Modularization and Integration of Stock-and-Flow Models using ChatPySD

Justin Hoffmann[1,2], Ricardo Büttner[1], Bo Hu[2]
[1] University of the Federal Armed Forces Hamburg
[2] University of the Federal Armed Forces Munich

Abstract

This paper proposes a new method for modular development and integration of System Dynamics models. Existing System Dynamics models can be decomposed into several modules, or modules can be directly built as required. These modules each embody specific functions, can operate independently, and are less complex than the entire model, making them easier to understand. With the aid of ChatPySD, several modules can be converted into Python modules, and then combined by ChatGPT into a Python application, realizing the complete model function. The purpose of developing this method is to lay the foundation for establishing a System Dynamics module library using the Python language.

Keywords: ChatPySD, PySD, ChatGPT, module library

Introduction

"Modularity has been used in several disciplines as a structured way to approach complexity, to increase efficiency, to reduce repetitive work, to decrease errors, and to create cumulative learning. The System Dynamics field has yet to benefit from modularity as a systematic method for developing models." With these two sentences, Elmasry and Größler (2016) clearly introduced their research efforts and achievements in adopting modular models in the field of supply chain. A modular approach has also been demanded and implemented in the healthcare sector, where multiple models from various areas, such as population dynamics, disease dynamics, healthcare, and healthcare financing are loosely coupled (Djanatliev et al. 2012). For the construction industry, Karl (2016) introduced a System Dynamics Library (SDL) approach, to enable "multidisciplinary teams to develop joint models from varying perspectives".

As systems become more complex, the demand for robust architectures and effective methodologies rises. Eberlein and Hines (1996) advocated for standardized modules called "molecules" to improve model quality. Tignor and Myrtveit (2000) distinguished between classes and submodels in System Dynamics, requiring submodels to be standalone. In their modeling of dryland salinity in Australia, Khan and McLucas (2008) adhered to the V-model approach, placing a strong emphasis on integrating multiple modules into a comprehensive model. These modules were not only designed to be executed independently, but were also intended to have their own user interfaces with control elements. Yeager et al. (2014) presented a new platform for System Dynamics modeling that supports detailed and object-oriented modeling. To enhance the integration of diverse System Dynamics models and data, an MDaaS architecture (Arto et al. 2014) was proposed, which is characterized by models and data sources acting as services that communicate with each other solely through the exchange of state variables (Figure 1).

In this paper, we aim to follow the aforementioned approach and utilize the newly introduced ChatPySD method (Hu 2025). Built upon the Python library PySD (Martin-Martinez et al. 2022) and ChatGPT 4's ADA (see, e.g., Koçak 2025), this method is expected to provide crucial support for integrating the submodels.

We will first present the proposed procedure for modularization and integration. This procedure will then be demonstrated through two examples, followed by a concluding discussion and summary.

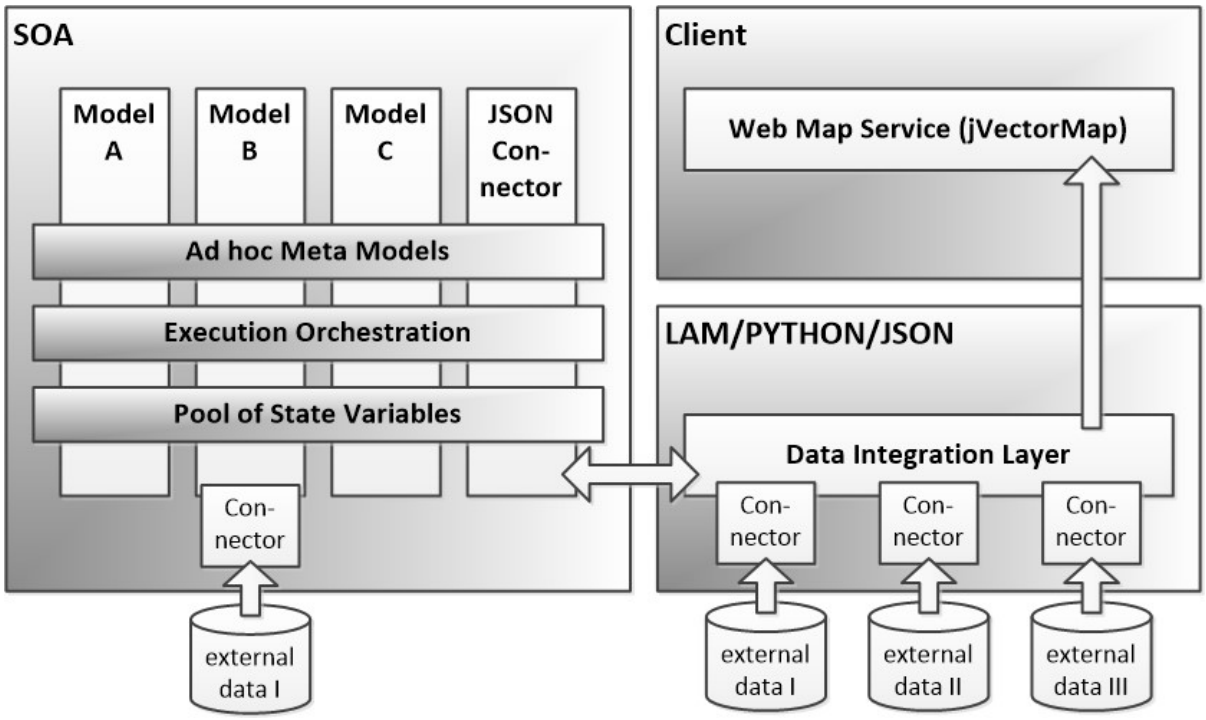

Figure 1: Service-oriented Integration of Models and Data (Arto et al. 2014)

Modularization and Integration Concept

Gamma et al. (1994) emphasized the importance of finding appropriate objects, determining object granularity, and specifying object interfaces as the primary concerns in software design. Meyer (1997) proposed five fundamental requirements for modularization: decomposability, composability, understandability, continuity, and protection.

System Dynamics modules can be developed through a mixed approach of decomposing existing models and building individual modules from scratch. The modules thus generated can be combined into a final model. We have the following requirements for each module:

1. It must have a clearly defined functional scope and accordingly include one or more core stocks.
2. The number of nodes should not exceed a certain limit, such as 70 nodes, and should be easily displayed within a single page of the user interface of the model development and simulation environment, such as Vensim®.
3. It should be capable of independent simulation within the model development and simulation environment. This means that the module's input parameters should only include stocks and constants, while output parameters can include several other variables in addition to these.

The third point is particularly crucial from a practical modeling and simulation perspective. Modules that meet this criterion can be independently tested and optimized. Several such modules can be integrated into a single model using the following steps:

1. Convert all modules to Python modules using PySD or ChatPySD. The advantage of using ChatPySD lies in its ability to leverage the capabilities of ChatGPT-4.
2. Create a Python main program to load all the integrated Python modules and establish a common variable table for all modules, including stocks and output variables.
3. Before simulating each module, initialize all stocks in the model with values from the common variable table. The module is then simulated for one time step, and the

current values of all stocks and specific variables are stored back in the common variable table.
4. Each time step of the overall model simulation consists of sequentially calling the integrated modules for simulation.
5. The entire model simulation is completed by iterating through all time steps according to the design.

Decomposing an existing model into several modules is, to some extent, a process of re-modeling. The following method can generally be used:

1. Based on the functions assigned to the module, determine one or several core stocks of the module, i.e., the stocks that the module's operation will change;
2. Delete all variables and constants that will not cause changes in these core stocks;
3. The remaining variables can be concentrated into one page and stored separately as a module.

Integration using ChatPySD

In this section, we will demonstrate the process of modularization and integration through two examples.

URBAN1

In this section, we present an example of modularization and integration of the URBAN1 model (Ghaffarzadegan et al. 2011). We first decomposed the original model into three modules using Vensim® and then recomposed them into a integrated model with the help of ChatGPT, using the ChatPySD approach (Hu 2025) to embed the model into ChatGPT. To this end, the URBAN1 model was segmented into three subsystems. The subsystems used are thematically related but independent systems.

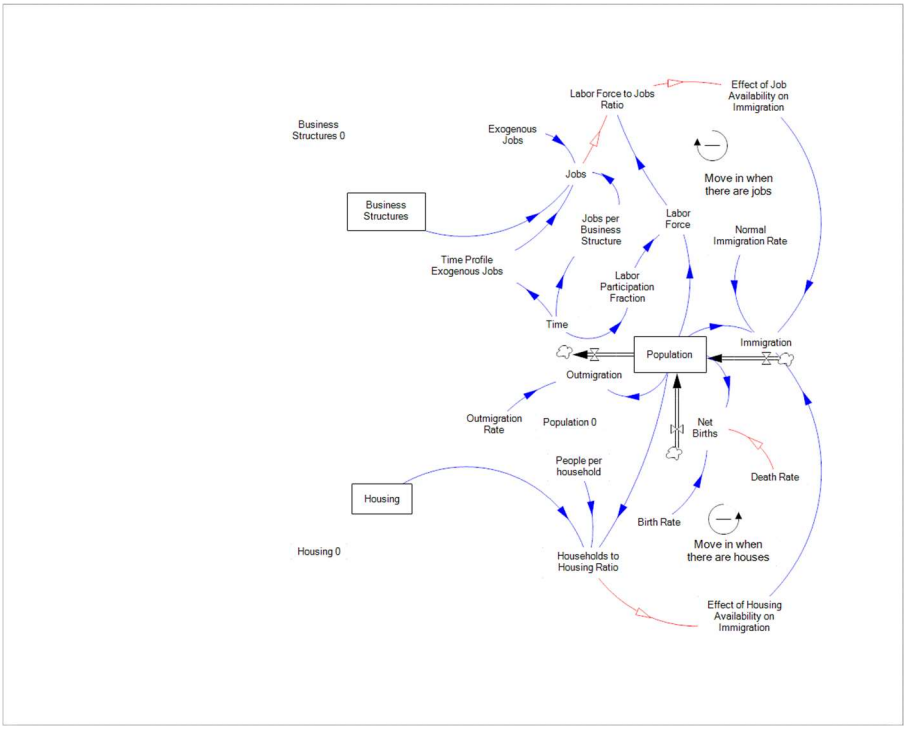

Figure 2: Module Population of URBAN1

The first module revolves around the stock of `population`. Figure 2 illustrates the subsystem that describes population dynamics based on the birth rate, mortality rate, and migration processes. Population growth is particularly influenced by other factors such as housing and job opportunities. The module retains the two stock variables representing `housing` and `business structures`, while removing all components from the original model that change them.

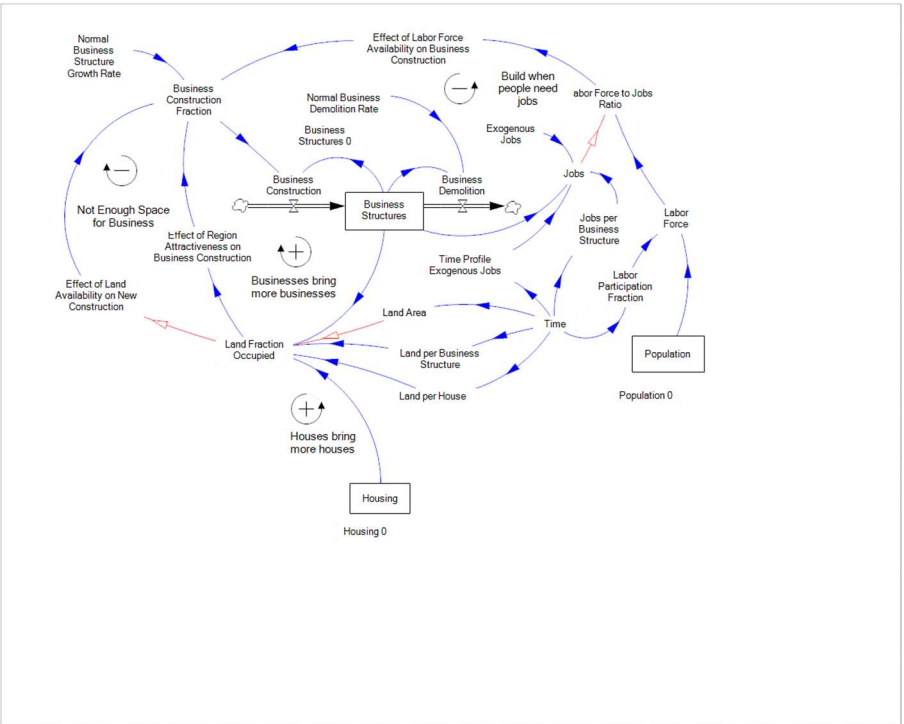

Figure 3: Module Business of URBAN1

The second module is centered around the stock of `business structures`. Figure 3 shows the subsystem that describes the mechanisms of business expansion, job creation, and land utilization. Business growth is also influenced by other conditions such as labor supply and land resources. The module retains the two stocks of `housing` and `population` and their relationship with `business structures`, while removing all components from the original model that change `housing` and `population`.

The third module encompasses the `housing` subsystem. Figure 4 illustrates the processes of housing construction, demolition, and availability in response to demographic changes and land constraints. Housing development is influenced by factors such as land availability, regional construction dynamics, and the relationship between housing needs and available space. As the population grows, the demand for housing rises, altering household distribution and shaping further expansion. The module retains the two stocks of `population` and `business structures` and their relationship to housing development, while removing all components from the original model that change population and company numbers.

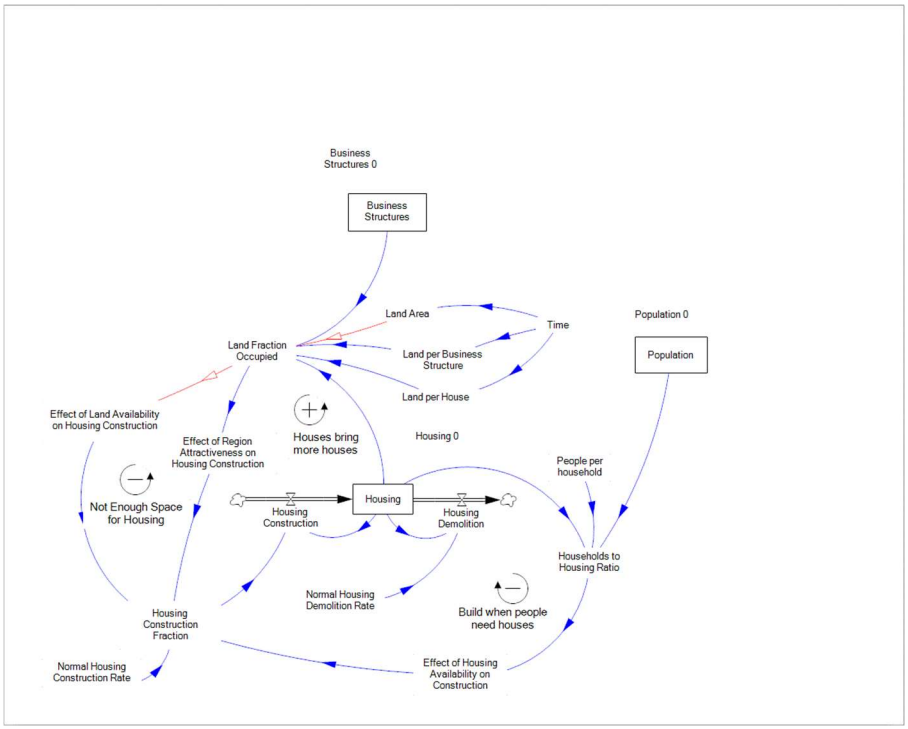

Figure 4: Module Housing of URBAN1

The modules share the characteristic that they incorporate each other as relevant factors via the stocks. We implemented this approach to represent interdependencies and to enable reintegration with ChatGPT. To this end, we inserted the three modules - Population, Housing, and Business - along with the ChatPySD elements into ChatGPT-4o.

In the following, we present a summary of the actual course of our conversation with ChatGPT-4o, supplemented by additional explanations. We provide the complete record of this conversation in the appendix, along with the generated (and partially manually edited) Python modules.

Prompt: (After uploading the ChatPySD files and Urban s5 - Housing.py, Urban s5 - Business.py, Urban s5 - Population.py) I have got here three modules of a system dynamics model. Would you mind running each of them for me?

Response summary: ChatGPT initially did not recognize that we had provided pysd.py. After minor adjustments, the models were successfully executed. Each of the three models was run individually, and the respective calculation results were visualized using ace_tools.

Prompt: Please don't use ace_tools because it is not available for me. Please plot the results of the three modules.

Response summary: ChatGPT used Matplotlib to generate a line chart for each submodel.

Prompt: Fine. Please use following initial values: Housing: 14000 Business: 1000 Population: 50000

Response summary: ChatGPT assigned the predefined initial values only to the model whose filename exactly matched the specified designation, instead of applying them to all variables with the same name across all models. New simulations were then conducted using these values.

Prompt: Please set these initial values in all three modules.

Response summary: ChatGPT did not integrate the initial values into all three modules.

Prompt: You have not set all three initial values in Housing and in Business. Please do it.

Response summary: ChatGPT successfully applied the predefined initial values to each model. Additionally, simulations were conducted, and ChatGPT provided a graph for each module based on the results.

From this point onward, the embedding of the three models using the ChatPySD approach was successful. The next step focuses on re-integrating the three modules into a single model.

Prompt: Perfect! Now please do following for me: - Instead running all three modules for 150 years, - run each of the three modules only 1 step and then transfer the shared stock values to each other - and do this for 150 times (years)

Response summary: ChatGPT executed each of the three modules for one time step at a time, representing a year. After each time step, the stock values (Housing, Business Structures, and Population) were updated and transferred between the modules. This process was repeated for 150 time steps.

Prompt: Please show me the whole code.

Response summary: ChatGPT provided the desired code. The code demonstrates that it loads the three modules and initializes the shared stock values (Housing: 14,000, Business Structures: 1,000, Population: 50,000). Each module runs one step at a time for 150 years, with the stock values being updated and transferred between the modules after each step.

Prompt: Would you mind putting all three curves into a single graph?

Response summary: ChatGPT created a line plot illustrating the development of stock values from the modules over a 150-year period.

Prompt: Please use a second scalar for business structures and housing.

Response summary: ChatGPT created a dual y-axis line plot using Matplotlib, where the primary y-axis represents Population, and the secondary y-axis represents Housing and Business Structures. The plot displays Population as a red line, Housing as a blue dashed line, and Business Structures as a green dotted line, with labels, a legend, and a grid for clarity.

Prompt: I am not sure about your code housing_result = housing_model.run(return_timestamps=[t]). Please change it to housing_result = housing_model.run(params={"INITIAL TIME": t, "FINAL TIME": t+1}) And please do the same to business_result population_result

Response summary: ChatGPT adjusted the time parameterization as requested. This resulted in an error, which was resolved. The error occurred because PySD does not support reset_state(), which had been used previously. The problem was solved by reloading the models at each time step to ensure a consistent starting point. The simulation was executed step by step, updating stock values accordingly and visualizing the results using a dual y-axis plot.

The previous process illustrates the successful integration of the modules through human-machine interaction. This human-machine interaction enabled the modules to be connected via stock variables and the desired values to be initialized. Furthermore, it facilitated the collaborative resolution of issues during the integration and simulation process. The conversation between the user and ChatGPT played a central role in this process. Through the user's instructions, the successful execution of the simulation for the newly integrated model was achieved. Thus, effective human-machine interaction can significantly influence the success of the process and the quality of the results.

Power2045

In this section, we present an additional example of modularization and integration. The original Power2045 model (Hu et al. 2024) has been divided into four modules, each representing a subsystem for renewable energy supply:

- **WDPV**: electricity consumption and production from wind, solar, and other renewable sources;
- consumer-side **load shifting**;
- pumped **storage** hydropower;
- **dispatchable** thermal power plants and plants for synthetic fuel production.

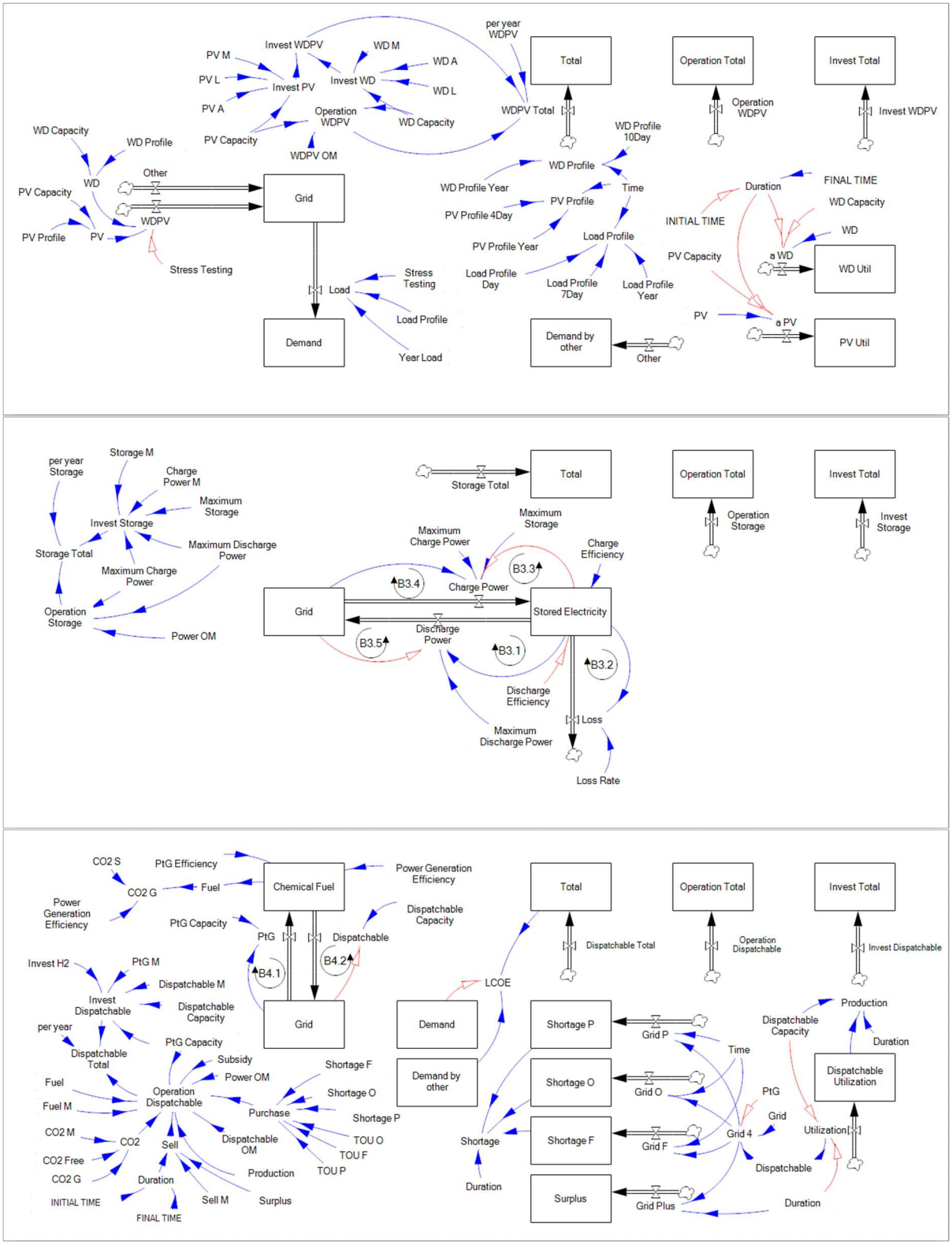

Figure 5: Three modules of the Power2045 model (from top to bottom): WDPV, electricity storage, and dispatchable generation

As shown in Figure 5, three of the modules consider both the technical and economic aspects of the subsystems. A model assembled from these three modules enables a detailed hourly simulation of electricity supply over a year. The WDPV, Storage, and Dispatchable modules are executed sequentially in each of the 8760 time steps, reflecting a hierarchy of energy utilization. Renewable energy is prioritized, followed by mechanical energy storage, and finally, conversion to chemical energy or dispatchable generation. To maintain consistency throughout the simulation, state variables are shared globally across all modules, while other variables are kept local within each module.

All three modules include a stock `Grid` that balances electricity generation and consumption. The mechanisms embedded in the storage and dispatchable modules ensure a largely balanced grid, potentially including the purchase of electricity from external sources.

A majority of the variables within these three modules are dedicated to economic analysis. Each subsystem's investment, operational costs, and annualized total costs are computed. By summing up the relevant stocks and dividing by the total annual electricity consumption, the Levelized Cost of Energy (LCOE) is calculated. Notably, the electricity generation and costs of other renewable energy sources are not included in these calculations.

One of the four modules, consumer-side load shifting, only considers power engineering aspects and does not include economic calculations (Figure 6). This module can be easily integrated with the previous three modules to form a new model: in each time step, WDPV, load shifting, storage, and dispatchable are executed sequentially, sharing state variables to prioritize the use of wind, solar, and other renewable energy sources, followed by load shifting to balance peak and off-peak demand, and finally using mechanical and chemical energy storage and dispatchable generation to maintain grid stability.

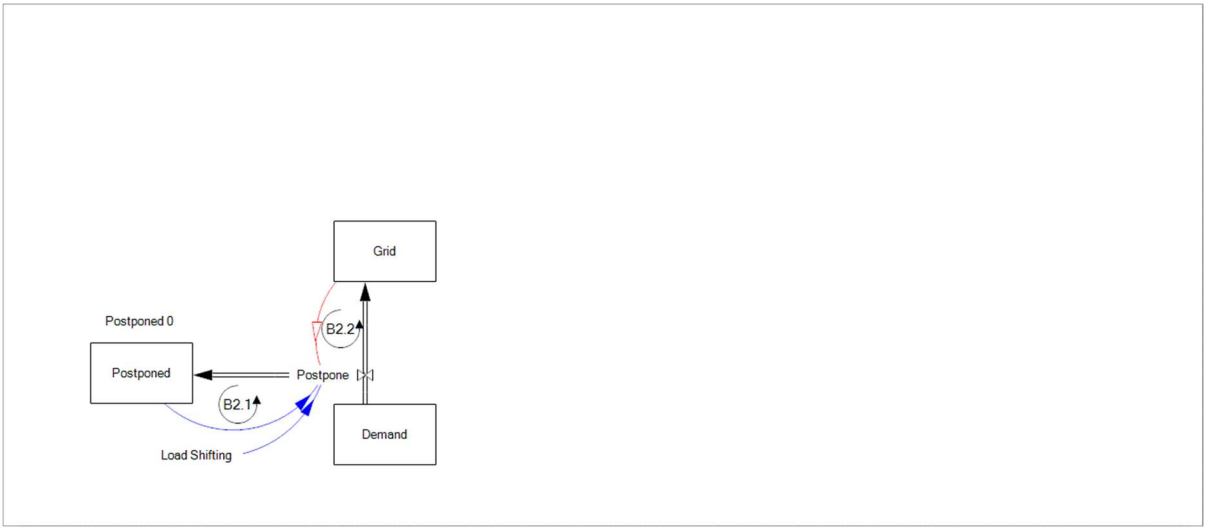

Figure 6: An addition module of the Power2045 model: load shifting

The results of the simulation using the assembled model are consistent with the results obtained from the original model.

Discussion and Conclusion

This report introduced a method for decomposing existing System Dynamics models into several modules, which, along with other directly established modules, can be independently simulated, reflect certain functions, and are significantly less complex than the entire model, making them easier to understand. With the aid of ChatPySD, the modules are converted into Python modules, and then recombined using ChatGPT into a Python application that retains

the functionality of the complete model. Using two published models as examples, this approach initially demonstrated its feasibility.

The purpose of developing this method is to lay the foundation for building a library of System Dynamics modules using the Python language. As shown in Figure 7, the main function of such a module library is to provide modules that can be integrated to create System Dynamics applications. These applications can run in a regular Python environment to output the expected text and graphical results, and can also run with the aid of ChatGPT's ADA to produce more detailed analysis results. The modules in the library can also be run individually in ChatGPT's ADA and generate results.

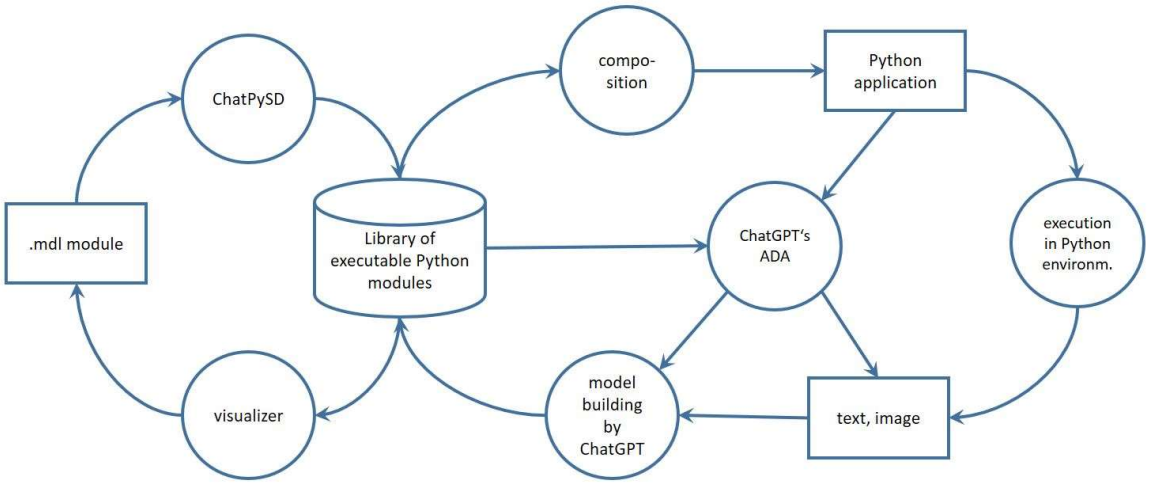

Figure 7: building a module library with ChatPySD

A current research focus is to generate new System Dynamics modules and optimize them with the support of ChatGPT, based on various text and graphical information combined with existing modules. At the same time, establishing a suitable method for generating stock-and-flow diagrams from existing Python modules to help modelers build new modules or optimize existing ones is also a key task.

A major drawback of the method described in this report is its low level of automation. For example, the process of splitting existing models into modules is currently entirely manual, which increases the chance of errors. We will develop corresponding software to strive for the highest degree of automation in splitting and reduce the error rate.

An obvious limitation of this method is that modelers need to have a certain level of Python programming skills, which in many cases limits the practicality of this method. To address this, we have also developed a method for implementing modularization and integration within the Vensim environment, and we plan to present this at this conference.

The use of generative AI to support System Dynamics modeling and simulation has tremendous potential for development. Putting modularization back on the agenda can facilitate the transition of system dynamics from an art to a computational science (Pruyt & Kwakkel 2012). This approach enables the fusion of human-created and AI-generated system dynamics modules within the same simulation application, thereby advancing the practice of Hybrid Intelligence (see, e.g., Dellermann et al. 2019) in the field of System Dynamics.

References

Arto et al. 2014 Klaus Arto, Bo Hu, Armin Leopold, Silja Meyer-Nieberg, Goran Mihelcic, Jan Stutzki, Tim Tepel: Modellbasierende Früherkennung politischer Krisen: Prototyp einer serviceorientierten Plattform. *Multikonferenz der Wirtschaftsinformatik, 278-293,* Paderborn, 26.-28.02.2014

Dellermann et al. 2019 Dominik Dellermann, Philipp Ebel, Matthias Söllner, Jan Marco Leimeister: Hybrid intelligence. *Business & Information Systems Engineering, **61.5**: 637-643,* 2019

Djanatliev et al. 2012 Djanatliev, A., German, R., Kolominsky-Rabas, P., & Hofmann, B. M.: Hybrid simulation with loosely coupled system dynamics and agent-based models for prospective health technology assessments. *Proceedings of the 2012 winter simulation conference (WSC), 1-12,* IEEE, 2012, December

Eberlein & Hines 1996 R. Eberlein, and Hines, J.: Molecules for modelers. *Proceedings of the International System Dynamics Society,* System Dynamics Society, Cambridge, 1996

Elmasry & Größler 2016 Elmasry, Aly, and Andreas Größler: Modularity in System Dynamics: representing closed-loop supply chain configurations. *2016 International System Dynamics Conference,* 2016

Gamma et al. 1994 Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns : elements of reusable object-oriented software. ISBN 0-201-63361-2, Addison-Wesley, 1994

Ghaffarzadegan et al. 2011 Ghaffarzadegan, Navid, John Lyneis, and George P. Richardson: How small system dynamics models can help the public policy process. *System Dynamics Review, **27.1**: 22-44,* 2011

Hu 2025 Hu B: ChatPySD: Embedding and Simulating System Dynamics Models in ChatGPT-4. *System Dynamics Review, **41:1**,* 03.02.2025, doi:10.1002/SDR.1797

Hu et al. 2024 Bo Hu, Bo Zhang, Yonghua Li, Junshen Zhang: Towards carbon neutrality: Optimizing generation and storage capacities in Germany and carbon pricing in China. *Sustainable Production and Consumption, **46**: 703-716,* Elsevier, May 2024

Karl 2016 Karl, Christian K.: System Dynamics Libraries-An approach to develop modular-oriented simulation models. *2016 International System Dynamics Conference,* 2016

Khan & McLucas 2008 Khan, Naeem U., and Alan C. McLucas: A Case Study in Application of Vee Model of Systems Engineering to System Dynamics Modelling of Dryland Salinity in Australia. *School of Information Technology and Electrical Engineering,* 2008

Koçak 2025 Koçak, D.: Examination of ChatGPT's Performance as a Data Analysis Tool. *Educational and Psychological Measurement, 00131644241302721,* 2025

Martin-Martinez et al. 2022 Eneko Martin-Martinez, Roger Samsó, James Houghton, Jordi Solé: PySD: System Dynamics Modeling in Python. *Journal of Open Source Software, **7(78)**: 4329,* 2022

Meyer 1997 Meyer, Bertrand: Object-oriented software construction. Second Edition, Prentice Hall, Englewood Cliffs, 1997

Pruyt & Kwakkel 2012 Erik Pruyt and Jan H. Kwakkel: A Bright Future for System Dynamics: From Art to Computational Science and Beyond. *2012 International System Dynamics Conference,* 2012

Tignor & Myrtveit 2000 Tignor, Warren, and Magne Myrtveit: Object Oriented Design Patterns and System Dynamics Components. *Proceedings of the International System Dynamics Society,* 2000

Yeager et al. 2014 Yeager, Larry, Thomas Fiddaman, and David Peterson: Entity-based system dynamics. *Proceedings of the international system dynamics conference,* Delft, 2014