A Technique for Estimating the Progress of System Dynamics

David W. Peterson
Ventana Systems Inc.
david@ventanasystems.com

Matthias O. Müller
Ventana Systems Inc.
matthias@ventanasystems.com

Work in Progress - March 21st 2026

Paper to appear in the
Proceedings of the 44th conference of the System Dynamics Society, July 20-24 2026, Delft.

Abstract

System dynamics (SD) started with a qualitative "servo" methodology, but has evolved into a quantitative, data-based, predictive capability that resembles the techniques of model predictive control (MPC). This paper introduces a simulation test bed to quantify how much this transition has changed the capability of SD to improve the performance of systems. The first use of the test bed suggests that the gain in capability may have been large. Because the MPC use of SD models relies on the predictive power of SD models, we also tested the sensitivity of the results to errors in prediction, and discovered that the MPC use of SD models has high payoff even if the predictions are imprecise. Both results must be regarded as preliminary, but they suggest the evolution of SD has greatly improved its potential effectiveness and impact.

1 Introduction

The Servo paradigm

System Dynamics was developed by Jay Forrester starting in 1956, leading to the seminal book *Industrial Dynamics* published in 1961. There were several foundational principles that seemed curious to some people at that time. Forrester downplayed the importance of data and considered the calibration of models to data to be unnecessary. This anti-data stance alienated scientists and economists who focused on data analysis. He also asserted that good models shouldn't be used to make predictions. This seemed contrary to the scientific method, where a theory (model) is deemed successful only if it correctly predicts novel phenomena. How can you judge a model useful if it doesn't make correct predictions? There was a third curiosity. Forrester wasn't interested in measuring the complete current state of the company or calibrating a model to match data; he only wanted a few measurements as inputs to the servo controlling logic. The simulation model was used to design and test the servo rule, but was not to be used in day-to-day control, which was the role of the servo rule.

These three assertions/attitudes are out of step with what turned out to be modern approaches to system dynamics. As this paper will illustrate, the field can and does do much better now. But although they seemed odd to critics, they were a self-consistent and coherent view of how to make the world better. We'll call it the Servo paradigm, because before he pioneered the; invention of modern computers and system dynamics, Jay Forrester was a world-class designer of servo-control devices. In the mathematical and practical world of servomechanisms, data is of little importance, predictions are in some senses irrelevant, and it is unnecessary to measure the state of a system to improve its state. Figure 1 shows the overall process of the servo paradigm of system dynamics.

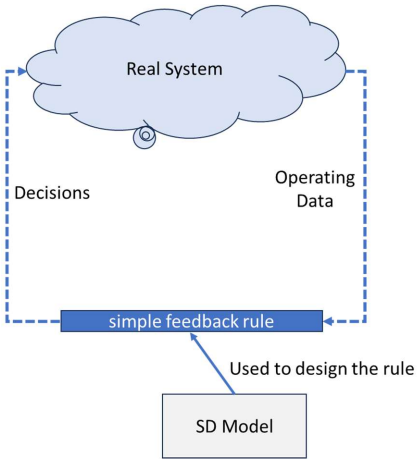

Figure 1: *Servo Paradigm. System Dynamics model is used (offline) to design and test a simple, algebraic feedback rule to compute a decision stream. The feedback rule's purpose is to steer the real system from any unfavorable behavior model into the desired mode. The modes are qualitative; no attempt is made to predict specific numerical trajectories.*

The Model Predictive Control (MPC) paradigm

System dynamics has since evolved to often embrace modern, compute-intensive statistics, validation testing, calibration of the model to match historical data, and numerical predictions with explicit confidence bounds. It then becomes possible to introduce decisions and policies that steer the system as efficiently as possible from its current state to a desired, even optimal state. With the modern tools, this transition can be both faster and more secure, and the end state can be chosen to optimize measures of system behavior. The techniques for doing this are now described in the control theory literature as "model predictive control" (Rawlings et. al., 2024) and many of them are documented in the modern definitive system dynamics text *Business Dynamics* (Sterman 2000). Model Predictive Control is the logical combination of modern SD models and big computation. Instead of simple feedback policies, the SD model itself is the core of the feedback loop. It is like optimizing a servo for the actual scenario as the scenario unfolds. The result is robust and higher performing than servo solutions, and it is particularly well-suited for real-world applications of SD, when policies can be regularly changed as history unfolds, such in model-based management. Figure 2 shows the MPC paradigm of system dynamics.

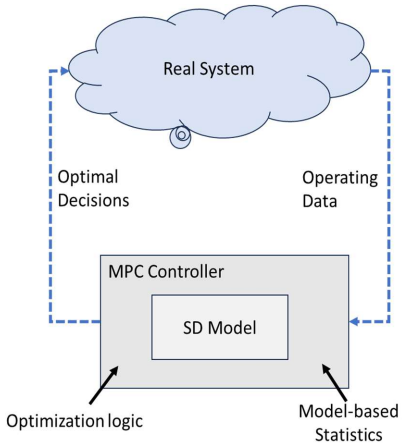

Figure 2: *Model Predictive Control (MPC) Paradigm. MPC uses the System Dynamics model as the core of an online detection and optimization feedback control. Its purpose is to detect the state of the real system and steer it onto an optimal trajectory.*

The control strategy of MPC uses the system dynamics model to anticipate future behavior and determine optimal control actions over a finite planning horizon. At each control step, an optimization problem is solved over this horizon, and the first segment of the optimal solution is applied as the current decision. As soon as new data makes it possible to measure an updated state of the real system, the process repeats with a new optimal solution, of which the first segment is implemented. In this way, MPC replaces the simple feedback equation of a servo with a full predictive simulation model in the control loop. MPC is an iterative cycle, where each iteration may range from seconds in a robot, to minutes in chemical plants to months in business decisions (see Figure 3).

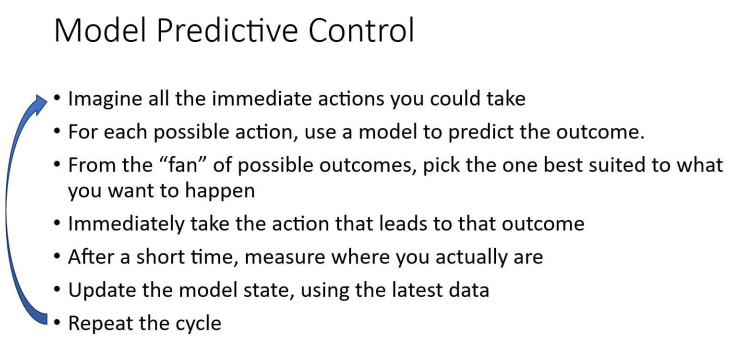

***Figure 3**: The MPC iteration.*

Hypothesis and test bed

We posit that the transition to data-tested, predictive SD models has enabled more effective improvements to the performance of real-world systems. To test this hypothesis, we built a simulation test bed in which the servo approach can be directly compared with the model predictive control approach. The test bed consists of one model that represents the real system to be improved. A second model represents a system dynamics model of the first model. Both models contain representations of decision making that resemble servo control logic, likely with inferior parameter choices. The second "controller" model can be used to improve the servo parameters or logic, as in the original paradigm of SD. But in the same test bed, the second model can be embedded as an active part of a control logic that replaces the servo with model predictive control logic. This latter use requires that as the "real" model steps through time, it obtains decisions not from its servo equations, but from repeated full optimization of the controller model to compute the best decisions. Therefore, we can use the test bed to compare servo results with model predictive control results for exactly the same "real" system, the same SD model, facing exactly the same starting conditions and inputs.

The remainder of this paper is structured as follows. In section 2, we describe the computation structure used to test our hypothesis. Section 3 provides an in-depth analysis of the test results. In section 4 we note some aspirational targets for the further evolution of SD, and in section 5 we offer a discussion and conclusions.

2 Methodology: Simulating the MPC Loop

Simulating MPC: two models create a single simulation

To explore MPC in simulations, two models must interact. One model represents the system to be managed (the *main model*), and the other, the *controller model* is a used to compute optimal decisions in the main model. In this way, there is a feedback loop from main model to the controller model and back to the main model (as shown in figure 2). Some of the flows in the main model, instead of being immediate functions of the states of the main model, are instead computed by optimizations of the controller model. The controller model is in general a simpler version of the main model, echoing the fact that any system dynamics model is a simplification of the real system. However, to keep things clear, this paper uses a controller model which is the same structure and complexity as the main model.

Architecture for simulating MPC

The implementation relies on a code base in the Python programming language that interfaces with the Vensim simulation environment through its dynamic link library (DLL). Two separate DLL instances are loaded, each operating an independent copy of the model. This dual-instance architecture allows the main model to advance through simulated time while the controller model independently performs optimizations without disturbing the main model's state.

The interface layer provides methods for core simulation operations — running simulations, setting parameter values, loading and unloading configuration files — as well as for optimization, where Vensim's built-in optimizer is directed by specifying an optimization control file (defining the decision variables and their bounds) and a payoff definition file (defining the objective function to be maximized or minimized). Critically, the interface also exposes Vensim's *gaming mode*, which allows a simulation to be paused at discrete intervals, its state to be inspected, and parameter values to be modified before the simulation is resumed.

The simulation of MPC requires the interplay between two models. The procedure operates as follows:

1. Initial Optimization. Before the main model begins its time evolution, the controller model performs a preliminary optimization over a limited time horizon starting from the initial time. This optimization determines the initial values of the decision variables — that is, the policy levers available to the decision-maker. The results of this optimization are captured from Vensim's output file and parsed into a dictionary that maps variable names to their optimized values.

2. Main Model Initialization. The main model is configured with the full simulation time horizon and run once without optimization to establish a baseline trajectory. It is then placed into

gaming mode, which enables stepwise advancement through time. The optimized decision variable values obtained from the controller's initial optimization are injected into the main model as its starting policy settings. A game interval is configured to determine how many time steps the main model advances before control is relinquished back to the controller.

3. Iterative Control Loop. The main model is then advanced through time in discrete increments defined by an *update interval*. At the end of each increment, the following sequence is executed:

- **State Transfer:** The current values of all stock (level) variables in the main model are read out using the gaming interface. These values represent the full dynamic state of the system at the current point in time.
- **Controller Synchronization:** The controller model is re-initialized so that its simulation begins at the current time of the main model. Each stock variable in the controller is set to match the corresponding value just read from the main model, using dedicated initial-value parameters. This ensures that the controller's optimization starts from the actual current state of the system, rather than from a potentially divergent trajectory.
- **Rolling-Horizon Optimization:** The controller model's final time is set to the current time plus the *optimization interval* — a configurable parameter that defines how far into the future the optimizer looks. The optimization is then executed, yielding a new set of decision variable values that are optimal with respect to the payoff function over this forward-looking window.
- **Policy Update:** The optimized decision variable values are extracted from the controller's output and injected back into the main model through the gaming interface, updating the policy settings before the next time increment begins.

This cycle of advance–observe–optimize–update repeats until the main model reaches its final time. If the remaining simulation time at the end is shorter than the update interval, the game interval is adjusted accordingly to ensure the entire time horizon is covered. The gaming session is then formally terminated.

State Mapping Between Models

A central requirement of MPC is the faithful reproduction of the main model's state within the controller. This is accomplished through a mapping dictionary that associates each stock variable in the main model (e.g., "Production Capacity," "Backlog," "Sales Agents") with a corresponding initial-value parameter in the controller model (e.g., "Production Capacity INITIAL," "Backlog INITIAL," "Sales Agents INITIAL"). The controller model must be structurally prepared so that each stock accepts an exogenous initial condition, allowing its state to be

overwritten at the beginning of each optimization cycle. This structural modification is a prerequisite for applying the MPC approach to any given System Dynamics model.

3 Analysis: Testing the performance of servo vs MPC

To estimate the relative performances of servo control and MPC, we performed a series of simulations of the two methods under various scenarios.

The tests were performed by simulations of a (slightly altered version of) Forrester's corporate growth model (Forrester 1968). We used two copies of the model. One copy, the *Main Model*, played the role of a "real" company. The second copy, the *Controller Model*, played the role of a system dynamics model of the same company. The structure of Forrester's model is shown in Figures 4 and 5.

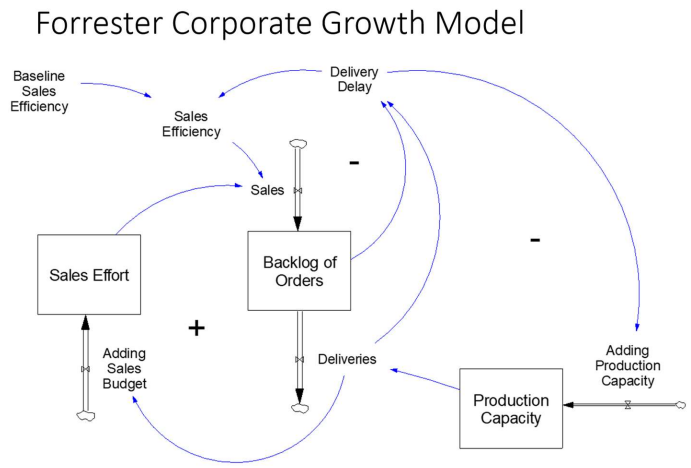

***Figure 4**: simplified overview of Forrester's model (Forrester 1968).*

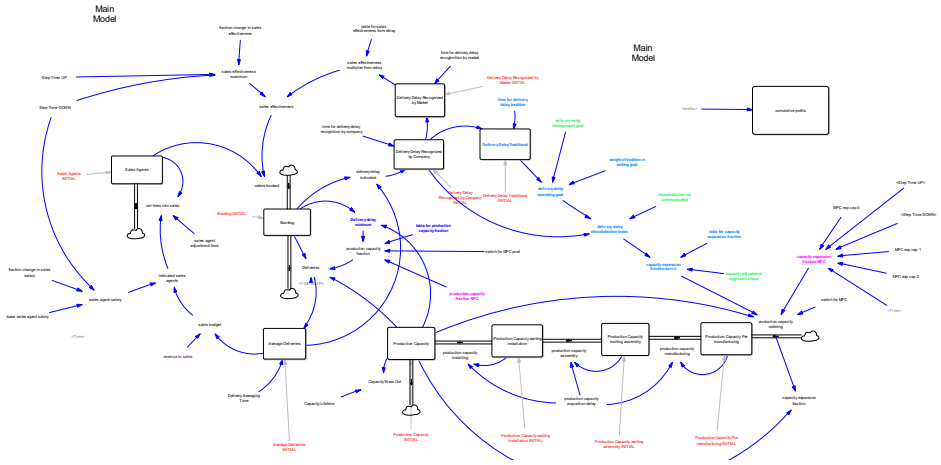

Figure 5: *The detailed structure of the main model (slightly modified from Forrester, 1968). The descending chain of variables colored green on the right side of the diagram constitute a servo logic for controlling delivery delay by adjusting production capacity.*

A single simulation of the Main model emulates several years of the actual company, but with the capacity-control servo deleted, leaving the capacity-expansion decision as an exogenous decision, implemented as a gaming variable. Each month the main model simulates, it obtains its capacity expansion input by waiting for the Controller model to perform an optimization, searching for the pattern of capacity expansions over several years that will maximize the cumulative profits. When the optimization is complete, the first month of that optimal series of capacity expansions is passed back to the main model, which then implements that new capacity order as part of simulating one month's "real" progress. After one month, the main model stops, the state variables at the end of the month are passed to the controller model, and the controller model repeats its optimization starting at the new time with the updated state variables from the main model, and the cycle continues until the main model has completed its simulation of 140 months.

External disturbances

Control strategies must be robust against disturbances. If there were no disturbances, it would be relatively easy to maintain desired performance. Therefore, any control strategy must be tested under a wide range of potential disturbances.

For simplicity and clarity, the only disturbances considered will be sudden changes in the efficiency of sales, as might be caused by a competitor introducing a new product. The disturbances will always have the time pattern shown in Figure 6. At month 25, sales productivity suddenly jumps to a new value, stays there until month 75, when it abruptly returns to its original value. The starting value is taken from Forrester's published model, at 400 units sold per person-month of sales effort. Figure 6 shows an example in which sales effectiveness drops in half and then recovers. We will also consider other magnitudes, for a total of five scenarios: -50% (as shown in the figure), -25%, 0 (no change), +25%, and +50%.

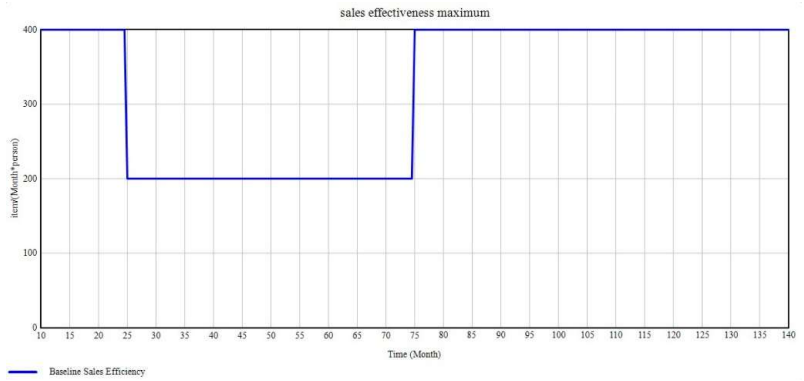

Figure 6: *The disturbance pattern used in the tests of this paper. The vertical magnitude may vary or even reverse, but each simulation uses this timing and shape.*

The comparison baseline: optimal servo control

The servo parameters were optimized to maximize cumulative profits, averaged over the five scenarios defined by the five magnitudes of change in sales effectiveness. The performance of the optimized servo varies from different degrees of disturbance, but the same servo parameters are used in each scenario. This is realistic, because only one servo can be used, but it must be designed to cope well with whatever unpredicted disturbance might occur.

We then performed five simulations, testing optimized servo controller in each of the five disturbance magnitudes. The results are shown in table 1. Unsurprisingly, more profits are made when the disturbance is more in favor of sales effectiveness.

	Cummulated Profits
Scenario	**Servo**
-50%	9.93
-25%	62.6
0%	180
25%	378
50%	709

Table 1: *Performance of the optimized servo controller in each of five scenarios, with varying degrees of disturbance in sales effectiveness.*

The relative performance of MPC

For each of the five scenarios, we also simulated the performance of MPC. In each simulation, the main model's equations for calculating new capacity orders were disabled. The main model was initialized but then paused while its initial conditions were transmitted by the Python code to the controller model (see table 2). While the main model was paused, waiting for a capacity decision, the controller model was simulated many times by the Vensim optimization algorithm to find the future monthly series of capacity-expansion decisions which would lead to maximum predicted profits. The first month of that series was then transmitted by the Python code back to the main model. Having received its capacity expansion decision (which would have normally come from servo logic within the model), the main model then simulated its first month of activity, and was again paused, cycle iterated. For each scenario, this iteration was repeated 140 times to yield a single simulation of the main model, supported by repeated optimizations by the controller model to supply the main model with capacity-expansion amounts for each month.

The results are shown in Table 2, an extension of Table 1. Its third column ("MPC") shows the performance of MPC under each disturbance scenario. The final column ("MPC % Gain") displays the percentage improvement obtained by using MPC in place of the optimized servo. MPC is significantly superior in every one of the tested scenarios, and the gain in performance grows with the magnitude of the disturbance, in both the bad and good directions.

	Cummulated Profits		
Scenario	**Servo**	**MPC**	**MPC % Gain**
-50%	9.93	27.2	180%
-25%	62.6	77.4	25%
0%	180	196	9%
25%	378	442	12%
50%	709	937	32%

Table 2: *Results of five MPC experiments (extension of table 1).*

Commented [M1]: Fixed by Matthias, based on numbers in the comparison email by David sent on 2026-03-19

Sensitivity to Prediction Errors

So far, for clarity we have dealt with a Controller Model which exactly matches the structure of the Main Model, thereby emulating an unrealistic situation in which a system dynamics model is making perfect predictions of a complex real company and its competitors and customers. In any actual application, the controller model, no matter how sophisticated as a model, will be a simplification of the real system, and its predictions will be imperfect. It is important, therefore, to test the effect of prediction errors on the performance of MPC.

For clarity, we used the following method to represent prediction errors. For each of the five disturbance scenarios, we focused on the -50% disturbance as the “real” scenario, meaning the main model always experienced a -50% depression in sales effectiveness. But we assigned a different sized depression in the controller model. For example, if the controller model has a sales effectiveness depression of zero, then it is operating with a prediction error of 50% in absolute percentage, or +100% of the actual depression of 50%. We performed such a simulation five times, with the controller model operating with five different magnitudes of prediction error. These five results could then be compared with the perfect prediction case with -50% sales effectiveness depression, from line 1 of Table 2.

The results are shown in Table 3, and they show that MPC can be highly robust to prediction errors. Because the controller model is imbedded in a feedback loop, even if its predictions are seriously in error, it still does a better job than the optimized simple servo, unless the prediction error is extreme. When the disturbance is a 50% drop in sales effectiveness, MPC is more than twice as good as the optimized simple servo, so long as the MPC prediction of the disturbance is less not much more than 50% in error.

	-50% Scenario			
Prediction for MPC	% Error	Cummulated Profits - MPC (M$)	% Lost from Error	% Better than Servo's 9.9
-75%	-50%	25.1	8%	154%
-50%	0	27.2	0%	175%
-25%	+50%	26.7	2%	170%
0%	+100%	24.1	11%	143%
25%	+150%	16.9	38%	71%
50%	+200%	7.05	74%	-29%

***Table 3**: Sensitivity of MPC to errors in the model's predictions.*

Commented [M2]: New figure, as sent by David on 19th of March 2026

How can MPC perform so well when its predictions are potentially so bad? Figure 7 shows the capacity expansion decisions made by MPC in the case that the real disturbance is -50%, but the MPC controller model falsely predicts that the disturbance will be only half that: -25%. When the disturbance first hits at month 25, the MPC as expected, reacts less than optimally. It continues to order new capacity. But as the resulting excess capacity builds up, the controller model finds it can increase profits by reducing capacity expansion. The controller model continues to falsely predict the depression in sales will immediately fall to -50%, but the more excess capacity accumulates, the more the capacity order is reduced by the controller's calculation of predicted profits.

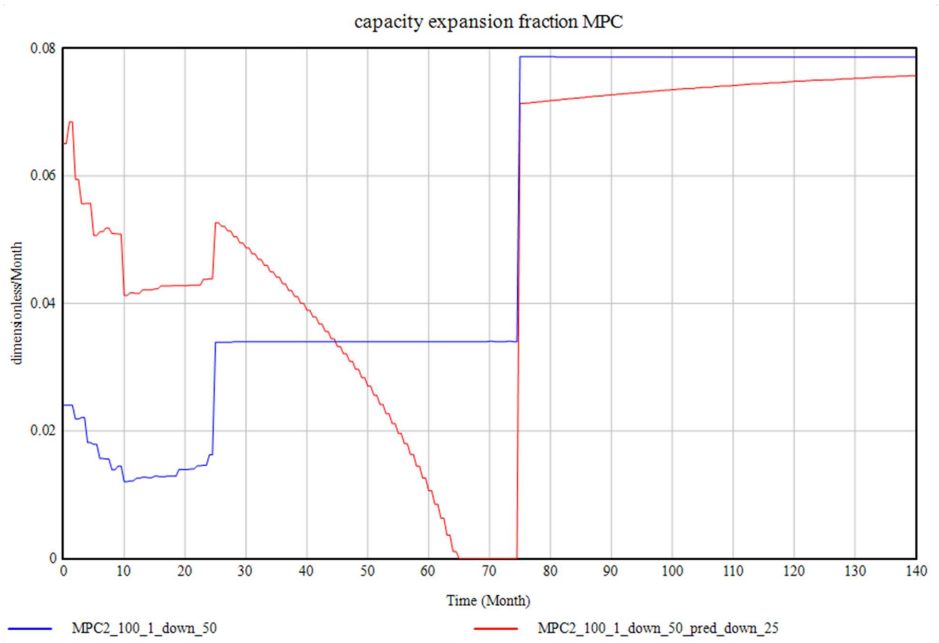

***Figure 7**: Capacity expansion trajectories of MPC with perfect forecast, vs. 50% error. The red trace shows how MPC reacts when it predicts a 25% drop in sales, but actually the drop it twice that. In other words, MPC has predicted only half the magnitude of the actual problem.*

Commented [M3]: New figure sent by David on 19th of march 2026

4 Cautions and Aspirations

We have asserted that the modern practice of applied SD has moved away from a qualitative focus on modes and servo logic, and has come to resemble model predictive control (MPC). The first results from our test bed suggest that this transition may have greatly amplified the ability of SD to improve the performance of systems.

However, there is one aspect of MPC that has not yet become common in SD applications. MPC uses its predictive model to compute decisions at regular intervals over long stretches of time. SD practice still tends to create a model and use it perhaps only once to find and promote a superior decision. Subsequent, regular use of the model as an ongoing policy is rare.

For example, in the case of an insurance company (Corcoran et. al., 2012), a system dynamics model accurately predicted the future behavior of a competitor, the company acted on that prediction, and everything came out OK. But the company did not implement full MPC, stopping short of updating the model and policy optimization each month while waiting for the predicted event to occur. The omission of updates clearly increases the risk from prediction errors, and a good use of the new test bed will be to quantify the expected loss in performance from such omissions.

Observant readers might argue that comparing MPC vs optimize servos is not a fair comparison, as MPC has more degrees of freedom compared to traditional optimized servos. In the optimized servo setting, we keep the parameters of the model constant throughout the entire runtime of the model and rely on the servos to manage the system. Whereas in MPC, we are changing the values of key decision parameters as the simulation unfolds over time. However, this exactly supports our findings that rather than using a model as part of a “one-time” engagement to derive policies and strategies we should be looking to incorporate model use as part of ongoing management.

However, our preliminary results suggest serious improvements are possible if organizations would imbed SD models in ongoing decision streams, as in MPC. We hope to see organizations grow to take advantage of the full capabilities of system dynamics models, with resulting major improvements in performance.

5 Conclusion and next steps

In this paper, we have observed that system dynamics practice has evolved from a qualitative, "servo" approach to a quantitative, data-based, predictive capability that resembles the control theory field of Model Predictive Control (MPC). Our hypothesis was that the transition to data-tested, predictive SD models has enabled more effective improvements to the performance of real-world systems. The first experiments suggest that SD has indeed become more effective.

Further research could better quantify the preliminary magnitudes from this study, and could break down the gains attributed to specific aspects of system dynamics practice.

References

Corcoran, S., A.K. Graham, W.B. Arthur and D.W. Peterson (2012): A competitor's strategy unclothed: How indirect measurement justified not fighting an insurance price war. Proceedings of the 30th International Conference of the System Dynamics Society, St. Gallen. Available online from: https://proceedings.systemdynamics.org/2012/proceed/papers/P1297.pdf [2026-03-17].

Forrester, J.W. (1961): Industrial Dynamics. Productivity Press, Cambridge, MA.

Forrester, J.W. (1968): Market Growth as Influenced by Capital Investment. Industrial Management Rev. (MIT), 9(2), 83-105.

Rawlings, J.B., D.Q. Mayne, and M.M. Diehl (2024): Model Predictive Control: Theory, Computation, and Design, 2nd Edition. Nob Hill Publishing, Santa Barbara, CA.

Sterman, J.D. (2000): Business Dynamics. Irwin McGraw-Hill, Boston, MA.