Lessons Learned From Modelling a Complex Residential Building Energy System in TRNSYS

Cem Kalkan[1], Colin Ward[1], Jean Duquette[1], Fidel Khouli[1], Mehmet Akif Ezan[2]
[1]Carleton University, Ottawa, Canada
[2]Dokuz Eylul University, Izmir, Turkey

Abstract

The TRNSYS software is a preferred choice for modelling the operation of diverse multi-component thermofluid systems. Thanks to its versatile architecture and associated libraries, TRNSYS enables users to develop custom components with a high level of accuracy and speed. However, challenges persist regarding how to properly integrate these components into larger-scale models and ensure correct functionality via program tracing. In this study, two key TRNSYS limitations are discussed: 1) the computation time related to coupling C++, MATLAB®, and Python codes with TRNSYS simulations, and 2) the need to emulate thermal inertia in closed loop fluid systems and the potential pitfalls that may occur if this aspect is not handled correctly. Examples are presented to illustrate these limitations and results are quantified via the calculation of simulation speed and modelling error. Potential solution methods are explained to help current and future TRNSYS users develop higher accuracy models.

Introduction

Numerical modelling is typically employed in the design and analysis of multi-component thermofluid systems as it enables designers to quickly investigate different alternatives and their effect on overall system performance. As the complexity of these systems increases, the level of effort needed to develop representative models also increases. Therefore, to reduce the designers' workload and speed up the design process, adequate modelling and simulation software is required. The TRNSYS software has been developed to address this need.

TRNSYS enables users to design and develop a new system or improve the performance of an existing system using its versatile architecture and associated component libraries. TRNSYS component libraries include a large number of models (referred to as Types) that can be used for modelling essential system components such as heat exchangers, storage tanks, and controllers. In addition to the pre-existing component libraries, the architecture of TRNSYS allows users to develop and integrate custom component models.

Literature Review

TRNSYS allows its users to model a wide variety of system applications. Although there are no restrictions on the types of applications in which TRNSYS can be used, the software has found many use cases in complex building-integrated renewable energy applications (especially those comprising solar energy systems). For example, in one study Reda et al. (2016) numerically investigated the performance of two design configurations (*i.e.*, parallel and series) of a solar-driven thermal system that included solar thermal collectors, an absorption chiller, storage tanks, and a district heating connection for heating and cooling an office building located in Finland. They developed a model of their system in TRNSYS using its standard component libraries and were able to conclude that the parallel configuration performed better than the serial configuration. In another study, Sokhansefat et al. (2017) carried out a performance analysis for a solar-thermal cooling system composed of solar collectors, a storage tank, an auxiliary heater, an absorption chiller, a cooling tower, and an air handling unit using TRNSYS and its standard component libraries. In the first part of the study, they modelled an actual system located in Iran, and in the second part of the study, they investigated the effects of varying different system parameters (*i.e.*, solar collector area, storage tank volume, auxiliary heater setpoint temperature, mass flow rate of the collector loop, and collector slope) on the system's performance. Results showed that the system's performance could be improved by roughly 28% using the optimum parameter values.

Even though the number of models found in the TRNSYS component libraries is vast, users often need to develop custom component models to simulate unique processes that are not part of the standard TRNSYS package. TRNSYS allows users to create and add custom component models that are developed using several common programming languages such as Fortran, C++, MATLAB®, and Python. The TRNSYS kernel, which is coded in Fortran and runs essential simulation functions such as simulation time advancement, convergence checks, etc., calls the DLL file of the component model during simulation iterations at each timestep. Some programming languages (*e.g.*, Fortran and C++) can create the component DLL file directly whereas

other programming languages (*e.g.,* MATLAB® and Python) require an interface DLL file to link the custom component model code to the TRNSYS kernel (Klein et al., 2017).

Custom component models developed in C++ can be seamlessly integrated into TRNSYS simulations as native Fortran components after compiling the code to a DLL file and creating a component proforma in TRNSYS Simulation Studio (TRNSYS, 2020). Many examples of studies with custom components developed in C++ can be found in the literature. For example, Pintaldi et al. (2017) investigated the performance of a high-temperature solar cooling system composed of parabolic trough collectors, a storage tank (both sensible and latent types were considered), an auxiliary heater, an absorption chiller, and a cooling tower in a commercial building located in Australia. The component models for the absorption chiller and the latent heat storage tank were developed using the C++ programming language. They compared the performance of the latent and sensible heat storage options using a parametric analysis. Their results showed that the latent heat storage system had higher storage efficiency than the sensible heat storage system.

In contrast to the connection between C++ and TRNSYS, the MATLAB® – TRNSYS connection requires an interface component (*i.e.*, Type 155) to execute commands in MATLAB® scripts. Type 155 runs the script which contains the component model and exchanges inputs, outputs, and other important information with the TRNSYS kernel as the simulation timesteps proceed using the Component Object Model (COM) interface. Due to the variety of libraries that it contains from various research areas, MATLAB® custom component models may provide an advantage by shortening the development time for certain model types (especially those containing control applications, and a solution to a system of equations). An example of a study in the literature that uses Type 155 was conducted by Li et al. (2018). They simulated the performance of a heating system comprising an air-source heat pump, an open-air swimming pool, and a phase change material (PCM) storage tank. The open-air swimming pool and PCM storage tank models were developed using MATLAB®. They carried out a parametric study of the heating system to minimize the volume of the PCM storage tank by analyzing the effects of several decision variables. Their results indicated that it was possible to reduce the storage tank volume and achieve economic savings.

Similar to the custom components developed in MATLAB®, those developed in Python also require an interface component. To link a Python script to TRNSYS, it is possible to use Type 163, Type 169, or Type 3157. Type 163 uses an input and output file to exchange data between the Python script and TRNSYS in each timestep of the simulation. Due to the usage of file write/read functions, this component might not provide a fast connection. Type 169, on the other hand, is a wrapper C++ component that can execute Python scripts. The data exchange between the TRNSYS kernel and Python script is controlled by Type 169. A potential issue with Type 169 is that it does not allow for calling of external libraries in the Python script, and this might significantly reduce one's ability to develop custom components for different applications. Additionally, Types 163 and 169 do not perform initialization steps like other TRNSYS types. More details regarding the use of Types 163 and 169 can be found in the TRNSYS documentation (TRNSYS, 2021). The third alternative interface component, Type 3157, is a recently developed component that uses the C Foreign Function Interface (CFFI) package to connect Python scripts to TRNSYS. Thanks to its versatile coding structure, Type 3157 allows developers to call external Python libraries and perform initialization steps like other native TRNSYS components. Therefore, it has significant advantages over Types 163 and 169. Additional details regarding the use of Type 3157 can be found in Bernier et. al (2022). Although Type 3157 possesses great potential, only a limited number of studies that employ the component can be found in the literature due to its recent development. For example, Kirschstein et al. (2023) investigated the performance of a floor heat exchanger used for heating and cooling the buildings of a small district. In the study, Type 653, which simulates a lumped capacitance model of a radiant floor, was extended by adding effectiveness calculations that were carried out using a custom Python component linked by Type 3157.

Although many options are available for linking custom component models in TRNSYS, not all are alike from a computational speed standpoint. Although this issue is of paramount importance while simulating large and complex system models (due to time and resource constraints), it has not been covered in the literature. Only one study in the literature mentioned that simulations using a custom component model in MATLAB® took significantly longer than simulations without the custom model (Bava and Furbo, 2017). To the authors' knowledge, there is no mention of the computational times associated with simulating custom component models developed in C++ or Python.

Another important TRNSYS issue that is seldom (if never) discussed in the literature involves the inclusion of pipe components in fluid loops. Real-life thermofluid systems often contain fluid loops in which the working fluid is circulated between different components via pipes. However, many studies in the literature do not include pipe components in their system models since it is assumed that there are no heat losses from the loop pipes. Although omitting pipes does not pose an issue from a heat transfer standpoint, it can cause significant convergence problems and affect model accuracy, especially in complicated systems with several components and loops. The reason for this is due to the thermal inertia of the heat transfer fluid inside of the loop pipes being neglected. A potential solution

to this problem is to ensure that all loops considered in any given simulation contains a pipe that has enough capacity to hold the fluid flow for at least one timestep (TESS, 2010). Although this solution sounds attractive due to its ease of implementation, it only works if the mass flow rate of fluid circulating through the pipes remains constant in time; unless steps are taken to ensure that the dimensions of the pipes are precisely updated whenever there is a change in the mass flow rate. Failure to implement these steps when simulating variable mass flow systems (*e.g.*, in a variable speed pumping system) can lead to significant error propagation.

The objective of this study is to address the issues mentioned above in the following two ways:

- Compare the required computation times in TRNSYS for simulating two distinct simplified system models that each contain custom components developed in C++, MATLAB®, and Python. The simplified models are of an absorption chiller and a Photovoltaic Thermal (PV/T) collector.
- Display the potential integration problems that may arise when the thermal inertia of a heat transfer fluid in a loop is neglected (*e.g.*, via omitting pipe components in the model), and provide recommendations on how to avoid this problem via the addition of custom delays. A simplified loop system comprising heat exchangers, pumps, and fluid heaters is developed for illustrative purposes.

Methods

To compare the computation times associated with the various custom components, two simplified system models containing different levels of complexity are developed in C++, MATLAB®, and Python: (*i*) an absorption chiller model, and (*ii*) a PV/T collector model. As the development of these models is not central to the focus of this study, only a summary of their formulation is presented herein.

Figure 1 shows a representative schematic of the absorption chiller modelled in this study. The absorption chiller has four vessels (*i.e.*, a generator, condenser, absorber, and an evaporator), and these vessels are connected to other peripheral systems (which are not included in this study – *e.g.*, a heat source, a cooling tower, and a load) using three external water connections which interface with the generator, the absorber-condenser, and the evaporator, respectively. The external connection in the generator is used to supply the required heat to operate the absorption chiller, the external connection that goes through the absorber and condenser removes the heat released from the vessels, and the external connection in the evaporator is used for providing cooling. To simulate the operation of the absorption chiller, a second-order multivariate regression model is used, which is generated based on an experimental dataset. In this model type, the heat flow (W) from/to the external water connections ($\dot{Q}_k$) is expressed as (Labus et al., 2013)

$$\begin{aligned}\dot{Q}_k = {} & \beta_{0,k} + \beta_{1,k}T_{gen}^{in} + \beta_{2,k}T_{ac}^{in} + \beta_{3,k}T_{eva}^{in} \\ & + \beta_{4,k}T_{gen}^{in}T_{ac}^{in} + \beta_{5,k}T_{gen}^{in}T_{eva}^{in} \\ & + \beta_{6,k}T_{ac}^{in}T_{eva}^{in} + \beta_{7,k}\left(T_{gen}^{in}\right)^2 \\ & + \beta_{8,k}(T_{ac}^{in})^2 + \beta_{9,k}(T_{eva}^{in})^2\end{aligned} \quad (1)$$

where $\beta_{0,k}$ to $\beta_{9,k}$ are constants of the regression model, k represents the external water connection (*i.e.*, generator, absorber-condenser, or evaporator), and T_{gen}^{in}, T_{ac}^{in}, and T_{eva}^{in} (°C) are the external connection inlet temperatures associated with the generator, absorber-condenser, and evaporator, respectively.

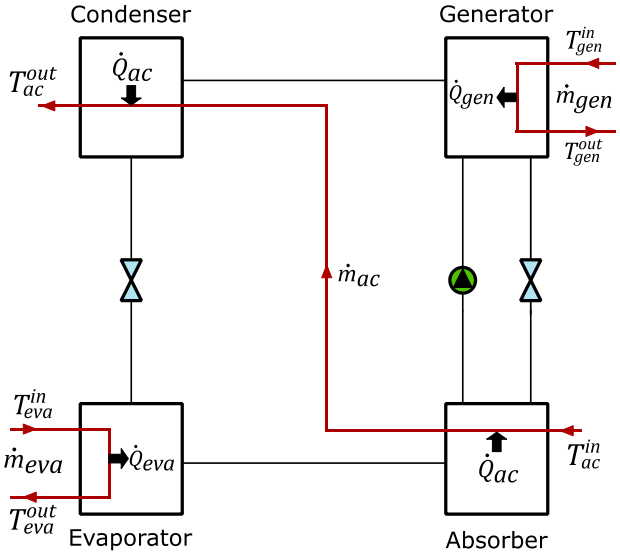

Figure 1. A representative schematic of the absorption chiller model

The constants in the regression model ($\beta_{0,k}$ to $\beta_{9,k}$) are calculated using an experimental dataset which contains 19 steady-state test results with various inlet temperatures for the external connections. The experimental dataset can be found in Labus et al. (2011). Eq. (1) can be applied to all three external connections to calculate the heat flow to/from the absorption chiller. The outlet temperature of the external connections can then be calculated using its corresponding energy balance. For example, the generator outlet temperature (T_{gen}^{out}) is given by

$$T_{gen}^{out} = T_{gen}^{in} - \frac{\dot{Q}_{gen}}{\dot{m}_{gen}c_{p,gen}} \quad (2)$$

where $\dot{Q}_{gen}$ is the heat flow (W) to the generator, and $\dot{m}_{gen}$ and $c_{p,gen}$ are the mass flow rate (kg/s) and the specific heat capacity (J/kg °C) of the water in the external connections,

respectively. More details about the absorption chiller model can be found in Labus et al. (2013).

Figure 2 shows a representative schematic of the PV/T collector that is modelled in this study. The collector contains a glass cover, an air gap, PV cells, an absorber plate, box-shaped flow channels, and an insulation layer. To model the PV/T collector's operation, an energy balance equation is solved for each layer using discretization along the flow direction. For example, the energy balance equation for the glass layer is expressed as

$$m_g c_{p,g} \frac{\partial T_g}{\partial t} = k_g A_c \frac{\partial T_g}{\partial x} + \dot{q}_{solar} + \dot{q}_{conv} + \dot{q}_{rad} + \dot{q}_{conv,in} + \dot{q}_{rad,in} \quad (3)$$

where m_g is the mass of glass node (kg), $c_{p,g}$ is the specific heat of glass (J/kg °C), T_g is the temperature of glass node (°C), k_g is the thermal conductivity of glass (W/m°C), A_c is the cross-sectional area of the glass node, and $\dot{q}_{solar}$ (W) represents the absorbed portion of the incoming solar irradiance. The other terms in Equation 3, *i.e.*, $\dot{q}_{conv}$, $\dot{q}_{rad}$, $\dot{q}_{conv,in}$, and, $\dot{q}_{rad,in}$ (W) represent the convective heat loss to the ambient, the radiative heat loss to the ambient, the convective heat transfer to the air gap, and the radiative heat transfer to the PV layer, respectively. Similarly, an energy balance equation can be written for all the other PV/T collector layers. The resulting equation system is solved using the finite-volume method with the implicit scheme. Details of the equation system and information related to the terms included in the equations can be found in Kalkan et al. (2022). For the solution to the system of equations, the backslash operator (\), NumPy library, and Eigen library are used in MATLAB®, Python, and C++, respectively.

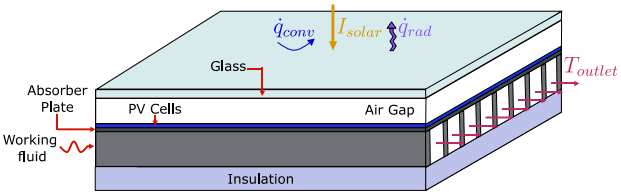

Figure 2. A representative schematic of the PV/T collector model

For both models, the DLL file of the C++ based component is compiled using Microsoft Visual Studio 2022, the connection for the MATLAB® based component (formulated in MATLAB® 2016b) is established using Type 155, and the connection for the Python based component (formulated in Python 3.10) is established using Type 3157. Simulations are carried out using a computer that has the following setup: Intel(R) Core (TM) i5-7200U–2.50 GHz-4 cores CPU, 4 GB RAM, Windows 10–64-bit Operating system.

To illustrate and quantify the problems that can arise from neglecting the thermal inertia in fluid loops, a simplified representative system is developed. Figures 3 and 4 show the simplified system without and with custom delay components, respectively. It is worth noting that a separate scenario containing pipe components is not considered in this study due to the issues mentioned in the introduction related to variable mass flow rate systems. The simplified system contains one main loop and two secondary loops. The main loop is connected to each secondary loop via a constant effectiveness heat exchanger (*i.e., HEX-1* and *HEX-2*). Each secondary loop is equipped with a fluid heater to simulate heat addition from other sources. A pump is included in all loops to drive fluid flow.

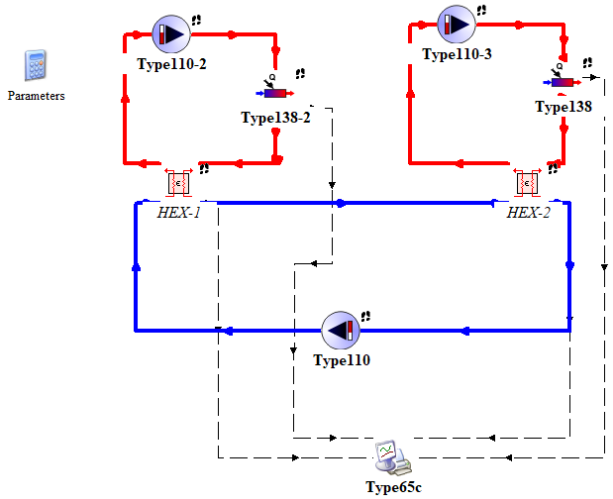

Figure 3. Schematic of a simplified system in which thermal inertia is not accounted for.

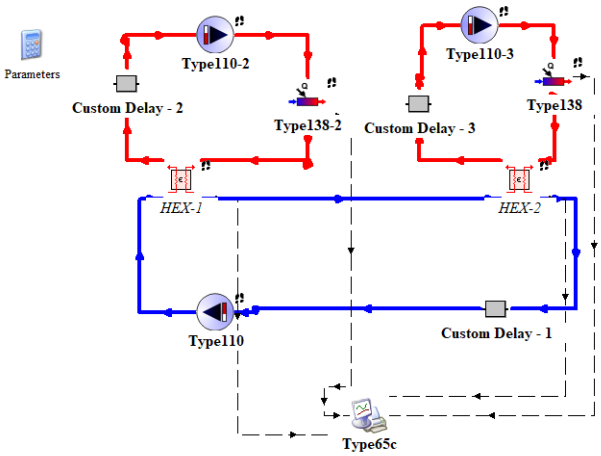

Figure 4. Schematic of a simplified system in which thermal inertia is accounted for using custom delay components.

The main purpose of the secondary loops is to increase the temperature of the fluid in the main loop. In its current representation, the main loop of the simplified system does not have a clearly defined purpose (*e.g.*, supplying heat to an external load). However, similar temporal situations may

arise in any given thermofluid system when there is either no call for a load (*e.g.*, when the working fluid in a solar collector loop is circulated using a bypass line to increase its temperature) or when the temperature of the working fluid in the loop is not high enough for its intended use (*e.g.*, the fluid temperature is lower than the required generator inlet temperature of an absorption chiller).

Types 91, 110, and 138 from the TRNSYS 18 Standard Component Library are used to model the heat exchangers, pumps, and fluid heaters, respectively. The custom delay component shown in Figure 5 is coded in the C++ programming language. Its sole purpose is to delay the inputs for one simulation timestep. Therefore, it requires no additional information from the user. In addition to this advantage, the component can be easily manipulated when the fluid flow in the loop is interrupted.

Results

To verify the accuracy of model outputs associated with the different custom components, the aforementioned simplified system models are simulated using prescribed input conditions.

For the verification case of the absorption chiller model, a fixed schedule is followed, which allows operation only between 10 am and 5 pm. During operational hours, the generator, absorber-condenser, and evaporator inlet temperatures are set to 80ºC, 25ºC, and 17ºC, respectively, while the flow rates are set to 1200 kg/hr, 2000 kg/hr, and 1200 kg/hr, respectively. Figure 5 illustrates the daily outlet temperatures of the generator, absorber-condenser, and evaporator using the custom components developed in C++, MATLAB®, and Python. As shown in the figure, all component model outputs are identical. A similar trend is observed for all other component model outputs (not shown here).

Similarly, for the verification case of the PV/T collector model, a fixed flow schedule is followed where the working fluid is allowed to flow through the collector between 10 am and 5 pm, and the inlet fluid temperature is kept constant at are obtained from the CWEC file for the location of Ottawa, ON for the period between August 1st – August 31st . Figure 6 shows the daily collector inlet and outlet temperatures using the various custom components. All component model outputs are in close agreement with only a negligible difference in the order of 10^{-10} between them. This negligible difference exists because of the distinct algorithms that are used for solving the equation system, and the way in which the data type of the stored variables is treated.

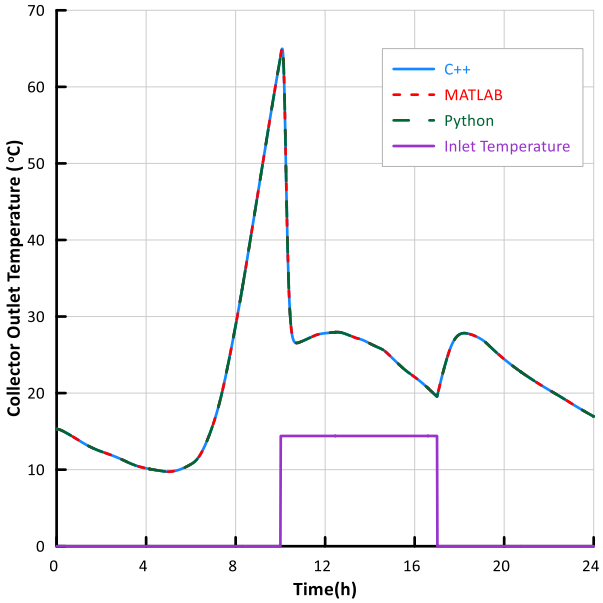

Figure 6. Daily *PV/T collector inlet and outlet temperatures using prescribed input* conditions *and custom components developed in C++, MATLAB®, and Python*.

To compare the computation times of the various custom component models, the simulation time of the verification cases are extended to one month. This time extension is implemented to lower the effect of start-up functions in

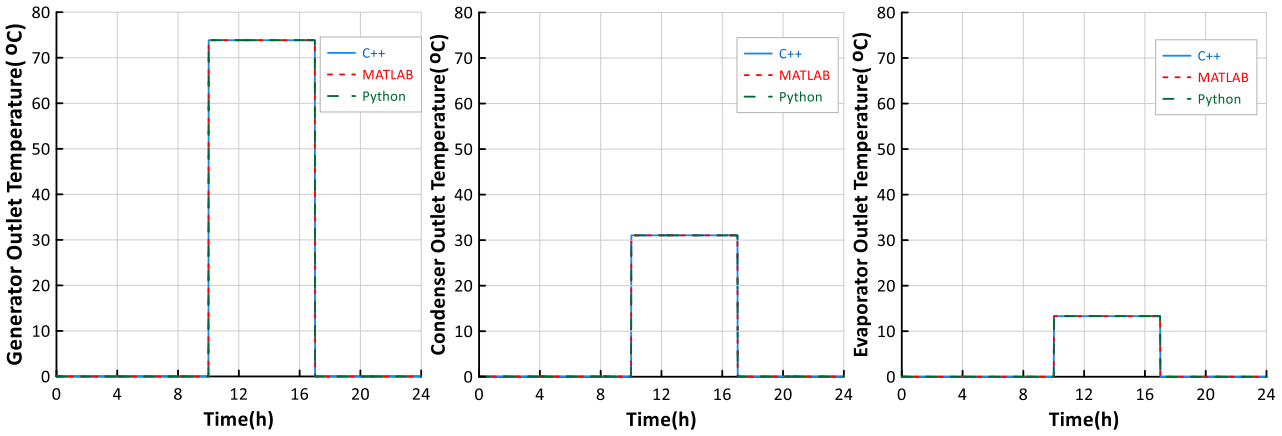

Figure 5. Daily outlet temperature of generator, condenser, and evaporator using prescribed input conditions and custom components developed in C++, MATLAB®, and Python.

TRNSYS kernel via simulation outputs while in debug mode. To eliminate the potential effect of unrelated processes (*e.g.*, background operations) on the computation time, each simulation case is run three times and then averaged. Table 1 shows the average computation time associated with each custom component model over one month of simulation.

Table 1: Average computation times for the absorption chiller and the PV/T collector model using three different custom component models over one month of simulation.

Model	C++	Python	MATLAB®
Absorption Chiller	0.04 s	5 s	1272 s
PV/T Collector	26 s	123 s	1300 s

A comparison of the results in Table 1 shows that the PV/T collector model requires more computation time than the absorption chiller model for all the custom component models. This behavior is expected since the PV/T collector model is more computationally intensive than the absorption chiller model. The PV/T collector model requires solving an equation system, whereas the absorption chiller model is composed of several simple equations. The computation time of the PV/T collector model relative to the absorption chiller model using C++ and Python is greater by a factor of approximately 600 and 25, respectively. However, there is hardly any increase in computation time while using the MATLAB® model. This result indicates that there might be a delay in the data exchange between Type 155 and the TRNSYS kernel. For each simplified system model, the computation time is lowest for the C++ custom component model and highest for the MATLAB® custom component model, with the Python custom component model fitting somewhere in between.

To investigate the thermal inertia effect on the simplified representative system, a test case simulation is conducted for 10 minutes of operation. The initial temperature of the fluid in the loop pipes is set to 20°C. The fluid heater setpoint temperature and maximum heating rate are defined as 60°C and 7200 kJ/hr, respectively. The mass flow rate for main and secondary loops are selected as 400 kg/hr and 200 kg/hr, respectively. Figure 7 shows the change in *HEX-1* inlet and outlet temperatures, and *HEX-2* outlet temperature on the main loop side of the system throughout iterations of the first timestep for the case in which no custom delay component is added to the system loops. Since there is no thermal inertia in the main loop, the outlet temperature at the end of the first iteration immediately becomes the inlet temperature of *HEX-1* in the second iteration. Thus, the outlet temperature of *HEX-2* is updated using each new guess of the inlet temperature. This constant feedback effect continues until the fluid heaters in the secondary loops reach the setpoint temperature defined in the simulation parameters (*i.e.*, 60°C). Eventually, the temperatures in the main loop also end up at a temperature level close to that of the heater setpoint temperature. Since convergence is reached in 21 iterations, which is lower than the default iteration limit before issuing a warning (*i.e.*, 30), the TRNSYS kernel accepts the outputs in this iteration as the final temperatures of the first timestep. However, a quick comparative check of the power level of the fluid heaters and the temperatures in the loop before the first timestep and after the first timestep shows that the fluid heaters do not have enough capacity to achieve these levels of thermal output. Therefore, the obtained results after the first timestep are inaccurate. Although the simplified representative system demonstrated in this study is a small-scale application, similar situations occur whenever the thermal inertia in the loop is neglected. In the case where thermal inertia is present in one or more components of the loop (*e.g.*, a thermal storage tank), convergence can be reached more rapidly due to a decrease in the rate of change of inlet/outlet temperatures, which will result in smaller errors than those illustrated in the simplified representative system presented in this study. However, this will not eliminate the computation time spent on the multiple calls for all the system components. This situation may cause a significant increment in computation time especially when the computation time required for the component model is relatively expensive (*e.g.*, when simulating transient component models).

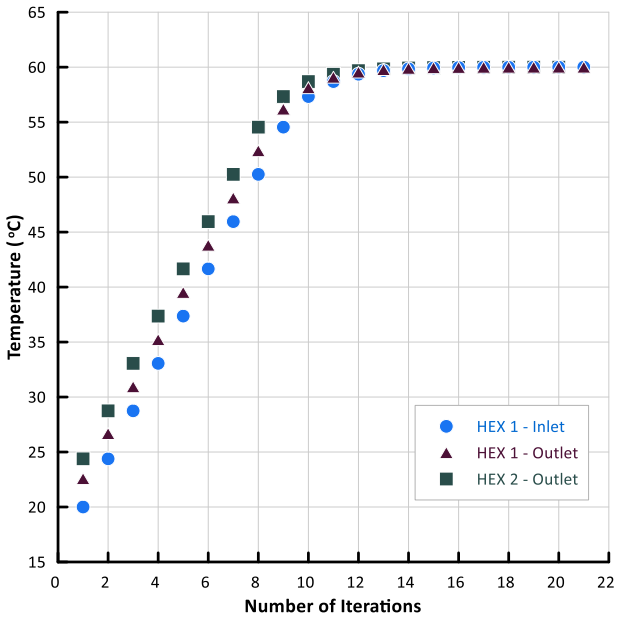

Figure 7. Change of heat exchanger inlet and outlet temperatures over the iteration of the first calculation timestep

To assess the thermal inertia effect of the fluid inside of the loops of the simplified representative system, custom delay components are added. Figure 8 shows the *HEX-1* and *HEX-*

2 outlet temperatures for the system without and with custom delay components. As explained above, the outlet temperatures quickly approach (after 1 minute) the setpoint temperature of the heaters (*i.e.*, 60ºC) when custom delay components are not included. In the case when custom delay components are included, on the other hand, the *HEX-1* and *HEX-2* outlet temperatures increase gradually over the course of ten minutes of operation and do not reach the high temperature levels of the former case. The latter case is a more accurate representation of the thermal processes occurring in the simplified representative system.

Conclusion

In this study, an absorption chiller model and a PV/T collector model were simulated in the TRNSYS 18 environment using custom component models from C++, MATLAB®, and Python. The models were verified using simple test cases. The computation time required for the models was investigated via test cases over the course of one month of operation. Additionally, a simplified representative system model was developed (without and with custom delay components) to assess the effect of thermal inertia in the loop pipes. The main findings of the study are as follows:

- In terms of the custom component models used, C++ provided the fastest solution for both the absorption chiller and PV/T collector models. The C++ component was followed in computational speed by the Python component and then the MATLAB® component. Due to the significant computation time difference between MATLAB® and other custom component models considered in this study, users are recommended to avoid using MATLAB® custom components unless a unique MATLAB® library is required. Considering the recent trends in scientific tool development, users may find Python libraries that satisfy the demand for a MATLAB® unique library. C++ based custom components showed a significant advantage in computation times, thus, they are the preferred choice in cases in which total simulation time is a concern (*e.g.*, parametric or optimization studies, simulations with transient models, or complex systems). However, using C++ custom components may lead to increased development time due to the complex nature of the programming language (*e.g.*, the need to declare variables, memory management, etc.,). To remedy this problem, Python and C++ might be combined (*e.g.*, model development and validation using Python, then simulation using a C++ component which is created based on the validated Python component). This approach may decrease the development time since the Python component does not require compilation of the code after a code change.
- Although TRNSYS allows users to generate results quickly via simulation, care must be taken especially when the integration of multiple system components via fluid loops is required. Failure to account for the thermal inertia present in fluid loops can result in inaccurate outputs and/or increased computation time. Thermal inertia can be simulated by including pipe components (for constant flow systems) from the standard TRNSYS library, or by including delay components (for constant/variable flow systems) from either the standard TRNSYS library or via the addition of customized models.

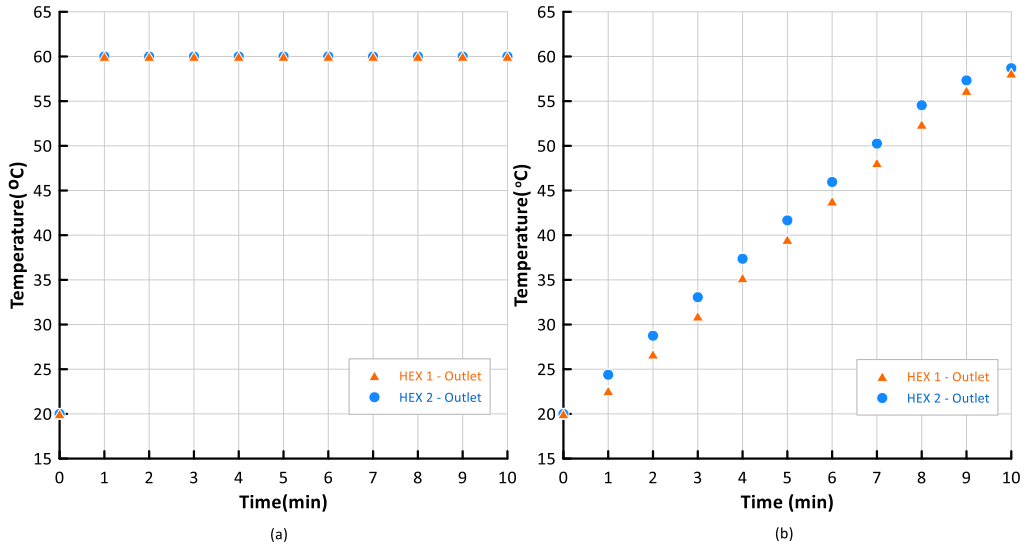

Figure 8. HEX-1 and HEX-2 outlet temperatures (a) without and (b) with custom delay components in the loops of the simplified representative system.

Acknowledgement

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC). The authors would like to thank the developers of the Eigen template library, NumPy package, Type 155, and Type 3157 for the development of the libraries/types and associated detailed documentation.

References

Bava, F. and Furbo, S. (2017). Development and validation of a detailed TRNSYS-Matlab model for large solar collector fields for district heating applications. *Energy*, *135*, pp.698-708.

Bernier, N., Bruno M., Kummert, M. (2022). Calling Python from TRNSYS with CFFI. Polytechnique Montréal. DOI: 10.5281/zenodo.6523

Kalkan, C., Duquette, J., Khouli, F., Keblawi, A., Ezan, M.A., (2022) Development of a Dynamic PV/T-Driven Combined Cooling, Heating and Power System Model for Use in the Canadian Residential Sector. *Proceedings of eSim 2022: 12th Conference of IBPSA-Canada*,Ottawa (Canada)

Kirschstein, X., Reber, J., Zeus, R., Schuster, M., Bishara, N. (2023). Modelling of Floor Heating and Cooling in Residential Districts. *Energies, 16(15)*, p.5850.

Labus, J. (2011). Modelling of small capacity absorption chillers driven by solar thermal energy or waste heat. Ph.D. Thesis, Universitat Rovira I Virgili

Labus, J., Bruno, J.C. and Coronas, A. (2013). Performance analysis of small capacity absorption chillers by using different modeling methods. *Applied Thermal Engineering*, *58*(1-2), pp.305-313.

Li, Y., Huang, G., Xu, T., Liu, X., Wu, H. (2018). Optimal design of PCM thermal storage tank and its application for winter available open-air swimming pool. *Applied Energy*, *209*, pp.224-235.

Pintaldi, S., Sethuvenkatraman, S., White, S., Rosengarten, G. (2017). Energetic evaluation of thermal energy storage options for high efficiency solar cooling systems. *Applied Energy*, *188*, pp.160-177.

Reda, F., Viot, M., Sipilä, K., Helm, M. (2016). Energy assessment of solar cooling thermally driven system configurations for an office building in a Nordic country. *Applied Energy*, *166*, pp.27-43.

Sokhansefat, T., Mohammadi, D., Kasaeian, A., Mahmoudi, A.R. (2017). Simulation and parametric study of a 5-ton solar absorption cooling system in Tehran. *Energy conversion and Management, 148*, pp.339-351.

TESS, (2010). TESS HVAC Library Examples - Restaurant with Solar Absorption Cooling

TRNSYS, (2021). TRNSYS 18 - a TRaNsient SYstem Simulation Program – Mathematical Reference.

TRNSYS, (2020). TRNSYS 18 - a TRaNsient SYstem Simulation Program – Programmer's Guide.