Proceedings of the 2008 Winter Simulation Conference
S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler eds.

CONSTRUCTING BUSINESS SIMULATIONS WITH SERVICE PATTERNS

Richard B. Lam

IBM T.J. Watson Research Center
1101 Kitchawan Road, Route 134
Yorktown Heights, NY 10598 USA

ABSTRACT

Typically, system dynamics-based simulations of business processes are constructed in an ad hoc manner, with a modeler creating low-level components and defining interrelationships one-by-one. As the numbers of process components or variables grow, the resulting model becomes ever more difficult to manage and extend. This paper discusses the definition and use of high-level business patterns that have predefined system dynamics sub-models. These patterns enable rapid construction of arbitrarily complex system dynamics models of business processes through abstraction, reuse, and unification of sub-model elements.

1 INTRODUCTION

A number of authors (e.g., see Affeldt 1999) address the difficulties associated with ad-hoc construction of system dynamics models. The bottoms-up selection of stocks, flows, and variables, interconnected for appropriate modeling of a particular domain, requires system dynamics expertise beyond that of many domain experts. In many instances, this hampers the use of system dynamics as a viable platform for simulation.

In an early paper, Tignor (2001) asserts that a design patterns approach (Vlissides et.al. 1995) is needed for system dynamics modeling to achieve further impact. Such patterns, or building blocks, provide reusable solutions to modeling problems within a given context. Tignor also provides a summary of other authors' pattern approaches to modeling.

Hines (2005) describes a "molecules" approach, implemented in at least one system dynamics software package. Molecules are defined as sets of primitive stocks, flows, and auxiliary components connected to form substructures. The intent was specification of a catalog of "atomic" substructures that captured various system dynamics concepts (e.g., level, decay, smoothing, etc.). These substructure models are subsequently reused to construct more complex system dynamics models.

Unfortunately, the original categorization of the molecules showed that the individual molecules were defined at different levels of abstraction – from primitives such as "Stock" up to higher-level concepts like "Stock protected by Stock". In a subsequent working paper (Hines 2005), this discrepancy was corrected and a specialization hierarchy was created, starting with initial components of stocks, flows, and policies, where policies represent decision rules which control flows and thus affect accumulations (i.e., stocks). The resulting catalog of models allows substitution of sub-models within other models to ensure that as the components grow in numbers and relationships, the model maintains its conceptual validity and functionality.

Bauer and Bodendorf (2005) and Madachy (2006) also present component-based modeling approaches that allow construction of complex models from primitive components or microstructures.

From the number of papers in this area, it seems clear that pre-built patterns describing partial models (either generic or domain-specific) are essential to rapid construction of system dynamics models. In this paper, I discuss a small set of patterns specifically for simulation of business services. The patterns are driven by cause-effect diagrams that delineate business decision points that are an inherent part of each sub-model. Then, whenever a pattern or set of combined patterns are instantiated to run a system dynamics simulation, the decision points specify the user inputs or controls on the process being modeled. In addition, the pattern components can be renamed, reused and combined in different business contexts, as illustrated in some example simulations.

2 BUSINESS SERVICE PATTERNS

2.1 Profit

The first business service pattern, Profit, is illustrated by the cause-effect diagram in Figure 1. This is a simple calculation-based pattern that relates unit price (P) and unit cost (C) to the unit profit (P – C), actual profit margin (M = (P – C)/C), and gap (G = T - M) between the actual profit margin and a target profit margin (T). The signs on the connectors represent the effect of increasing the component at the connector's tail on the value of the component at the connector's head.

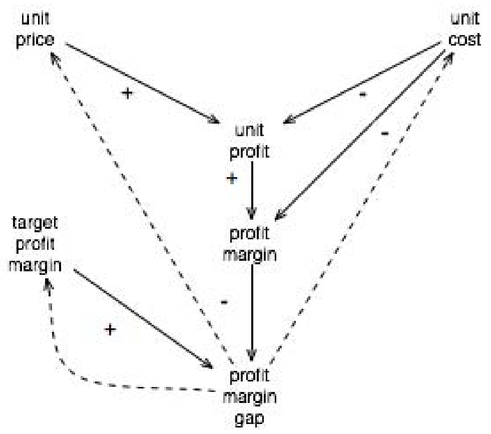


Figure 1: The Profit pattern cause-effect diagram

The dashed connectors represent three typical decisions that a business services user would make based on looking at the profit margin gap. There are only three business decisions possible that will affect the gap: change the unit price, change the unit cost, or modify the target profit margin itself.

Such computations occur repeatedly in business service simulations, so it is appropriate to take the cause-effect model and design a system dynamics sub-model that captures the components and decisions (Figure 2). Note that diamonds in this figure illustrates the decision points.

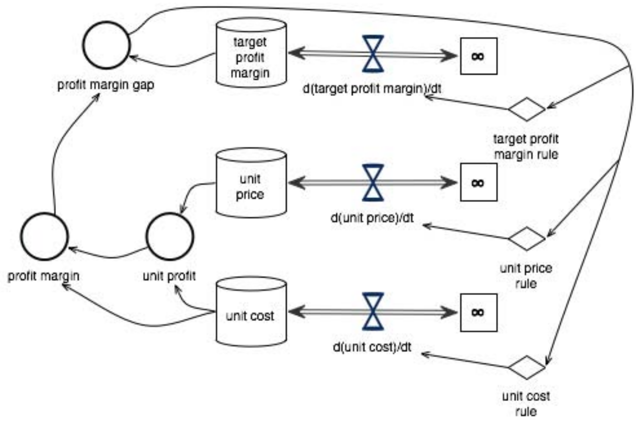


Figure 2: The systems dynamics implementation of the Profit pattern

Each decision affects the flow, or change, that occurs to change one of the three stocks that ultimately affect the value of the profit margin gap that triggered the decision. In a later section, I discuss how this pattern is combined with other patterns to create more complex business service simulations.

2.2 Sense and Respond

A more general pattern, actually a meta-pattern, is Sense and Respond. The cause-effect diagram of this pattern is shown in Figure 3. It will become clear in a moment why this is a meta-pattern. In general, this pattern deals with demand, capacity (or supply), and the gap between them.

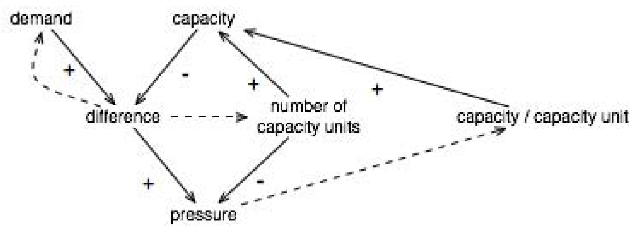


Figure 3: The Sense and Respond meta-pattern

Here there is a difference, or gap, between demand and capacity. Capacity is computed generically by multiplying the total number of capacity units by the available capacity of each unit. The pressure felt by a capacity unit is computed by dividing the difference (gap) by the total number of capacity units – this is analogous to the physical definition of pressure, which is the total force acting on a unit area. In this case, the force is the unsatisfied demand and the unit area is represented by the number of capacity units that can act to satisfy the demand.

With these relationships defined, a business user would examine the value of difference and make two possible decisions: modify the incoming demand, or change the number of capacity units acting to fill the demand. In addition, a business user would look at the value of pressure and make a decision to change the capacity per capacity unit. It is interesting that this decision could be realized

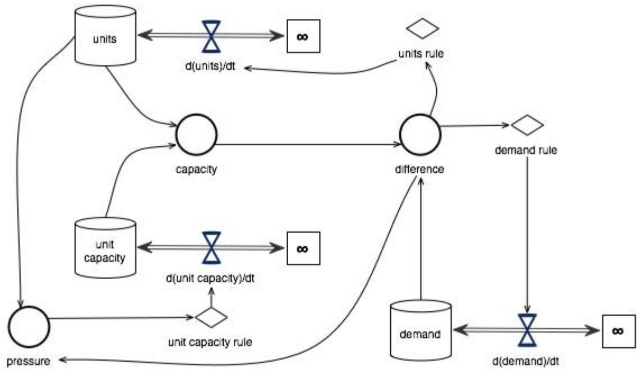


Figure 4: The system dynamics view of the Sense and Respond meta-pattern

in two ways, depending on the context. One way is by increasing (or decreasing) the capacity expected from each unit (effectively making a capacity unit produce more (or less). The second way is to change the number of capacity units (the denominator) that act to generate a given capacity (the numerator). The net effect is a change in output quality or production time).

Figure 4 shows the system dynamics sub-model corresponding to Figure 3, again with three decision nodes that affect the flows. This system dynamics meta-model can be used to implement any of the next three patterns.

2.2.1 Work Resource Pattern

We can now see why Sense and Respond is a meta-pattern. Consider the notions of demand and capacity applied to a services business where we are interested in the resources applied to a services contract. We can re-label the Sense and Respond pattern components to reflect their use in a work resource model.

As shown in Figure 5, demand and capacity refer to the amount of services work demand and services work capacity for a business. The difference between them is relabeled as work, referring to either a gap or glut, depending on whether there are sufficient resources available to service the incoming demand.

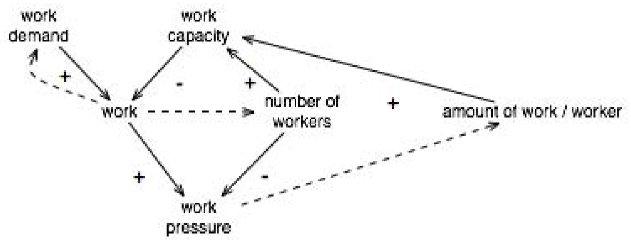


Figure 5: The Work Resource pattern

Work pressure is defined as the work gap or glut divided by the number of workers – thus this value reflects the amount of extra work each worker would face based on the demand. So business decisions on changing the incoming work demand or total number of workers are driven off of the work gap/glut value, while work pressure affects the amount of work per worker. If the work pressure is high, this either forces the amount of work for a given worker to increase (i.e., work harder and/or work longer hours to produce more), or forces the number of workers responsible for a given amount of work to decrease (perhaps with a decrease in work quality).

This pattern is consistent with the results reported by Oliva and Sterman (2001) in their analysis of quality erosion in services. The pattern also works at different organizational levels of a services business, where it can be used to model teams, departments, service towers, etc.

2.2.2 Market Share Pattern

Another specialization of Sense and Respond is a Market Share pattern, where target market share represents the demand component, and total sales represents the capacity acting to meet the demand. The total sales is given by the number of salespersons multiplied by the sales per salesperson (or sales target). The market share gap is the difference between the desired or target market share and the actual market share captured due to sales.

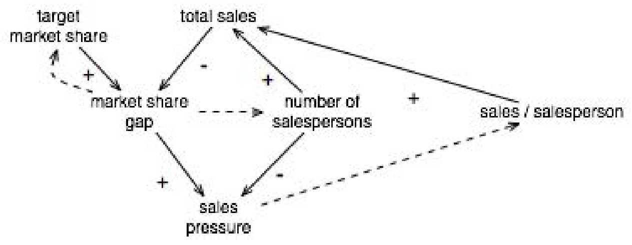


Figure 6: The Market Share pattern

Sales pressure is the market share gap divided by the total number of salespersons, and reflects the number of additional sales that each salesperson must make to satisfy the target market share set by the business. This sales pressure drives the decision on how to set each salesperson's sales target.

2.2.3 Revenue Pattern

A third instantiation of the Sense and Respond pattern considers a services business from a revenue standpoint. The Revenue pattern is shown in Figure 7. In this case, a business decision-maker would examine the revenue gap between a target revenue number and the actual revenue realized. Actual revenue is computed by multiplying the number of clients by the amount of revenue per client (in the case of fixed price service contracts). The revenue pressure is the amount of revenue gap that would need to be filled by each client, either through a service price increase to the client or the addition of more clients.

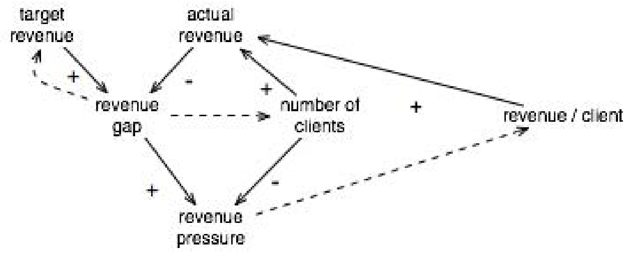


Figure 7: The Revenue pattern

3 PATTERN-BASED SIMULATIONS

The small set of four patterns above provides a rich space for modeling service businesses. The patterns have been used for rapidly constructing system dynamics simulations for a number of internal business modeling projects. Two simple examples are presented here.

3.1 A Revenue-Profit example

The Revenue and Profit patterns can be combined rapidly to construct a simulation where we wish to model the business impact of decisions on target revenue, target profit margin, and unit price. In this example, we assume a services business has fixed price contracts for a replicable service to be delivered. Thus, we can combine the Revenue and Profit patterns by unifying their two respective components, revenue per client and unit price. The resulting pattern diagram is shown in Figure 8.

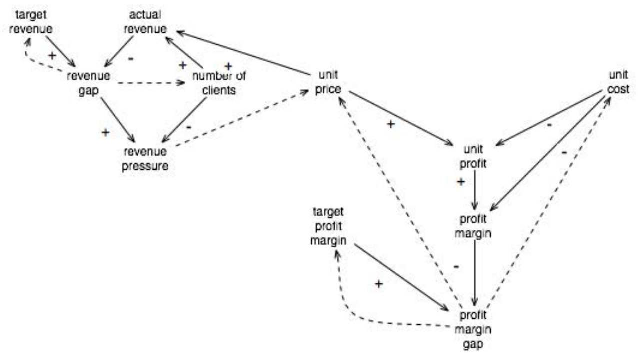


Figure 8: A combination of two patterns, with the revenue per client component in the Revenue pattern unified with the unit price component in the Profit pattern

In combining the patterns, it is immediately apparent to a modeler that there are five possible decisions to allow a user to make when running the simulation in a user interface (UI). The decision to change the unit price is now based on both revenue pressure and profit margin gap. Also, the unification of revenue per client with unit price also unifies the components unit capacity rule with unit price rule, and d(unit capacity)/dt with d(unit price)/dt, so there are fewer components than in the two models considered separately. The resulting system dynamics model is depicted in Figure 9.

An implementation of this pattern combination was written in a Python script. The Revenue and Profit pattern instances were created from base classes defining the pattern components and connections. A `unify()` method was called to unify the components discussed above. The script runs on a server that carries out the system dynamics computations and communicates via XML messages with a client program. The client implements a user interface for setting initial conditions, providing users with the ability to make decisions within the model, and plotting the results. A screen capture of the user interface for this example, in this case written with NetLogo, is shown in Figure 10.

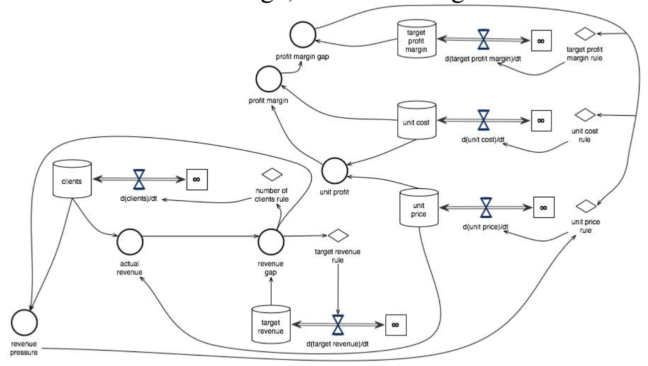
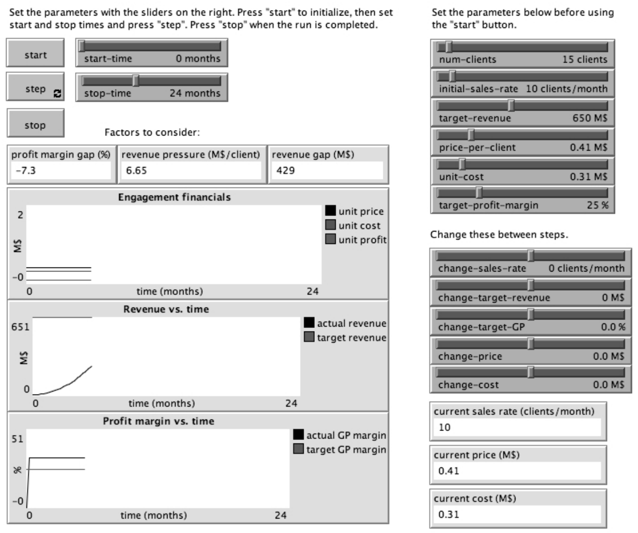


Figure 9: The system dynamics model resulting from a combination of the Revenue and Profit patterns



Set the parameters with the sliders on the right. Press "start" to initialize, then set start and stop times and press "step". Press "stop" when the run is completed.

start-time 0 months
stop-time 24 months

Factors to consider:

profit margin gap (%) -7.3
revenue pressure (M$/client) 6.65
revenue gap (M$) 429

Set the parameters below before using the "start" button.

num-clients 15 clients
initial-sales-rate 10 clients/month
target-revenue 650 M$
price-per-client 0.41 M$
unit-cost 0.31 M$
target-profit-margin 25 %

Change these between steps.

change-sales-rate 0 clients/month
change-target-revenue 0 M$
change-target-GP 0.0 %
change-price 0.0 M$
change-cost 0.0 M$

current sales rate (clients/month) 10
current price (M$) 0.41
current cost (M$) 0.31

Figure 10: The NetLogo user interface to the Revenue-Profit system dynamics simulation. The UI contains five controls corresponding to the decision points in the combined patterns.

3.2 A Three-pattern example

The previous model can be extended further by including the Market Share pattern. Suppose we wish to look at the influence of pricing decisions on market share as well as on profit margin. We need to connect the unit price (unified again with revenue per client from the Revenue pattern) with the total sales component in the Market Share pattern. This can be done by introducing an external component, say sales effectiveness, that is computed based on unit price. Then sales effectiveness is introduced as a factor that modifies the computation of total sales in Market Share.

In a similar manner, we might wish to connect the number of salespersons in the Market Share pattern to the unit cost, so that a decision to increase the number of salespersons is now linked to cost and profit margin. Thus, the complexity of a single decision becomes apparent in its effects on components and other subsequent decision types.

Figure 11 shows one possible combination of these three patterns, and Figure 12 is a NetLogo screen capture for a simulation built using the combined patterns.

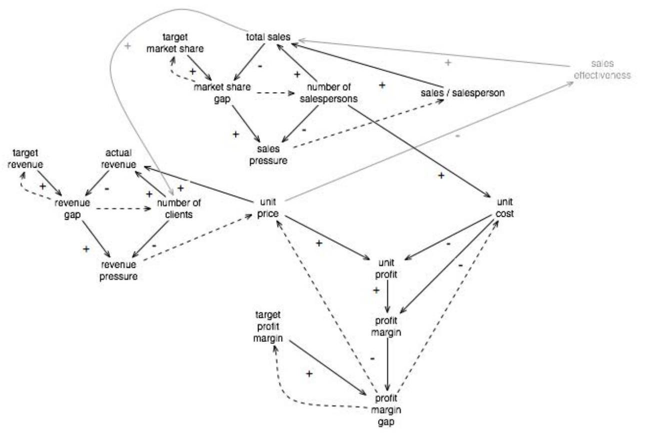
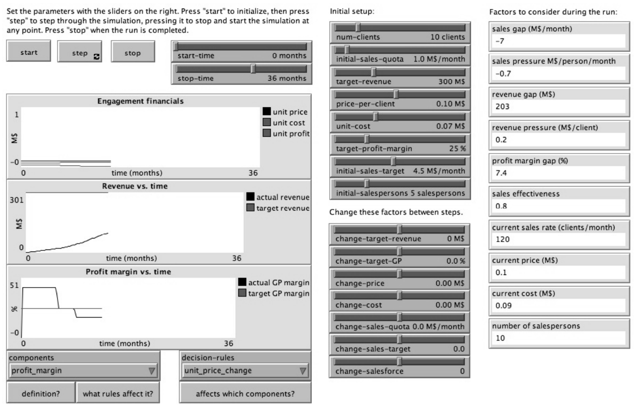


Figure 11: A possible combination of the Revenue, Profit, and Market Share patterns for a hypothetical services business.



Set the parameters with the sliders on the right. Press "start" to initialize, then press "step" to step through the simulation, pressing it to stop and start the simulation at any point. Press "stop" when the run is completed.

start-time 0 months
stop-time 36 months

components: profit_margin
decision-rules: unit_price_change

Initial setup:

num-clients 10 clients
initial-sales-quota 1.0 M$/month
target-revenue 300 M$
price-per-client 0.10 M$
unit-cost 0.07 M$
target-profit-margin 25 %
initial-sales-target 4.5 M$/month
initial-salespersons 5 salespersons

Change these factors between steps.

change-target-revenue 0 M$
change-target-GP 0.0 %
change-price 0.00 M$
change-cost 0.00 M$
change-sales-quota 0.0 M$/month
change-sales-target 0.0
change-salesforce 0

Factors to consider during the run:

sales gap (M$/month): -7
sales pressure M$/person/month: -0.7
revenue gap (M$): 203
revenue pressure (M$/client): 0.2
profit margin gap (%): 7.4
sales effectiveness: 0.8
current sales rate (clients/month): 120
current price (M$): 0.1
current cost (M$): 0.09
number of salespersons: 10

Figure 12: The NetLogo UI for the model in Figure 11

Another feature of the UI is a built-in voice interface. This voice interface does a text-to-speech rendering of answers to three different types of questions. These questions allow the user to ask for:

1. a definition of any component, which is obtained from the pattern's component specification (e.g., "The revenue gap is the difference between the actual and target revenue. This variable's value is expressed in units of dollars.");
2. a list of the rules that affect the value of any component – this list is obtained by following the cause-effect diagram links of the pattern combination from the component back to the possible decision rules (e.g., "Profit margin is affected by the following decision rules: change in unit price, change in unit cost, change in number of salespersons.");
3. a list of the components that are affected by any particular decision rule – this list is obtained by following the cause-effect diagram links from the decision rule to the list of affected components (e.g., "Changing the target profit margin affects the following components: profit margin gap.").

The text-to-speech interface is useful to avoid showing the user a complex cause-effect or system dynamics diagram that requires significant explanation. It provides the user with feedback on the potential effects of their decisions, either before or during a simulation run.

4 SUMMARY

This paper presents a set of service patterns based around typical decisions made by business stakeholders. The patterns capture standard computations in addition to demand vs. capacity (or target vs. actual), pressure, quality, and productivity relationships, while delineating the input parameters that go into business decision-making. By combining these patterns through reuse, unification of components common to multiple patterns, and connections among components through other components external to the patterns, complex system dynamics models can be constructed rapidly. The decision points in the pattern models dictate the inputs that a user interface designer must include in a simulation UI based on the resulting model. And the cause-effect diagram that results from combining the patterns with each other or external components affords extra UI functionality for users to determine the impact of their decisions.

REFERENCES

Affeldt, J.F. 1999. Application of System Dynamics (SD) Simulation to Enterprise Management. In *Proceedings of the Winter 1999 Simulation Conference*, ed. P.A. Farrington, H.B. Nembhard, D.T. Sturrock, and G.W. Evans, 1496-1500.

Bauer, C. and F. Bodendorf. 2005. Component-Based Composition of System Dynamics Models. In *Proceedings of the 19th European Conference on Modelling and Simulation*, ed. Y. Merkuryev, R. Zobel, and E. Kerckhoffs, ISBN 1-84233-112-4.

Hines, J.H. 2005. Modeling with Molecules 2.02. Available via <http://www.vensim.com/molecule.html> [accessed June 3, 2008].

Hines, J.H. et.al. 2005. Construction by Replacment: A New Approach to Simulation Modeling. MIT Center for Coordination Science Working Paper No. 225, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Madachy, R. 2006. Reusable Model Structures and Behaviors for Software Processes. In *Software Process Change*, LCNS 3966:222–233.

Oliva, R. and J.D. Sterman. 2001. Cutting Corners and Working Overtime: Quality Erosion in the Service Industry. *Management Science* 47:7, pp. 894-914.

Tignor, W. 2001. Design Pattern Language and System Dyanmics Components. Available via <http://www.systemdynamics.org/conferences/2001/papers/Tignor_1.pdf> [accessed June 3, 2008].

Vlissides, J., R. Helm, R. Johnson, and E. Gamma. 1995. *Design patterns: Elements of reusable object-oriented software*. Reading: Addison-Wesley.

AUTHOR BIOGRAPHIES

RICHARD LAM is a Research Staff Member at the IBM T.J. Watson Research Center in Yorktown Heights, NY. He has a Ph.D. in physical science and a broad range of interests spanning science, business simulations, mathematics and software engineering. He currently focuses on risk analytics with applications to complex business processes.