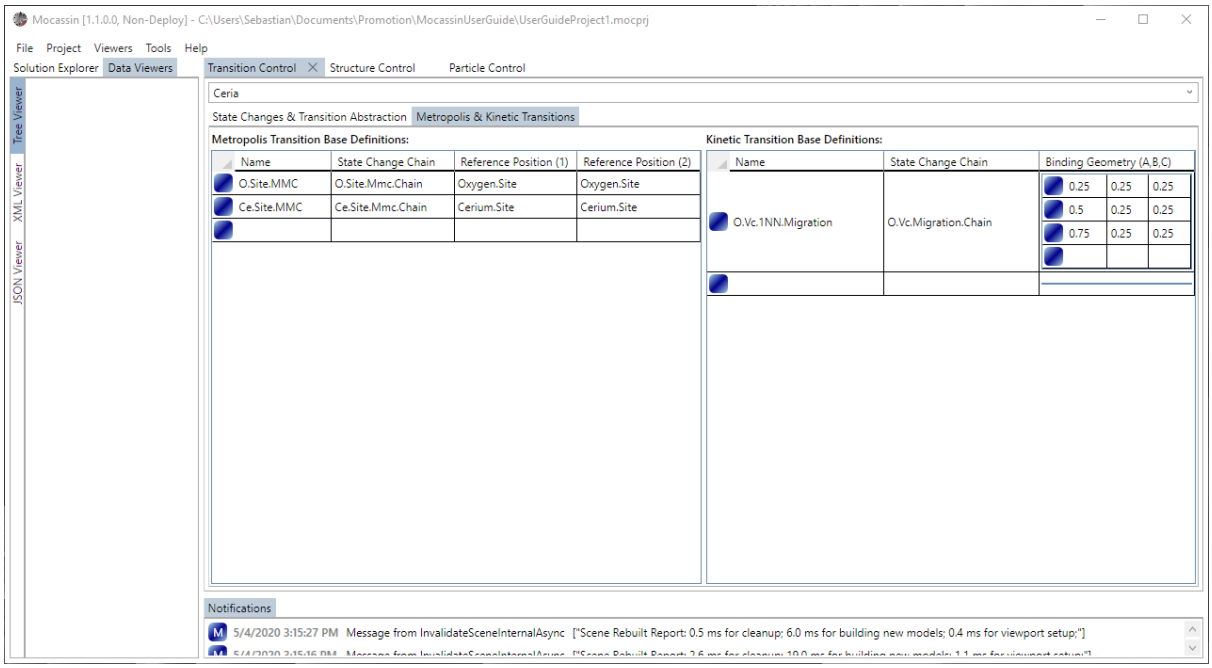

Figure 11: The MMC & KMC transition definition by binding the process chains to geometries.

quantify your interaction model. The following model options & objects are available in the "Environment Ranges & Filters" sub-tab:

I. **Stable Cutoff Radius:** The range cutoff for interactions between stable sites
II. **Stable Interaction Filter:** Define hollow sphere filters around stable position
    a. ***Name:*** The UI display name
    b. ***Min. Radius***: The start radius of the filter in [Å]
    c. ***Max. Radius:*** The end radius of the filter in [Å]
    d. ***Center Site:*** The reference site around which to place the filter
    e. ***Partner Site:*** The interacting partner lattice site that is filtered out
III. **Unstable Environment:** Environments around unstable sites (Provided by MOCASSIN)
    a. ***Name:*** The UI display name
    b. ***Center:*** The center site, auto set by MOCASSIN
    c. ***Cutoff Radius:*** The cutoff radius for interactions around the unstable site
    d. ***Interaction Filters:*** ➔ analogue to stable case, fixed center site

Based on these settings, MOCASSIN performs a radial search and reduction routine to generate the required list of pair interactions.

Note: It is irrelevant if center/partner site on stable interaction filters are inverted. For consistency reasons, they will be applied to both partners.

Example Part I: Define the environment settings for ceria

i. Open a new "Project→Energy Model Control" and target your model
ii. Set the stable cutoff radius to 6.5
iii. Set the cutoff of the environment generated for the oxygen migration site to 3
iv. Select the oxygen migration site environment and create an interaction filter in the data grid below the selection grid to abide by the above rule:
    a. Set the min. range to 0 and the max range to 3
    b. Select the *partner reference site* to be the oxygen site

The settings should look as shown in Figure 12, with validation reports notifying about 11 stable interactions in the system and 1 asymmetric interaction for the oxygen migration site.

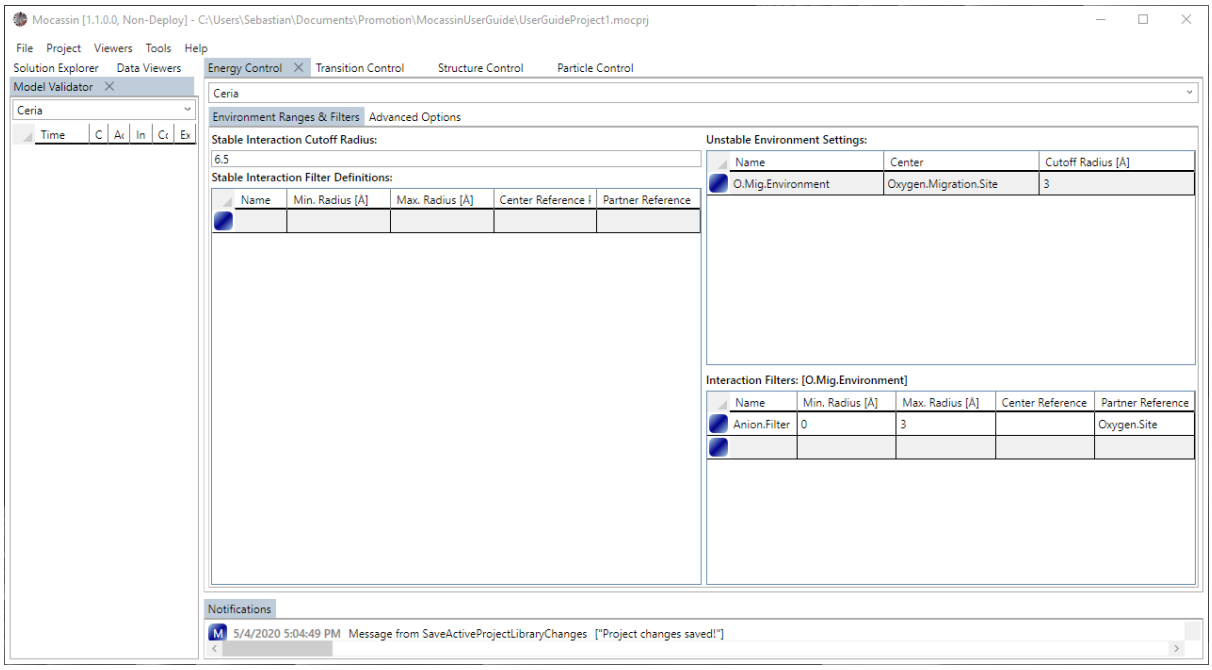

Figure 12: The energy control "Environment Ranges & Filters" tab with the settings for ceria.

Important: Unstable sites are only allowed to have interactions with stable sites they do not share any KMC event with. Defining non-zero interactions between unstable and stable sites that are directly related during any KMC event is a silent error, yielding a meaningless simulation. They can be either filtered out, or the affiliated interactions must be set to zero (default) later. In advanced cases, this rule can be circumvented if two distinct KMC events share the same transition site with distinct mobile particle states. This allows to model each interaction in such a way that it has non-zero interactions with the sites affiliated with the currently inactive event only.

The second sub-tab of the energy control "Advanced Options" gives access to local defect site energies and definition of custom interaction cluster geometries. The defect energy data grid is auto generated and contains an entry for every site/particle permutation that exists in the structure model. The following settings and model objects can be defined:

I. **Interaction Group:** Defines a cluster interaction by center site and surroundings
   a. ***Name:*** The UI display name
   b. ***Center Site:*** Select the center site that serves as the cluster origin point
   c. ***Surroundings:*** Define the absolute vectors of positions that belonging to the cluster

II. **Position Defect Energy:** Sets local defect energies for each site/particle permutation
   a. ***Particle:*** The affiliated particle (read only)
   b. ***Cell Site:*** The affiliated site (rad only)
   c. ***Energy:*** The affiliated local energy in [eV]

Example Part II: Define the migration edge cluster for ceria

i. Open a new "Project→Energy Model Control" and target your model
ii. Select the "Advanced Options" sub-tab
iii. Create a new *interaction group* entry that includes the 1NN cation positions around the oxygen migration site to create a "migration edge":
   a. Select the oxygen migration site as center position
   b. Enter (0.5 0.5 0.0) ➔(0.5 0.0 0.5) as surrounding geometry if you chose (½ ¼ ¼) as unstable reference site

Pair interactions and clusters can be visualized in the 3D viewer, this however requires creation of parameterization templates as explained later.

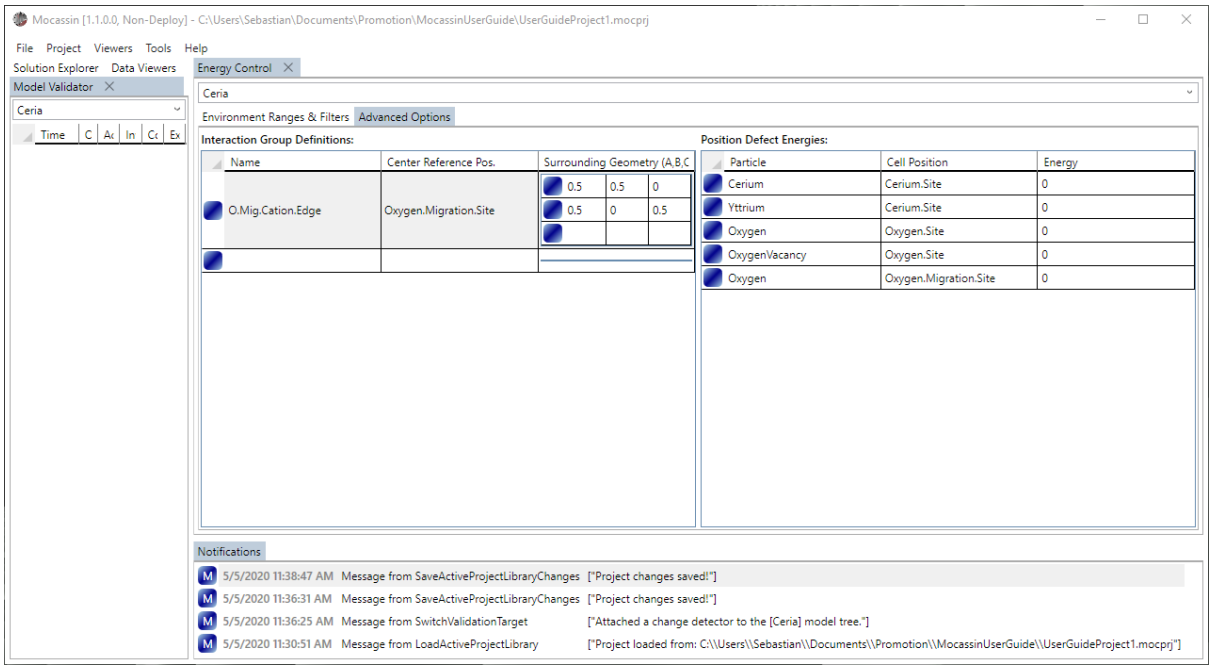

Figure 13: The energy control "Advanced Options" tab with an added ceria migration edge cluster around the unstable site.

Note: Interaction groups are defined as geometric paths with the originally entered coordinates of the center site as the first position. All "surrounding" positions must have a defined (non-filtered) pair interaction with the center to be accepted.

Important: Clusters are processed and permuted using symmetry, and will be symmetry extended during the P1 translation, potentially causing partial overlapping. If your cluster does not produce a set of unique self-projections, then the cluster geometry is ambiguous, a red flag for an inconsistent model definition.

Additional information & tips:

- Some KMC programs favor direct modelling of transitions over modelling of the affiliated states, this approach is not supported by MOCASSIN
- Clusters around stable sites are problematic for MMC/KMC simulations with multiple active events. There is no mechanism in MOCASSIN to ensure that each cluster is also properly defined for each member of the surrounding geometry.
- An A-B pair interaction is treated as B-A equivalent in all cases to ensure consistency. This currently causes the limitation that A-B and B-A must have identical energy, even if the interacting sites are not identical, making A-B and B-A distinct interactions
- The validation system might report detection of "chiral pairs" reducing the number of unique pair interactions. This is caused by the fact that pair interactions contain a direction information and their left/right orientation is differentiated by the space group symmetry. They cannot be modeled independently due to the A-B to B-A consistency rule

4.3.9 Simulation model

The simulation model control can be access by "Project→Simulation Model Control" and is used to define KMC/MMC simulation blueprints with a set of fallback values serving as defaults. These blueprints are used to define which previously created MMC or KMC transitions belong to a single type of simulation. When a simulation database is built later, one such blueprint must be selected for each simulation. This way it is very simple to test the effects of enabling or disabling certain events in a simulation. The following model objects and specific settings can be manipulated:

I. **Metropolis Simulation Base:** Defines a metropolis simulation blueprint
   a. ***Name:*** The UI display name
   b. ***Transitions:*** Assign/Select the allowed MMC transitions
   c. ***Break Check Sample:*** Number of simulation cycles used for averaging the lattice energy when checking for energy fluctuation break conditions
   d. ***Relative Break Tolerance:*** Relative tolerance of energy fluctuation between two checks that causes an automated break for the simulation (Set to 0 to disable)

II. **Kinetic Simulation Base:** Defines a kinetic simulation blueprint
   a. ***Name:*** The UI display name
   b. ***Transitions:*** Assign/Select the allowed KMC transitions
   c. ***Relaxation MCSP:*** The number of MCSP performed before the main run
   d. ***Normalization* Probability**: Set the upper acceptance limit in range (0…1], values below one deliberately over-normalize the system
   e. ***Electric Field Vector:*** Define the direction of the electric field as a vector in unit cell coordinates
   f. ***Electric Field Modulus:*** Define the electric field strength in [V/m]

Note: It is usually not required to manipulate the automatically set general fallback values in any way. The relevant entries (MCSP, temperature, …) are later quantified for each job in the simulation build templates. Due to data processing, the electric field direction vector for KMC cannot be changed in these templates. It is currently a constant for each simulation definition and multiple simulation blueprints are thus required to model different field orientations.

Example Part I: Define a simulation base of the oxygen migration

i. Open a new "Project→Simulation Model Control" and target your model
ii. Select the "Kinetic Simulations" sub-tab
iii. Create a new *simulation base* entry for your oxygen migration:
    a. Select your previously created oxygen migration from the drop-down menu in the first transition entry
    b. Select the new transition and set a field direction (optional, defaults to (1 0 0))

Your data should look like shown in Figure 14. Depending on your version, the validation system might report several "value over warning limit" issues for the simulation, related to no longer required or outdated validation rule settings, that can be safely ignored.

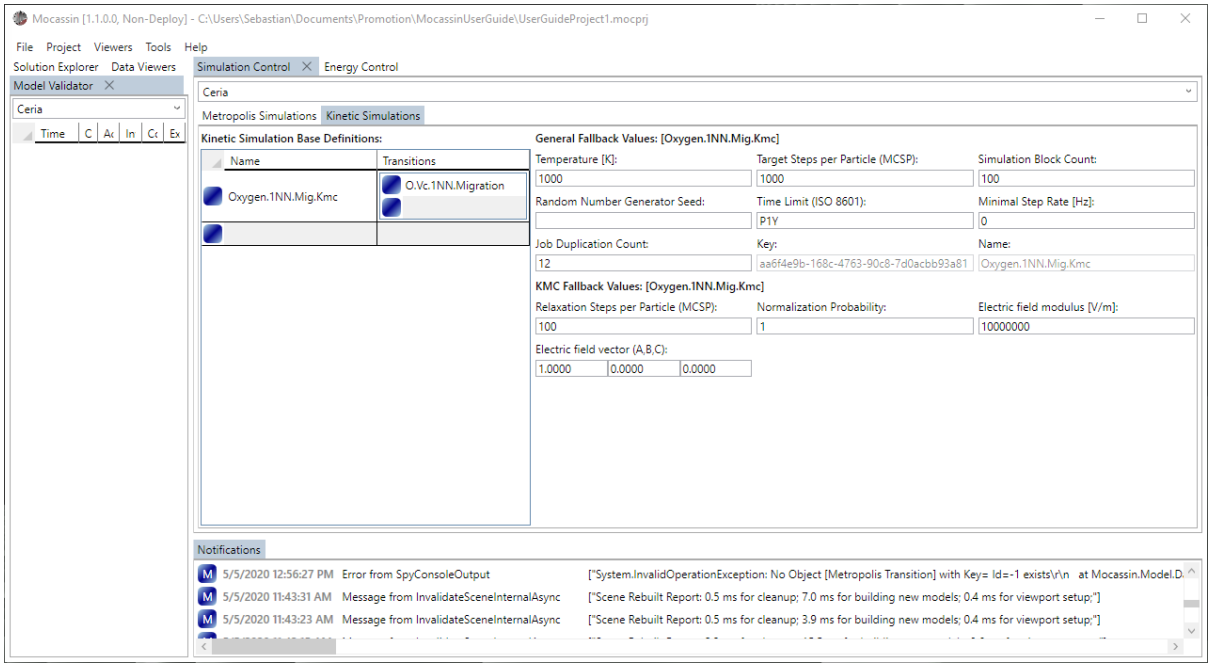

Figure 14: The "Kinetic Simulations" tab of the simulation control showing a KMC base simulation with one migration event.

Example Part II: Define all possible MMC simulation bases for ceria

i. Open a new "Project→Simulation Model Control" and target your model
ii. Select the "Metropolis Simulations" sub-tab
iii. Create three *simulation base* entry for all possible options:
   a. Oxygen lattice MMC only by adding the oxygen MMC transition
   b. Cerium lattice MMC only by adding the cerium MMC transition
   c. Full MMC by adding both entries in the transition data grid

Your data should look like shown in Figure 15, again several ignorable warnings may occur.

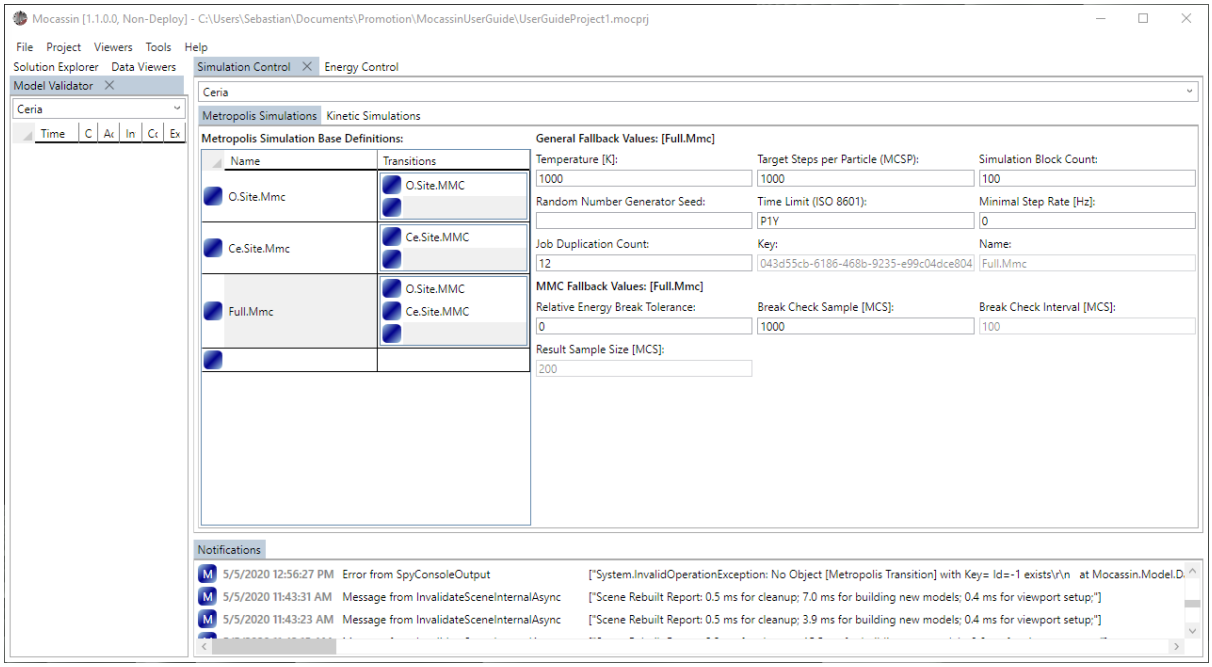

Figure 15: The "Metropolis Simulations" tab of the simulation control showing all possible MMC simulation base simulations for ceria.

Additional information & tips:

- Due to experience, it is not recommended to use the MMC break tolerance system. It is difficult to find pairs of tolerance and test sample length that yield useful simulation breaks.

4.3.10 Lattice model

Lattice/Supercell generation for simulation uses uniform random placement based on doping definitions. The lattice model accessed by "Project→Lattice Model Control" provides the controls for defining the existing doping types, which are quantified later. The following model objects and settings are supported by the "Building Blocks" sub tab:

I. **Building Block:** Defines a base unit cell (currently fixed default occupations)
   a. ***Name:*** The UI display name
   b. ***Occupation:*** Select the default occupation of each site (currently disabled)

The "Block Doping" sub tab additionally provides the controls required for creating doping schemes:

I. **Occupation Exchange:** Defines a change of site occupation
   a. ***Name:*** The UI display name
   b. ***Cell Position:*** Select the site on which the change occurs
   c. ***Old Particle:*** Select the original particle
   d. ***New Particle:*** Select the replacement particle
II. **Doping:** Defines a doping scheme that supports auto charge balancing
   a. ***Name:*** The UI display name
   b. ***Auto Charge Balance:*** On/Off switch for automatic charge compensation
   c. ***Application Order:*** Set an index for application order (ascending)
   d. ***Primary Exchange:*** The primary exchange
   e. ***Dependent Exchange:*** The dependent exchange that is used for charge balancing
   f. ***Building Block Binding:*** Assign a building block (currently default only)

Note: The building block system is currently limited to one default building block that uses the first entries of each particle set on each site as the default occupation. Any doping is applied to a supercell build from this default unit cell. Thus, the GUI can currently create supercells that can be described as a doped host matrix only.

Example: Define the yttrium doping logic for the ceria host matrix

i. Open a new "Project→Lattice Model Control" and target your model
ii. Select the "Building Blocks" sub-tab and check if the default block has the correct occupation (if not you have to change the order of particles in the affiliated *particle set*)
iii. Switch to the "Block Doping" sub-tab
iv. Create two *occupation exchange* items to describe the yttrium doping and the corresponding creation of oxygen vacancies:
   a. The first affects the cerium site, with cerium as the old particle and yttrium as the new particle
   b. The second affects the oxygen site, with oxygen as old particle and oxygen vacancy as new particle
v. Create one *doping* item to model the solving of yttria in the ceria host matrix:
   a. Activate auto charge balancing
   b. Select the cerium to yttrium exchange as the primary exchange
   c. Select the oxygen to oxygen vacancy exchange as the dependent exchange
   d. Select the default building block if not already selected

Your data should look like shown in Figure 16 and can be quantified for each simulation later.

Important: Activating automatic charge balancing enforces proper balancing in all cases by rounding a doping fraction to the closest value less or equal to the input that yields integer population counts for the requested supercell size. As in any finite lattice simulation, exact doping is only possible if the doping can be written as an integer fraction for the specific supercell size.

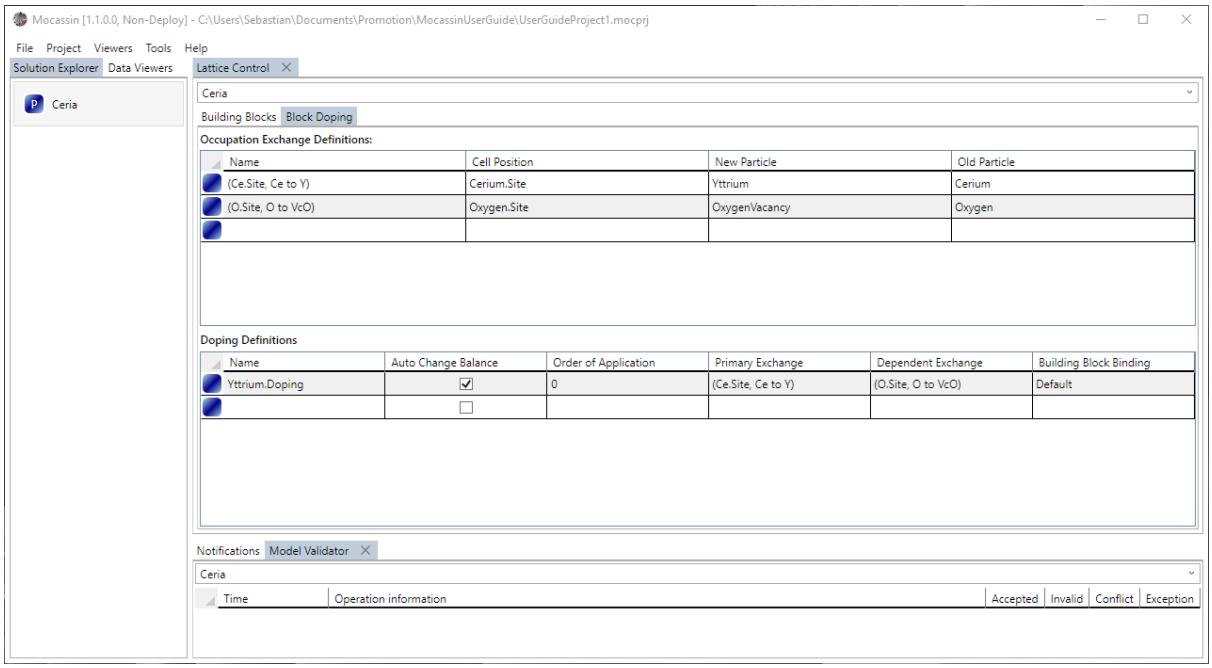

Figure 16: The "Block Doping" tab of the lattice control showing the model items to describe yttria dissolution into a ceria host matrix..

Additional information & tips:

- The order of application can be used to create consecutive doping logic, e.g. creation of oxygen vacancies by reduction and then repopulation by hydration

4.4 3D Model Viewer

[There is currently no explanation for this UI element]

4.5 Simulation Building

4.5.1 Parameterization with customization/parameterization templates

Interaction energies and KMC attempt frequencies are parameters that may change in a model or vary depending on the calculation method (DFT, pair potentials, …). Thus, MOCASSIN uses your model definitions to create a *parameterization or customization template* that contains all settable values and permutations. These are accessed, created, and deleted, through the "Project→Customization Template Control" tab. This control has two selection boxes at the top, the first selects the target model as before, the second selects the target template. Right click into the drop-down popup or the arrow button of the template selection box brings up the context menu with the following actions:

I. **New Template:** Creates a new template based on the targeted model
II. **Migrate Selected:** Creates a new template based on the current state of the target model and tries to salvage data from the original template where possible (Generates report)
III. **Duplicate Selected:** Creates a plain deep copy of the current template without ensuring that contained components remain valid in the model (Deprecated)
IV. **Delete Selected:** Deletes the currently selected template

Example I: Create a parameterization template for the ceria model

i. Open a new "Project→Customization Template Control" and target your model
ii. Open the drop-down menu for customizations right next to the model target box
iii. Right click into the drop-down popup and select "New Template" from the context menu
iv. If your model is valid a new customization will be added. If validation errors exist, an error box will be displayed instead

After creation, templates are listed in the solution explorer tab and can be renamed there. Some contents can be visualized with the 3D viewer by selecting the created template as the content source. The customization template control is divided into four sub-tabs that allow access to the following parameters of your model:

I. **KMC Transition Rule Set:** Describes the different states of a KMC event
   a. ***Name:*** The UI display name (has an auto generated default)
   b. ***Source Transition:*** The affiliated transition definition (read only)
   c. ***Set Content:*** Gives access to the unique rules of the set
      i. ***Name:*** Name the rule (has an auto generated default)
      ii. ***Hidden Count:*** Informs how many hidden/equivalent rules exist (read only)
      iii. ***Frequency:*** Set the attempt frequency for the transition rule
II. ***Stable Pair Interaction Set:*** The symmetry reduced set of stable-stable pair interactions
   a. ***Name:*** The UI display name (has auto generated default)
   b. ***Content Count:*** The number of unique permutations in the set (read only)
   c. ***Center Cell Position:*** The first interacting site (read only)
   d. ***Partner Cell Position:*** The second interacting site (read only)
   e. ***Distance:*** The distance between the sites in [Å]
   f. ***Chiral Partner:*** Defines the chiral partner if it exists (read only)
   g. ***Set Content:*** Gives access to the unique permutations of the interaction
      i. ***Name:*** The UI display name (auto generated, read only)
      ii. ***Center:*** The particle occupying the first site (read only)
      iii. ***Partner:*** The particle occupying the second site (read only)
      iv. ***Energy:*** The energy of the interaction permutation in [eV]
III. ***Unstable Pair Interaction Set:*** Analogue to the stable case for unstable centers
   a. ***… (see stable set)***
IV. ***Group Interaction Set:*** The symmetry reduced set of defined cluster interactions
   a. ***Name:*** The UI display name (auto generated default)
   b. ***Content Count:*** The number of unique permutations in the set (read only)
   c. ***Source Interaction:*** Defines which group definition the set belongs to (read only)
   d. ***Center Cell Position:*** Defines the origin/center site of the group (read only)
   e. ***Set Content:*** Gives access to the unique permutations of the interaction
      i. ***Name:*** The UI display name (auto generated, read only)
      ii. ***Center Particle:*** The particle occupying the center/origin site (read only)
      iii. ***Energy:*** The energy of the interaction permutation in [eV]

Note: The contents of parameterization templates are generated by MOCASSIN. It is not possible to manually add or remove entries. If a model is subjected to breaking changes, e.g. addition of a new cell position, existing templates must be migrated before further usage by using the affiliated context menu item.

Additional information & tips:

- Leaving interaction sets at zero or another constant value for all included permutations causes them to be "optimized away" by the solver. Thus, having a few "dead" interactions has little to no effect on the solver performance
- It can usually be avoided to model interactions between default occupations of the host lattice, e.g. oxygen and cerium in ceria
- If you want to model that two ions should not exist in a certain distance during MMC, just assign huge values to the affiliated pairs interactions, e.g. 10 eV. This requires at least one of the affiliated occupations to be mobile on the affiliated site.

4.5.2 Simulation definition with job templates

Simulations are generated by a translation system that converts model and parameterization into computable sets of symmetry extended data for the solver. Usually, many simulations with the same model and parameterization are used to create parameter sweeps with statistical error estimates. Therefore, MOCASSIN is designed to create simulations in huge sets and uses SQLite databases as containers to reduce the file clutter often affiliated with these types of simulations. The blueprint for job sets is called a *job template* and is accessed through "Project→Job Template Control". As with the parameterization template, a second drop-down menu exists at the top of the control, and the context menu provides the following actions:

I. **New Template:** Creates a new template for the targeted model
II. **Duplicate Selected:** Created a deep copy of the current template
III. **Delete Selected:** Deletes the currently selected template

After creation, the *job templates* are listed in the solution explorer, below the *customization templates* of the affiliated model. The general template layout for both KMC and MMC is:

I. **Job Collection:** Define a set of jobs with shared settings
   a. ***Name:*** The identification & UI display name
   b. ***RNG Seed*****:** Seed for the random number generator used for collection building

Example II: Set the values for a basic 1NN interaction model in ceria

i. Open a "Project→Customization Template Control" and target your model & template
ii. Select the "Transition Rules" sub-tab and select the only existing *KMC rule set*:
   a. Select "Rule.0" from the contents and set the attempt frequency to 10^{13} Hz
iii. Switch to the "Pair Interactions (Stable)" sub-tab:
   a. Select the cerium site to oxygen site interaction with the shortest range and set the yttrium to vacancy interaction to "-0.27" eV
   b. Select the oxygen site to oxygen site interaction with the shortest range and set the vacancy to vacancy interaction to "0.84" eV
iv. Switch to the "Group Interactions (All)" sub-tab and select the defined migration edge cluster:
   a. Set the permutation with two cerium ions to "0.52" eV
   b. Set the permutation with one cerium and one yttrium ion to "0.57" eV
   c. Set the permutation with two yttrium ions to "0.82" eV

The values for your cluster should look like shown in Figure 17.

c. ***Job Multiplier:*** Defines how many equivalent copies of each contained job are created for statistical evaluation. (Randomized data is recreated, not copied)
d. ***Simulation Base:*** Select the parent *simulation base* object from the model
e. ***Settings:*** Define the KMC/MMC job settings of the job collection
f. ***Jobs:*** The set of job config items, each defining a simulation setting
    i. ***Settings:*** Define the KMC/MMC job settings of the job
g. ***Selection Optimizer:*** Exclude site/particle pairs from event selection (ADVANCED!)

The settings on the template are structured in a hierarchy, following the principles "job overwrites job collection" and "job collection overwrites simulation base". Additionally, both *job* and *job collection* data grid provide duplication functions to define huge sets of simulations quickly.

A template can contain both MMC and KMC collections which differ in the possible settings. The following settings exist for jobs:

I. **General Settings:** Exist for KMC & MMC
    a. ***Temperature:*** Set the simulation temperature in [K]
    b. ***Time limit:*** Limit the max. runtime for the simulation as [dd.hh:mm:ss]
    c. ***MCSP:*** The minimal successful step count that must be reached for the simulation to terminate automatically as [MCSP]
    d. ***Min. Success Rate:*** Defines the minimal average MCS/s rate that a simulation must reach in [Hz].
    e. ***Execution Flags:*** Allows to set multiple execution flags that perform on/off switching of certain features
        i. ***NoSaving:*** Prevents the solver from generating checkpoint files
        ii. ***NoJumpLogging:*** Prevents expensive KMC jump histogram recoding
        iii. ***UseFastExp:*** Fast exponential approximation by N. Shraudolph
    f. ***Instruction String:*** Attaches a custom routine instruction string (ADVANCED!)
    g. ***Doping Values:*** Defines the **relative** fractions of replacement [0…1) according of the previously defined doping definitions
II. **MMC Settings:** MMC exclusive
    a. ***Break Tolerance:*** Define a relative energy fluctuation break tolerance
    b. ***Break Sample Size:*** Define the sample size for break tolerance checking
III. **KMC Settings:** KMC exclusive

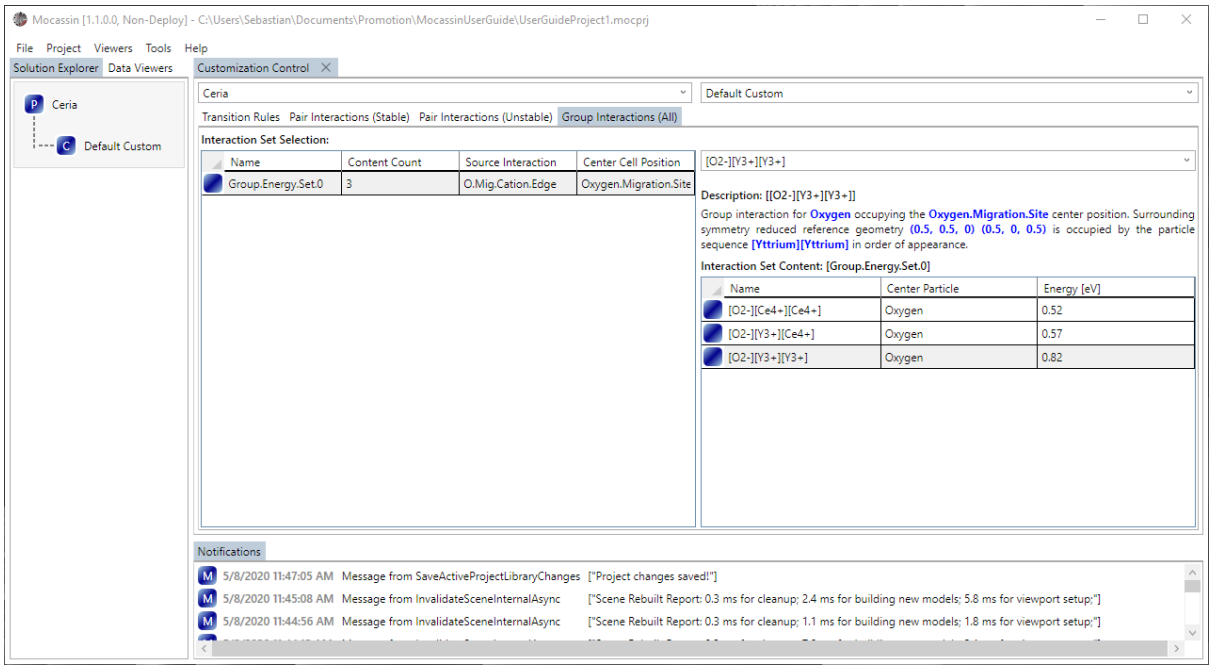

Figure 17: The "Group Interactions" sub-tab of the customization template control with the ceria migration edge.

a. ***Relaxation MCSP:*** The minimal successful steps that are done for normalization and relaxation before the main run in [MCSP]
b. ***Norm. Probability***: Define a new upper acceptance limit in range (0…1) that is used to normalize the simulation
c. ***Electric Field Modulus***: Set the magnitude of the electric field in [V/m]
d. ***Max Frequency***: Renormalizes the highest attempt frequency to a new value and corrects all other frequencies accordingly

Important: Using the "NoSaving" flag prevents simulation state files from being written to disk. These files are required for simulation restart and in-depth evaluation with the C# API.

Important: Selection optimization is a process by which rejection of KMC/MMC events due to site blocking can be reduced by removing redundant event origin points from the process. E.g. in a vacancy diffusion case, selecting only the minority species (vacancies) is more efficient. Currently, MOCASSIN has no mechanism to automate creation and/or check consistency of these optimizations. Thus, proper selection optimization is up to the user or must be left empty.

Important: Doping values are relative to each other and relative to the sublattices they are applied to. I.e. if a first doping replaces 50% of A on a sublattice by B and the second doping replaces 50% of B on that sublattice by C, than the system has 50% A, 25% B and 25% C.

Example II: Create a new job template for the ceria model

i. Open a new "Project→Job Template Control" and target your model
ii. Open the drop-down menu for customizations right next to the model target box
iii. Right click into the drop-down popup and select "New Template" from the context menu
iv. If your model is valid, a new job template will be added

Example II: Define a simple MMC simulation job for oxygen sites in ceria

i. Open a "Project→Job Template Control" and target your model and template
ii. Select the "Metropolis Job Packages" sub-tab
iii. Add a new *job collection* entry in the affiliated data grid:
    a. Set the job multiplier to 1
    b. Set the simulation base to your previously defined oxygen site mmc simulation
    c. In the *job collection* settings set the temperature to 1000K and the steps to 200 MCSP
iv. Add a new *job config* entry in the affiliated data-grid:
    a. Set a cell size (defaults to 10x10x10)
    b. Set the previously defined yttrium doping to 0.05
v. For your *job collection* add an entry to the selection optimizer data grid that defines the oxygen particle to be removed as selectable from the oxygen site (optional)

Your control should look like shown in Figure 18. The selection optimization entry causes the solver to only use the oxygen vacancies as event source. If the vacancy fraction in the lattice is low, this greatly decreases the chance of an MMC event failing due to same occupation on both sites.

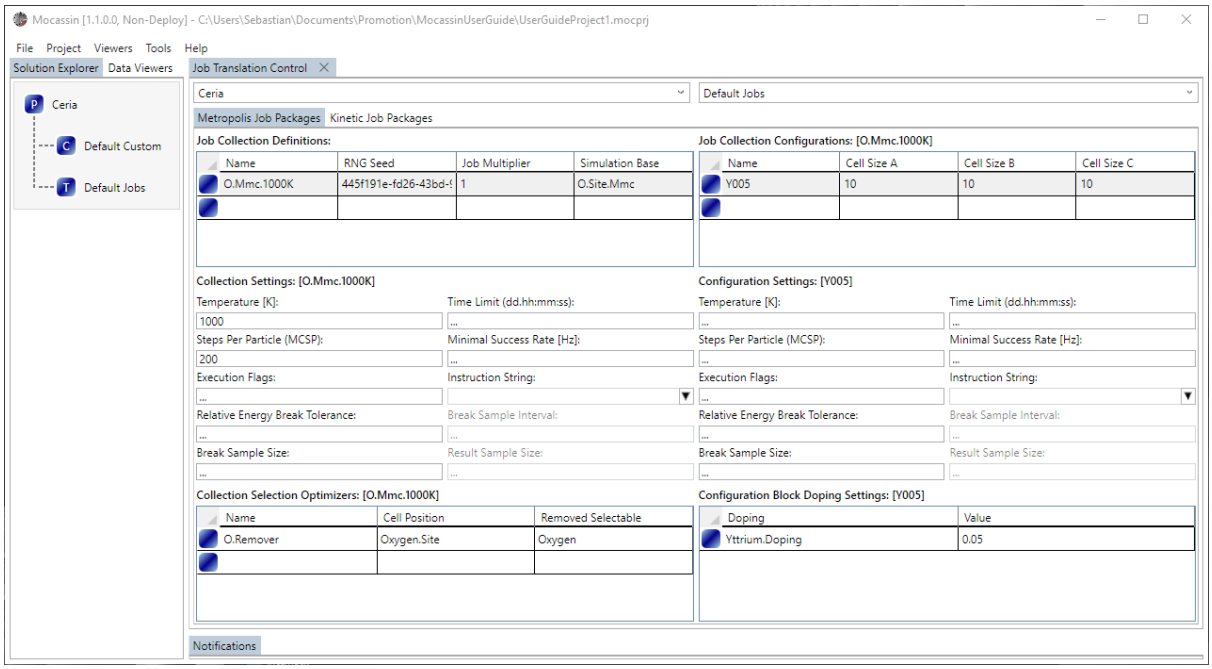

Figure 18: The "MMC Job Packages" sub-tab of the job template control with a simple MMC simulation definition.

Example III: Define a simple KMC simulation job for oxygen sites in ceria

vi. Open a "Project→Job Template Control" and target your model and template
vii. Select the "Kinetic Job Packages" sub-tab
viii. Add a new *job collection* entry in the affiliated data grid:
   a. Set the job multiplier to 1
   b. Set the simulation base to your previously defined oxygen vacancy KMC simulation
   c. In the *job collection* settings set the temperature to 1000K, the steps to 200 MCSP, and the relaxation steps to 100 MCSP
ix. Add a new *job config* entry in the affiliated data-grid:
   a. Set a cell size (defaults to 10x10x10)
   b. Set the previously defined yttrium doping to 0.05
x. For your *job collection* add an entry to the selection optimizer data grid that defines the oxygen particle to be removed as selectable from the oxygen site (optional)

Your control should look like shown in Figure 19. The selection optimization entry causes the solver to only use the oxygen vacancies as event source. If the vacancy fraction in the lattice is low, this greatly decreases the chance of an KMC event failing due to site blocking.

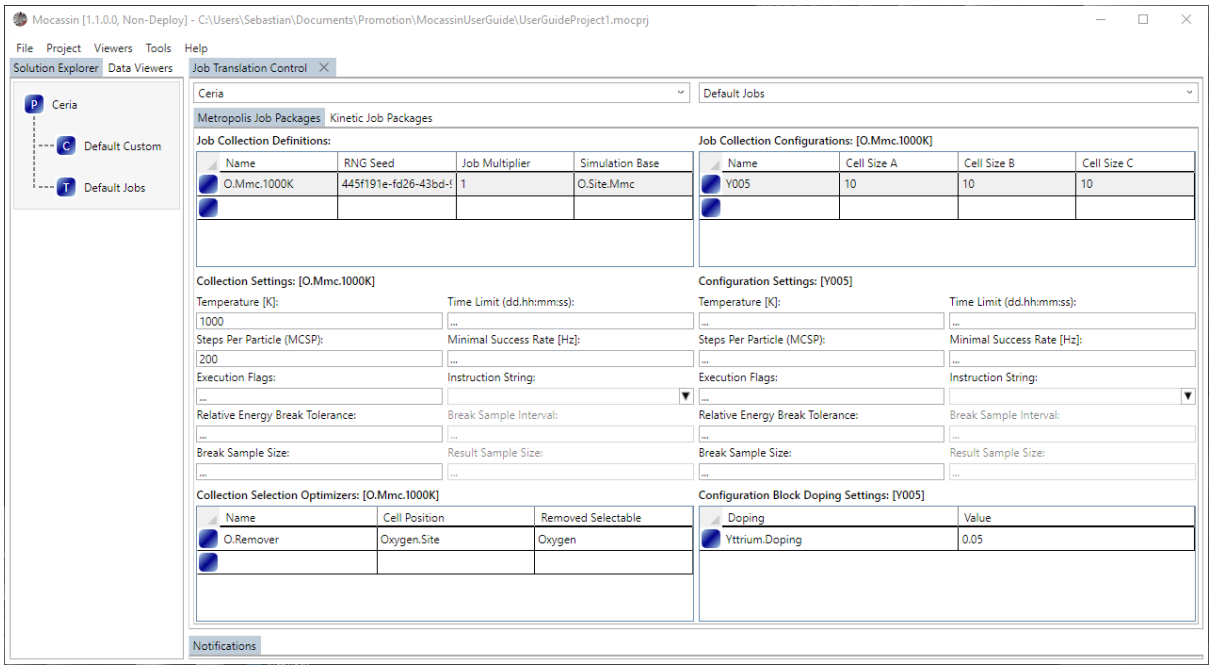

Figure 19: The "KMC Job Packages" sub-tab of the job template control with a simple KMC simulation definition.

Additional information & tips:

- The most common parameter sweep is "doping over temperature" and can be generated efficiently by defining one *job collection* for the lowest temperature with a set of *jobs* that defines the "doping" series. The *job collection* can then be duplicated with the context menu and the temperature can be adjusted for each collection.
- Using fast exponential approximation is usually recommended. The average deviation from normal calculation is around 0.01% (For the 0…1 range required in Monte Carlo) and causes a simulation result bias that is indistinguishable from statistical noise in most cases.
- *Rule of thumb:* A supercell should be at least 2.5x the size of your interaction range in all directions to avoid periodic boundary effects.
- *Rule of thumb:* Simulating a larger amount of medium sized cells from 8x8x8 to 12x12x12 is usually more efficient than simulating a small set of huge cells.

4.5.3 Local deploy of simulation databases

Currently, simulation databases are built and deployed to the localhost, there is no FTP/SFTP support. For this purpose, the simulation builder exists, and a new control tab can be created by the "Tools→Simulation Builder" menu item. The builder translates *build instruction* entries, consisting of model, customization template, and job template, into SQLite job databases. After creation of a database, the control shows the job meta information table within the right data grid. The table is part of the database and defines the job settings for each job context id to identify the jobs later.

Example I: Deploy your defined ceria simulation set

i. Open a "Tools→Simulation Builder" and target your ceria model
ii. Create a new *build instruction* entry:
    a. Select your *customization template* as customization source
    b. Select your *job template* as job set source
iii. Use the "Select File" button to choose a deploy path
iv. Press the "Start Build" button

Your control should look like shown in Figure 20. If you did not define more jobs than the examples suggested, then the meta data table on the right should show two entries. Both jobs are now contained in the ".msl" file your chose as deploy path.

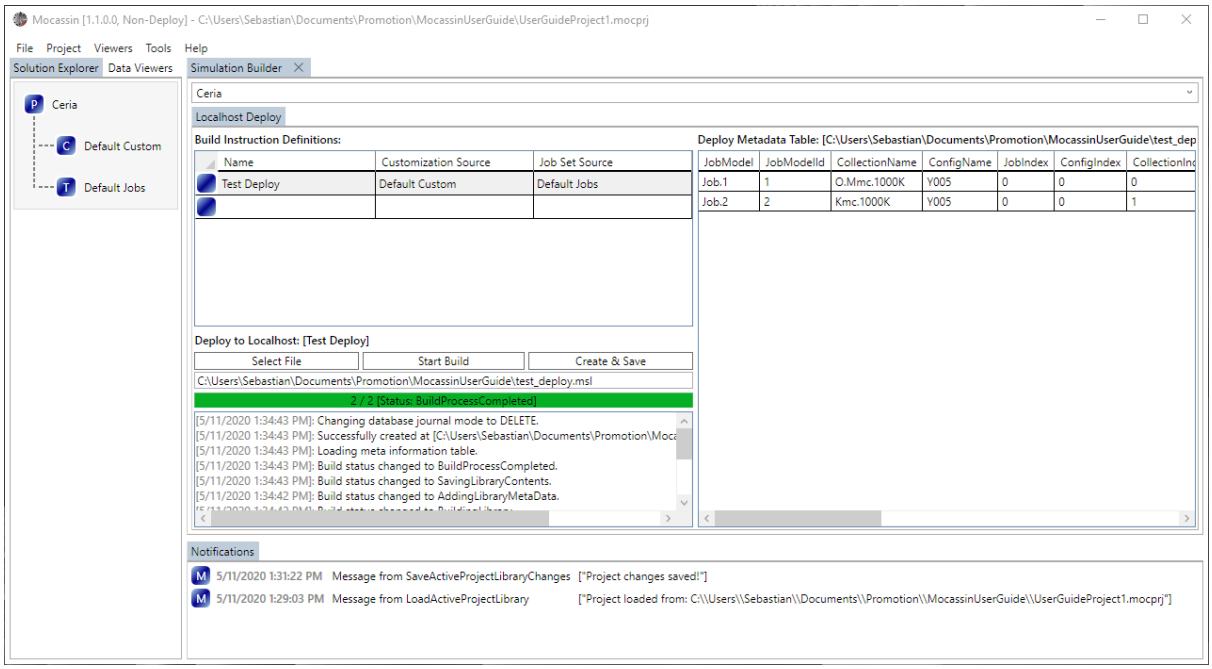

Figure 20: The localhost deploy system after translation of the defined ceria model and test job template.

Note: The build system uses all available CPU cores for lattice creation. This may cause your system to become irresponsive for a few seconds when building large simulation sets.

Important: SQL basics are required to query the meta information from the ".msl" file later. MOCASSIN does not provide the data in any other fashion.

Additional information & tips:

- Deployed simulation files have the file extension ".msl", they are however nothing more than SQLite databases and can be used as such.
- The ".msl" files contain the XML serialized model data and the C# API can rebuild the model for evaluation without requiring the original ".mocprj" file

5 Simulation

5.1 Using the simulator

The MOCASSIN solver executable "Mocassin.Simulator" (Linux) or "Mocassin.Simulator.exe" (Windows) is a single core application that is controlled by command line arguments. The following arguments are mandatory:

I. **-dbPath <file>** Absolute path to the ".msl" file that serves as data source
II. **-jobId <integer>** The id of the job the solver should process
III. **-ioPath <directory>** Path to the folder for data/file write and read operations

Multiple optional set of arguments can be defined:

I. **-stdout <name>** Defines a filename to which the standard output should be redirected. The file is written into the "ioPath" folder
II. **-extDir <directory>** Defines where the system should look for extension routines (advanced)
III. **-outPluginPath <file>** Define a .dll/.so library path from which an alternate output function should be loaded
IV. **-outPluginSymbold <name>** Define the function name which is loaded from the output plugin
V. **-engPluginPath <file>** Define a .dll/.so library path from which an alternate energy calculation function should be loaded
VI. **-engPluginSymbold <name>** Define the function name which is loaded from the energy plugin

Starting a simulation can be simply done by calling the simulator from a command line window. For simulations on Windows machines, MOCASSIN comes with a simple CMD script "StartMocassinSimulator.cmd" that prompts the user for the required parameters. On Linux, just input the following line into a shell after giving the simulator execution rights:

Mocassin.Simulator -dbPath <msl_file> -jobId <integer> -ioPath <directory>

The simulator generates the following files in the defined I/O directory:

I. **"run.mcs":** The main run state file (MMC & KMC)
II. **"prerun.mcs":** The relaxation run state file (KMC only)
III. **"<outname>":** The stdout redirect (if -stdout is set)
IV. **"stderr.log":** The redirected stderr stream

Additionally, a temporary "*.mcs.backup" file is created each time a state file is written to prevent data loss if a simulation is terminated at the wrong moment.

Important: If saving is enabled, the disk space requirement for a simulation is at least 2.5x of a single "*.mcs" file for an MMC simulation, and 3.5x times for a KMC simulation including a relaxation run.

Additional information & tips:

- The MOCASSIN solver performance drops drastically if the data no longer fits into the CPU cache. Keeping the models clean and using small/medium sized lattices can significantly boost performance, especially when running several simulations in parallel where the cache space per simulation is limited.
- The best way to make the simulator available on a host system is to create a command or alias for your preferred shell that targets the simulator executable. This way, the simulator can be started easily from any terminal of the system

5.2 Concurrent execution

In most cases, it is recommended to run MOCASSIN on a computing cluster where jobs can be executed in parallel. As each simulation is an independent sequential process, concurrent execution

is trivially realized through a glue script that targets a parallelism model of your choice. MOCASSIN provides a basic Python3.6 startup wrapper that can handle shared, distributed and hybrid execution cases. Using distributed/hybrid job startup requires an MPI installation and the "mpi4py" package to be installed as described in section 4.

The execution wrapper "mocassin_mt.py" can be used either with or without MPI execution and is configurated with the "moccasin_mt.cfg" file. It is recommended to leave most settings at "auto" and let the script choose the execution method. The config file entry "AutoSearchPath" should be set to target the parent directory of "Mocassin.Simulator". The script is generally called in the following way:

```
$MPI_EXEC $MPI_FLAGS python3 mocassin_mt.py db=<msl_path> jobs=<sequence>
```

The prefixed $MPI_EXEC and $MPI_FLAGS describe the MPI execution command and the affiliated flags, e.g. number of ranks, respectively, and are left empty if MPI is not used. The "db=<msl_path>" describes the absolute path to the "*msl" file and the "jobs=<sequence>" defines which jobs are started. The <sequence> must be comma separated string of job context ids, e.g. "1,3,6". The "auto" execution mode translates your settings into the "logical setup". For example, running the script with 2 MPI ranks and a sequence with 4 jobs, leads to "hybrid" execution with 2 ranks @ 2 cores. Defining none or a single rank of MPI causes a "shared" execution with #cores equal to the number of jobs in the sequence.

Note: MOCASSIN ships with a submit system for the SLURM workload manager that can automatically bundle and submit jobs in a convenient fashion with more advanced sequence strings. The submit script uses a basic XML template system, from which an execution script with parameters for "moccasin_mt.py" is generated. It is most likely not difficult to modify the XML template and "slurmsubmit.py" in such a way, that it can be used for other batch systems as well.

Important: The concurrency wrapper does not ensure that simulations are packaged in a "smart" way based on their expected runtime. This causes mixing of long a short simulation and may lead to unnecessary waste of resources.

Additional information & tips:

- When simulating a new system/model, users should always run a small test set and check the integrity/consistency of the results before simulating a full parameter sweep
- Large KMC parameter sweeps accumulate a significant disk space requirement when run in parallel. Users should use the small test set to estimate the space requirement before blindly running an entire "*msl" file at once

5.3 Job submitting with SLURM

For the specific case of the SLURM batch workload system, MOCASSIN provides a submit system for jobs. The system consists of an abstract submit script "slurmsubmit.py" that builds job scripts based on an XML template and loads provider scripts for program command argument generation. The matching provider script "mocsim_slurm.py", which translates the original argument string into a generator function for command line arguments for the "moccasin_mt.py" execution wrapper, is included. Before usage, the XML template should be configurated to your requirements and the executable entry needs to be set to target "moccasin_mt.py" on the executing machine. Startup is performed by running the following command:

```
python3 slurmsubmit.py <path_to_xml_template> db=<path_to_msl> jobs=<job_sequence>
```

In contrast to directly calling "moccasin_mt.py", the "mocsim_slurm.py" generator can parse mixed range information, e.g. "1,2,4-6,10-12", and is sliced into packages matching the settings in your template. By default, if the last package is smaller than the requested product of ranks and cores per rank, the system will automatically switch to an MPI only setup with the correct number of ranks. For example, passing a sequence with 11 jobs to a 2 ranks 2 core template will be submitted as two 2-ranks-2-cores hybrid jobs and one 3-ranks-1-core MPI job.

Important: Due to the way the python import statement works, it is required that “slurmsubmit.py” and “mocsim_slurm.py” are placed into the same directory.

5.4 Collecting results

Result raw data of the simulator is stored in the “*.mcs” state files, which can be opened using a reader class provided by the MOCASSIN C# evaluation API. Commonly used, simple to calculate properties, e.g. ion conductivities or diffusion coefficients, are also written as a statistics text dump for each particle into the standard out. Currently, the simulation dumps checkpoint file and statistics for each percent of progress. Using the meta data table in the “*.msl” file, many common evaluations can be done by just extracting relevant data from the text output file. To collect all results generated using “moccasin_mt.py” back into the source database, the simple script “moccasin_collector.py” is provided. It can be used with the following command:

```
python3 mocassin_collector.py <abolute_path_to_msl> <parent_directory_of_job_folders>
```

The script collects the “*.mcs” files and the “stdout.log” from all generated job folders based on the folder names. The second parameter is optional, if left empty the script uses the parent directory of the “*.msl” file as root directory.

Important: The script changes the database journal mode to “WAL” temporarily, which is not supported by the parallel solver startup. If the script execution is interrupted for any reason, the database must be reverted to “DELETE” mode manually if to be used with “moccasin_mt.py” again.

Note: The C# evaluation API provides a context class for the populated “*.msl” file. The context allows to query jobs, offers some evaluation tools, and provides several processing helpers, e.g. model context reconstruction or decoding/de-mapping of indexed data.