Fiftieth Annual Allerton Conference
Allerton House, UIUC, Illinois, USA
October 1 - 5, 2012

Splash: Simulation Optimization in Complex Systems of Systems

Peter J. Haas, Nicole C. Barberis, Piyaphol Phoungphol, Ignacio G. Terrizzano,
Wang-Chiew Tan, Patricia G. Selinger, and Paul P. Maglio*

Abstract— **Decision-makers increasingly need to bring together multiple models across a broad range of disciplines to guide investment and policy decisions around highly complex issues such as population health and safety. We discuss the use of the Smarter Planet Platform for Analysis Simulation of Health (Splash) for cross-disciplinary modeling, simulation, sensitivity analysis, and optimization in the setting of complex systems of systems. Splash is a prototype system that allows combination of existing heterogeneous simulation models and datasets to create composite simulation models of complex systems. Splash, built on a combination of data-integration, workflow management, and simulation technologies, facilitates loose coupling of models via data exchange. We describe the various components of Splash, with an emphasis on the experiment-management component. This latter component uses user-supplied metadata about models and datasets to provide, via an interactive GUI, a unified view over all of the parameters in all of the component models that make up a composite model, a mechanism for selecting the factors to vary, and a means for allowing users to easily specify experimental designs for the selected factors. The experiment manager also provides a mechanism for systematically varying the inputs to the composite models. We show how the experiment manager can be used to implement some simple stochastic-optimization functionality by implementing the Rinott procedure for selecting the best system. We also implement a sensitivity-analysis method based on a fractional-factorial experimental design. We demonstrate this technology via a composite model comprising a financial-rate model and a healthcare payer model.**

I. INTRODUCTION

Simulation-based optimization is a powerful and increasingly popular approach to the design and operation of highly complex systems over a wide variety of domains. For instance, the current list of test problems in the SimOpt.org library [1] includes applications to vehicle routing, supply chains, healthcare facilities, fisheries management, finance, call centers, voting machines, air transportation networks, and more. Other recent application domains have included electrical grids [2] and environmental policymaking [3]. Methodology for simulation optimization has developed along with applications; see, e.g., Chapters 17–21 in [4].

Currently, simulation optimization algorithms are typically applied to individual, domain-specific simulation models to solve relatively contained optimization problems. Simulation is increasingly being used, however, to guide investment and policy decisions around highly complex issues such as population health and safety [5]. In this setting, decision-makers increasingly need to bring together multiple models across a broad range of disciplines. Such model composition is required to capture the behavior of complex "systems of systems" and gain synergistic understanding of highly complicated problems, avoiding unintended consequences of policy, investment, and operational decisions; see, e.g., [6], [7] in the setting of food, climate, and health. This composition task is extremely hard, because domain experts have different worldviews, use different vocabularies, sit in different organizations, and have often invested considerable effort in developing and implementing their models using different programming paradigms and development platforms. So how might widely disparate simulation models be combined, and what are the implications for simulation-optimization methodology? These questions are the focus of this paper.

Our approach to enabling cross-disciplinary modeling and simulation is embodied in the Smarter Planet Platform for Analysis and Simulation of Health (Splash). Splash [8] is a prototype platform for combining existing heterogeneous simulation models and datasets to create composite simulation models of complex systems. To facilitate interdisciplinary collaboration and model re-use, Splash facilitates loose coupling of models via data exchange, building upon and extending existing data integration technology. The component models run asynchronously and communicate with each other by reading and writing datasets. Typically, data transformations between models are needed to ensure compatibility. Such transformations are designed semi-automatically in Splash: the modeler uses an intelligent GUI to specify a given transformation, and Splash then automatically compiles the specification into runtime code. The key ingredient for detecting incompatibilities and designing transformations, as well as for executing composite models, is user-supplied metadata about each component model and dataset. Splash shares some features with the CIShell open-source platform for software interoperability [9], but is specifically tailored to simulation modeling and analysis, providing tools for semi-automated data transformation, time-series alignment, experiment management, and more, as discussed in what follows.

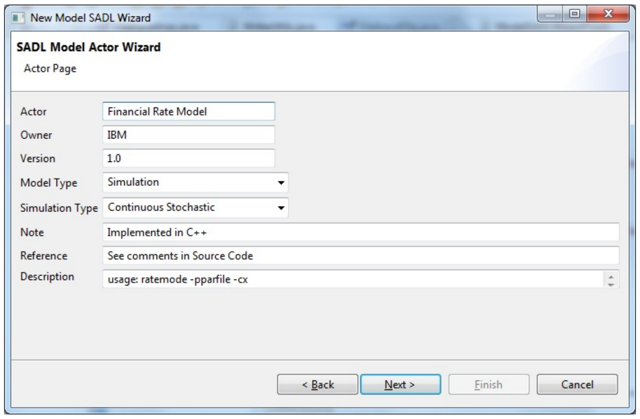
Our design philosophy, inspired by recent developments in information management, contrasts with more traditional approaches. These include (1) writing a single monolithic model, (2) creating component models that are then compiled together (see, e.g., [10], [11]), (3) adopting common standards and interfaces such as DEVS [12] or OpenMI [13], or (4) deploying distributed-simulation frameworks such as the High Level Architecture [14]. In the latter approach, custom communication logic is added to pre-existing models,

*The authors are with IBM Research–Almaden, San Jose, CA 95120, USA phaas@us.ibm.com which are then run in a tightly synchronized manner based on the exchange of time-stamped events. All of these existing approaches have drawbacks that hinder cross-disciplinary collaboration. Monolithic models can be difficult and expensive to build, verify, validate, and maintain—see, e.g., [15] or [16, pp. 4–6]—and require fine grained collaboration across disciplines and organizations. Both traditional component modeling and distributed simulation approaches typically require extensive re-coding of existing models, as well as unrealistic requirements with respect to use of common standards across heterogeneous scientific and engineering disciplines; see [8] for further discussion. Splash attempts to overcome these barriers by combining and extending information-integration, workflow-management, and simulation technologies.

In Section II, we describe the key elements of Splash that enable the design and execution of a composite simulation model. This discussion both summarizes and updates the description of the Splash platform given in [8]. We then (Section III) describe Splash's experiment-manager component, which allows systematic execution of a composite model over different sets of experimental parameters specified by the user. This component requires extensions to the original metadata language, and provides a GUI that consolidates the entire set of model parameters and permits design of experiments. Experiments designed in the GUI can be described using an "experiment markup language" (EML) and saved for future re-use or modification. Next, we show how the new experiment manager and the EML language can be used to implement some simple stochastic-optimization and sensitivity-analysis functionality. Specifically, we implement (Section IV) an optimization component that allows Splash to select the best value for a control variable from among a small set of feasible values, using the Rinott [17] selection procedure, as enhanced by Nelson and Matejic [18]. We also implement (Section V) a main-effects analysis capability, based on an underlying two-level fractional-factorial design; see, for example, the textbook of Allen [19, pp. 70–73]. In Section VI, we apply this technology to a small composite model comprising a simple financial-rate model together with a healthcare payer model based on Park et al. [20]. We conclude in Section VII with a discussion of future directions.

II. COMPOSITE MODELING WITH SPLASH

In Splash, domain experts contribute, and use, component simulation models and data sources. (Statistical models and optimization models can also be handled by the system.) Contributors register their models and data sources in the *Splash repository*. A designer of a composite model can then discover these components, connect them together, set up and run simulation experiments, and subsequently analyze, visualize, and share the results. The new composite model, as well as any useful datasets generated during the simulation experiments, can be registered in the repository and thus be made available to other model designers. In this section, we


New Model SADL Wizard

SADL Model Actor Wizard

Actor Page

Actor	Financial Rate Model
Owner	IBM
Version	1.0
Model Type	Simulation
Simulation Type	Continuous Stochastic
Note	Implemented in C++
Reference	See comments in Source Code
Description	usage: ratemode -pparfile -cx

```
<actor name="Financial Rate Model"
  actor_type="model"
  model_type="Simulation"
  simulation_type="Continuous stochastic"
  owner="IBM"
  version="1.0"
  note="Implemented in C++"
  reference="See comments in Source Code">
  <description>
    usage: ratemode -pparfile -cx
  </description>
...
```

Fig. 1. SADL Wizard screenshot and snippet of resulting SADL code

describe the design process for a composite model and how a single simulation run is executed.

A. *Registration of Models and Datasets*

Models and data must be registered with Splash before they can be used, to create *Splash model actors* and *Splash data actors*. These "actors" are components of Splash that encapsulate the framework's knowledge about the various models and data sources. This knowledge is specified via metadata files, written in the *Splash Actor Description Language (SADL)*, that are created by the system as part of the registration process.

The SADL file for a model actor is created via a SADL "wizard", and contains pointers to the SADL files for the model's input data sources and the output datasets; each of these data sources and sinks is represented as a Splash data actor. The model SADL file also contains information on where and how the model is to be accessed and executed, as well as "provenance" data such as the model's owner, model type, the history of edits made to the SADL file, a summary description of the model's functionality, and pointers to references—such as scientific papers, URLs, and reviews—about the interpretation, assumptions, applicability, and quality of the model; see Figure 1. As can be seen from the figure, the description language uses an XML-style syntax.

The SADL file for a data actor specifies information such as the data schema, data-source location, commands to access the data, and so on. A schema may be specified in industry-standard XSD format (a dialect of XML). The SADL file also describes important characteristics of each attribute (i.e., field) in the data-source records, such as measurement units, a description of the semantics of the attribute, general constraints on the data—e.g., weight must lie between 0 and 400 pounds and pre-tax income must exceed after-tax income. Such information is crucial for sensible composition of models, as well as automated error checking and problem detection. SADL files for certain data sources, such as those containing time series data, have additional metadata, for instance, documenting that a time series comprises regular observations in continuous time with one tick equal to 15 minutes. Similarly, the SADL file may contain geospatial metadata. The current prototype primarily handles data sources that correspond to files on disk; in future work we will extend Splash to handle other types of data sources such as database management systems and data from web-service calls.

Our prototype does not yet support a search capability over the Splash repository, but this functionality will become increasingly important as the size of the repository grows. We envision that to create a composite model in Splash, a user will search through the repository using keyword queries or structured queries over the SADL metadata. Using ontologies and ranking techniques, the discovery component will retrieve and display the "most relevant and compatible" models and data, in a manner reminiscent of web-service discovery [21].

B. Designing Composite Models

After identifying a suitable set of component models in the repository, a modeler then puts together these models in the Splash design environment. The current prototype relies on the Kepler scientific workflow system [22] to provide a visual design environment; indeed, our "actor" terminology derives from Kepler. A user designs a composite model by dragging icons from the repository window and dropping them into the design workspace; these icons represent Splash data actors and model actors, as well as *mapper actors* that execute data transformations. The user then connects the components and configures the mapper actors (as described below).

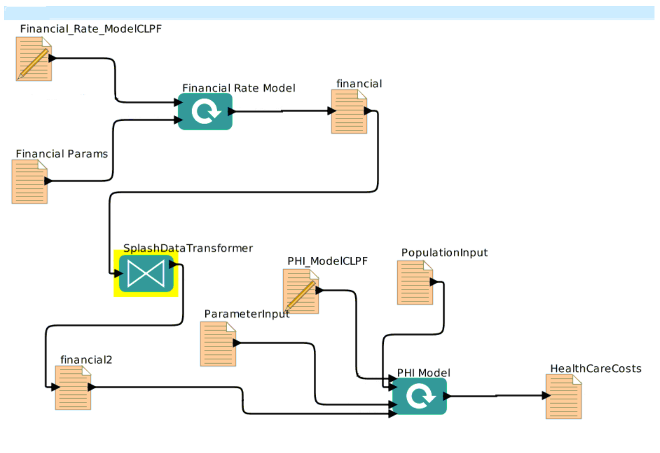


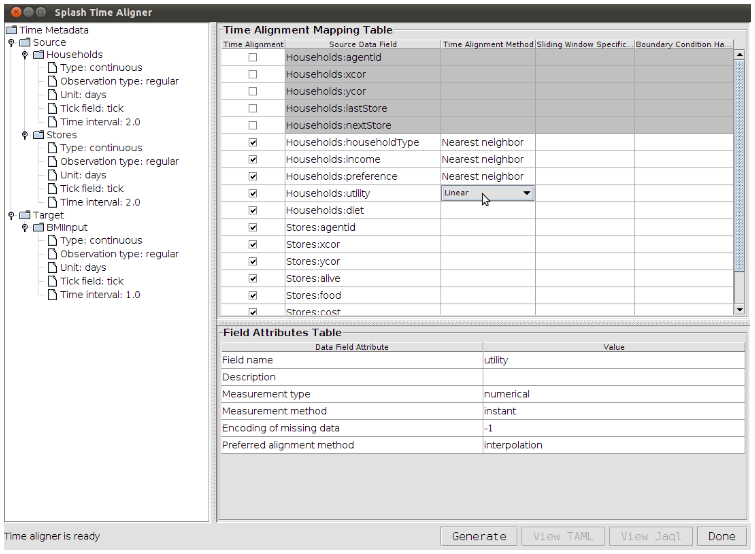
Fig. 2. Design environment showing a simple composite healthcare model

Figure 2 shows a screenshot of a simple composite model comprising two component models. The PHI (Predictive Health Institute) model is derived from[1] an agent-based simulation model developed by Park et al. [20] to explore the economic performance of a wellness program for managing heart disease and diabetes under alternative payment models (capitated—i.e., fixed—payments to the healthcare provider, outcome-based payments, or a combination of the two). The PHI model takes as input a time series of the annual healthcare inflation rate, general economic inflation rate, and discount rate (cost of capital). This time series is provided by the displayed financial-rate (FR) model, which is an extremely simple random-walk model used here for demonstration purposes. We will use this composite FR-PHI model as a running example.

The output dataset of the FR model is passed through a Splash mapper actor named *SplashDataTransformer* to create the corresponding input dataset of financial rates for the PHI model. In this very simple model, the mapper actor essentially copies the data unchanged. In general, however, output data produced by one or more upstream "source" models may need to be combined together and transformed in sophisticated ways to create input data for a downstream "target" model. Splash provides a number of tools to detect source/target mismatches and allow the semi-automatic creation of data transformations. In particular, if a source model outputs a time series which is then input to a target model, Splash will use the metadata in the SADL files for these models to automatically detect time mismatches—e.g., if the source model outputs data every second but the target model needs data every 0.5 seconds—and will pop up a visual interface (see Figure 3) that allows the user to choose a time alignment method for each data field in the time series; in this case, a menu of interpolation methods such as linear, cubic-spline, or nearest-neighbor will be displayed. Depending on the relative time granularity of the source and target time series, as well as the type of data item to be imputed (an instantaneous measurement, cumulative amount since last tick, or cumulative amount since start of measurement period) the interface can also display a menu of appropriate aggregation or allocation transformations. Splash will also automatically glean from the SADL metadata the information needed to handle missing data, "boundary" data points at the beginning or end of a time series, and so on. In ongoing work, we are adding corresponding transformation capabilities for geospatial data.

In a similar manner, Splash provides a visual interface as in Figure 4 for the interactive design of "structural" data transformations from multiple source schemas to an input schema. The interface is based on an extension of the Clio [23] tool for design of schema mappings. The user draws a line from a source attribute to a target attribute to identify a correspondence between them; future versions of Splash will use schema-matching technologies to automatically suggest such correspondences. Various transformations can be specified to merge together one or more source

[1]The model in [20] is programmed in AnyLogic; we use a simplified version of the model written in Python and provided to us by the authors.


Splash Time Aligner

```
Time Metadata
  Source
    Households
      Type: continuous
      Observation type: regular
      Unit: days
      Tick field: tick
      Time interval: 2.0
    Stores
      Type: continuous
      Observation type: regular
      Unit: days
      Tick field: tick
      Time interval: 2.0
  Target
    BMIInput
      Type: continuous
      Observation type: regular
      Unit: days
      Tick field: tick
      Time interval: 1.0
```

Time Alignment Mapping Table

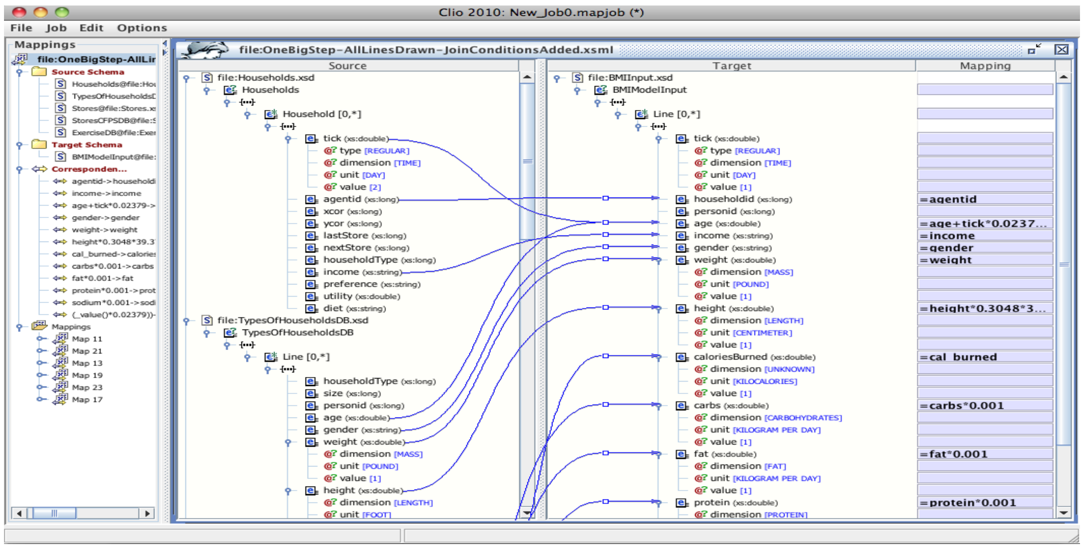
Time Alignment	Source Data Field	Time Alignment Method	Sliding Window Specific...	Boundary Condition Ha...
☐	Households:agentid			
☐	Households:xcor			
☐	Households:ycor			
☐	Households:lastStore			
☐	Households:nextStore			
✓	Households:householdType	Nearest neighbor		
✓	Households:income	Nearest neighbor		
✓	Households:preference	Nearest neighbor		
✓	Households:utility	Linear		
✓	Households:diet			
✓	Stores:agentid			
✓	Stores:xcor			
✓	Stores:ycor			
✓	Stores:alive			
✓	Stores:food			
✓	Stores:cost			

Field Attributes Table

Data Field Attribute	Value
Field name	utility
Description	
Measurement type	numerical
Measurement method	instant
Encoding of missing data	-1
Preferred alignment method	interpolation

Time aligner is ready

Fig. 3. Visual interface for designing time-alignment transformations.


Clio 2010: New_Job0.mapjob (*)

```
Mappings
file:OneBigStep-AllLir
  Source Schema
    Households@file:Hou
    TypesOfHouseholdsD
    Stores@file:Stores.xs
    StoresCFPSDB@file:S
    ExerciseDB@file:Exer
  Target Schema
    BMIModelInput@file:
  Corresponden...
    agentid->householdi
    income->income
    age+tick*0.02379->
    gender->gender
    weight->weight
    height*0.3048*39.3
    cal_burned->calories
    carbs*0.001->carbs
    fat*0.001->fat
    protein*0.001->prot
    sodium*0.001->sodi
    (_value()*0.02379))-
  Mappings
    Map 11
    Map 21
    Map 13
    Map 19
    Map 23
    Map 17
```

file:OneBigStep-AllLinesDrawn-JoinConditionsAdded.xsml

Source

```
file:Households.xsd
  Households
    {...}
      Household [0,*]
        {...}
          tick (xs:double)
            type [REGULAR]
            dimension [TIME]
            unit [DAY]
            value [2]
          agentid (xs:long)
          xcor (xs:long)
          ycor (xs:long)
          lastStore (xs:long)
          nextStore (xs:long)
          householdType (xs:long)
          income (xs:string)
          preference (xs:string)
          utility (xs:double)
          diet (xs:string)
file:TypesOfHouseholdsDB.xsd
  TypesOfHouseholdsDB
    {...}
      Line [0,*]
        {...}
          householdType (xs:long)
          size (xs:long)
          personid (xs:long)
          age (xs:double)
          gender (xs:string)
          weight (xs:double)
            dimension [MASS]
            unit [POUND]
            value [1]
          height (xs:double)
            dimension [LENGTH]
            unit [FOOT]
```

Target / Mapping

```
file:BMIInput.xsd
  BMIModelInput
    {...}
      Line [0,*]
        {...}
          tick (xs:double)
            type [REGULAR]
            dimension [TIME]
            unit [DAY]
            value [1]
          householdid (xs:long)              =agentid
          personid (xs:long)
          age (xs:double)                    =age+tick*0.0237...
          income (xs:string)                 =income
          gender (xs:string)                 =gender
          weight (xs:double)                 =weight
            dimension [MASS]
            unit [POUND]
            value [1]
          height (xs:double)                 =height*0.3048*3...
            dimension [LENGTH]
            unit [CENTIMETER]
            value [1]
          caloriesBurned (xs:double)         =cal_burned
            dimension [UNKNOWN]
            unit [KILOCALORIES]
            value [1]
          carbs (xs:double)                  =carbs*0.001
            dimension [CARBOHYDRATES]
            unit [KILOGRAM PER DAY]
            value [1]
          fat (xs:double)                    =fat*0.001
            dimension [FAT]
            unit [KILOGRAM PER DAY]
            value [1]
          protein (xs:double)                =protein*0.001
            dimension [PROTEIN]
```

Fig. 4. Visual interface for designing structural data transformations.

attributes to create a target attribute, and the system will automatically add simple transformations such as measurement-unit corrections (e.g., kilograms to pounds).

The data-transformation specifications from each of the foregoing design steps is saved in a file, using a specification language—XML schema mapping language (XSML) for structural mappings and a novel time alignment markup language (TAML) for time alignments. The transformations can later be reloaded into the GUIs for re-use or modification.

C. Code Generation and Execution of Composite Models

Once the user has finished designing a data transformation, the system automatically generates runtime code to execute the transformation during a simulation run. This step serves to instantiate a Splash mapper actor, which can be stored in the Splash repository. Because the execution of large, high-resolution models can generate enormous amounts of data, it is important to ensure that the transformations be carried out in a fast and scalable manner. Currently, Splash compiles a data transformation into JAQL [24] code, which will execute the transformation on the Hadoop platform for massive parallel MapReduce processing of data. See [25] for a detailed description of Splash's time-alignment framework and [26] for a description of a novel MapReduce algorithm for cubic-spline interpolation in Splash, based on distributed stochastic gradient descent.

To perform an individual simulation run of a composite model, the current prototype uses Kepler's "director" mechanism to orchestrate the execution of the component Splash model actors and mapping actors. In our example, the FR model executes first, followed by the *SplashDataTransformer* mapper and then the PHI model. In general, it is possible to execute multiple groups of component models and mappers in parallel, if allowed by the structure of the composite model. Currently, Splash most easily handles composite models in which the data flow can be represented as a directed acyclic graph, so that there is always a clear notion of upstream and downstream models. In ongoing work, we are exploring different ways of approximating bi-directional causality between a pair of models.

For our simple example, all models and data sources reside on the same computer as Splash. In general, Splash can execute models remotely. Indeed, upon invocation, a Splash model actor or mapping actor simply synthesizes an appropriate command-line string for executing the model or mapping code and sends it to an appropriate destination. This remote execution capability can be important if certain models must be executed on specialized hardware or executed behind a firewall for security or privacy reasons. We intend to eventually enhance Splash with mechanisms for automatically reconfiguring parts of a simulation-experiment workflow among distributed data and models, moving models to data or vice versa, applying filtering or data-reduction operations to datasets before transmitting them over a network to a downstream model, avoiding redundant computations within a run, and so on. This rich research area related to experiment-execution optimization can be viewed as a major extension to database query-optimization technology as described, for example, in [27, Ch. 16].

We envision two modes of model execution. The first mode supports debugging and "test driving" a composite model, both when the model is first created and later on, if the model produces unusual or counterintuitive results that merit deeper investigation. For this type of execution, scientific-workflow functionality can be very valuable, in that it is easy to send copies of the intermediate data produced by component models to, say, a visualization actor for plotting or a statistical-analysis actor to run diagnostics or perform validation. (Kepler currently has native support for the *R* statistical package, for example.) The second type of execution comprises "production runs" where the model is executed under a range of different inputs and parameter values to study its behavior for purposes of calibration, validation, prediction, sensitivity analysis, and optimization. This latter mode of operation lies within the purview of the experiment manager component, which we discuss next.

III. EXPERIMENT MANAGEMENT

The Splash experiment manager forms the foundation for sensitivity analysis and simulation optimization. Key challenges in this setting include (1) providing a unified view over all of the parameters in all of the component models that make up a composite model, (2) providing a mechanism for selecting the factors to vary, and (3) allowing users to easily specify experimental designs for the selected factors. An additional challenge is to provide a mechanism for systematically varying the inputs to the composite models; in general, each model will have been developed in isolation, with no view towards having the model be part of an ensemble. We discuss Splash's solutions to each of these issues in the following sections. As will be seen, extensions to the SADL metadata language play a crucial role in enabling experiment management. The goals of our experiment manager are similar to those of the script-based Nimrod toolset for computational experiments in a grid-computing environment [28].

We use standard experimental-design terminology throughout: an *experiment* comprises the systematic variation of a finite set of *factors* to study their effect on system behavior. We focus throughout on the case where each factor can take on multiple values, or *levels*. By fixing each factor at one of its levels, we obtain a specific experimental *condition*. Because composite models are often stochastic in nature, we typically want to run multiple Monte Carlo *replications* for each condition that we study. The design specification for a simulation experiment then comprises a set of conditions to simulate, along with the number of replications for each condition. (The number of replications can vary from one condition to another.) One standard experimental design is the *full-factorial* design, in which every possible condition is simulated. For k factors with two levels each, for example, there are 2^k conditions, with multiple replications required for each condition. Typically, this design is too expensive and so the usual goal is to find a design with fewer conditions that will still capture the most important relationships between factor values and system behavior; see, e.g., [29], [30], [31].

A. Specifying Factors

We divide Splash data sources into *data files* whose values remain constant over an experiment and *parameter files* having one or more model parameters that correspond to experimental factors, and hence are varied over the course of the experiment. Referring to Figure 2, for example, the *PopulationInput* data source comprises a file that contains physical characteristics for each member of a simulated population (age at start of simulation, 10-year heart risk, and so on); this file is randomly sampled with replacement to produce new members of the simulation population when needed. In our experiments the characteristics of a population remain fixed, and so *PopulationInput* is viewed as a data file. On the other hand, the data source *ParameterInput* contains PHI model parameters that will be varied from condition to condition—such as the population growth rate, capitation amount per patient, terminal age at which the patient exits the program, and payment model (the relative fraction of the capitation versus outcome-based components of the healthcare provider's revenue)—and thus *ParameterInput* is considered to be a parameter file. Note that the *name* of the population data file might be considered a parameter in some experiments if it is of interest, say, to compare results for a given urban population to those for a given rural population.

We extend the SADL syntax to let an attribute in the input data source for a model be flagged as a potential experimental factor. Consider, for example, the snippet from the file `ParameterInput.sadl` shown in Figure 5. The parameter *terminalAge* is identified as a potential experimental factor, and the model provider has recommended

```
<attribute name="terminalAge"
  description="age at which person exits program"
  measurement_type="numerical"
  unit = year
  datatype="double"
  experiment_default_values="65 70 75"
  experiment_factor="true"
 />
```

Fig. 5. A snippet of `ParameterInput.sadl`

```
<attributes>
  <attribute
    name="population"
    description="population data file input to PHI_Model"
    datatype="string"
    experiment_factor="true"
    experiment_default_values="/default_dir/populationdata.csv"
    label="-o"
  />
  <attribute
    name="parameters"
    description="pathname for file containing parameter data"
    datatype="string"
    experiment_factor="false"
    experiment_default_values="/default_dir/data/params.csv"
    label="-p"
  />
  <attribute
    name="rseed"
    description="Pseudorandom number seed for PHI Model"
    datatype="integer"
    experiment_default_values="1234"
    experiment_factor="false"
    label="-r"
    random_seed="true"
/>
</attributes>
```

Fig. 6. A snippet of `PHI_ModelCLPF.SADL`

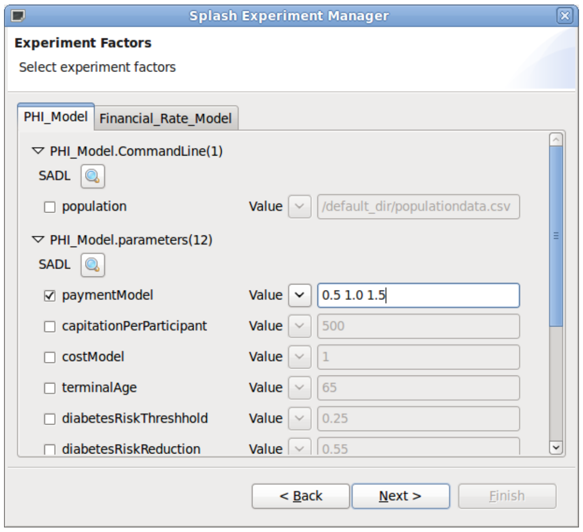
default low, medium, and high level values for this parameter as 65, 70, and 75. A file is considered to be a parameter file if `experiment_factor="true"` for at least one attribute in the file. (If a model parameter is supposed to stay constant throughout an experiment, then we set `experiment_factor="false"` and specify a single experiment default value.)

Besides being read from a file on disk, the other way in which parameters are routed to a component simulation model is via arguments given on the command line when invoking the model. For instance, the PHI model is invoked as

```
> PHI -o /default_dir/populationdata.csv \
      -p /default_dir/data/params.csv -r 132453 ...
```

where the command-line parameters include the name of the population characteristics file to use, the name of a file of additional parameter values to read, and a seed for the pseudorandom number generator used by the model.

In Splash, we conceptually view such data as being read from a "command-line parameter file" `PHI_ModelCLPF.csv` containing the data values, i.e., the two filename strings, the integer pseudorandom number seed, and so on. This data source is encapsulated as a Splash data actor and described by a file `PHI_ModelCLPF.sadl` that gives the data type, units, description, and so on, for each of the command-line parameters. See Figure 2, in which each model has a CLPF data source, and Figure 6,


Splash Experiment Manager

Experiment Factors

Select experiment factors

PHI_Model | Financial_Rate_Model

PHI_Model.CommandLine(1)

SADL

☐ population Value /default_dir/populationdata.csv

PHI_Model.parameters(12)

SADL

☑ paymentModel Value 0.5 1.0 1.5

☐ capitationPerParticipant Value 500

☐ costModel Value 1

☐ terminalAge Value 65

☐ diabetesRiskThreshhold Value 0.25

☐ diabetesRiskReduction Value 0.55

Fig. 7. Identification of experimental factors

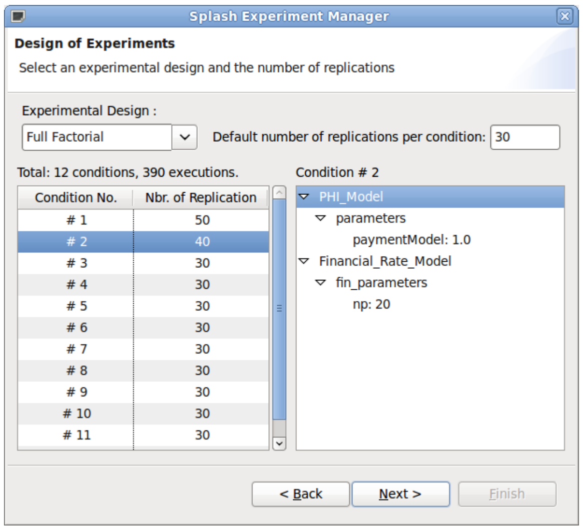
which displays a snippet of the SADL file for the PHI model CLPF. In most respects a CLPF file is treated just like any other parameter file.

B. Designing Experiments

Experiments are designed using a GUI that brings together all of the potential experimental factors in a composite model. Specifically, the experiment manager first identifies the set of "base" parameter files in a composite model, that is, those parameter files whose contents are not derived from the output of any component model. In the composite model of Figure 2, for example, the data sources *FinancialParams* and *PHI_ModelCLPF* correspond to base parameter files but *financial2* does not. The GUI then systematically reads the SADL files corresponding to each of the base input parameter files, and displays all of the parameters in each file that have been flagged as potential experimental factors; see Figure 7. As shown in the figure, the user then indicates which of the potential experimental factors should actually be treated as factors in the experiment of interest. Default levels for the selected factors are obtained from the SADL description, as discussed previously, and can be modified in the GUI.

The user can then specify the experimental design, either by selecting a standard design from a drop-down menu, or by explicitly specifying the desired set of conditions. Similarly, the user can specify a fixed number of Monte Carlo replications to use for all of the conditions, or can specify a desired number of replications for each condition individually; see Figure 8.

The experimental design specification is saved in a file, using an experiment markup language (EML). As with the markup languages for data transformations, the EML representation of an experiment can be loaded into the experiment manager at a later time for purposes of experiment modification or re-execution. Figure 9 gives some snippets of an EML file, which specifies the composite model to run, the top-level directory for holding the simulation outputs, the set



Splash Experiment Manager

Design of Experiments

Select an experimental design and the number of replications

Experimental Design : Full Factorial

Default number of replications per condition: 30

Total: 12 conditions, 390 executions.

Condition No.	Nbr. of Replication
# 1	50
# 2	40
# 3	30
# 4	30
# 5	30
# 6	30
# 7	30
# 8	30
# 9	30
# 10	30
# 11	30

Condition # 2

```
PHI_Model
  parameters
    paymentModel: 1.0
Financial_Rate_Model
  fin_parameters
    np: 20
```

Fig. 8. Design of an experiment

of factors and their levels, along with the specific level values and number of replications for each condition. Observe that, for the experiment described by the EML snippet, the entire population is treated as an experimental factor, with two levels corresponding to an urban and a rural population. The EML file also specifies a global pseudorandom number seed to use for the experiment; the issue of seed management is discussed in the next section.

C. Executing Experiments

For each simulation run in an experiment, Splash must first prepare all of the data sources expected by the component models. The procedure for a given data source depends on how the data is "routed" to the model.

Perhaps the simplest data source to deal with is the set of command line arguments for a given component model. In this case Splash simply generates the appropriate invocation command for the model—as in the PHI example of Section III-A—which is then executed on the machine where the model resides. Any argument values that change from condition to condition are obtained from the EML file, and any other unchanging, default argument values are obtained from the SADL file corresponding to the model's CLPF. The CLPF SADL file also contains information needed to format each command line argument, such as the command line flag, if any; see Figure 6. (If the interpretation of command-line parameters depends on the order of the parameters rather than explicit flags, then this order is captured in the CLPF SADL file.)

For a data source that corresponds to a file of parameter values, Splash needs to synthesize a version of the file having the correct value for each parameter, and then put the file in a location where the model expects it. Splash currently supports a few standard file formats, and we are currently implementing a template system, similar to that in [28], to deal with non-standard formats. The idea is that a model developer can take a "typical" input parameter file

```
<eml model="FR-PHI" date="06/22/2012">
  <archive>user/model/fin-phi/fin-phi.splash</archive>
  <experiment-directory>/user/expts</experiment-directory>
  <global-seed>1234</global-seed>

  <experiment-factors>
   <submodel name= "PHI_Model">
    <datasource name="PHI_ModelCLPF">
     <factor name="popdata">
        <value>"popUrban.dat"</value>
        <value>"popRural.dat"</value>
     </factor>
    </datasource>
    <datasource name="ParameterInput">
     <factor name="PaymentModel">
        <value>0.0</value>
        <value>0.5</value>
        <value>1.0</value>
     </factor>
    <factor name="CapAmount">
      <value>100</value>
      <value>3000</value>
    </factor>
    </datasource>
   </submodel>
  </experiment-factors>

  <experiments>
   <condition no="1" replication="20">
     <factor name="PHI_ModelCLPF#popdata"
        value="popUrban.dat"/>
     <factor name="ParameterInput#PaymentModel"
        value="0"/>
     <factor name="ParameterInput#CapAmount" value="100"/>
   </condition>
   <condition no="2" replication="10">
     <factor name="PHI_ModelCLPF#popdata"
        value="popRural.dat"/>
     <factor name="PaymentModel" value="0.5"/>
     <factor name="CapAmount" value="100"/>
   </condition>
  </experiments>
</eml>
```

Fig. 9. A snippet of an EML file

and create a template by replacing every specific data value that may vary across experimental conditions with a symbol that identifies the parameter. For instance, the line

```
TEMPERATURE = 35.2 PRESSURE = 120
```

would be replaced by

```
TEMPERATURE = @TEMP(5.1f) PRESSURE = @PRESS(4d)
```

to identify the parameters *TEMP* and *PRESS*. The template information can then be used to generate parameter files with varying values of temperature and pressure. The goal here is to enable a fairly generic mechanism for dealing with a large class of idiosyncratic file formats without forcing model developers to write a separate wrapper for each such format. This mechanism can also be used to parse nonstandard output files from source models or generate nonstandard input files for target models as part of the data transformations that occur during a simulation run of a composite model.

Putting a synthesized parameter file in the correct location can be as simple as putting the file in some preferred directory and synthesizing an appropriate command-line argument that specifies the filepath. For example, we can put `params.csv`, a parameter file for the PHI model, in directory `/default_dir/data/` and then create a CLPF as in Figure 6, which will in turn lead to the synthesis of an invocation command as in Section III-A. In a more difficult situation, the filepath might be hard-wired into the model's code. In this case, the expected filepath is specified in the model's SADL file, so that Splash can create the appropriate directory, copy the synthesized parameter file to this directory, and rename the file as needed, prior to model execution. Data read from standard input can be handled in a similar manner.

Splash has analogous capabilities for handling the output from simulation models. In general, the user specifies a top-level directory to hold the output files from a set of experiments, and Splash creates subdirectories for each (experiment, condition) pair. Each such subdirectory holds the output from the corresponding Monte Carlo replications. Output files with a hardwired destination may need to be copied from a model-specified directory to the user's preferred directory.

Currently, the Splash prototype executes Monte Carlo replications in a straightforward manner by repeatedly executing the entire composite model; each such execution invokes every component model at least once. Clearly, there are opportunities for improving efficiency. For example, suppose that one of the component models is deterministic, expensive to execute, and does not receive input from other component models. It may then be worthwhile to cache the model's output during the first replication; subsequent replications can simply read from the cache, avoiding repeated execution of the model. More generally, it might be advantageous to execute fewer replications for component stochastic models with low variability in the output than for models with high variability, and then combine sample paths in a bootstrap-like manner; such ideas are related to notions of "splitting" as in [32, Sec. 6.3].

We conclude this section by discussing one of the more challenging issues that arises when running experiments over stochastic composite models, namely, dealing with the pseudorandom number generators (PRNGs) in the various component models. Most stochastic simulation models use PRNGs, which take as input an integer value called a *seed* and apply a deterministic recursion to generate sequences of seeds that appear to be statistically random; this sequence of seeds forms a *cycle*, since the generator will eventually return to its starting state. The potential problem is that two component models might inadvertently use sequences of seeds that overlap, which would induce spurious statistical correlations between the models.

Splash currently handles PRNG seeding as follows. The SADL syntax for a component model allows specification of the PRNG(s) used by the model, and the mechanism by which the seed for the generator is set. In the easiest case, the initial seed is a parameter of the model that can be set by the experiment manager; see, for example, the PHI model invocation example in Section III-A, where the seed may be specified on the command line. The experiment manager allows specification of a global seed (see Figure 9) which is used by Splash to generate initial seeds for each replication of each component model. Provided that the PRNG for a component model is different from Splash's PRNG—currently the WELL19973a generator [33]—the initial seeds provided by Splash will map to locations on the component-model PRNG cycle that will appear to be chosen at random. If the length of the latter cycle is sufficiently long, then the cycle segments "consumed" during different replications will be unlikely to overlap and the statistical performance should be acceptable. If the generators are the same, then initial seeds can be chosen to explicitly avoid overlap of seed subsequences; see [34] for some pertinent techniques and analysis.

Some component models do not allow fine-grained control of seeds. For example, some models derive their initial seed from the system clock. Since both knowledge about and control over PRNGs may be limited, we expect that diagnostics and statistical testing will play an important role in avoiding erroneous or misleading simulation results. For example, in the debugging mode of model execution mentioned in Section II-C, the user could run statistical tests of independence on pairs of model output sequences that are supposed to be independent according to the model definition, perhaps after batching the outputs [4, Sec. 15].

IV. OPTIMIZATION

In this section, we describe how we can exploit the experiment manager functionality to support rudimentary simulation-based optimization over a composite model. As discussed below, the current Splash prototype uses the *R* statistical package for the supporting statistical calculations and final graphical display of results.

Our goal is to select the best value of a control variable from among a small number of feasible values. Here "best" means the value that maximizes an expected "reward" (typically revenue or profit). Specifically, we implemented the well known Rinott two-stage selection procedure for choosing the best system under an indifference-zone relaxation [17]. In our setting, each "system" corresponds to the composite model running under a fixed value of the control variable. Note that the control variable may actually correspond to a vector of model parameters, as long as the number of distinct parameter vectors considered is small. For example, one "control-variable value" might correspond to a capitation rate of \$200 per program participant per year and a payment-model factor of 0.1, whereas another value might correspond to a capitation rate of \$100 and a payment-model factor of 0.5. We chose to add the Rinott procedure first because of its simplicity and relative ease of implementation in Splash.

The general setting for the Rinott procedure is a small collection of systems $S_1, S_2, \dots, S_k$ (typically $k \leq 20$), where the expected reward of the system S_i is an unknown constant μ_i that can be only be estimated via stochastic simulation. That is, $\mu_i = E[Y_i]$, where Y_i represents the noisy output from a single simulation run of S_i. The goal is to select the system having the largest expected reward, ensuring that the probability of correct selection exceeds a specified constant C; e.g., take $C = 0.95$ to be 95% certain that the best system is selected. To make the computation tractable, we assume that any two systems having rewards within δ units of each other are considered equally acceptable, where the length δ of the "indifference zone" is specified by the user, based on practical considerations. Thus we consider a selection "correct" if the expected reward for the selected system is greater than or equal to $\mu^* - \delta$, where $\mu^* = \max_{1 \le i \le k} \mu_i$ is the maximum expected reward; if the difference in expected reward between the two best systems is greater than δ, then, with probability C, we will in fact have selected the best system. (Without an indifference zone, a huge number of Monte Carlo replications might be required to distinguish between two systems whose expected rewards are very close to each other.)

To allow additional inferences about the alternative systems, the algorithm also provides interval estimates for the set of quantities $\{\gamma_1, \gamma_2, \ldots, \gamma_k\}$, where $\gamma_i = \mu_i - \max_{j \neq i} \mu_j$. These quantities indicate the relative performance of the various systems. For example, suppose that we are comparing $k = 3$ systems and, unbeknown to us, $(\mu_1, \mu_2, \mu_3) = (1, 2, 7)$. Then $(\gamma_1, \gamma_2, \gamma_3) = (-6, -5, 5)$, indicating that S_3 has the highest reward and is, in fact, 5 units more profitable than the best (most remunerative) of the other solutions. Similarly, S_2 is 5 units less profitable than the best of the other solutions, and S_1 is 6 units less profitable than the best of the other solutions. The γ_i values are especially useful for identifying near-optimal solutions, which might be easier to implement in practice than the best solution while still incurring high rewards.

In more detail, the outputs of the basic Rinott selection procedure are used to provide a confidence interval $J_i = [a_i, b_i]$ for each γ_i. With probability C, these intervals are simultaneously correct, in that each interval J_i contains the unknown quantity γ_i [18, Th. 2]. The intervals provide probabilistic bounds on how suboptimal each system can be. Moreover, with probability C, intervals with $a_i < b_i \le 0$ correspond to systems that can be eliminated from being considered as the best, and a system with $0 \le a_i < b_i$ is unambiguously the best. A system with $a_i < 0 < b_i$ is a contender for being the best. This type of procedure goes by the name of "multiple comparisons with the best" (MCB).

The Rinott procedure is given as Algorithm 1. In stage 1, which corresponds to steps 2–5, n_0 initial replications are run for each system S_i. Based on the sample mean $\bar{X}_i$ and sample variance V_i of the reward for S_i, the algorithm computes (step 5) the total number of replications N_i of system S_i needed to make a correct selection with the required probability C. The quantity $h = h(n_0, k, C)$ appearing in step 5 is a tabulated constant whose value is based on the assumption that each stage-1 mean $\bar{X}_i$ is normally distributed. In stage 2, additional replications of S_i are run (step 6) to bring the total number of replications to N_i. Based on the final estimates $\bar{Y}_1, \bar{Y}_2, \ldots, \bar{Y}_k$ of the expected rewards $\mu_1, \mu_2, \ldots, \mu_k$, one of the S_i's is selected as the best system. In the final step, the algorithm computes confidence intervals for $\gamma_1, \gamma_2, \ldots, \gamma_k$.

Figure 10 illustrates how the optimizer component of

Algorithm 1 Rinott Selection Procedure with MCB

```
Require: $k$, $n_0$, $C$, $\delta$
 1: for $i = 1$ to $k$ do
 2:     Simulate $S_i$ to obtain $Y_{i,1}, Y_{i,2}, \ldots, Y_{i,n_0}$
 3:     $\bar{X}_i \leftarrow (1/n_0) \sum_{j=1}^{n_0} Y_{i,j}$
 4:     $V_i \leftarrow \big(1/(n_0 - 1)\big)\big(\sum_{j=1}^{n_0} (Y_{i,j} - \bar{X}_i)^2$
 5:     $N_i \leftarrow \max\Big(n_0, \lceil \frac{h^2 V_i}{\delta^2} \rceil\Big)$,
 6:     Simulate $S_i$ to obtain $Y_{i,n_0+1}, Y_{i,n_0+2}, \ldots, Y_{i,N_i}$.
 7:     $\bar{Y}_i \leftarrow (1/N_1) \sum_{j=1}^{N_1} Y_{i,j}$
 8: end for
 9: Select system with largest value of $\bar{Y}_i$ as the best
10: for $i = 1$ to $k$ do
11:     Form confidence interval $[a_i, b_i]$ for $\gamma_i$, where
            $a_i = \min\big(0, \bar{Y}_i - \max_{j \neq i} \bar{Y}_j - \delta\big)$
            $b_i = \max\big(0, \bar{Y}_i - \max_{j \neq i} \bar{Y}_j + \delta\big)$
12: end for
```

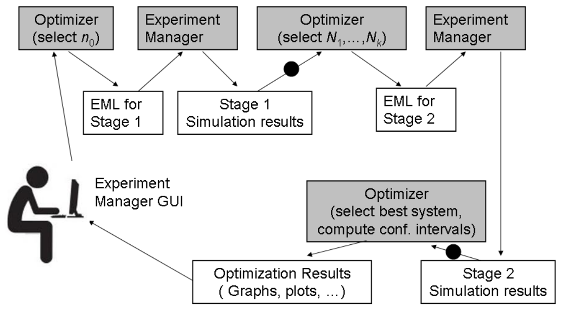

Optimizer (select n_0)
Experiment Manager
Optimizer (select $N_1,\ldots,N_k$)
Experiment Manager
EML for Stage 1
Stage 1 Simulation results
EML for Stage 2
Experiment Manager GUI
Optimizer (select best system, compute conf. intervals)
Optimization Results (Graphs, plots, ...)
Stage 2 Simulation results

Fig. 10. Optimization process flow

Splash uses the experiment manager to perform the Rinott procedure. The user, employing the Splash experiment manager GUI, specifies the feasible values of the control variable as a set of conditions, and also specifies the indifference zone δ and the probability C of correct selection. The optimizer component then determines the number n_0 of stage-1 replications and creates an EML file to run the experiments, which is then passed to the experiment manager for execution.[2] The optimizer then processes the output files created by the experiment manager to extract the stage-1 sample mean and variance of the reward for each condition simulated. In general, this extraction step might require the user to provide a routine for computing the reward from the output of a given simulation run; this component is represented in Figure 10 by a black circle. The optimizer then determines the number of stage-2 replications for each condition and creates an EML file, which is passed to the experiment manager for execution. When the experiment manager completes, the optimizer computes the overall sample means $\bar{Y}_1, \bar{Y}_2, \ldots, \bar{Y}_k$, which serve as the final estimates of the expected system rewards. The optimizer then selects as best the system S_i having the

[2]In the simplest implementation, n_0 is simply set to a fixed value between 20 and 50, to try and ensure that each stage-1 sample mean $\bar{X}_i$ is approximately normally distributed (by the central limit theorem). A more sophisticated approach would adaptively determine n_0 by executing some pilot runs and testing the sample means for approximate normality.

largest $\bar{Y}_i$ value and computes the MCB confidence intervals. Finally, the results are displayed to the user.

V. Sensitivity Analysis

Because a composite model will typically have many parameters, it is usually essential to determine which of the parameters have the greatest effects on system performance. These sensitive parameters can then be used for optimization purposes as described above. Sensitivity information is important in its own right: sensitive parameters can become a focal point for policy and investment decisions, and may also drive data-collection efforts, since such parameters must be estimated very carefully.

To demonstrate the use of the Splash experiment manager for efficient sensitivity analysis, we implemented several simple methods for main-effects assessment described in [19]. Specifically, "high" and "low" values for each of a small set of k factors are specified by the user, based on practical considerations. The sensitivity-analysis component then generates an orthogonal fractional-factorial design using *R*'s *FrF2* library. For $k = 7$ factors, one such design is as follows:

A	B	C	D	E	F	G
-1	-1	-1	1	1	1	-1
1	-1	-1	-1	-1	1	1
-1	1	-1	-1	1	-1	1
1	1	-1	1	-1	-1	-1
-1	-1	1	1	-1	-1	1
1	-1	1	-1	1	-1	-1
-1	1	1	-1	-1	1	-1
1	1	1	1	1	1	1

This design prescribes eight experimental conditions, one per row. Each column corresponds to a factor (here labeled A–G). The symbol "-1" (resp., "1") in the ith row and jth column indicates that the jth factor is to be set to its low (resp., high) value in the ith experiment. The design is orthogonal in that the columns are mutually orthogonal: the inner product of any two columns equals 0. Importantly, each factor is set to its low and high value equally often in the experimental conditions, i.e., a factor is low in four conditions and high in the remaining four conditions. The plan is to run n i.i.d. Monte Carlo replications for each condition and average the results. The goal here is to run enough replications so that the distribution of each average is approximately normal, and so we typically choose n between 20 and 50. The design is called fractional factorial since the number of experimental conditions tested is much less than the 2^k possible conditions. In our example, there are $2^7 = 128$ possible conditions, of which only eight are run. With $n = 20$ Monte Carlo replications, this represents a reduction from 2,560 to 160 total simulation runs of the composite model.

Based on the foregoing specification, Splash synthesizes an EML file embodying the above experiment, and sends it to the experiment manager for execution. After the experiments have been run, the results can be displayed in a main-effects plot. For each factor, the plot displays the average system response over the four conditions in which the factor is low and over the four conditions in which it is high—the *effect size* for a factor is the difference between these averages. The resulting plot indicates both the direction of the response (increasing or decreasing) as the factor value increases, as well as the relative magnitude of the responses. To determine the statistical significance of the effects, the effect sizes can be displayed in a normal probability plot (sometimes called a *Daniel plot*). The k effect sizes are plotted, in increasing order, against the $1/k$ quantiles of the standard normal distribution. If there were no factor effects, then the observations would fall roughly on a straight line; deviations from such a line indicate significant effects. Splash uses the *MEPlot* and *DanielPlot* functions in *R*'s *DOE.base* package to create main-effects and Daniel plots.

VI. Case Study

To demonstrate Splash's optimization and sensitivity-analysis functionality, we experimented with the composite FR-PHI model described earlier. The model estimates a number of economic metrics associated with the wellness program; we focus on the profit to PHI, the wellness provider. A key control variable is the payment-model parameter α. A value of 0 corresponds to a pure capitation system where PHI receives a fixed dollar amount per program participant per year; a value of 1 corresponds to a pure pay-for-outcome system in which PHI is paid according to the (estimated) illness-related costs that are avoided for each participant due to PHI health interventions. A value between 0 and 1 corresponds to a combination of these payment methods. To make the model a bit more interesting from an optimization point of view for purposes of our demonstration, we modified the model slightly so that, as α increases and PHI revenue becomes increasingly linked to patient health outcomes, the healthcare providers spend increasing amounts of time and resources on each program participant during office visits, examinations, and so on, increasing healthcare delivery costs which counterbalance revenue increases due to improved health outcomes.

Figures 11 and 12 show the output from the Rinott optimization procedure. Conditions C1-C9 correspond to α values of $0.1, 0.2, \ldots, 0.9$. As can be seen in Figure 11, condition C5 (which corresponds to setting $\alpha = 0.5$) is selected as best; under an indifference zone value of $\delta = \$250,000$, this decision is correct with probability 95%. Figure 12 shows the corresponding MCB plot. As can be seen, values of α less then 0.5 or greater than 0.6 can be rejected as suboptimal with 95% confidence. At this level of confidence, there is no value of α that is unambiguously superior; the two values of $\alpha = 0.5$ and $\alpha = 0.6$ are both contenders to be the true optimal value. The confidence intervals show that neither solution yields a profit far from the true optimal profit, so both solutions are quite acceptable.

Figures 13 and 14 show the output from the sensitivity analysis. The parameters studied are the capitation amount,

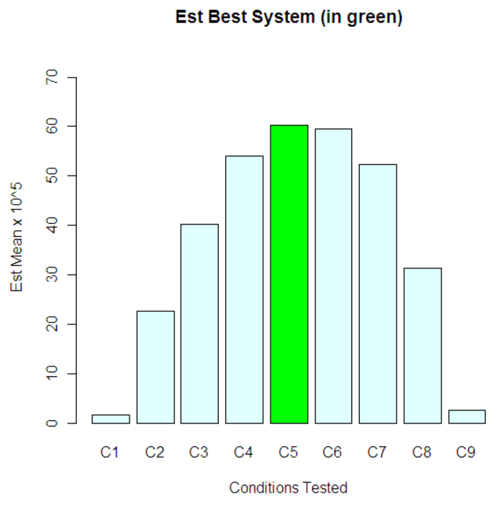


Fig. 11. Optimizer results: Best payment model

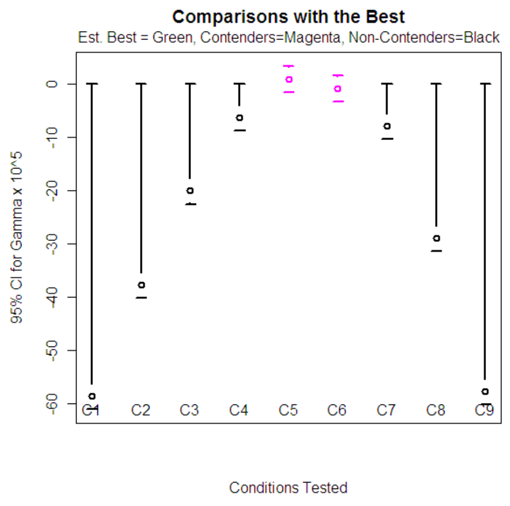


Fig. 12. MCB intervals for payment models

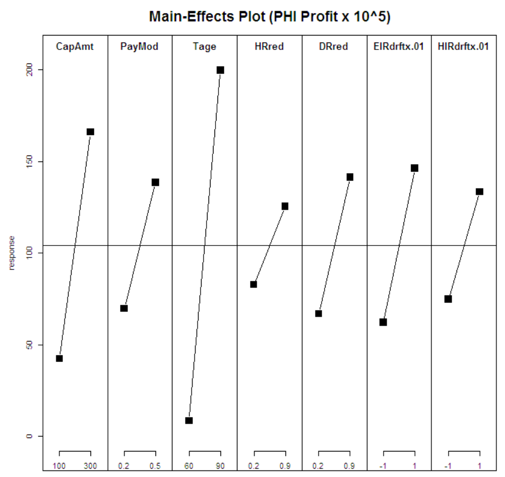


Fig. 13. Main-effects plot for composite model

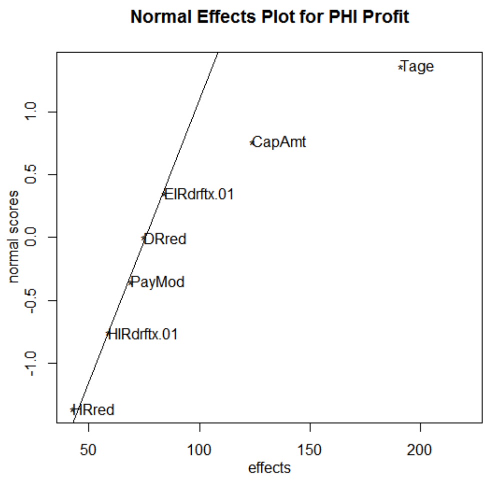


Fig. 14. Daniel plot of estimated effects

payment model parameter (α), terminal age at which participants leave the program, yearly heart-risk and diabetes-risk reductions due to PHI interventions, and the mean drift in the rates of economic inflation and healthcare-cost inflation. Figures 13 shows that the PHI profit is monotonically increasing in each of these parameters. Perhaps surprisingly, the most sensitive variable is the terminal age followed (probably not surprisingly) by the capitation amount. The Daniel plot in Figure 14 indicates that these two effects are the only statistically significant effects of those considered (based on the experiments conducted).

We emphasize that our experiments involve a modification of a simplified version of the model in [20], so the results reported here cannot be taken at face value. Our case study does, however, serve to illustrate the types of analyses that are possible in the current Splash prototype.

VII. CONCLUSIONS

We have illustrated the Splash platform for cross-disciplinary modeling and simulation, with an emphasis on recent experiment-management capabilities and potential application to simulation-based optimization and sensitivity analysis. Our approach combines simulation methodology with information-management technology, especially techniques for data exchange and workflow management. As can be seen, metadata about component models, datasets, transformations, and experiments plays an important role in enabling Splash's new functionality.

Splash is a work in progress, and we are planning to enhance the platform in a variety of ways. We will incorporate other simulation optimization algorithms, such as metamodel-based optimization, stochastic approximation, and discrete optimization over large feasible sets. One interesting challenge in the setting of discrete optimization is to try and coordinate PRNG seeds to apply efficiency improvement techniques such as common random numbers [4, p. 513]. We also intend to add more sophisticated techniques for sensitivity analysis, such as frequency-domain methodology [35], and factor screening, such as controlled sequential bifurcation and controlled sequential factorial design [36]. Other platform improvements, besides those discussed already, include supporting grid computing as in [28] and maintaining privacy and security of data and models where appropriate. The key goal, however, is enabling collaboration, and we envision that Splash will support a collaborative forum similar to ManyEyes [37], where data, models, and visualizations can be uploaded, shared, annotated, and rated by a community of users.

Acknowledgements

We wish to thank Trustin Clear, Rahul Basole, Hyunwoo Park, and William Rouse for the use of their PHI model. We also wish to thank Cheryl Kieliszewski, Yinan Li, and Ronald Mak for their contributions to Splash.

References

[1] R. Pasupathy and S. G. Henderson, "SIMOPT: A library of simulation optimization problems," in *Proc. Winter Simul. Conf.*, 2011, pp. 4080–4090.

[2] D. Phan and S. Ghosh, "A two-stage non-linear program for optimal electrical grid power balance under uncertainty," in *Proc. Winter Simul. Conf.*, 2011, pp. 4227–4238.

[3] Z. Hu and J. Cao, "Robust simulation of environmental policies using the DICE model," in *Proc. Winter Simul. Conf.*, 2010, pp. 1295–1305.

[4] S. G. Henderson and B. L. Nelson, Eds., *Simulation*, ser. Handbooks in Operations Research and Management Science. Amsterdam, The Netherlands: Elsevier, 2006, vol. 13.

[5] Institute of Medicine, *For the Public's Health: The Role of Measurement in Action and Accountability*. The National Academies Press, 2010.

[6] H. Godfray, J. Pretty, S. Thomas, E. Warham, and J. Beddington, "Linking policy on climate and food," *Science*, vol. 331, no. 6020, pp. 1013–1014, 2011.

[7] T. T. Huang, A. Drewnowski, S. K. Kumanyika, and T. A. Glass, "A systems-oriented multilevel framework for addressing obesity in the 21st century," *Preventing Chronic Disease*, vol. 6, no. 3, 2009.

[8] W. C. Tan, P. J. Haas, R. L. Mak, C. A. Kieliszewski, P. G. Selinger, P. P. Maglio, S. Glissmann, M. Cefkin, and Y. Li, "Splash: a platform for analysis and simulation of health," in *ACM Intl. Health Informatics Symp. (IHI)*, 2012, pp. 543–552.

[9] K. Börner, "Plug-and-play macroscopes," *Commun. ACM*, vol. 54, no. 3, pp. 60–69, 2011.

[10] D. A. Ford, J. H. Kaufman, and I. Eiron, "An extensible spatial and temporal epidemiological modelling system," *Int. J. Health Geographics*, vol. 5, no. 4, 2006.

[11] W. D. Collins, C. M. Bitz, M. L. Blackmon, G. B. Bonan, C. S. Bretherton, J. A. Carton, P. Chang, S. C. Doney, J. J. Hack, T. B. Henderson, J. T. Kiehl, W. G. Large, D. S. Mckenna, B. D. Santer, and R. D. Smith, "The community climate system model version 3 (CCSM3)," *J. Climate*, vol. 19, pp. 2122–2143, 2006.

[12] G. A. Wainer, *Discrete-Event Modeling and Simulation: A Practitioner's Approach*. CRC Press, 2009.

[13] J. Gregersen, P. Gijsbers, and S. Westen, "OpenMI: Open modelling interface," *Journal of Hydroinformatics*, vol. 9, no. 3, pp. 175–191, 2007, http://www.openmi.org.

[14] F. Kuhl, R. Weatherly, and J. Dahmann, *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*. New Jersey: Prentice Hall, 1999.

[15] R. W. Conway and J. O. McClain, "The conduct of an effective simulation study," *INFORMS Trans. Education*, vol. 3, no. 3, pp. 13–22, 2003.

[16] P. K. Davis and R. H. Anderson, *Improving the Composability of Department of Defense Models and Simulations*. Santa Monica, CA: RAND Corporation, 2003.

[17] Y. Rinott, "On two-stage selection procedures and related probability-inequalities," *Commun. Statist. Theor. Meth.*, vol. A7, pp. 799–811, 1978.

[18] F. J. Matejcik and B. L. Nelson, "Two-stage multiple comparisons with the best for computer simulation," *Oper. Res.*, vol. 43, no. 4, pp. 633–640, 1995.

[19] T. T. Allen, *Introduction to Discrete Event Simulation and Agent-Based Modeling*. Springer, 2011.

[20] H. Park, T. Clear, W. B. Rouse, R. C. Basole, M. L. Braunstein, K. L. Brigham, and L. Cunningham, "Multi-level simulations of health delivery systems: A prospective tool for policy, strategy, planning and management," *Service Science*, 2012, to appear.

[21] M. Klusch, B. Fries, and K. P. Sycara, "Automated semantic web service discovery with OWLS-MX," in *Proc. AAMAS*, 2006, pp. 915–922.

[22] "Kepler Scientific Workflow System," http://kepler-project.org/.

[23] L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth, "Clio grows up: From research prototype to industrial tool," in *SIGMOD Conference*, 2005, pp. 805–810.

[24] "JAQL: Query Language for JavaScript Object Notation (JSON)," http://code.google.com/p/jaql, 2009.

[25] Y. Li, P. J. Haas, W.-C. Tan, and I. Terrizzano, "Data exchange between simulation models: Getting the time right," IBM Research – Almaden, San Jose, CA, Tech. Rep., 2012, forthcoming.

[26] P. J. Haas and Y. Sismanis, "On aligning massive time-series data in Splash," in *Intl. Workshop on End-to-end Management of Big Data (BigData 2012)*, 2012.

[27] H. Garcia-Molina, J. D. Ullman, and J. Widom, *Database Systems: The Complete Book*, 2nd ed. Pearson Education, 2009.

[28] B. Bethwaite, D. Abramson, F. Bohnert, S. Garic, C. Enticott, and T. Peachey, "Mixing grids and clouds: High-throughput science using the Nimrod tool family," in *Cloud Computing: Principles, Systems and Applications*. Springer, 2010, pp. 219–237.

[29] J. P. C. Kleijnen, *Design and Analysis of Simulation Experiments*. Springer, 2008.

[30] J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn, "Design and analysis of computer experiments," *Statist. Sci.*, vol. 4, no. 4, pp. 409–423, 1989.

[31] S. M. Sanchez and H. Wan, "Better than a petaflop: The power of efficient experimental design," in *Proc. Winter Simul. Conf.*, 2011, pp. 1441–1455.

[32] B. L. Fox and P. W. Glynn, "Discrete-time conversion for simulating finite-horizon Markov processes," *Siam J. Appl. Math.*, vol. 50, no. 5, pp. 1457–1473, 1990.

[33] F. Panneton, P. L'Ecuyer, and M. Matsumoto, "Improved long-period generators based on linear recurrences modulo 2," *ACM Trans. Math. Software*, vol. 32, no. 1, pp. 1–16, 2006.

[34] F. Xu, K. Beyer, V. Ercegovac, P. J. Haas, and E. J. Shekita, "$E = MC^3$: Managing uncertain enterprise data in a cluster-computing environment," in *ACM SIGMOD*, 2009, pp. 441–454.

[35] D. J. Morrice and L. W. Schruben, "Simulation sensitivity analysis using frequency domain experiments," in *Proc. Winter Simul. Conf.*, 1989, pp. 367–373.

[36] H. Shen and H. Wan, "A hybrid method for simulation factor screening," in *Proc. Winter Simul. Conf.*, 2006, pp. 382–389.

[37] "Many Eyes," www-958.ibm.com/software/data/cognos/manyeyes/.