Article

Journal of Service Research
2015, Vol. 18(1) 75-89
© The Author(s) 2014
Reprints and permission:
sagepub.com/journalsPermissions.nav
DOI: 10.1177/1094670514541002
jsr.sagepub.com
SAGE

Improving Productivity in Design and Development of Information Technology (IT) Service Delivery Simulation Models

Anton Beloglazov[1], Dipyaman Banerjee[2], Alan Hartman[3], and Rajkumar Buyya[4]

Abstract

The unprecedented scale of Information Technology (IT) service delivery requires careful analysis and optimization of service systems. The simulation is an efficient way to handle the complexity of modeling and optimization of real-world service delivery systems. However, typically developed custom simulation models lack standard architectures and limit the reuse of design and implementation artifacts across multiple models. In this work, following the design science research methodology, based on a formal model of service delivery systems and applying an adapted software product line (SPL) approach, we create a design artifact for building product lines of IT service delivery simulation models, which vastly simplify and reduce the cost of simulation model design and development. We evaluate the design artifact by constructing a product line of simulation models for a set of IBM's IT service delivery systems. We validate the proposed approach by comparing the simulation results obtained using our models with the results from the corresponding custom simulation models. The case study demonstrates that the proposed approach leads to 5–8 times reductions in the time required to design and develop related simulation models. The potential implications of the application of the proposed approach within an organization are quicker responses to changes in the business environment, more information to assist in managerial decisions, and reduced workload on the process reengineering specialists.

Keywords

IT service delivery, service science, service systems, formal model, simulation, software product lines

Introduction

With the advent of Cloud computing and the proliferation of Information Technology (IT) in all industries, services and their delivery become crucial in the business world. In addition, nowadays such services are often delivered on an unprecedented scale. For example, there are mobile service companies in India, such as Bharti Airtel and Vodafone, that have more than 150 million customers (Telecom Regulatory Authority of India 2013). When services are delivered on a large scale, it is essential to optimize the service delivery processes, as even small inefficiencies have a huge economic impact.

IBM is one of the largest IT service companies and requires design and optimization of numerous IT service delivery lines at IBM's Global Delivery Centers in India and other geographies, where a range of services are delivered by large teams. Each service line follows different processes for the service delivery and supports a range of customers with different service level agreements (SLAs). Currently, the optimization of each service line requires the design and implementation of a specialized service simulation model, which takes significant time and resources. This work is intended to improve the productivity and reuse of design and implementation artifacts in the global optimization team by developing a more efficient service simulation model design and implementation process.

To address the identified problem, we follow the design science research methodology proposed by Peffers et al. (2007) and the design science research guidelines defined by Hevner et al. (2004). According to the formal model of service delivery systems introduced by Banavar et al. (2010), we separate the logic of service specific processes from the general processes of service delivery and formalize interactions between the components of a service delivery system. To capture the variability among IT service delivery systems, we propose a novel application of the software product line (SPL) methodology to the development of families of IT service delivery simulation models. Based on the defined service delivery system model and

[1] IBM Research, Melbourne, Australia
[2] IBM Research, New Delhi, India
[3] IBM Research, Haifa, Israel
[4] Department of Computing and Information Systems, The University of Melbourne, Australia

Corresponding Author:
Anton Beloglazov, IBM Research, Melbourne, Australia.
Email: anton.beloglazov@au.ibm.com

analysis of the SPL knowledge base, we derive a unified modeling language (UML) product line design and development methodology tailored to building simulation models of service delivery systems as an adaptation of the SPL approaches proposed by Gomaa (2004) and Pohl, Böckle, and Linden (2005).

Applying the derived product line methodology and knowledge from analysis of IBM's sample IT service delivery systems, we design a model of a product line of IT service delivery simulation models. The model can be used to develop and implement simulation models of IT service delivery systems, while reducing the required time and resources. We incorporate extensibility into the proposed model via variation points and implement it as a framework in the AnyLogic simulation environment (Borshchev and Filippov 2004), while documenting the variability and providing a set of reusable design and implementation assets. Instead of representing a single model, the proposed product line of simulation models spans a variety of simulation models in the IT service delivery domain. To create a concrete simulation model of a service delivery system, a model designer specializes the product line by specifying a set of required features, parameters, and components. By specialization, the model designer transforms the product line into the desired concrete model without the necessity for redevelopment of the model architecture, and thus reducing the development complexity and time.

The result of design science research is a purposeful IT artifact addressing an important problem according to Hevner et al. (2004). The problem addressed in this work is the lack of a standard architecture and efficient design methodology that could increase productivity and reuse in the design and implementation of IT service delivery simulation models. This work contributes four innovative IT artifacts:

1. A model of IT service delivery systems encompassing a description of the components of a service delivery system and interactions between them in the form of a workflow, as well as formal definitions of service request (SR) characteristics, resource capabilities, and dispatching and resource allocation processes.
2. A methodology for building product lines of simulation models adapted from a combination of the SPL approaches proposed by Gomaa (2004) and Pohl, Böckle, and Linden (2005).
3. A product line of IT service delivery simulation models designed by following the derived product line development methodology.
4. A proof-of-concept instantiation of the product line for a set of related service delivery systems implemented and evaluated using the AnyLogic simulation environment.

According to the definitions of research contributions in design science given by Hevner et al. (2004), we argue that the research contributions of this work are the models and methods listed previously, as well as the proof-of-concept simulation models implemented based on the instantiation of a product line of simulation models for a set of IBM's related IT service delivery systems. The combination of the artifacts produced by this work contributes to the solution of a heretofore unsolved problem, namely, low productivity and reuse in the design and implementation of IT service delivery simulation models.

The remainder of the article is organized as follows. In the next section, we discuss the previous research in the areas of service science and service simulation models, followed by the research methodology applied in this work. Next, we design a model of an IT service delivery system. Then, we introduce our SPL methodology tailored to the design and development of simulation models of service delivery systems. We apply the proposed methodology to the design and development of a product line of IT service delivery simulation models. This is followed by an evaluation case study aimed at validating the applicability and showing the efficiency of the proposed approach. We conclude the article with a discussion and directions for future research.

Related Work

A substantial amount of research has been done in the exploration, formalization, management, engineering, and optimization of service systems (Zhang 2008). Maglio et al. (2006) stated that according to a measurement of the labor job percentages, the economies of the world are shifting labor from agriculture and manufacturing of goods into services. The authors noted that service systems have gained tremendous importance, as they are involved in numerous areas, for example, education, public services, and IT services. They claim that to fully understand service systems, it is necessary to establish a new academic discipline, service science. Chesbrough and Spohrer (2006) seconded the call of Maglio et al. (2006) in their manifesto for service science. Spohrer et al. (2007, 2008) laid a theoretical foundation in the form of definitions of the service science abstractions. These research efforts show that studying, understanding, modeling, and optimization of service systems are important research challenges.

Alter (2010) proposed a framework for describing and analyzing service systems. The proposed framework is focused on the business perspective of service systems and can be used to describe any such system, for example, IT service systems involving compositions of web services. However, the framework does not cover the optimization of service systems, rather it is applied to formalize a body of knowledge and organize the information about services. In contrast, we propose an architecture and an implementation of a product line of IT service delivery simulation models, which can be used for the rapid development of simulation models of services and their simulation-based optimization.

There have been a number of research efforts aimed at the development of simulation frameworks for modeling computing systems that conform to the service-oriented architecture (SOA) principles. For example, Sarjoughian et al. (2008) proposed a SOA-compliant simulation framework supporting the hierarchical composition of service models. In comparison, the scope of our research is broader, as the proposed product line of IT service delivery simulation models covers not just software systems but a more general class of service delivery systems involving human resources and business processes.

Modeling service systems is a complex task, which is explained by the complexity of modeling people, their knowledge, activities, and intentions. Maglio et al. (2006) defined the complexity of service systems as "a function of the number and variety of people, technologies, and organizations linked together in the value creation networks." The high complexity of service delivery systems makes simulation the most suitable way to model and optimize such systems, which is indicated by a range of research works aimed at the development of custom simulation models of service delivery systems.

Diao et al. (2011) designed and implemented a simulation model of a global IT service delivery system to support model-based decision making. The modeled IT service delivery system is within the scope of the current work. The difference is that the simulation model design and development approach proposed in this article is more general and can be applied to creating a family of related models and managing their variability in an efficient way. Anerousis, Diao, and Heching (2011) investigated the trade-off between the cost of service delivery and service quality using discrete-event simulation. The simulation model was applied to minimize the total cost of service delivery under a number of constraints. Similarly, the model falls within the scope of the simulation model design and development approach proposed in this article. Our approach allows the optimization of service delivery systems through parameter variation, for example, using the OptQuest optimization engine built into AnyLogic.

Prashanth et al. (2011) proposed a stochastic optimization algorithm for minimizing the staffing skill levels and shifts of workers under SLA constraints in service delivery systems. The approach proposed in this article is aimed at the efficient design and development of IT service delivery simulation models and does not limit the choice of the optimization algorithm to be applied for optimizing the models. In other words, any suitable optimization algorithm can be applied once a simulation model of the IT service delivery system of interest is designed and implemented.

Diao and Heching (2012) proposed a closed loop performance management scheme integrating service level and service operations management to improve SR dispatching policies. The idea is to enhance SLA management of the dispatching process by leveraging the information about the request severity and SLA target time, as well as dynamic SLA attainment levels in the system. The solution is orthogonal to the simulation model design and development approach proposed in this article, and can be integrated by implementing the corresponding dispatching and resource allocation algorithms that take advantage of the data collected by the system monitor discussed in the next section.

Vemuri (1982) applied discrete-event simulation to model a university research support service system. Kennedy (1969) and Zaki, Ahmed, Cheng, and Parker (1997) developed discrete-event simulation models to optimize a community health service system and an emergency service system, respectively. Rohleder, Bischak, and Baskin (2007) implemented a simulation model based on the system dynamics approach to model a patient service system. Klievink and Janssen (2009) built a simulation model of a public service delivery system using game theory. These articles successfully addressed their specific problems; however, a service delivery simulation framework could vastly simplify and speed up the model design and development processes. In this work, we address this issue by developing and implementing a product line of IT service delivery simulation models that relieves model designers from the necessity for the redevelopment of a simulation model architecture, and significantly increases the reuse of design and implementation artifacts across multiple models.

Research Methodology

We follow the design science research methodology proposed by Peffers et al. (2007). The methodology defines a process of conducting research in design science consisting of a sequence of six activities applied iteratively in order to refine the outcomes of each step based on the obtained knowledge and experience. The research process followed in this work is problem centered, which means it is initiated by a problem definition. The problem definition has been derived from the need for reducing the amount of manual work and creating opportunities for efficient reuse of design and implementation assets in IBM's global service optimization team. The application of the six activities of the research methodology can be summarized as follows:

1. Problem identification and motivation, as given in the introduction.
2. Definition of the objectives of the desired solution, as discussed in the introduction and related work sections.
3. Design and development of the desired artifacts, which in this research comprise creating a model of an IT service delivery system, deriving an efficient product line development methodology tailored to the construction of product lines of simulation models, and designing a model of a product line of IT service delivery simulation models.
4. Demonstration of the solution by instantiating a product line of simulation models for a set of IBM's related IT service delivery systems followed by the implementation of the designed simulation models in the AnyLogic simulation environment.
5. Evaluation of the design artifacts through validation of the simulation models by construction, as well as statistical comparison of the results obtained from the base simulation model of the product line with the results from the corresponding custom simulation model, followed by a comparison of the design and development time required to implement new but related models.

6. Communication of the problem, its importance, produced design artifacts, and evaluation results to practitioners, as well as managerial audiences.

In addition to applying the design science research methodology proposed by Peffers et al. (2007), we follow the widely accepted guidelines for conducting and presenting design science research defined by Hevner et al. (2004), which together with the work by Walls, Widmeyer, and Sawy (1992) created the foundation of the research methodology discussed previously.

A Model of an IT Service Delivery System

According to Hevner et al. (2004), rigorous methods should be applied to both construction and evaluation of a design science research artifact. In relation to the design process, we analytically model a general IT service delivery system through rigorous mathematical formulations based on a previously defined formal model of service delivery systems introduced by Banavar et al. (2010). According to their definition, *a service system* is a network of providers and clients coproducing value through service performances. We model a service system as a system consisting of a set of clients submitting SRs; a set of service providers, each containing a service delivery system; and a service operating system (OS) management system.

Clients request services (SRs) from a service provider by submitting SRs that contain a set of SR characteristics and an entity defining the input for the service, for example, data consumable by process tasks, other SRs or processes. SR characteristics are used by the service delivery system to dispatch the SR to the appropriate process and allocate the necessary resources. Defined more formally, *a service delivery system* is a set of interacting entities, such as people, processes, and products, which are involved in the delivery of one or more services. We model a service delivery system as a system of four major components:

1. A process model.
2. A resource model.
3. A value model.
4. A service OS.

Service processes are sets of tasks connected in the form of directed graphs with deterministic/nondeterministic, conditional/nonconditional transitions. A service is delivered through the execution of one or more service processes via performing sequences of steps defined by these processes. A service delivery system contains a finite set of process types that define the service processes provided by the system. Once an SR is received and the appropriate process type is selected, an instance of the process type is created to serve the SR. The process instance has access to the data included in the SR, which can be used to pass parameters to the process instance.

Resources are allocated for a process instance on a per task basis. Each process task contains information about resource capabilities that are required for its execution. A process task can also contain information about its performance characteristics depending on the number and capabilities of the allocated resources. Process tasks can create new SRs, that is, a process can initiate other processes during its execution. Depending on the process type and the input provided by the SR, the result of the process execution returned to the client can vary, for example, modified input entity, new data. It is important to note that the purpose of a process execution may differ from the result returned to the client, that is, the result may represent metadata describing the actual outcome of the process. In other words, a process can have side effects.

A resource, in the context of a service delivery system, is a broad term that spans IT resources (e.g., computing power, storage, and communication lines), people, facilities (e.g., equipment and physical space), and so on. In other words, a resource is an entity whose amount or capabilities can be measured, and which must be utilized in order to deliver a service. Resources can be expendable and nonexpendable. Banavar et al. (2010) defined resources as capability containers, that is, resources are characterized by the capabilities they exhibit. Each capability can have multiple levels, for example, representing different skills of people or performance characteristics of equipment. A resource is described by the maximum levels of its capabilities, under the assumption that it can also serve the purpose of resources with lower capability levels. In addition, resources are described by the cost associated with their usage per unit of time.

Coproduction in service systems stands for enhancing the value accrued by all the stakeholders (i.e., the clients and providers) as a result of the service execution. A service delivery system contains a value model that defines how the value of the service provider and a client is enhanced during, and as a result of, a service performance. Each process task influences the client's and provider's value appreciations. Apart from the value in terms of the cost, the value of a task can represent the task's contribution, or importance, to the service delivery process. This information can be used to optimize a service delivery process in order to maximize the combined value appreciation by eliminating tasks that bring low value to the process. Another aspect of the value model is the adjustment of the value appreciation according to the SLAs negotiated between a client and a service provider (Dhanesha, Hartman, and Jain 2009). For example, if the SLAs have been violated, the client's and provider's value appreciations must be adjusted in order to reflect this violation. In this case, the provider could be penalized for causing an SLA violation by transforming some of its value into the client's value.

The service OS is that part of a service delivery system that is responsible for managing the resources and processes (Banavar et al. 2010). In our work, we model the service OS as consisting of the following components: contract negotiator, dispatcher, process manager, resource manager, system monitor, complaint resolver, and failure manager. Each component is modeled as a state-dependent control object and is defined as a process. The service OS management system is the component of a service system that manages the service OS processes

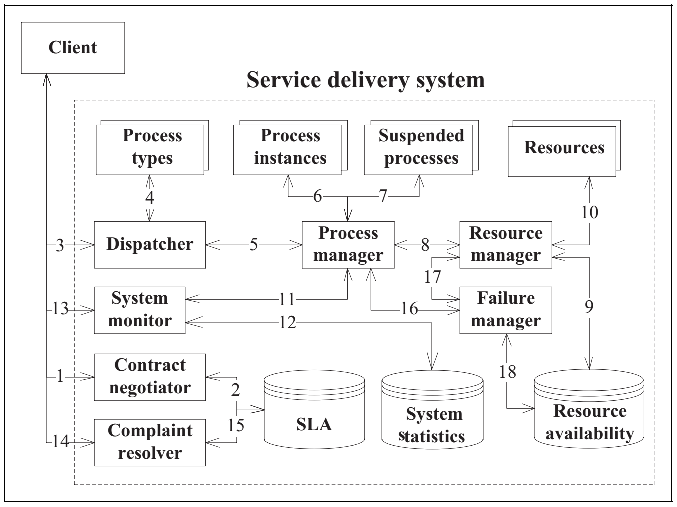

Figure 1. The service delivery system workflow.

themselves. The purpose of this component is to monitor the operation of the service delivery system and adjust its behavior according to predetermined policies. The service OS management system is capable of adding and removing instances of service OS processes, as well as modifying their parameters or changing algorithms. For example, to satisfy the quality of service requirements, extra instances of service OS processes can be created due to an increasing rate of incoming SRs. The operation of the service delivery system components as parts of the service delivery system workflow is detailed in the next section.

Service Delivery System Workflow

Having introduced the main components of the service system model, we can now describe a typical service delivery system workflow shown in Figure 1. For this example, we assume that there is only one instance of each service OS process. However, depending on a specific scenario, it is possible to have multiple instances of dispatchers, process managers, resource managers, and so on. Moreover, the number of instances can be dynamically adjusted according to the policy of the service OS management system. All the service OS components are connected through queues that can be priority-based or can enforce any other policy. Another purpose of the queues is to distribute SRs among multiple instances of service OS processes.

Service delivery starts when a client submits an SR to the service delivery system. Once submitted to the system, the SR gets into the dispatcher's queue. The dispatcher processes SRs sequentially fetching them one by one from the queue. It analyzes the SR's characteristics and selects the appropriate process type to serve the request. The dispatcher forwards the SR along with the selected process type to the process manager's queue.

The purpose of the process manager is to manage the process life cycle: (1) instantiation of the process; (2) scheduling of the process tasks; (3) management of the process preemption; and (4) destruction of the process on completion. Once the SR is received by the process manager, it creates a new instance of the process according to the process type assigned to the SR by the dispatcher. The process manager schedules the process tasks by organizing the control flow around them. The resource allocation is done on a per task basis. Thus, prior to the actual execution of a task, the process manager submits a request for resource allocation for this task to the resource manager. During the resource allocation phase, the process is suspended.

When the request for resource allocation is received, the resource manager fetches the resource availability (RA) data from the RA database to determine the set of resources that are currently available. The RA data reflect the state of resources and can represent the information about busy resources, shifts of the workers, weekends, resource downtimes, and so on. Once a set of available resources is determined, the resource manager applies a resource allocation algorithm to find a subset of resources that correspond to the SR characteristics and the resource capabilities required by the process task.

As an example, sophisticated resource allocation algorithms may also initiate preemption of processes with a lower priority, or apply statistical analysis of the request arrival distribution, and possibly keep resources in reserve to manage the overall performance of the system over time. If some of the resource requirements cannot be satisfied, the request is queued until the resources are available. According to its resource allocation policy, the resource manager selects the actual resources to be allocated to the process task out of the set of suitable and available resources. Then, the resource manager updates the availability data of the allocated resources by setting their state to "busy." It is also possible that the resource manager for various reasons allocates multiple independent tasks at the same time to a single resource. In that case, the resource multitasks by either switching between the tasks in a time-shared manner, such as round-robin, or processing the tasks sequentially in the order of their priorities, while the inactive tasks become suspended.

Once the resources are allocated, the process manager resumes the suspended process task and initiates its actual execution. Upon the completion of the task execution, the resources are released, and the control flow is passed to the next process task if it is required. The results of the process execution and the statistical data are passed to the system monitor, which stores the data and, if necessary, checks whether the performance requirements have been satisfied. The service execution result and statistical data are then sent back to the client as a result of the service invocation.

The contract negotiator handles the process of negotiating the service contract terms with the client and altering them when needed. If the contract terms have been violated during or as a result of the service delivery, the client may lodge a complaint, which is then processed by the complaint resolver. The role of the failure manager is to monitor and handle any failures that occur in the system. If a failure occurs, the failure manager requests suspension of the involved processes and release of the resources, and marks the resources experiencing the problem as "unavailable." Once the problem is eliminated, it sets the state of the resources to "available" and sends a request for resumption of the suspended processes.

Dispatching SRs

Clients submit SRs to a service delivery system from *a set of SRs* $R = \{r_i | i = 1, 2, ..., R\}$. Each SR is described by *a set of characteristics* and contains data that can later be passed to a process task. The characteristics are used by the dispatcher and resource manager to select the appropriate process type and allocate the required resources. A service system contains a set of n SR characteristics, with each of them containing a set of possible values. In other words, there are n sets C_j of *SR characteristic values*, where $j = 1, 2, \ldots, n$. An SR is assigned a value for each characteristic. That is, each SR r_i is described by an n-tuple of the values of n SR characteristics $(c_1, c_2, \ldots, c_n)$, where $c_1 \in C_1, c_2 \in C_2, ..., c_n \in C_n$. The set C of all possible n-tuples of n SR characteristic values is defined in (1).

$$C = \{(c_1, c_2, ..., c_n) | c_1 \in C_1, c_2 \in C_2, ..., c_n \in C_n\} \quad (1)$$

The relation between the set of SRs and the set of all possible n-tuples of SR characteristic values is defined as a function $f : R \to C$.

SRs are assigned the appropriate process types by a dispatcher. The dispatcher analyzes the characteristics of an SR and according to its dispatching algorithm assigns the corresponding process type. A service delivery system contains *a set of process types* $P = \{p_k | k = 1, 2, ..., P\}$. The relation between the set of all possible n-tuples of SR characteristic values and the set of process types is defined as a function $g : C \to P$. Therefore, we can define *a dispatching algorithm d* as a composite of the functions f and g, $d = g \circ f$. The domain of d is R; the range of d is P.

Example. Let SRs of a service delivery system focused on IT infrastructure support be described by three characteristics: SR type $C_1 = \{\text{hardwareProblem, softwareProblem}\}$, SR priority $C_2 = \{\text{lowPriority, highPriority}\}$, and SR complexity $C_3 = \{\text{simple, moderate, complex}\}$. In this case, an SR reporting a low-priority complex software problem is described by a triple (*softwareProblem*, *lowPriority*, *complex*). For this example, the set X of all possible triples of SR characteristic values in the system is defined as shown in equation (2).

$$X = \{(c_1, c_2, c_3) | c_1 \in C_1, c_2 \in C_2, c_3 \in C_3\}. \quad (2)$$

Resource Allocation

The purpose of a resource is to provide one or more capabilities that can be employed during a service process execution in order to deliver the service. *A capability* is defined by a totally ordered or unordered set of its levels. An example of a capability can be the level of Java development skills $X = \{x_l | l = 1, 2, ..., X\}$, which is totally ordered under the $\leq$ relation. Another example of a totally ordered set of capability levels is the central processing unit (CPU) speed of a server $Y = \{y_p | p = 1, 2, ..., Y\}$. An example of a capability with an unordered set of its levels is the tools that a person can work with $Z = \{z_q | q = 1, 2, ..., Z\}$. A resource can have multiple capabilities, which are defined by the maximum level of each capability provided by the resource if the capability levels are ordered, or by an enumeration of the capability levels if they are unordered. It is assumed that a resource can provide a capability at all levels lower than its maximum if the capability levels are ordered.

Let A_i be one of the m available resource capabilities described by a set of its levels. Then, the resources of a service delivery system are described by m sets of *capability levels* for each capability A_i, where $I = 1, 2, \ldots, m$. The set A of all possible m-tuples of resource capabilities is defined in (3).

$$A = \{(B_1, B_2, ..., B_\mu) | B_1 \subset A_1, B_2 \subset A_2, ..., B_\mu \subset A_\mu\} \quad (3)$$

The capabilities of each resource are described by an m-tuple from A. There can be several resources with the same capabilities. Let the system contain *a set of resources* $E = \{e_i | i = 1, 2, ..., \mathrm{E}\}$. Then, there is a relation between the resources and their capabilities $q : E \to A$. At any point in time t, a resource e_i is characterized by its state $s_t(e_i) \in S$, which shows, for example, whether the resource is occupied or can accept further SRs. Since several resources can have the same capabilities and taking into account the time-dependent states of resources, the inverse relation of q is defined as $q^{-1} : A, S \to \wp(E)$, where $\wp(E)$ is the power set of E.

Each process type contains *a set of tasks* $T = \{t_u | u = 1, 2, ..., \mathrm{T}\}$. Resources are allocated on a per task basis. The relation between process tasks and their minimum required resource capabilities is defined by a function $h : T \to A^l$, where l is the number of resources required by the process task.

Based on the given definitions, a set of l sets of resources satisfying the capability requirements of a task $t_u \in T$ can be determined as follows: (1) applying h to the task t_u to obtain a set of required capabilities and (2) applying q^{-1} to each of the l required capabilities and a state from S to obtain a set of suitable resources for each capability. Once a task is mapped on a set of suitable resources, the selection of a particular resource or subset of the suitable resources is done by *a resource allocation algorithm* of the resource manager. The resource mapping function and resource allocation algorithm can be arbitrarily complex and can accept extra parameters as required by the modeler.

According to the defined resource allocation algorithm, the resource manager evaluates the SR characteristics, required and available resources, required performance characteristics, and possibly other factors, and then allocates the appropriate resources to the process task. For example, for a particular process task, the resource manager is able to map the specified priority and complexity of an SR to the Java development skills and CPU performance of a server required to meet the priority and complexity requirements.

There are multiple possible scenarios of resource allocation, which depend on the system requirements. Some examples of resource allocation scenarios are listed subsequently:

1. If there is one or more available resources for each required set of capabilities, the resource allocation algorithm selects one resource out of the available set of resources for each required set of capabilities.
2. If the task is high priority and there are no suitable resources available (e.g., they are all busy), the task can be assigned to busy resources, which suspend their current tasks and switch to the high priority task. The multitasking strategy is not fixed by the model. For instance, apart from the mentioned priority-based preemption, multitasking can be implemented by time-sharing round-robin scheduling. The suspended tasks can be tied down to the original resource, or can be moved to a global queue, from which other resources can retrieve and resume them. When the high priority task is finished, the resources switch back to the suspended tasks if those are still not processed.
3. If the task is low priority and there are no suitable resources available, the task is moved to a priority-based queue of deferred tasks to wait for the required resources to become available.

In addition, a process task may contain information on its performance characteristics depending on the number of allocated resources (if multiple resources with the same capability are supported by the process task) and different levels of the resource capabilities.

Example. Continuing the example from the previous section, assume that the system receives an SR with the following characteristics (*databaseProblem*, *lowPriority*, *complex*). The SR (SR_1) reports an issue of slow response time of some Database (DB) queries in a production system, which requires investigation and optimization. By analyzing the SR characteristics, the dispatcher assigns the SR to a process consisting of the following tasks: (1) creation of a DB snapshot for testing purposes requiring the capabilities of a DB server and DB administrator; (2) profiling and optimization of the problematic DB queries requiring the capabilities of a DB layer programmer at the complex level; and (3) deployment of the optimized DB layer in the production system requiring the capabilities of a DB administrator. Based on the analysis of the resource capabilities required by the process tasks, the resource manager instantiates the process and assigns a DB server to all the tasks, DB administrator to Tasks 1 and 3, and DB layer programmer capable of handling complex problems to Task 2.

While Task 2 is still being served, a new SR (SR_2) comes into the system with the following characteristics (*businessLogicModification*, *highPriority*, *simple*) requiring an addition of a new field in the *customer* domain model. The SR gets dispatched to a process type consisting of the following tasks: (1) modification of the DB schema requiring the capabilities of a DB layer programmer with basic skills; (2) modification of the business logic layer requiring the capabilities of a business logic layer programmer; and (3) deployment of the system. Due to the unavailability of other staff members and high priority of the new SR, Task 1 gets allocated to the same DB programmer working on Task 2 of SR_1. The DB programmer suspends the processing of SR_1 and switches to SR_2. The suspended task is kept assigned to the programmer and gets moved into his queue of deferred tasks. Once Task 1 of SR_2 is completed, the DB programmer resumes the work on Task 2 of SR_1. In this context, the partial execution of the task 2 of SR_1 is represented by elements of F, which in a simple case could just be percentages of the task completion. The resources are released upon the completion of the corresponding tasks. When the processes are completed, the results are sent to the clients reporting the outcomes of the processes and required statistical information.

Designing a Product Line of IT Service Delivery Simulation Models

The SPL methodology is motivated by the desire to improve software development efficiency and maximize code reuse. First introduced by Clements and Northrop (2001), this methodology extends the software architecture design approach to span whole families of software systems. SPL explicitly captures the commonality and variability of different but related software products. The explicit description of variability is necessary to document the anticipated changes and the places where these changes may occur. Clements and Northrop (2001) defined an SPL as "a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way."

In our work, we apply the SPL methodology to model and design an architecture of a product line of IT service delivery simulation models. The SPL methodology allows the modeling of a variety of simulation models, reducing the time and cost of the development of new simulation models of IT service delivery systems. According to our survey and analysis of different SPL approaches and their applicability to the case of modeling a product line of simulation models, we have chosen a combination of the SPL approaches proposed by Gomaa (2004) and Pohl, Böckle, and Linden (2005). In particular, we apply the integrated variability model and phases of the PLUS approach proposed by Gomaa (2004) including dynamic modeling, while splitting the feature model into external features, which are derived from use cases, and internal features, which are defined by the architectural variability that is not dependent on the use cases. Moreover, we use the feature model as a representation of all the variability introduced in the product line, which like the orthogonal variability model proposed by Pohl, Böckle, and Linden (2005) creates a single view of variability. In contrast to complete decoupling of variability information at lower architectural levels as in the orthogonal variability model, the

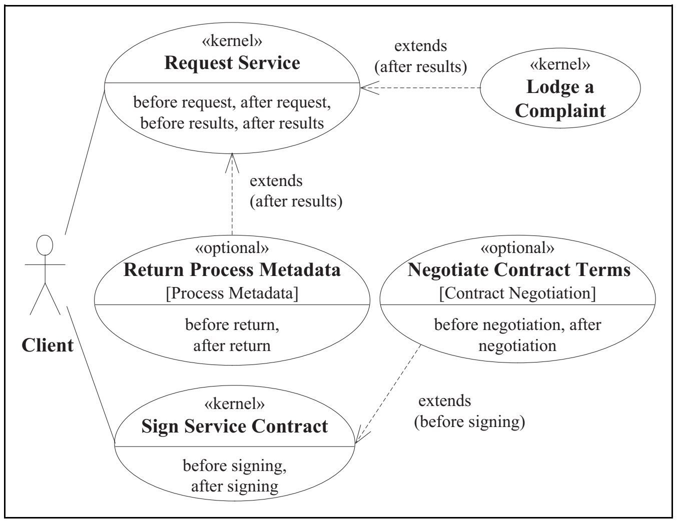

Figure 2. The use case diagram.

variability is defined via feature conditions, which simplifies the development.

Use Case Modeling

The first phase of the Product Line UML-based Software Engineering (PLUS) approach proposed by Gomaa (2004) is use case modeling. During this phase, functional requirements for the product line are specified and modeled as UML use case diagrams. During the use case modeling, functional requirements are defined as sequences of interactions between actors (users of the system) and the system, which is considered to be a black box. In other words, use cases define the external functional requirements of the system. Functional requirements and the associated use case diagrams are divided into three categories: kernel, optional, and alternative. Kernel use cases capture functional requirements that must be met by all members of the product line. Optional use cases are needed only by some members of the product line. Alternative use cases describe functional requirements that are different for different members of the product line. Apart from the categorization of use cases, there are other mechanisms of incorporating variability into a use case model. Small variations can be expressed as conditional alternative branches in a use case. Use cases can extend each other through extension points using the "extends" relationship and conditions. The "includes" relationship can be used to include an abstract use case into several independent use cases.

The main use case of an IT service delivery system is when a client requests service from the system by submitting an SR. Therefore, we define the "Request Service" use case as a kernel use case (Figure 2). During use case modeling, the system is considered to be a black box; therefore, the "Request Service" use case contains two basic actions: A client submits an SR and the system returns the SR processing results. To provide flexibility in the design of future systems, the use case definition includes four extension points: <before request>, <after request>, <before results>, and <after results>. These extension points can be used by other use cases to extend the basic functionality of the system.

Another kernel use case is "Sign Service Contract," which represents the necessity to establish a service contract agreement between a client and the provider prior to the actual service delivery. Often, service providers offer default service contract terms that are implicitly accepted by clients by submitting SRs. However, this is not always the case; therefore, the use case includes two extension points: <before signing> and <after signing> the contract. An example of a use case extension is an optional use case "Negotiate Contract Terms" that extends "Sign Service Contract" at the <before signing> extension point. This use case is enabled if the "Contract Negotiation" feature is supported by the system. This use case represents the ability of a client to negotiate and adjust the service contract terms before requesting the service from the service provider.

"Lodge a Complaint" is a kernel use case that extends the "Request Service" use case at the "after results" extension point. This use case is invoked when a client disputes the results of the service delivery due to, for example, a violation of the SLAs. The "Return Process Metadata" use case is an optional extension of the kernel "Request Service" use case at the <after results> extension point. This use case is enabled if the system supports the "Process Metadata" feature. If it is enabled, the data describing the process of the service execution, such as performance characteristics or any required intermediate results, are collected and returned to the client along with the service process execution results.

Feature Modeling

Once the functional requirements of a system are captured in a use case model, the next step is to derive features from the use cases. A feature is a requirement or characteristic that is provisioned by one or more members of the product line. Features are used to differentiate members of the product line and determine common and optional functionality. Features can be kernel, optional, alternative, or parameterized. Features may depend on each other using the "requires" or "mutually includes" relationships.

According to the PLUS approach, all features are derived from the use case model. However, based on the idea of internal variability proposed by Pohl, Böckle, and Linden (2005), we extend this by dividing features into external and internal. External features capture the external functional requirements of the system and are derived from the use cases. On the other hand, internal features are invisible to the users of the system and are determined by its internal architecture. Internal features are architectural features in the sense that they define variations of the internal system architecture without affecting the external functional requirements. Therefore, internal features usually define lower level variability and depend on a subset of the external features. A combination of the external and internal feature models provides a single view

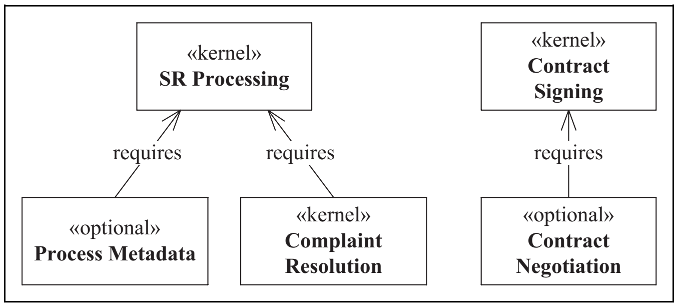

Figure 3. The external features diagram.

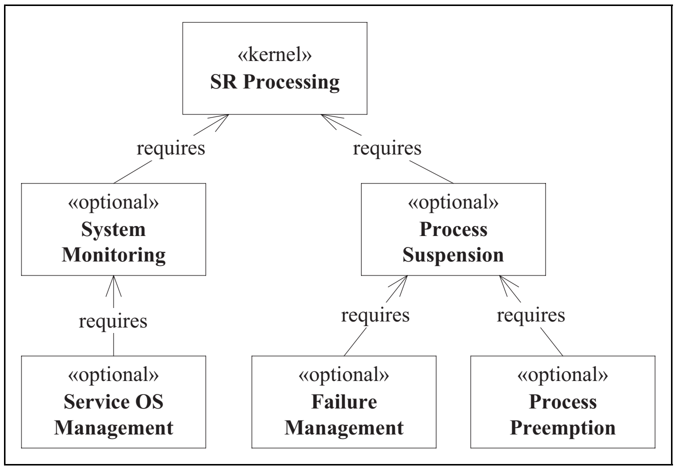

Figure 4. The internal features diagram.

of the product line's variability, as all variations are defined using feature conditions.

Since external features are derived from use cases, we define three kernel features: "SR Processing," "Contract Signing," and "Complaint Resolution." We also define two optional features: "Process Metadata" and "Contract Negotiation." The relationships between these features are depicted in Figure 3. According to the use case model, the "Process Metadata" feature depends on the kernel "SR Processing" feature, whereas the "Contract Negotiation" feature depends on the "Contract Signing" feature. Therefore, in the feature model, we define that the "Process Metadata" and "Contract Negotiation" features require the kernel "SR Processing" and "Contract Negotiation" features, respectively, via the "requires" relationship. We define five optional internal features that depend on the kernel "SR Processing" feature: "System Monitoring," "Service OS Management," "Process Suspension," "Failure Management," and "Process Preemption." The relationships between the internal features are depicted in Figure 4.

The "System Monitoring" feature represents the capability of the service delivery system to continuously monitor the system operation and collect data describing the system state, such as metrics of the performance, cost, and so on. The "Service OS Management" feature describes an additional ability of the service delivery system to manage its service OS processes by real-time changes of their parameters, algorithms, number of instances, and so on, according to predetermined policies and data collected by the system monitoring component. The "Process Suspension" feature allows the system to suspend the execution of service processes and resume them later. The "Failure Management" feature applies the process suspension to interrupt the service processes that are using the resources affected by a failure. The processes are resumed when the failure is eliminated. The "Process Preemption" feature allows the resource manager to preempt service processes that are utilizing the resources, which must be allocated to SRs with higher priorities.

In our approach, the feature model plays the role of the variability model proposed by Pohl, Böckle, and Linden (2005). It is used to define the variability via feature conditions. We apply the defined feature model to specify feature conditions at the lower levels of the product line architecture, such as the static and dynamic models. The PLUS approach applies iterative development; therefore, all the variability introduced at the lower architectural levels is reflected in the feature model during the next iterations. The hierarchies of external and internal features can be further extended when necessary using the "extends" and "mutually includes" relationships and introducing additional optional or alternative features.

Static Modeling

In static modeling, the objects participating in the system's operation are determined and categorized using stereotypes; relationships between them are established using UML class diagrams. At this stage, the variability is defined through interfaces, abstract classes, inheritance, and parameterization. Classes can be kernel, optional, and alternative, corresponding to the defined external and internal features. For our model, we use a convention that all the classes in the static model are represented by interfaces and, therefore, each of them can be extended to implement the variability.

According to the PLUS approach, we divide the objects into two main categories: state dependent objects and entities. State-dependent objects are the objects that implement the control flow of the system. These objects are active in the sense that apart from reacting to external events, they can initiate activities themselves independently of the external environment and other state dependent objects. On the other hand, entities are passive objects, they do not perform actions; rather these objects are acted on by state-dependent objects and are used to store data and states of other objects in the system.

Figure 5 shows the kernel and optional state dependent objects constituting our model as well as their relationships. According to the model, the service delivery system consists of four kernel state dependent objects: Dispatcher, Resource Manager, Process Manager, and Complaint Resolver; and four optional state-dependent objects: OS Manager, Failure Manager, System Monitor, and Contract Negotiator. The functionality of these objects corresponds to the service delivery system workflow introduced earlier.

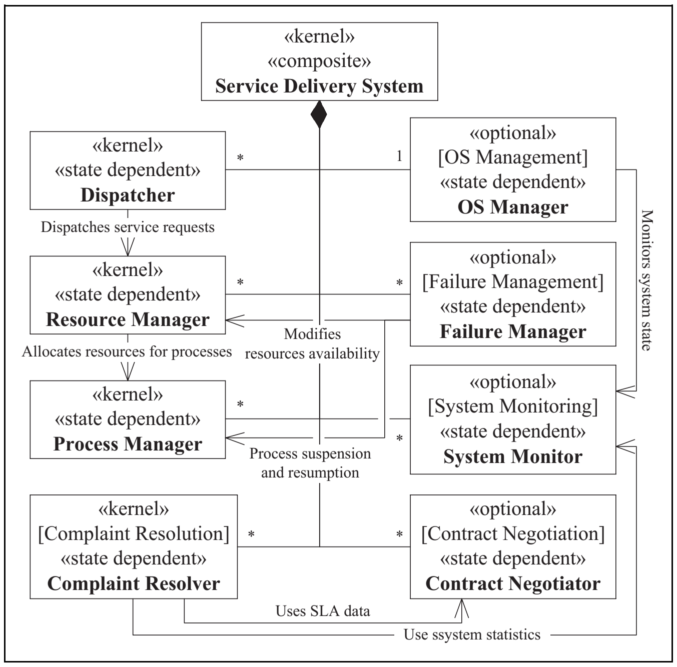

Figure 5. The static model diagram of state-dependent objects.

We split the entity model into three parts: SR, resource, and process type–related entities. An SR contains a single collection of SR characteristics, which in turn contains multiple SR characteristics. The model of resource-related entities divides the resources into two categories: OS resources and regular resources. OS resources are utilized by service OS processes, such as Dispatcher, Resources Manager, and so on. OS Manager's decision on adding and removing instances of OS processes may rely on the availability of OS resources. In contrast, regular resources are allocated to service processes. All resources contain information on their availability using the resource availability data abstraction. Similar to SRs, each resource embodies a collection of resource capabilities containing a set of resource capability values.

Similar to the resource entity model, process types are split into OS process types and regular processes types. OS process types represent classes of service OS processes (e.g., Dispatcher and Resource Manager), whereas instances of service processes are created based on regular process types. Regular process types contain sets of process tasks that are executed in order to deliver the service. Each instance of a process contains the corresponding process type data abstraction and is stored in a process pool.

Dynamic Modeling

During the dynamic modeling phase, UML communication diagrams are developed to represent message passing, control, and sequencing between the objects comprising the system. Communication diagrams model different scenarios and are parameterized by internal and external features. Communication diagrams can be kernel, optional, and alternative. Variant diagrams show the impact of particular variation points in the original diagram and depict only the required modifications of the original diagram. Using UML state charts, state machine modeling describes the internal behavior of state-dependent objects defined during the static modeling phase. Due to the lack of space, we do not include the dynamic model and state machine model diagrams in this article; however, the interested reader may find them online.[1]

Implementation in AnyLogic

We have implemented the proposed model of a product line of IT service delivery simulation models as a Java framework in the AnyLogic simulation environment (Borshchev and Filippov 2004). The AnyLogic simulation environment has been chosen for its approach to the model development, which is a combination of graphical interface-based modeling using an extensive library of components with the ability to program the behavior of custom components in Java. The implementation of our framework features generic interfaces and abstract classes with pre-implemented base functionality. The framework contains all the necessary components, which can be used directly, extended, or completely overridden if required. This adds to the flexibility in the design of simulation models, as the base architecture and components of the framework are reusable and customizable in accordance with the product line model.

Evaluation

Evaluation is an essential component of a design science research process (Hevner et al. 2004). In our work, the resulting IT artifacts are built upon each other in the following sequence: (1) the model of an IT service delivery system is a base for defining requirements and deriving a methodology for building product lines of IT service delivery systems; (2) the methodology is applied to design a model of a product line of IT service delivery simulation models; and finally (3) the product line model is used to build an instantiation of a product line of a set of IBM's related IT service delivery systems represented by a base model and a number of its major modifications, as discussed in the following sections. This sequence of derivations automatically validates the functionality and completeness of the artifacts, as well as their consistency with the requirements and constraints of the problem, as long as the final artifact, that is, the instantiation, is validated and evaluated.

Since the whole proposed approach of applying the SPL methodology to designing families of IT service delivery simulation models is novel, a comparison with alternative solutions is not feasible. Therefore, we evaluate the instantiation of the product line of IT service delivery simulation models in the following two ways:

1. We construct a proof-of-concept product line of several related IBM's IT service delivery systems and implementing the corresponding simulation models in the

Table 1. SR Parameter Probability Distributions.

	Account 1	Account 2	Account 3	Account 4	Account 5	Account 6
SR complexity probabilities						
Low complexity	0.92	0.76	0.75	0.62	0.89	0.68
Medium complexity	0.08	0.24	0.25	0.38	0.11	0.32
SR priority probabilities						
Priority 1	0.09	0.43	0.04	0.36	0.00	0.33
Priority 2	0.04	0.14	0.48	0.13	0.02	0.08
Priority 3	0.86	0.40	0.34	0.46	0.46	0.56
Priority 4	0.01	0.03	0.14	0.05	0.52	0.03

Note. SR = service requests.

AnyLogic simulation environment. We validate the correctness of the implemented model by establishing a statistical equivalence between the results produced by the base simulation model of the product line and the results from the corresponding custom simulation model using real data as the input for simulations.

2. We show that the artifacts satisfy the defined objectives of reducing the development efforts by comparing the time taken to produce a set of related simulation models based on the product line with the time required to produce the corresponding custom simulation models estimated by an experienced IT service delivery simulation model designer.

Proof-of-Concept Validation

In order to validate the proposed approach, we implemented a proof-of-concept product line of simulation models of a family of real-world IT service delivery systems in the domain of data center management (Banerjee, Dasgupta, and Nirmit, 2011). The related IT service delivery systems were represented by a base simulation model of the product line and a set of related simulation models produced through major modifications of the base model. The simulation models have been implemented using the AnyLogic simulation environment and the framework described in the previous section. The validation consisted in showing the correctness and applicability of the proposed product line of IT service delivery simulation models to modeling real-world systems through a comparison of simulation results produced by a model implemented using the proposed framework with results produced by the corresponding custom simulation model that has been specifically designed and implemented to model the service delivery system.

In the domain of data center management, customer-owned IT infrastructures are supported and managed by specialized third-party service providers. Customers submit SRs that are remotely processed by Service Workers (SWs) of the service providers. The base IT service delivery system simulated in this case study is one of IBM's IT service delivery systems, which is described in detail by Banerjee, Dasgupta, and Nirmit (2011).

The modeled service delivery system serves several customer accounts. SRs submitted by each account are characterized by their complexity and priority, which are mapped onto SR characteristics. There are two levels of complexity $C_1 = \{\text{lowComplexity, moderateComplexity}\}$, and 4 priority levels $C_2 = \{1, 2, 3, 4\}$. In our experiments, we used real workload traces and statistics collected from the modeled IBM's service delivery system. Both the product line based and custom simulation models were supplied with the same input data. The workload traces used in the simulation have been obtained from six customer accounts, which are anonymized due to confidentiality concerns.

Table 2. Service Time Distribution Parameters.

	Priority 1	Priority 2	Priority 3	Priority 4
Low complexity SRs				
Mean	42.02	53.20	32.73	42.52
SD	51.29	53.99	35.14	38.90
Moderate complexity SRs				
Mean	56.83	68.13	50.74	55.50
SD	57.47	65.10	50.37	48.26

Note. SD = standard deviation; SR = service request.

The complexity and priority of an SR follow probability distributions shown for each account in Table 1. The arrival times of SRs follow an exponential distribution with the arrival rate changing every hour on a 7-day cycle for each account independently. Due to the large number of arrival rates, we do not include them in this article. Each SR requires a period of time to be processed, which is referred to as the service time and is log-normally distributed with the mean and standard deviation defined separately for each combination of complexity and priority, as shown in Table 2. SRs waiting in the queue are ordered based on their priority: high priority SRs are served first. Moreover, if no resource is available for a just arrived high priority SR, a lower priority SR is preempted, and the released SW starts working on the high priority SR. The preempted SR waits in a queue for the SW to become available and resume the processing.

At every point in time, there are a number of SWs each capable of serving SRs with some maximum level of complexity. The capabilities of SWs are mapped onto the corresponding resource capabilities from the set of available resource capabilities $A_1 = \{\text{low, medium, high}\}$. An SW is available between the specified start and end times of his shift at specific days of week, which are modeled as resource availability changing depending on the time of the day and day of the week. Normally, a highly skilled SW works on SRs with the maximum complexity, which the SW is capable of serving. There is a policy, referred to as *the swing policy*, according to which highly skilled SWs start working on SRs with a lower complexity when a specified threshold on the number of pending SRs with the lower complexity is exceeded. For the purposes of this case study, a pull-based dispatching policy was applied (Banerjee, Dasgupta, and Nirmit, 2011). According to this policy, an SW that becomes available fetches the next SR from the global prioritized queue of SRs.

The metric of interest in this case study is the response time. The response time is the time period from the moment when an SR enters the system to the moment when its processing is completed and it exits the system. We focus on this metric, as it describes the overall system operation encompassing the dispatching of SRs, resource allocation, preemption, and so on. We aim to obtain a nonstatistically significant difference in the response times produced by the model implemented using the proposed approach and the custom model.

We conducted a set of *t*-tests to compare the response times obtained from the two simulation models for all combinations of accounts, complexity, and priority. The resulting *p* values are listed in Table 3. All the *t*-tests resulted in *p* values > .05, which means that there are no statistically significant differences in the response times produced by the models. Therefore, we accept the null hypothesis that there is no difference in the response time produced by the two simulation models. This fact demonstrates the correctness of the proof-of-concept simulation model implemented based on the proposed framework and product line of IT service delivery simulation models.

Table 3. *p* Values From the Conducted *t*-Tests for Comparing the Response Times Produced by the Simulation Models.

	Priority 1	Priority 2	Priority 3	Priority 4
Low complexity SRs				
Account 1	0.948	0.870	0.194	0.627
Account 2	1.000	0.206	0.222	0.064
Account 3	0.982	0.806	0.964	0.991
Account 4	0.921	0.655	0.601	0.578
Account 5	0.993	0.845	0.923	0.875
Account 6	—	0.990	0.797	0.084
Moderate complexity SRs				
Account 1	0.855	0.786	0.383	0.893
Account 2	0.886	0.589	0.518	0.240
Account 3	0.869	0.965	0.983	0.432
Account 4	0.513	0.831	0.701	0.777
Account 5	0.729	0.841	0.902	0.924
Account 6	—	0.850	0.476	0.793

Note. SR = service requests.

Table 4. Design and Development Time Comparison.

Modification	Product Line	Custom Model
Push-based dispatch	5 hours	40 hours
High-medium swing	2 hours	12 hours
Rest breaks of SWs	3 hours	15 hours

Note. SWs = Service Workers.

Comparison of Design and Development Time

In order to evaluate the productivity gains brought by the proposed approach, we implemented three major modifications in the initial service delivery simulation model described in the previous section to produce three new models with similar but different functionality. The aim of this experiment is to evaluate the time taken to implement new but related models compared with implementing the corresponding modifications in the custom simulation model. The first modification was the implementation of a push-based dispatching policy, in which the dispatcher assigns SRs to SWs as soon as they arrive (Banerjee, Dasgupta, and Nirmit, 2011). Each SW maintains his own queue of pending SRs prioritized by the deadline. If the shift of an SW ends before all of his SRs have been processed, the remaining SRs are reinserted into the dispatcher's queue. This dispatching policy supports preemption: Each SW maintains his own queue of preempted SRs waiting for the SW to become available and resume the processing.

The second modification was the implementation of the swing policy similarly to the original model, but in addition to assigning low complexity SRs to medium skill SWs, moderate complexity SRs are assigned to high skill SWs. The third modification was the implementation of rest breaks during the shifts of SWs. Each SW can take three rest breaks during his shift, where the total duration of breaks is calculated from the specified efficiency parameter of SWs. The duration of each of the three breaks is calculated according to the 1:4:1 proportions, respectively. A break is scheduled randomly during every one third of the shift. SWs cannot accept new SRs during their breaks. If an SR is currently being served, its processing is suspended at the beginning of the SW's break and resumed at the end of the break.

The development time taken for the implementation of the described modifications along with time estimates for the corresponding custom model modifications suggested by an experienced model designer are shown in Table 4. The results indicate that for these particular modifications, the application of the proposed framework and product line of IT service delivery simulation models has led to 5–8 times reductions in the development time compared with the time required to implement these modifications in the custom simulation model. This is explained by the fact that in contrast to the custom model, the proposed model takes advantage of the standard architecture provided by the product line of IT service delivery simulation models, its flexibility and extensibility leading to significant productivity gains, while providing correct simulation results as demonstrated by our case study presented in the previous section.

Discussion

Hevner et al. (2004) argue that design science research must be presented both to technology-oriented and management-oriented audiences. Due to the confined space, the current article focuses on the technological component of the communication to provide sufficient details enabling the implementation of the proposed artifacts. This is important to allow organizations and practitioners to take advantage of the benefits provided by the artifacts. This section is tailored to managerial audiences and clarifies the implications of the proposed solutions and effective ways of applying the produced artifacts within organizations. In addition, following Gregor and Hevner (2013), contributions of this work to the existing literature in the area are discussed.

This article describes a methodology and tools for modeling and optimizing the operational aspects of service delivery systems. The artifacts are relevant to service managers and process engineers who need to improve the performance of a variety of similar but distinct service delivery processes. Simulation is an efficient approach to both evaluating service delivery systems during the design process, and optimizing existing service delivery systems by varying parameters of the service processes. However, the design and development of simulation models of service delivery systems is a resource and time consuming task requiring highly skilled specialists in process engineering.

The related literature explored design, modeling, simulation, and optimization of service delivery systems as discussed in the related work section. Many research efforts focused on the analysis and optimization of particular instances of service delivery systems, for example, Kennedy (1969), Zaki, Ahmed, Cheng, and Parker (1997), and Anerousis, Diao, and Heching (2011). This work extends the current body of knowledge in several ways. First of all, it distills the components of an IT service delivery system and their interactions, and mathematically formalizes the concepts of SR characteristics, resource capabilities, and dispatching and resource allocation processes, thereby building upon and extending the work of Maglio et al. (2006), Spohrer et al. (2007, 2008), and Banavar et al. (2010). This work contributes a methodology for developing product lines of simulation models based on the SPL approaches proposed by Gomaa (2004) and Pohl, Böckle, and Linden (2005). It also demonstrates the application of the methodology to the design and development of a product line of IT service delivery simulation models. Finally, it presents a proof-of-concept instantiation of the product line utilizing the AnyLogic simulation environment (Borshchev and Filippov 2004), which validates and showcases the proposed approach.

The proposed approach is general in a sense that it can be applied to creating families of simulation models of IT service delivery systems and managing their variability in an efficient way. Instead of developing a special simulation model for each service delivery system, model designers can now develop a product line of domain-specific IT service delivery simulation models. Then, to instantiate a simulation model of a concrete IT service delivery system, they can specialize the developed product line by specifying the required features, parameters, and components at the predefined variation points. The product line development methodology enables flexibility of the approach by efficiently managing extensions of existing components and additions of new components and variation points.

In addition to increasing the productivity of model designers by enabling the reuse of both architectural and implementation artifacts across the multiple models, the approach allows the optimization of IT service delivery systems through the application of various optimization techniques, such as metaheuristics. The potential implications of the application of the proposed approach within an organization are quicker responses to changes in the business environment, more information to assist in managerial decisions, and reduced workload on the process reengineering specialists.

Conclusions and Future Directions

This work applies the design science research methodology to address the problem of excessive manual work and low reuse of design and implementation assets in the development of IT service delivery simulation models in IBM's global service optimization team. We have applied an adapted SPL methodology to design and develop a model of a product line of IT service delivery simulation models. We based the proposed model on the formal model of service delivery systems introduced by Banavar et al. (2010).

We have implemented a proof-of-concept instantiation of the proposed product line model for a set of IBM's related IT service delivery systems as a Java framework in the AnyLogic simulation environment, which has been used to validate and evaluate the proposed approach. The results of the simulation experiments have shown that the model implemented based on the product line produces correct results, while due to the reuse capabilities and extensibility provided by the product line reducing the design and development time required for producing new but related simulation models by 5–8 times compared with developing the corresponding custom simulation models.

In summary, this work makes the following contributions: (1) a workflow model of an IT service delivery system and formal definitions of SR characteristics, resource capabilities, and dispatching and resource allocation processes; (2) a methodology for building product lines of simulation models adapted from a combination of the SPL approaches proposed by Gomaa (2004) and Pohl, Böckle, and Linden (2005); (3) a product line of IT service delivery simulation models designed by following the proposed methodology; and (4) a proof-of-concept instantiation of the product line for a set of related IT service delivery systems.

An important future research direction is further mathematical formalization and analysis of the IT service delivery system model, which would provide more insights and improve our understanding of service delivery systems. Another interesting research direction is extension of the proposed model to accommodate a more general class of service delivery systems spanning other domains, such as public service delivery. A product line of service delivery simulation models following such a generic model could be applied to a wide variety of service delivery systems. An expanded application area would lead to the creation of multiple variants of system components that can be efficiently managed using the techniques of the product line development methodology.

Acknowledgment

The authors would like to thank Julia Rubin (IBM Research, Haifa, Israil) and the anonymous reviewers for their constructive comments and suggestions on enhancing the article.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) received no financial support for the research, authorship, and/or publication of this article.

Note

1. Extra materials. http://github.com/beloglazov/jsr-2014.

References

Alter, Steven (2010), “Service System Fundamentals: Work System, Value Chain, and Life Cycle,” *IBM Systems Journal*, 47 (1), 71-85.

Anerousis, Nikos, Yixin Diao, and Aliza Heching (2011), “The Cost of Service Quality in IT Outsourcing,” in *Proceedings of the 2011 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, Dublin, Ireland 773-784.

Banavar, Guruduth, Alan Hartman, Lakshmish Ramaswamy, and Anatoly Zherebtsov (2010), “A Formal Model of Service Delivery,” in *Handbook of Service Science*, Paul P. Maglio, Cheryl A. Kieliszewski, and James C. Spohrer, eds. Service Science: Research and Innovations in the Service Economy, New York, NY, USA: Springer, 481-507.

Banerjee, Dipyaman, Gargi Dasgupta, and Nirmit Desai (2011), “Simulation-Based Evaluation of Dispatching Policies in Service Systems,” in *Proceedings of the 2011 Winter Simulation Conference*, 779-791.

Borshchev, Andrei and Alexei Filippov (2004), “AnyLogic—Multi-Paradigm Simulation for Business, Engineering and Research,” in *Proceedings the 6th IIE Annual Simulation Solutions Conference*, 1-17.

Chesbrough, Henry and Jim Spohrer (2006), “A Research Manifesto for Services Science,” *Communications of the ACM*, 49 (7), 35-40.

Clements, Paul and Linda Northrop (2001), *Software Product Lines*. Boston, MA, USA: Addison-Wesley.

Dhanesha, Ketki A., Alan Hartman, and Anshu N. Jain (2009), “A Model for Designing Generic Services,” in *Proceedings of the IEEE International Conference on Services Computing*, 435-442.

Diao, Yixin and Aliza Heching (2012), “Closed Loop Performance Management for Service Delivery Systems,” in *Proceedings of the 2012 IEEE Network Operations and Management Symposium (NOMS)*, 61-69.

Diao, Yixin, Aliza Heching, David Northcutt, and George Stark (2011), “Modeling a Complex Global Service Delivery System,” in *Proceedings of the 2011 Winter Simulation Conference (WSC)*, 690-702.

Gomaa, Hassan (2004), *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. Boston, MA, USA: Addison-Wesley.

Gregor, Shirley and Alan R. Hevner (2013), “Positioning and Presenting Design Science Research for Maximum Impact,” *MIS Quarterly*, 37 (2), 337-356.

Hevner, Alan R., Salvatore T. March, Jinsoo Park, and Sudha Ram (2004), “Design Science in Information Systems Research,” *MIS Quarterly*, 28 (1), 75-105.

Kennedy, Fredric D. (1969), “Development of Community Health Service System Simulation Model,” *IEEE Transactions on Systems Science and Cybernetics*, 5 (3), 100-207.

Klievink, Bram and Marijn Janssen (2009), “Improving Integrated Service Delivery: A Simulation Game,” in *Proceedings of the 10th Annual International Conference on Digital Government Research*, 73-78.

Maglio, Paul P., Savitha Srinivasan, Jeffrey T. Kreulen, and Jim Spohrer (2006), “Service Systems, Service Scientists, SSME, and Innovation,” *Communications of the ACM*, 49 (7), 81-85.

Peffers, Ken, Tuure Tuunanen, Marcus A. Rothenberger, and Samir Chatterjee (2007), “A Design Science Research Methodology for Information Systems Research,” *Journal of Management Information Systems*, 24 (3), 45-77.

Pohl, Klaus, Günter Böckle, and Frank van der Linden (2005), *Software Product Line Engineering: Foundations, Principles, and Techniques*. New York, NY: Springer-Verlag.

Prashanth, Lakshmanrao A., Horabailu L. Prasad, Nirmit Desai, Shalabh Bhatnagar, and Gargi Dasgupta (2011), “Stochastic Optimization for Adaptive Labor Staffing in Service Systems,” in *Proceedings of 9th International Conference on Service Oriented Computing (ICSOC)*, 487-494.

Rohleder, Thomas R., Diane P. Bischak, and Leland B. Baskin (2007), “Modeling Patient Service Centers with Simulation and System Dynamics,” *Health Care Management Science*, 10 (1), 1-12.

Sarjoughian, Hessam, Sungung Kim, Muthukumar Ramaswamy, and Stephen Yau (2008), “A Simulation Framework for Service-Oriented Computing Systems,” in *Proceedings of the Winter Simulation Conference*, 845-853.

Spohrer, Jim, Paul P. Maglio, John Bailey, and Daniel Gruhl (2007), “Steps Toward a Science of Service Systems,” *The Computer Journal*, 40 (1), 71-77.

Spohrer, Jim, Stephen L. Vargo, Nathan Caswell, and Paul P. Maglio (2008), “The Service System is the Basic Abstraction of Service Science,” in *Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, 104-113.

Telecom Regulatory Authority of India (2013), “Information Note to the Press,” Press Release No. 92, Telecom Regulatory Authority of India, New Delhi.

Vemuri, Seshagiri Rao (1982), “A Simulation Based Methodology for Modeling a University Research Support Service System,” *Socio-Economic Planning Sciences*, 16 (3), 107-120.

Walls, Joseph G., George R. Widmeyer, and Omar A. El Sawy (1992), “Building an Information System Design Theory for Vigilant EIS,” *Information systems research*, 3 (1), 36-59.

Zaki, Ahmed S., Hsing Kenneth Cheng, and Barnett R. Parker (1997), “A Simulation Model for the Analysis and Management of an Emergency Service System,” *Socio-Economic Planning Sciences*, 31 (3), 173-189.

Zhang, Liang-Jie (2008), “EIC Editorial: Introduction to the Body of Knowledge Areas of Services Computing,” *IEEE Transactions on Services Computing*, 1 (2), 62-74.

Author Biographies

Anton Beloglazov is a researcher at IBM Research—Australia. He works in the areas of Cloud computing, distributed systems, and simulation. He holds a PhD in Computer Science from the University of Melbourne, Australia. This work has been done during his internship at IBM Research Laboratory in Bangalore, India.

Dipyaman Banerjee is a researcher at IBM Research—India. He works in the areas of Cloud computing, and analysis and optimization of IT service systems. He holds an MS degree in Computer Science from the University of Tulsa, Oklahoma. His research interests also include distributed systems, machine learning, and game theory.

Alan Hartman is a senior researcher at IBM Research—Israel. He leads the research efforts on data privacy issues. He was the service science lead for IBM Research India from 2008 to 2011. He holds a PhD in Mathematics from the University of Newcastle, Australia. He has published widely on combinatorial mathematics, software engineering, and service science since 1977.

Rajkumar Buyya is professor of Computer Science and Software Engineering; and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft, a spin-off company of the University, commercializing its innovations in Grid and Cloud Computing.