SIMES User Manual v1.0

1. Overview

Hybrid Electrical Energy Storage (HEES) system is an electrical energy storage (EES) that consists of many heterogeneous electrical storage elements such as lithium-ion battery, lead-acid battery and supercapacitor. A HEES system mainly comprises of energy sources, loads, EES banks, charge transfer interconnects (CTI) and converters. The HEES system can store energy from the sources and supply energy to the loads. Energy are stored in EES banks. Since the voltage of different banks, sources and loads usually do not match each other, converters are used to connect different components to a common interconnect, i.e. CTI.

HEES simulator is a software tool used for simulating the operation of an HEES system, including charge allocation, charge migration, charge replacement, etc.

2. SIMES Composition

SIMES consists of three main modules, namely Parser, Simulator and Visualizer.

Parser is implemented based on TinyXML (http://www.grinninglizard.com/tinyxml/), a lightweight XML parser. Parser is responsible for parsing the input data from an XML file which describes the HEES system.

Simulator simulates the operation of a HEES system parsed from the input XML file.

Visualizer is a tool that help user to create the input XML file using a graphical interface.

Below is the tree view of the SIMES package structure:

```
simes/
├─── simulator
│    ├─── banks
│    ├─── bin
│    ├─── converters
│    ├─── core
│    ├─── cti
│    ├─── examples
│    ├─── lib
│    ├─── loads
│    ├─── managers
│    ├─── parser
│    └─── sources
├─── tinyxml
└─── visualizer
     ├─── bin
     └─── genfile
```

3. SIMES Input File Format

The input to SIMES is an XML file describing the HEES system and optionally with the commands to be issued to Simulator to run the simulation.

Below is what an legitimate input file should look like:

```
<project>
	<name>[Project_Name]</name>
	<comp type=["bank"/"source"/"load"/"manager"]>
		<name>[Component_Name]</name>
		<derived type = [Derived_Type]>
			<[Component_Property]>[Property_Value]</[Component_Property]>
		</derived>
	</comp>
	<comp type="cti">
		<name>[Component_Name]</name>
		<derived>
			<[Component_Property]>[Property_Value]</[Component_Property]>
		</derived>
	</comp>
	<comp type="converter">
		<name>[Component_Name]</name>
		<port_a>[Port_A_Component_Name]</port_a>
		<port_b>[Port_B_Component_Name]</port_b>
		<derived type = [Derived_Type]>
			<[Component_Property]>[Property_Value]</[Component_Property]>
		</derived>
	</comp>
	<sensor target=[Target_Name] property=[Target_Propert]/>
	<cmd time=[Time] type="finish"/>
	<cmd time=[Time] type="set" target=[Target_Name]>
		<[Component_Property]>[Property_Value]</[Component_Property]>
	</cmd>
	<cmd time=[Time] type="get" target=[Target_Name]>
		<[Component_Property]></[Component_Property]>
	</cmd>
</project>
```

Each input XML file should have a node called <project> at the root level. Under project there should be a <name> child node which specifies the name of the project. The sensor output's filename is determined by the project name. Child nodes <comp> is used to declare a component, which has a attribute named "type". Type of a component can only be "source", "load", "bank", "cti", "converter" or "manager". The <name> child node of a component specifies the unique identifier of that component. A converter component has two child nodes in addition named <port_a> and <port_b>, which specifies to what components this converter is connected to. The derived type of a component is defined by attribute "type" in the <derived> child node. Properties specific to that derived type is set by child nodes under <derived>. Sensor child nodes specifies which value should be output and all commands to be executed by Simulator are declared by <cmd> child nodes. There are three types of commands: set, get and finish. Set/get command is used to set/get the property value of a component at the specified time. Finish command is used to run simulation till the specified time and then exit.

There are several example input XML files in the example subdirectory.

4. Simulator Basic Usage

Before compiling Simulator for the first time, run

```
make mkdir
```

to create the corresponding obj, lib and bin folders. Then run

```
make
```

to compile Simulator.

To run Simulator for a HEES system, use the command:

```
simulator Input_File [Options]
```

where Input_File is the name of the input XML file. Currently, the only option supported is "-i" which means run simulation in interactive mode, where the commands are read from console input instead of from the input file. The commands supported in interactive mode are:

```
sim Time

set Target Property Value

get Target Property

finish
```

The output is *ProjectName*.out, which contains all the values of the sensors at different time.

5. Simulator Advanced Usage

One can extend SIMES by introducing new models. It is done by implementing new C++ classes deriving from the existing base classes. For example, to implement a new energy storage bank model, one can implement a class deriving from CBankBase and implement these interfaces defined in CPort and CComponent classes (CBankBase inherits from these two classes). The definition of the new class may look like:

```
class CNewStorageModel : public CBankBase
{
private:
        ...
public:
        CNewStorageModel (void);
        ~ CNewStorageModel (void);
        virtual double GetStateofCharge() const;                        // Inherited from CBankBase
        virtual double PortVoltage(double time, double current) const;  // Inherited from CPort
        virtual double MaxOutPortCurrent(double time) const;
        virtual void Reset();                                           // Inherited from CTiming
        virtual double NextTimeStep(double time, int precision) const;
        virtual void TimeElapse(double time, double timeElapsed);
        virtual bool SetProperty(const string &name, const string& value);
        virtual string GetProperty(const string &name) const;
        virtual bool SetSensor(const string &name, CSensor &sensor);
};
```

Function PortVoltage defines the voltage-current relationship of the storage bank's output. MaxOutPortCurrent specifies the maximum amount of current this storage bank can provide. NextTimeStep should return the maximum acceptable time step for simulation if all other conditions remain the same. TimeElapse function simulates the operation of this storage bank during the time step.

This new model should be added in the factory method of CBankBase in BankCreate.cpp before it can be instantiated in Simulator.

6. Visualizer Usage

To compile the visualizer, first run:

```
qmake
```

Then run:

```
make
```

And the executable can be found in the bin subdirectory.

Visualizer is used to "draw" the HEES system. One can add components to the canvas, connect converters to other components, set components' derived type and properties, and set sensors and commands. The created HEES system can be saved to an XML file as the input to Simulator.