Analysis and simulation of business solutions in a service-oriented architecture

M. Kano
A. Koide
T.-K. Liu
B. Ramachandran

Modeling and simulation of business processes is a powerful capability for analysis of business solutions in a service-oriented architecture (SOA). In this paper, we describe analysis techniques that are applicable during the design-time and runtime development of business solutions to estimate business process performance. During the design phase, our analysis framework converts the business process model, which is annotated with additional information, into a more granular model by using an underlying middleware model that describes all the middleware components in an SOA. The resulting model can then be evaluated in terms of performance and cost. We discuss a prototype implementation that uses WebSphere® Business Integration (WBI) Modeler and present the results of a case study. After the design phase, several services required to support the business process execution may exist, but others may need to be newly developed. Our runtime simulation framework supports this by allowing users to simulate implementation models consisting of real and simulated services for function and performance testing. Furthermore, when new services are available, they can be easily included in the analysis by switching from the Simulator to the new service. We discuss a prototype implementation for this capability that uses the Process Choreographer of WBI Server Foundation and present results of a case study.

INTRODUCTION

Service-oriented architectures (SOAs) have been proposed as a mechanism to address pressures of IT organizations to support alignment with business requirements that are changing at an increasing rate[1] and to simultaneously reduce costs. Moreover, enterprise architectures are heterogeneous and require integration of new technology with different types of existing technology in a flexible manner, while satisfying business performance needs. In order to alleviate the needs of constant change, the enterprise architecture should provide a platform-independent layer for building loosely coupled application services. By doing so, SOA can integrate

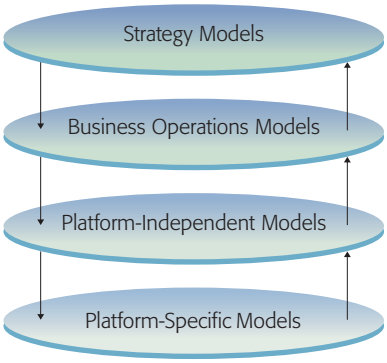

Figure 1
Model-driven enterprise

business processes within and between enterprises with the people and data that are required for their execution.

Web Services offer a specific approach to implement service-oriented architectures.[2] A Web service (WS) separates the interface specification from the implementation and is network-accessible through standardized XML (Extensible Markup Language) messaging. It provides a platform-neutral programming model that can be used to integrate loosely coupled business systems. The Web service is described using a standard, formal XML notation, called its service description, that can be published with a service registry. Service requestors may find the service through the registry and then directly bind to the service and invoke it.

In a related development, the Object Management Group, Inc. is promoting the concept of Model Driven Architecture** (MDA**) as an enabler of flexibility in IT infrastructures.[3] MDA allows machine-readable-application and data models to be defined that can enable (1) longer-term flexibility in implementation and integration and (2) ease of maintenance and testing. The models can be used to generate code that can be validated against requirements and tested against various infrastructures. MDA provides an approach to specify a system independent of its platform and further, to transform the system specification into a platform-specific specification. This approach consists of three types of models:

1. *Computation-Independent Model (CIM)*—This models the environment in which the system is expected to operate, thus specifying the system requirements, and acts as the bridge between domain experts and system architects.
2. *Platform-Independent Model (PIM)*—This is a platform-independent view of the system.
3. *Platform-Specific Model (PSM)*—Based on the PIM and the choice of a specific platform for implementation, a platform-specific view of the system can be developed, which is the basis for code generation.

Synthesizing the SOA and MDA concepts for business process management, the concept of a model-driven enterprise has been proposed to design, develop, deploy, and manage enterprise solutions.[4] This is a framework consisting of four layers of models (see ***Figure 1***). In the topmost layer, strategy models are used to specify the business objectives and context. Next, operational models describe business operations and how they achieve the strategic objectives. These two layers, when viewed together, are similar to the CIM layer in MDA, with the subtle difference that they constitute a business perspective rather than a system perspective. The bottom two modeling layers are execution models and implementation models, and these directly map to the PIM and PSM layers in the MDA framework.

The focus of this paper is to describe analysis techniques that are applicable to the model-driven enterprise framework. The analysis techniques address one of the core value propositions of the model-driven approach; that is, the models and the code generated from them can be validated against requirements, tested against various infrastructures, and used to directly simulate the behavior of the system being designed. As a result, the enterprise architecture can be changed more readily in response to changing business needs. The analysis techniques may be classified further as design-time or runtime, based on whether they are applicable to the models, to the generated code, or to both. The value proposition of such techniques is to identify potential issues and defects in the business solution earlier in the business-process-management life cycle, thereby helping reduce costs in designing and deploying the business solution.

In the second section of the paper, "Analysis Methods for the model-driven enterprise framework," we describe the different analysis techniques applicable to different layers in the MDA framework.

In the third section, "Design-time business performance modeling," we describe how to model and analyze business processes from the IT-infrastructure point of view, applicable during design time. In the fourth section, "Simulating PSMs," we describe how to model and analyze business processes during runtime. The proposed methods are also illustrated with case studies. We then conclude with remarks on further directions for this work.

ANALYSIS METHODS FOR THE MODEL-DRIVEN ENTERPRISE FRAMEWORK

By separating the platform-independent aspects of a solution from the platform-specific aspects and the resulting code, SOA and MDA support reuse of solution components and render the business solution more flexible and adaptable to changes in business requirements. Moreover, using machine-readable application and data models enables analysis, testing, and refinement during the design phase, rather than the implementation phase when the cost of changes is much higher.

In the CIM layer, several techniques such as Systems Dynamics and Value Modeling can be used to analyze different aspects of this layer. Systems Dynamics is an appropriate technique to analyze high-level implications of different strategies when sufficient information is not available to develop a detailed operational model of the business or system, but high-level information is available on the key system metrics and their relationships.[5] For example, Reference 5 has detailed descriptions using Systems Dynamics models of how supply chains can exhibit oscillations and instabilities that could be better managed by using different supply chain strategies. Value Modeling approaches are useful to assess the business value of different initiatives based on a value driver tree approach.[6]

When operational details of the business process are known, techniques such as discrete event simulation are relevant. First, the operational process is modeled by mapping the business process flows, identifying the activities and subprocesses contained therein in a hierarchical manner, and associating resource requirements and costs, where appropriate.

The entire process can be simulated to estimate different process metrics, such as resource utilizations and costs, and used to run different what-if process scenarios to identify a good process design.[7]

The analysis techniques described earlier are mature and have been used extensively to model and optimize business processes. Recently, transformation methods have been developed to generate queuing models automatically from a description of the business process, and these methods offer the promise of rapidly estimating steady-state metrics for the business process without performing simulation.[8]

There has been considerable work in the design-time and runtime analysis of IT infrastructures (several references cited in the sections "Design-time business performance modeling" and "Simulating PSMs."). Nonetheless, these are disconnected from operational business process models and hence are of limited value for analysis in the SOA context, because changes in the business models cannot be transformed to infrastructure models and analyzed automatically. In order to realize the value proposition of SOAs, one needs to be able to identify and model implications of process changes in terms of their impact on the IT infrastructure. These implications are addressed in this paper by using MDA as a framework to develop models at different levels and by applying appropriate analysis techniques to analyze the infrastructure.

The rest of the paper deals with techniques applicable to design-time and runtime analysis of IT infrastructures. By design-time analysis of an IT infrastructure, we refer to the analysis of an operational model of the business process with IT-level depth, that is, an operational model composed of IT-level activities and consuming IT resources. The overall objective here is to estimate resource and process bottlenecks, given a workload and deployment topology, and further, to identify a deployment topology that meets specified performance targets. By runtime analysis of an IT infrastructure, we refer to the analysis performed during an intermediate stage in the application development life cycle, wherein executable code has been generated from the PSM for some business solution components, and there may exist other solution components for which either code may not have been developed or code may exist, but may not yet be integrated with the business solution. Such situations may occur often when some parts of the business process are changed. In using the MDA approach, we may regenerate code for the components of the process that have changed but may want to test the process execution with the new components, without plac-

ing at risk the already existing components that are being executed for the business process. The overall objective here is to validate the application and process design and enable early functional testing of the business solution to identify and remove defects at an early stage of the development and integration life cycle, thereby reducing costs. Moreover, the performance of the application design can be tested against various deployment topologies to provide estimates for hardware sizing and middleware configuration.

DESIGN-TIME BUSINESS PERFORMANCE MODELING

The execution of most business processes requires the support of IT services to achieve the objectives of the business processes. The business processes designed for a business transformation project typically have objectives, such as improving human productivity, cutting cost, increasing process throughput, and improving the visibility of business operation for Business Performance Management (BPM). It is very crucial for a business to understand the potential benefits and cost in designing a business process with the supporting IT services.

IT services can be loosely classified into three types according to their primary benefits to a business. One type of IT service provides users with the information that is needed or helpful for completing a task in a business process. This type of IT service improves the productivity of human resources by shortening the time that it takes for users to complete a task, for example, a Web-portal application that assists users in filling out procurement orders. A second type of IT service automates the tasks of a business process by running business applications on computers. This type of IT service can reduce labor cost and human error, shorten the process cycle time (elapsed time between process instantiation and termination), and increase the throughput of a business process, among other benefits, for example, an enterprise-application-integration (EAI) system that sends an update message to an SAP application upon receiving a notification message from a Siebel Systems, Inc. application, signaling the change of customer contact information. An EAI system can automate the synchronization of customer contact information stored on different enterprise applications without human involvement. The third type of IT service provides visibility into the business operation for BPM, for example, a workflow-monitoring system that receives events about the status of workflow instances and presents a management dashboard for users. On the dashboard users can spot problematic workflow instances in real time and take actions, such as reassigning a work item or terminating a process instance. Some dashboards, for example, Websphere Business Integration (WBI) Monitor,[9] allow users to see the performance of workflow instances that have terminated in a historic view. The real-time and historic views can track how well business processes are meeting their goals so that timely management actions can be taken when necessary.

At design time, for any of the preceding three types of IT services, it is important for a business to be able to assess whether a given hardware configuration can meet the performance objective of the business process, which is often expressed in terms of process throughput, process cycle time, and user-perceived response time. It is equally important to be able to estimate the cost of the IT services that support the execution of a business process. The cost of IT services includes hardware, software licensing, and maintenance. The rest of this section focuses on assessing whether a hardware configuration can meet the performance requirement of the business process, given a set of supporting IT services.

Related work

Performance modeling of IT systems at design time has gained significant attention recently. The UML Profile for Scheduling, Performance and Time[10] is an OMG standard for designing a software system in UML** (Unified Modeling Language**) at design time. It uses activity diagrams, among others, to describe how software components interact in a scenario and the required resources in each activity. Various types of analysis, such as scheduling analysis for real-time systems and performance analysis, can be done by transforming the activity diagrams into analytical or simulation models that can be handled by various analysis techniques or tools. Our work recognizes the difficulty of specifying the resource demand of IT services at design time and proposes a middleware library that encapsulates prior knowledge of the performance characteristics of middleware components.

More detailed modeling of middleware servers for sizing the hardware for a business-process-integration solution has been attempted in Reference 11. Using a combination of layered queuing modeling

and the architecture of a business-process-integration middleware server, potential software bottlenecks, such as thread pools and database connection pools, that can impact system performance (especially on multiprocessor machines) are shown. However, it requires significant work to develop such a detailed model, and it may be impractical for business solutions that require many middleware components. We believe that the middleware model presented in this paper has the right level of detail for answering questions related to the cost and performance of a BPM solution at design time.

A business operations model with IT depth

In this section, we discuss design-time business performance modeling based on a business operations model. Our approach for design-time business performance modeling relies on first creating a knowledge store containing performance-related attributes of IT services that have been developed and benchmarked. The knowledge store is then referenced by a tool that maps the IT services required by a business process to the IT services contained in the knowledge store. To make this mapping easier, a business process can be annotated with the attributes used by the knowledge store, for example, the names of the IT services. After the mapping is done, the result is a business process with IT depth that is ready for performance analysis and that can answer what-if questions.

A business process model with IT depth can be used to relate the performance of a business process to the performance of the supporting IT services. For example, the number of business process instances to be handled per unit time determines the number of invocations of the supporting IT services per unit time, which in turn determines the throughput and response time of the IT services deployed on a given hardware configuration. The cycle time of the business process can then be obtained by summing up the elapsed time of the activities of the business process, which can be obtained from the elapsed time of the activities performed by human resources and the response times of the supporting IT services. Without specifying IT services in the context of a business process model, one cannot easily see how individual IT services impact business performance.

A business process can be modeled using a combination of predefined constructs, such as Tasks, Subprocesses, Fork, Merge, and Join. A detailed description of business process modeling is beyond the scope of this paper, but further details can be found in textbooks and product manuals, for instance Reference 7. The resource demand of an activity of a business operations model specifies the resources required to complete the activity. These can be human resources or IT resources. Resource demand for human resources is usually specified in terms of the type and quantity of roles required and their corresponding durations. Resource demand for IT resources can be specified similarly to demand for human resources at a high level. This high-level view is useful for the analysis of IT cost and cycle time of business processes at an early stage of the solution development cycle. However, if the objective is to estimate the hardware configuration required for solution provisioning, IT resources need to be specified at a finer granularity. This can be achieved by specifying the required IT resource as an IT service that can be realized by a multitier IT infrastructure. The mapping from an IT resource in a business operations model to an IT service can be described as an annotation to the model. A business operations model, thus annotated, can provide a holistic view of the required resources for the execution of a business process. Utilization and response time of IT resources can be analyzed in the context of the given business process. The specification of IT services as part of the annotation of a business process model is discussed in this subsection.

The performance of a business solution depends on the functions of the business solution, the middleware components used to realize the functions, and the hardware used to host the middleware components. An IT service that provides the function required for a business activity can be realized by one or more middleware components and enterprise-information-system (EIS) components, such as those provided by SAP AG and PeopleSoft, Inc. Typical middleware components include WebSphere Application Server, WBI message brokers, WBI InterChange Server (WICS), and DB2*. In addition, middleware components also include components based on J2EE** (Java 2 Platform, Enterprise Edition) that are developed on top of application servers such as WebSphere Application Server. An example of a J2EE-based middleware component is the Common Event Infrastructure (CEI).[12] When middleware and EIS components work in concert to complete a request to a particular

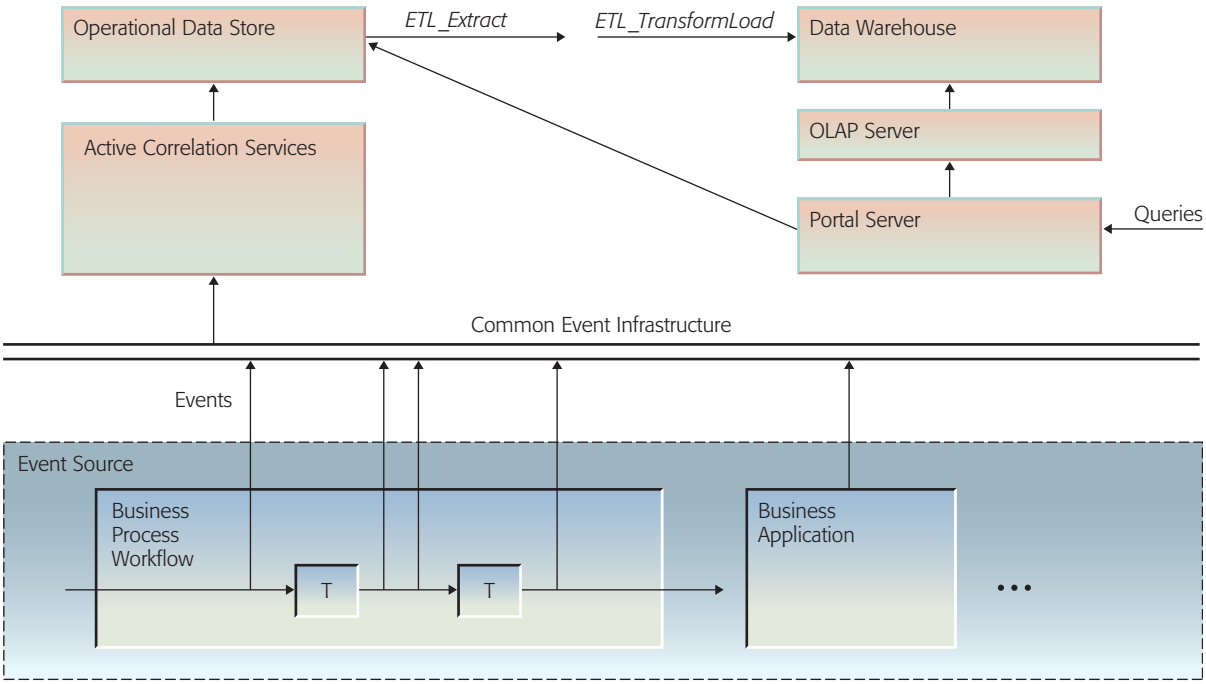

Figure 2
Middleware components of Business Performance Management

IT service, each component incurs certain CPU demand and I/O demand, collectively called hardware resource demand. *Figure 2* illustrates interactions of the middleware components of a general BPM solution, which are further detailed in the next subsection. The hardware resource demand is component-dependent and service-dependent. Quite often, for the same service and the same component, the hardware resource demand is further dependent on the parameters associated with a given IT service request. For example, the CPU demand for processing an event by CEI is found to depend on the size of the event, more specifically the number of extended elements in a Common Base Event (CBE) to be handled by CEI. In order to come up with an estimate of the required hardware configuration (i.e., capacity sizing), one needs to be able to characterize the hardware resource demand of middleware components. This characterization can be in terms of the IT services that are intended to be supported by the middleware components. This is the subject of the next subsection.

Middleware modeling

A middleware component provides a set of services according to the application logic deployed on the middleware. For example, WebSphere Application Server is middleware on which Enterprise JavaBeans** (EJBs**) and servlets can be deployed. The deployed EJBs and servlets determine the kinds of service that an application server can provide. Middleware components often work in concert to provide the IT services required by the activities in a business operations model; for example, DB2 can work with WebSphere Application Server to provide the IT services that the deployed servlets and EJBs intend to provide. To facilitate the specification of the IT services needed by the activities of a business operations model, it is desirable to reference a workload library that contains a list of workloads. A workload in a workload library specifies how middleware components interact to support a given IT service. The hardware resource demand required for each middleware component to support the IT services specified in a workload library is described in a middleware component library. The hardware resource demand of middleware components can be measured by benchmark tests performed in laboratories. The separation of the middleware component library from the workload library has the following advantages.

1. Middleware component libraries can be kept up to date by middleware product owners; whereas

```
<WorkloadLibrary>
   <Components>
            <Component ID="1" Name="CEI" />
            <Component ID="2" Name="ACS"/>
            <Component ID="3" Name="ODS"/>
            <Component ID="4" Name="PortalServer"/>
            <Component ID="5" Name="OLAPServer"/>
            <Component ID="6" Name="DW"/>
   </Components>
   <Workloads>
        <Workload ID="1" Name="EventProcessing">
             <Visit From="0" To="1" AvgNumOfCalls="1" Type="a"/>
             <Visit From="1" To="2" AvgNumOfCalls="1" Type="a"/>
             <Visit From="2" To="3" AvgNumOfCalls="1" Type="s"/>
        </Workload>
        <Workload ID="2" Name="WorkflowDashboardQuery">
             <Visit From="0" To="4" AvgNumOfCalls="1" Type="s"/>
             <Visit From="4" To="3" AvgNumOfCalls="1" Type="s"/>
        </Workload>
        <Workload ID="3" Name="BusinessDashboardQuery">
             <Visit From="0" To="4" AvgNumOfCalls="1" Type="s"/>
             <Visit From="4" To="5" AvgNumOfCalls="1" Type="s"/>
             <Visit From="5" To="6" AvgNumOfCalls="1" Type="s"/>
        </Workload>
        <Workload ID="4" Name="ETL_Extract">
             <Visit From="0" To="3" AvgNumOfCalls="1" Type="s"/>
        </Workload>
        <Workload ID="5" Name="ETL_TransformLoad">
             <Visit From="0" To="6" AvgNumOfCalls="1" Type="s"/>
        </Workload>
  </Workloads>
</WorkloadLibrary>
```

Figure 3
Workload library for Business Performance Management

workload libraries can be created and managed by various solution owners who decide which middleware components are used in a solution.

2. Business-process-solution designers can work with a high-level tool such as WBI Modeler, at the same time, letting the high-level tool reference the appropriate workload and middleware-component libraries that support the domain of the business processes to be designed. An example of a workload and a middleware-component library is discussed next in the context of BPM solutions.

A BPM solution provides visibility into corporate performance at the business-strategy and operations levels. The visibility comes from continuously monitoring business events, updating business performance metrics and key performance indicators (KPIs), identifying the occurrence of important business situations that need attention, and presenting alerts and exceptions to process owners to take timely management action. For the remainder of this section, we consider a generic BPM solution that has the following middleware components for monitoring business events and KPIs:

1. Common Event Infrastructure (CEI)
2. Active Correlation Services (ACS)
3. Operational Data Store (ODS)
4. Portal server
5. Online analytical processing (OLAP) server
6. Data warehouse (DW)

We consider five types of workloads in a BPM context. Figure 2 illustrates interactions among the middleware components in the five workloads, including the ETL (data extraction, transformation, and loading) function. The workload library specifies the middleware components that interact to provide the services required by each workload, as illustrated in ***Figure 3***. A component with an ID of 0 refers to an external component that is outside the scope of performance analysis. `Visit` specifies a connection between two middleware components. The `From` component calls the `To` component. `AvgNumOfCalls` specifies how

often the `From` component calls the `To` component after the `From` component is called in the workload. If `AvgNumOfCalls` is 1, the `From` component always calls the `To` component after the `From` component is called. `Type` specifies whether an invocation is synchronous (s) or asynchronous (a). The invocation type (synchronous or asynchronous) affects the response time of the IT service that the components intend to support. The five workload types are:

1. *EventProcessing*—Various types of transaction information are emitted as events with a common format into the CEI. Examples of event sources are business process workflows and inventory management systems. The details of event sources vary, depending on the specific business solution. The ACS captures events from the CEI and maps attributes of events to metrics and calculates several types of KPIs. The KPIs are stored in the ODS. Invocation of the ODS is synchronous, but others are asynchronous.
2. *WorkflowDashboardQuery*—This workload is for real-time observation of workflow progress and performance. Each query is submitted from the workflow dashboard to the portal server, and then the portal server queries the ODS. All the invocations are synchronous. A workflow dashboard is outside the scope of our performance analysis and is abstracted as a component with an ID of 0.
3. *BusinessDashboardQuery*—This workload is submitted from the business dashboard to the portal server. The portal server calls the OLAP server, and then the OLAP server executes an analysis that queries the DW. All invocations are synchronous. A business dashboard is outside the scope of our performance analysis and is abstracted as a component with an ID of 0.
4. *ETL_Extract*—This workload extracts data from the ODS into an intermediate format such as XML for further processing. The data being extracted are those related to workflow instances that have been completed.
5. *ETL_TransformLoad*—This workload converts the operational data from an intermediate format into a format that can be loaded into the DW and moves the data into the DW.

Figure 4 illustrates a snippet of the middleware component library, which documents performance characteristics of the middleware components for different workloads. The hardware resource demand is given as a function of several solution-level parameters. For example, the hardware resource demand of CEI in serving the event-processing workload is given as a function of the event size in bytes. In general, this kind of hardware-resource-demand function has to be obtained by benchmark tests under a set of varying solution-level parameters. Without sufficient benchmark testing, point measurements can be logged in a middleware component library as potential starting points for extrapolating the hardware resource demand. In Figure 4, the service demand of each component depends on the following solution-specific parameters that are fed by the `AdditionalSolutionInfo.xml` file, as illustrated in ***Figure 5***.

- *EventSize*—Average size of events (in bytes)
- *NumOfMaps*—Number of metrics to be mapped from attributes of events
- *NumOfKPIs*—Number of KPIs to be calculated
- *NumOfSituations*—Number of situations to be detected
- *ComplexityFactorPerRecord*—Relative complexity of a record in a database

The `AdditionalSolutionInfo.xml` file also specifies generation rates (per second) of respective workloads.

In this paper, we approximate service demands as linear functions of a set of application-specific parameters for illustration purposes. For example, the component ODS is called in three workloads: *EventProcessing*, *WorkflowDashboardQuery*, and *ETL_Extract*. Each service demand is calculated by a service demand function specified in the middleware component library. The unit of measurement for service demands is seconds (sec). For example:

Service Demand (sec) of the ODS in *EventProcessing* workload $= 0.006 \cdot (NumOfMaps) + 0.006 \cdot (NumOfKPIs) + 0.0055 \cdot (NumOfSituations) + 0.0075$.

Service Demand (sec) of the ODS in *WorkflowDashboardQuery* workload $= 0.006$.

Service Demand (sec) of the ODS in *ETL_Extract* workload $= 0.001 \cdot (ComplexityFactorPerRecord) + 0.001$.

In general, service demand functions are middleware-component-specific and application-specific.

```
<ComponentLibrary>
    <Component ID="1" Name="CEI">
        <Workload Name="EventProcessing">
            <ServiceDemandModel ModelName="LinearModel">
                <ModelPara Name="EventSize" Coefficient="6.25e-7"/>
                <ModelPara Name="Constant" Coefficient="0.00444"/>
            </ServiceDemandModel>
        </Workload>
    </Component>
    <Component ID="2" Name="ACS">
        …
    </Component>
   <Component ID="3" Name="ODS">
        <Workload Name="EventProcessing">
            <ServiceDemandModel ModelName="LinearModel">
                <ModelPara Name="NumOfMaps" Coefficient="0.006"/>
                <ModelPara Name="NumOfKPIs" Coefficient="0.006"/>
                <ModelPara Name="NumOfSituations" Coefficient="0.0055"/>
                <ModelPara Name="Constant" Coefficient="0.0075"/>
            </ServiceDemandModel>
        </Workload>
        <Workload Name="WorkflowDashboardQuery">
            <ServiceDemandModel ModelName="LinearModel">
                <ModelPara Name="Constant" Coefficient="0.006"/>
            </ServiceDemandModel>
        </Workload>
        <Workload Name="ETL_Extract">
            <ServiceDemandModel ModelName="LinearModel">
                <ModelPara Name="ComplexityFactorPerRecord" Coefficient="0.001"/>
                <ModelPara Name="Constant" Coefficient="0.001"/>
            </ServiceDemandModel>
        </Workload>
    </Component>
    <Component ID="4" Name="PortalServer">
        …
    </Component>
    <Component ID="5" Name="OLAPServer">
…
      </Component>
    <Component ID="6" Name="DW">
        …
    </Component>
</ComponentLibrary>
```

Figure 4
Middleware component library

They should be determined empirically. Modeling tools should have built-in support for more complex service demand functions, for example, nonlinear or piecewise linear models.

Computational examples

Using WBI Modeler, we have developed a prototype called the BPM Performance Advisor for the middleware modeling methodology and applied it to the generic BPM solution considered in this section. In this prototype, we annotate the business operations model of WBI Modeler indirectly by creating an XML file (`AdditionalSolutionInfo.xml`) that describes the characteristics of the business process being monitored and the workload intensity, for example, the event-generation and query-generation rates and the ETL-initiation rate. In general, the event-generation rate depends on the number of workflow activities per workflow instance and the number of business measures to be evaluated that are defined in the business operations model. The performance analysis is performed by using fluid models that have been described elsewhere in detail, including the methodology and assumptions.[8] Alternatively, the discrete event simulation engine in WBI Modeler can also be used for this purpose.

```
<AdditionalSolutionInfo>
        <SourceEvent SourceName="BusinessProcessWorkflow" GenerationRate="10.0" EventSize="1500"
                                        NumOfMaps="2" NumOfKPIs="2" NumOfSituations="2"/>
        <WorkflowDashboardQuery SourceName="WorkflowDashboard" GenerationRate="5.7"/>
        <BusinessDashboardQuery SourceName="BusinessDashboard" GenerationRate="10.4"/>
        <ETL GenerationRate="1.157e-5" ComplexityFactorPerRecord="1.5"/>
</AdditionalSolutionInfo>
```

Figure 5
Additional Solution Information

We consider the following case study to illustrate the utility of the BPM Performance Advisor. An enterprise, ABC Corp., is interested in deploying a BPM solution to monitor and manage their business performance on an ongoing basis. Because the customer base of ABC Corp. is growing at a significant rate, they want to ensure that their BPM solution is deployed on an appropriately sized infrastructure. In this context, the BPM Performance Advisor can be used to answer two kinds of questions: 1) Given a workload and a deployment topology, where is the resource bottleneck? 2) Given a workload, what deployment topology can satisfy the performance objective? When analyzing alternative topologies, BPM Performance Advisor allows two possible options, either change the machine type or change the cluster by increasing the number of nodes in the cluster.

In using the BPM Performance Advisor to address this problem, the business processes to be monitored in the BPM solution are first modeled using WBI Modeler. The attributes of the BPM solution, such as the sources of the business events and the number of KPIs monitored, are then specified in the BPM Performance Advisor. Business demands in future years are modeled as different scenarios to examine the performance of different deployment topologies and to identify the topology that can satisfy the performance objective for the BPM solution (e.g., less than 10 seconds of average query response time under given event/query generation rates).

In this prototype, a list of machine types and their performance ratings (Rperf[13]) are stored in a hardware library illustrated in ***Figure 6***. The prototype assumes that the service demand associated with a physical solution component in handling a request can be "scaled" to estimate the new service demand when machine type or number of nodes is changed. In this paper, for simplicity, the performance of the machine is proportional to Rperf and the number of nodes.

Figure 7 illustrates a deployment topology, that is, the machine on which each middleware component is deployed. Attributes of each machine are machine type (BrandModel), number of nodes, and utilization target. Note that default utilization target is 1.0 (100 percent). For this case study, ***Table 1*** shows the resource utilization statistics for each server. This shows that all the servers will meet the utilization targets for the average traffic specified in

```
<ServerLibrary>
    <Server BrandModel="IBM p640 P3II_375 1-way" Rperf="1"/>
    <Server BrandModel="IBM p640 P3II_375 3-way" Rperf="2.55"/>
    <Server BrandModel="IBM p640 P3II_375 4-way" Rperf="3.47"/>
    <Server BrandModel="IBM p650 P4+_1450 2-way" Rperf="4.47"/>
    <Server BrandModel="IBM p650 P4+_1200 4-way" Rperf="8.05"/>
    <Server BrandModel="IBM p670 P4_1100 8-way" Rperf="24.18"/>
    <Server BrandModel="IBM p670 P4_1100 16-way" Rperf="34.66"/>
    <Server BrandModel="IBM p690 P4+_1900 32-way" Rperf="104.17"/>
</ServerLibrary>
```

Figure 6
Server library

```
<DeploymentTopology>
  <Machines>
    <Machine ID="1" BrandModel="IBM p640 P3II_375 1-way" NumberOfNodes="1"/>
    <Machine ID="2" BrandModel="IBM p640 P3II_375 1-way" NumberOfNodes="1" UtilizationTarget="0.9"/>
    <Machine ID="3" BrandModel="IBM p640 P3II_375 1-way" NumberOfNodes="1" UtilizationTarget="0.8"/>
    <Machine ID="4" BrandModel="IBM p640 P3II_375 1-way" NumberOfNodes="1" UtilizationTarget="0.7"/>
    <Machine ID="5" BrandModel="IBM p640 P3II_375 1-way" NumberOfNodes="1" UtilizationTarget="0.7"/>
    <Machine ID="6" BrandModel="IBM p640 P3II_375 1-way" NumberOfNodes="1" UtilizationTarget="0.8"/
  </Machines>
  <Deploy ComponentName="CEI" MachineID="1"/>
  <Deploy ComponentName="ACS" MachineID="2"/>
  <Deploy ComponentName="ODS" MachineID="3"/>
  <Deploy ComponentName="PortalServer" MachineID="4"/>
  <Deploy ComponentName="OLAPServer" MachineID="5"/>
  <Deploy ComponentName="DW" MachineID="6"/>
</DeploymentTopology>
```

Figure 7
Initial deployment topology

the `AdditionalSolutionInfo.xml` file (see Figure 5). ***Table 2*** shows response-time statistics for each workload. Note that if there are one or more components whose server utilization is over 1.0, the estimated response time is infinity, and the analysis result suggests that the system is not stable and the deployment topology needs to be changed.

Now, we can analyze the middleware performance under different what-if scenarios. For instance, we want to analyze the performance in a future time period when the arrival rate during peak traffic is four times that during average traffic. More specifically, generation rates of *SourceEvent*, *WorkflowDashboardQuery*, *BusinessDashboardQuery*, and *ETL_Extract* are 40.0/sec, 22.8/sec, 41.6/sec, and 4.628×10^{-5}/sec, respectively. Analysis results in the peak traffic scenario are included in Tables 1 and 2. Table 1 shows that machines 2, 3, 4, 5 and 6 will not meet target utilizations under peak traffic. In addition, since utilization of machines 2, 3, 4, 5 is over 100 percent, response times of *EventProcessing*, *WorkflowDashboardQuery*, *BusinessDashboardQuery*, and *ETL_Extract* are infinity. For this type of situation, when several machines are overutilized, the BPM Performance Advisor provides two types of recommendation to change the deployment topology:

1. Change server types. The BPM Performance Advisor recommends a machine type from its server library in consideration of the utilization target, as illustrated in ***Table 3***. The appropriate machine type can be identified by using the approximation that resource utilization is inversely proportional to the performance ratio of the machine type. Note that the number of nodes is fixed in this usage mode.
2. Change number of nodes. The BPM Performance Advisor suggests the minimum number of nodes required to meet utilization targets, as illustrated in ***Table 4***. The minimum number of nodes can also be identified by using the approximation that resource utilization is inversely proportional to the number of nodes. Note that the machine type is fixed in this usage mode.

Table 1 Resource utilization for initial deployment topology under average and peak traffic

Resource Utilization (%)	Server 1 (IBM p640 P3II_375 1-way)	Server 2 (IBM p640 P3II_375 1-way)	Server 3 (IBM p640 P3II_375 1-way)	Server 4 (IBM p640 P3II_375 1-way)	Server 5 (IBM p640 P3II_375 1-way)	Server 6 (IBM p640 P3II_375 1-way)
Under Average Traffic	5.4	85.0	45.9	43.9	62.4	20.8
Under Peak Traffic	21.5	340.0	183.7	175.5	249.6	83.2

Table 2 Response Times (sec) of initial and new deployment topology under average and peak traffic

Workload Name	Response Time of Initial Topology under Average Traffic	Response Time of Initial Topology under Peak Traffic	Response Time of New Deployment Topology (changing server type) under Peak Traffic
EventProcessing	0.6509	Inf.	0.1459
WorkflowDashboardQuery	0.0182	Inf.	0.0134
BusinessDashboardQuery	0.2561	Inf.	0.0924
ETL_Extract	0.0046	Inf.	0.0035
ETL_TransformLoad	0.0069	0.0327	0.0032

Based on the first recommendation, we analyzed the new deployment topology shown in Table 3. The performance results of the new deployment topology (shown in the last column of Tables 2 and 3) showed utilization targets could be met even in peak traffic.

SIMULATION OF PSMS

MDA provides an approach to generate a PIM of a business process from the design-time business operations model and then transform the PIM into a PSM. By doing so, it facilitates longer-term flexibility in implementation integration and ease of maintenance. In the transformation from PIM to PSM, additional information on a specific platform is provided by an automated tool or software designers or both. There are many research publications[14–18] on this, and several commercial tools are also available.

Business Process Execution Language (BPEL)[19] is an industry-standard business process definition language, which specifies interconnection of Web Services. Several commercial tools support transformation from a business operations model into a BPEL model (PIM) that can be further deployed on a particular platform (PSM). To make the BPEL model executable, middleware-specific code needs to be added. For example, a business process model can be developed in WBI Modeler that can be converted into its BPEL representation. This representation can be imported into the WBI Server Foundation where platform-specific code can be developed that can enable the deployment of the business process.

SOA is a mechanism that enables the reconstruction of a business solution if business requirements change by changing only the interconnection of services. However, when business solutions adapt to constantly changing business requirements, it is

Table 3 Recommendation to change machine type under peak traffic

	Target Utilization (%)	Recommended Machine Type	New Resource Utilization (%)
Machine 1	100	IBM p640 P3II_375 1-way	21.5
Machine 2	90	IBM p650 P4+_1450 2-way	76.1
Machine 3	80	IBM p640 P3II_375 3-way	72.0
Machine 4	70	IBM p640 P3II_375 3-way	68.8
Machine 5	70	IBM p650 P4+_1450 2-way	55.8
Machine 6	80	IBM p640 P3II_375 3-way	32.6

Table 4 Recommendation to change number of nodes under peak traffic

	Target Utilization (%)	Recommended Node #	New Resource Utilization (%)
Machine 1	100	1	21.5
Machine 2	90	4	85.0
Machine 3	80	3	61.2
Machine 4	70	3	58.5
Machine 5	70	4	62.4
Machine 6	80	2	41.6

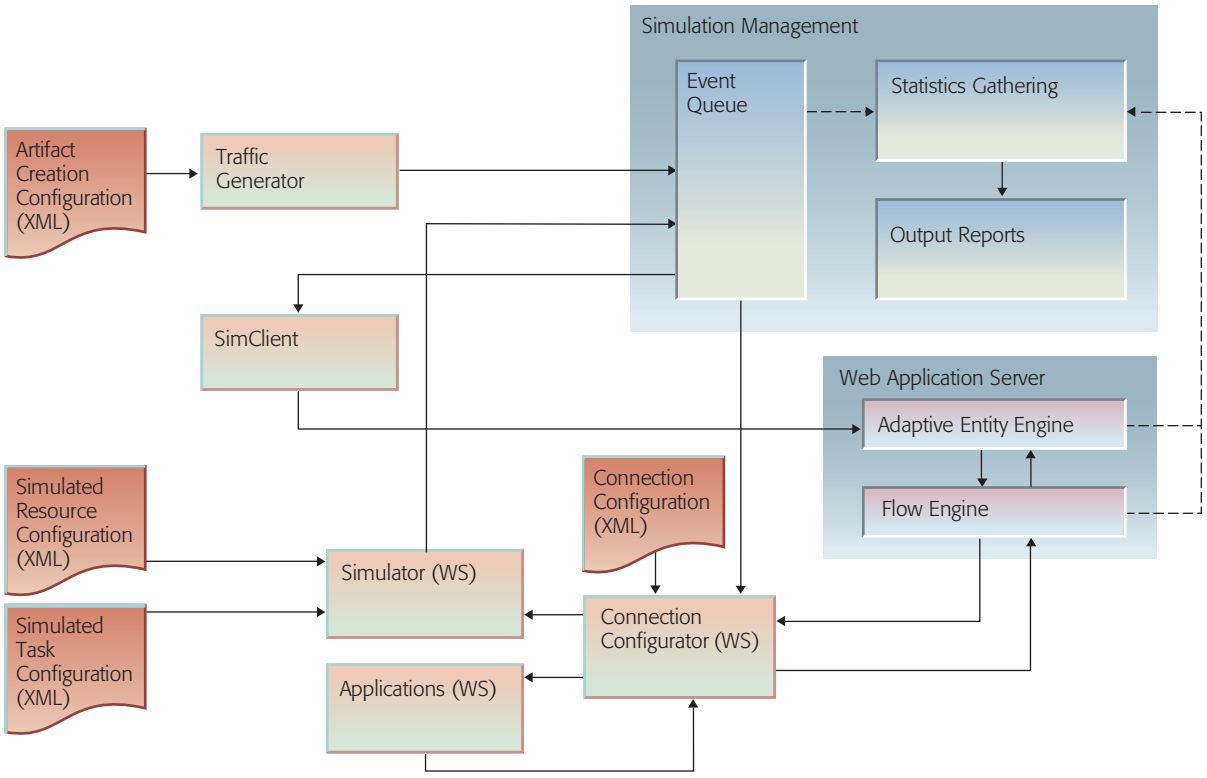

Figure 8
A framework for PSM simulation

often the case that several services exist but others need to be newly developed. Early analysis and testing of business solutions in the development phase could significantly contribute to the underlying SOA value proposition by reducing the overall time to deployment for the modified business solution. Our PSM simulation framework supports this by allowing users to simulate a PSM consisting of real and simulated services for function and performance testing. By including services available in the analysis, deeper insights into a business solution can be obtained, so that defects are detected as early as possible and do not propagate further in the solution development process.

Simulation framework

Figure 8 illustrates a framework for PSM simulation of business solutions. The Traffic Generator simulates client orders according to instructions from an artifact-creation configuration file. Each client order generated by the Traffic Generator is assigned a simulated submission date and other parameters, such as the client's name, the specific item ordered, the quantity of items ordered, and delivery instructions. These parameters are randomly assigned to each order. The details of these attributes vary depending on business solutions. When the simulation is started, client orders are generated and submitted to the Simulation Management component, where the orders are sorted based on their simulated submission dates and stored in the Event Queue.

The Event Queue keeps track of business processes in the implementation model by controlling the timing and invocation of events. There are two types of events associated with the implementation model simulation: (1) events associated with the arrival of client orders and (2) response events from the Simulator. The client orders, the first type of event, are generated by the Traffic Generator and stored in the Event Queue. The Event Queue sends stored client orders to the Web application server at scheduled time stamps, based on the submission date of the client orders. The Event Queue instructs a simulated client, (e.g., SimClient), to send the

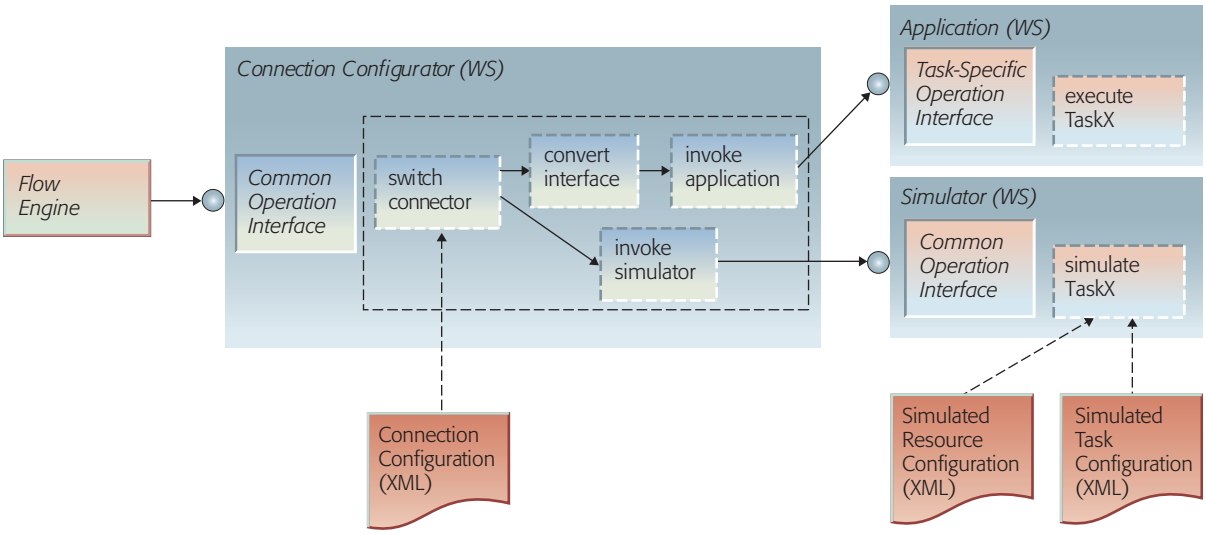

Figure 9
Switching between the Simulator and real solution components

client orders to an Adaptive Entity Engine[20] of the Web application server.

The Adaptive Entity Engine handles adaptive entities, which are state machines with state-transition logic. The state-transition logic is externally editable, and the Adaptive Entity Engine makes it possible to easily combine multiple processes and manage them using state-transition logic. An adaptive entity is generated for each client order and manages process instances for the client order. In addition, the Adaptive Entity Engine provides a function for scheduling a time-out event that is automatically invoked if no transition event occurs before a specified time elapses. This function can be used to detect potential functional defects in an application, as will be explained later in this section. In general, Flow Engines provide a time-out event and can also combine multiple data flows. However, the Adaptive Entity Engine is an optional feature of the framework that can easily change the logic of combinations of multiple processes without redeployment of process models to the Flow Engine. If the Adaptive Entity Engine is not used, the SimClient needs to be configured to directly invoke the Flow Engine.

A Flow Engine is novel technology that allows us to flexibly invoke a multitude of software assets and human tasks, according to business process models. Several Flow Engines supporting BPEL models are commercially available (see Reference 21 for an example). In our framework of runtime simulation, the Flow Engine invokes a Connection Configurator that can be switched between real and simulated solution components according to instructions in a connection configuration file. The Flow Engine does not need to know if the component that it has invoked is a real solution component or the Simulator. Thus, if several new solution components become ready, the user can easily switch from the Simulator to real solution components by editing the connection configuration file. The Flow Engine, therefore, does not need redeployment of the process models. After a process model is deployed, it can be used during all stages of the solution development phase. All changes are made in the connection configuration file.

Figure 9 illustrates the switching between the Simulator and real solution components by the Connection Configurator WS. The connection configuration file determines whether a task corresponds to a real application or to the Simulator, and provides information, including the types and names of the parameters of the input business object and the output business objects for each task.

If an invoked task is an actual application, the "switch connector" of the Connection Configurator invokes the real application by converting the interface to a task-specific interface. Then, the real response business object from the application is

converted to the common operation interface and sent back to the Flow Engine by the Connection Configurator. On the other hand, if an invoked task is connected to the Simulator WS, then the switch connector of the Connection Configurator simply invokes the Simulator and forwards the input business object to the Simulator. The Connection Configurator can then simply forward the input business object to the Simulator because the Simulator shares the interface with the Connection Configurator. The Simulator consumes simulated resources, incurs time delays and generates response business objects. Then the Simulator sends a response event to the Event Queue, consisting of the response business objects accompanied with a time stamp accounting for the simulated submission and completion dates of the task. Thus, the Event Queue sends the response business object back to the Flow Engine through the Connection Configurator at the scheduled time stamp. Note that the response event is the second type of event controlled by the Event Queue.

The delay time used by the Simulator refers to the time elapsed from when the solution component is invoked to when the solution component completes the task. More specifically, the delay time consists of the action cycle time required for execution of a task and the waiting time due to resource availability. Based on the delay time, the task completion time is calculated. In this case, the task submission and completion times that the Simulator sends to the Event Queue are time stamps of the simulation clock. Unlike traditional simulation, the implementation model simulation involves both real components executed with a real clock and simulated components executed with the simulation clock. When the Simulator is invoked, the Simulator gets a current time stamp from the real clock and then converts it to a time stamp for the simulated task submission using the simulation clock. Then, the delay time and the task completion time stamp are calculated. The Simulator also logs resource utilization and queues tasks, based on the simulation clock time. The Event Queue manages synchronization and conversion between real and simulation clocks. Further details on the synchronization and conversion of these clocks are discussed in the section "Synchronization of real and simulation clocks."

In the simulation framework, the single Simulator WS emulates a plurality of types of services according to instructions from the Simulator configuration files that store the behaviors of resources and tasks. First, the Simulator gets the name of the task to be emulated as an argument. Then, the Simulator consumes simulated resource, incurs action cycle time to execute the task, and generates the response business object, based on the task behavior specified in the task configuration file. In addition, the Simulator incurs waiting time due to resource availability and resource cost, according to the resource configuration file.

Modeling resource behavior during simulation is mainly relevant for identifying the implications for resource utilization, such as any resource bottlenecks and the resulting resource costs. This information is used to infer the cost/performance trade-offs for business integration solutions. The resource parameters include the name of the resource, the capacity of the resource, the cost of the resource, the resource-scheduling policy and resource availability. The resource cost is determined based on resource cost per use or resource cost per unit time. The resource-scheduling policy is FIFO (first in first out) or a priority queue. Resource availability is based on an availability pattern, such as holidays, weekends, and scheduled maintenance. The repetition frequency specifies the frequency at which the pattern repeats, such as weekly, daily, working days, and so forth. The start time and the end time determine the duration of the repetition.

Modeling task behavior during simulation is relevant for identifying the implications for overall cycle time, queuing behavior, resulting delays, and so forth. This is also used to infer the cost/performance trade-offs for business integration solutions. The task parameters include the task name, action-cycle-time distribution, resource requirements, input business objects, and output-business-object generation. The action-cycle-time distribution describes the distribution of the time elapsed during the execution of a task. The resource requirements describe what resources are required to complete the task and the quantity of each resource consumed. This description also includes the types and names of parameters of the input business object and the probability distributions of the parameters of the output business objects. Note that the Simulator functions independent of any specific business solution components. Therefore, new components to be simulated can be added easily to the implementation model simulation framework.

After a simulation of an implementation model is complete, the Statistics Gathering function gathers the simulation results from the Event Queue, the Adaptive Entity Engine, the Flow Engine, and the Simulator. The gathered statistics are somewhat similar to those available from traditional business process simulation (see, for example, WBI Workbench Version 4.2.4).[22] More specifically, the gathered statistics include tables describing client-order statistics (such as arrival times, completion times, cycle times, processing costs, and waiting times), resource statistics (such as utilization and total costs), and queue statistics for each task (such as average queue size, average queue waiting time, and maximum queue size). This can be useful for functional and performance testing of the business integration solution. The gathered statistics are provided to the user in output reports. The Statistics Gathering function also identifies whether an invoked solution component has been completed. This can be of value from the perspective of functional testing of business solutions. Our framework can be used for functional testing if the interaction between the workflow process and applications is stateless. Further work is necessary to model state transitions in interactions with end applications to enable this framework to be used more broadly. In order to use this framework more broadly, the Simulator WS needs to obtain the state as an argument and have some internal logic to simulate the behavior of the application, which would require application-specific simulators and is not within the scope of this paper.

Synchronization of real and simulation clocks

Unlike traditional simulation, the implementation model simulation method involves both real components executed in real time (e.g., the Flow Engine, the Adaptive Entity Engine, real solution components, and network) and simulated components, executed in virtual time (e.g., client orders and simulated solution components). Therefore, simulated components should be synchronized to real-time simulations involving real components. In a related work, Lendermann et al. also deal with synchronization issues between the real clock and simulation clock in the context of integrating discrete-event simulation models with framework-based business applications.[23] In our framework, the Event Queue handles the synchronization and consists of four subcomponents, the Priority Queue, the Event Invoker, the Response Event Handler, and the Flow Engine Tracker. These components interact with each other as illustrated in *Figure 10*.

In the Priority Queue, client order events and response events are sorted by their scheduled dates. The scheduled dates can be submission dates of client orders or task-completion dates. Client-order events are generated and stored in the Priority Queue by the Traffic Generator at simulation startup. In contrast, response events are sent from the Simulator at simulation runtime and submitted to the Priority Queue through the Response Event Handler. The scheduled dates are time stamps of the simulation clock. The events in the Priority Queue are invoked by the Event Invoker. During simulation runtime, the Event Invoker reads the first event in the Priority Queue and estimates the event invocation time by converting its scheduled time from the simulation clock into the equivalent time on the real clock. Then, at the scheduled time stamp of the real clock, the Event Invoker invokes the event; that is, the Event Invoker lets the SimClient send client orders to a Web server or sends a response message to the Flow Engine through the Connection Configurator. The Response Event Handler receives response events with time stamps accounting for the task submission and completion dates (according to the simulation clock), and the response business objects. Then, the Response Event Handler compares the task-completion date with the scheduled date of the first event in the Priority Queue. If the task-completion date of the response event is after the first event, then the Response Event Handler simply inserts the event into the Priority Queue. Otherwise, the Response Event Handler inserts the event and notifies the Event Invoker that the first event in the Priority Queue is updated. Then, the Event Invoker rereads the first event and prepares the invocation for it.

How is the simulation clock synchronized with the real clock? The naïve way would be synchronization by offsetting the time difference between the time stamp of the simulated submission date of the first client order and the real time stamp at simulation startup. However, this naïve synchronization, or a complete real-time execution, could take a prohibitively long time and not be of practical utility if most solution components are simulated in an early stage of the solution development phase or if a process to be simulated contains human tasks.

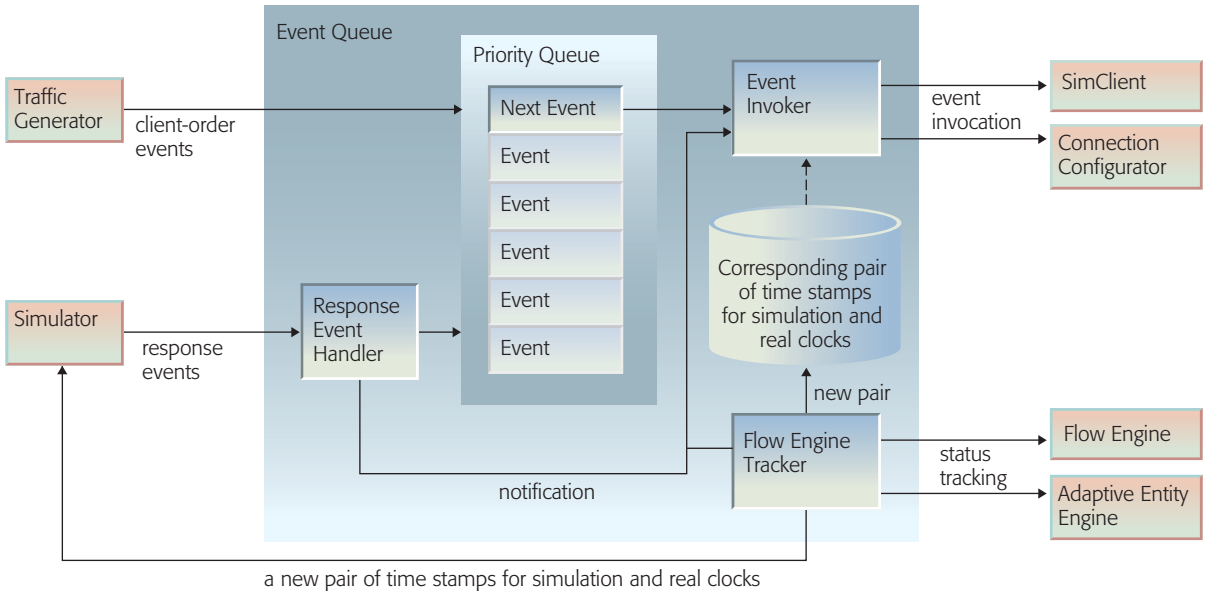

Figure 10
Subcomponents of the Event Queue

Generally, human tasks need to be simulated through the solution development phase and require much longer action cycle time than IT tasks. Therefore, we need methods of effectively synchronizing the simulation clock and real clock and shortening testing time. In this context, this paper proposes a method for time compression.

The proposed method periodically checks the Flow Engine and the Adaptive Entity Engine to see if any real solution components are active. When no real components are busy, intermediate durations are compressed (see ***Figure 11***). This is accomplished by a component called the Flow Engine Tracker, which uses the APIs of the Flow Engine and the Adaptive Entity Engine that provide information on the status of each process instance and adaptive entity in execution. More specifically, there are two types of compression. The first type is compression of intervals of client orders. By polling the Flow Engine and the Adaptive Entity Engine, it is possible to identify whether there are any client orders in execution or not. At any time, if there are no client orders in execution, intervals between client orders can be compressed (see duration B in Figure 11). The second type is compression of delay time in the Simulator. Regarding all client orders in execution, if all process instances for them are waiting for response events from the Simulator, the duration until the next event is invoked can be compressed. This type of duration can be detected by using the APIs of the Flow Engine and the Adaptive Entity Engine and checking the contents of the Priority Queue (see duration A in Figure 11).

When the Flow Engine Tracker detects a compressible duration, it logs the pair of corresponding time stamps of real and simulation clocks (see ***Figure 12***) and notifies the Event Invoker and the Simulator that the conversion function between real and simulation clocks is updated. Then, the Event Invoker re-converts the simulated scheduled date of the first event into its real scheduled date and prepares its invocation. Note that, by use of the pairs of corresponding time stamps of real and simulation clocks, any time stamp of the real clock can be converted into the corresponding time stamp of the simulation clock, and vice versa, as follows:

Let $(S_0, R_0), (S_1, R_1), \ldots, (S_n, R_n), (S_{n+1}, R_{n+1}) \ldots$ be the pairs of corresponding time stamps of real and simulation clocks, sorted by their time sequences, and let R_x be a time stamp of the real clock. By use of the corresponding time-stamp logs, R_x can be converted into the corresponding time stamp of the simulation clock, S_x. First, a real time stamp R_{n-1} (in the log) adjacent to the R_x time stamp is identified ($R_{n-1} < R_x < R_n$). Then, S_x can be calculated by

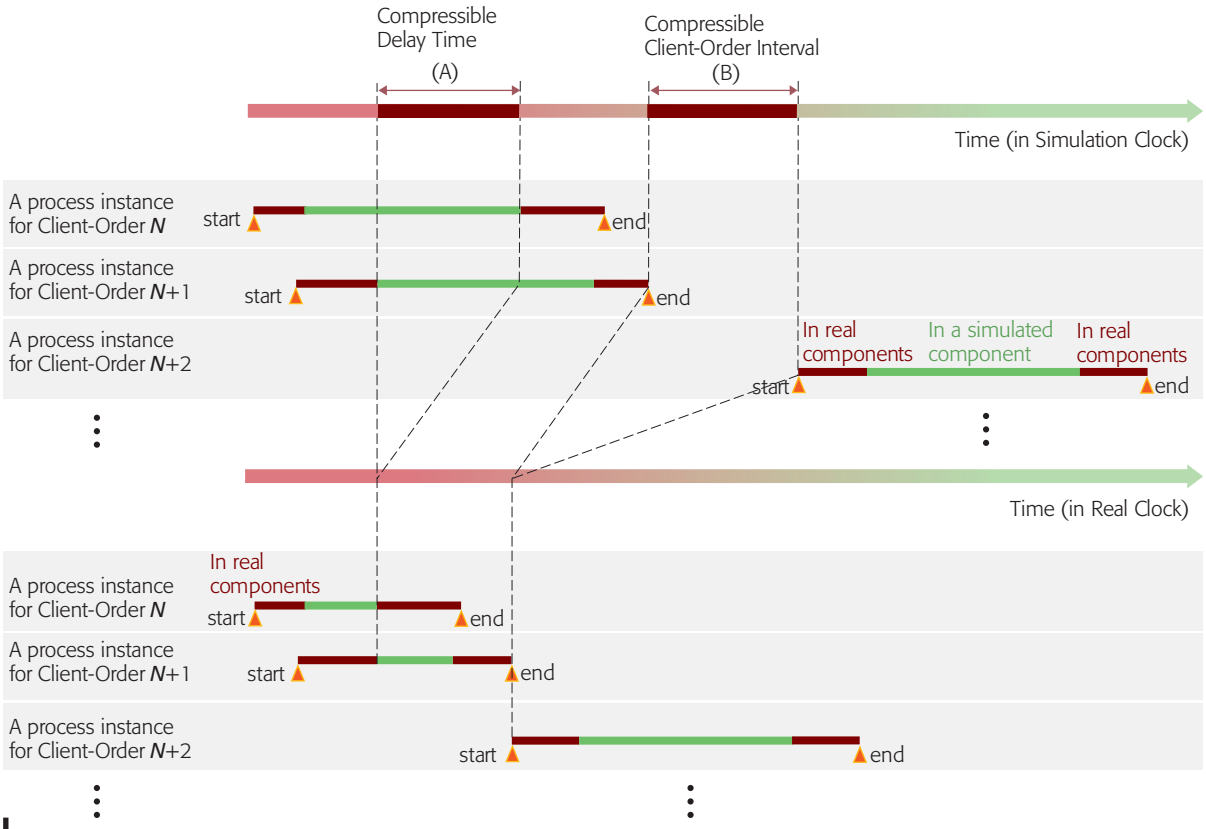

Figure 11
Compressible delay time and client-order interval

equation (1).

$$S_x = S_{n-1} + (R_x - R_{n-1}). \quad (1)$$

Reverse conversion is conducted in similar manner.

$$R_x = R_{n-1} + (S_x - S_{n-1}). \quad (2)$$

Note that equations (1) and (2) are only applicable outside the compressed times in the simulation clock. During simulation runtime, the latest pair of time stamps of real and simulation clocks allows the Simulator and the Event Invoker to conduct conversion between time stamps of the real and simulation clocks.

When a simulation is complete, statistics logged in real clock times need to be converted into those in simulation clock times. Regarding process cycle times, because the start time and completion time of the real clock for each process can be converted respectively into those of the simulation clock by use of the corresponding logs of time-stamp pairs, the simulated process cycle time can be obtained. Resource utilization and queue statistics for simulated solution components are logged in simulation clock times by the Simulator. Resource utilization and queue statistics for real solution components are logged by other profiling tools in real clock times, and these statistics in real clock times need to be converted. Because the compression operation is conducted only when real components are in the idle state, the resource utilization in real clock time, U_{real}, can be easily converted into that in simulation clock time, U_{sim}, as follows:

$$U_{sim} = U_{real} \times \frac{T_{real}}{T_{sim}}, \quad (3)$$

where T_{sim} is the testing time using the simulation clock, and T_{real} is the testing time using the real clock. Therefore, all the statistics logged in real clock time can be converted into those in simulation clock time.

Computational example

We have developed a prototype for simulation of implementation models using a BPEL Flow Engine

in the Process Choreographer of WBI Server Foundation. An executable BPEL flow was designed by using WebSphere Studio Application Developer Integration Edition Version 5.1. The prototype system was deployed on a Windows XP** computer with a 3.06 GHz Xeon** CPU and 1.5 GB RAM. We illustrate the PSM simulation capability now with an example.

Figure 13 illustrates a sample process for Web-based shopping for a personal computer (PC). When a customer visits the Web site, the customer profile is loaded, and then the customer browses the PC catalogs. The customer then checks different configuration options and checks their supply availability. After the customer identifies a satisfactory configuration that is available, the order is submitted. This Web shopping process consists of three subprocesses; an access-catalog process, a check-supply process, and an order-submission process. In the prototype system, each subprocess is executed in the Flow Engine, and an adaptive entity in the Adaptive Entity Engine manages the invocations of the three subprocesses. Because this is only an illustrative example, clients' behaviors such as thinking time and repetition of browsing catalogs were not considered, and only IT tasks were analyzed; that is, when an adaptive entity receives a completion event of a subprocess (e.g., an access-catalog process), the adaptive entity promptly invokes the next subprocess (e.g., a check-supply process).

Figure 14 shows an example of simulated client orders that was generated by the Traffic Generator according to an artifact-generation configuration file. The simulated client orders are stored in the Event Queue until the Event Queue instructs the SimClient to send a simulated client order to the Flow Engine at each scheduled submission time for a client order. With each submission of a client order, an adaptive entity and a process instance are generated. Then, the process instance invokes specific process tasks.

Table 5 shows sample simulation data sets for experiments with the implementation model simulation framework. We assumed that all the tasks are deployed on a WCBE (WebSphere Commerce Business Edition) server with two CPUs and that they consume the CPUs of the server. The cycle times of the tasks are assumed to be normally distributed with parameters listed in Table 5.

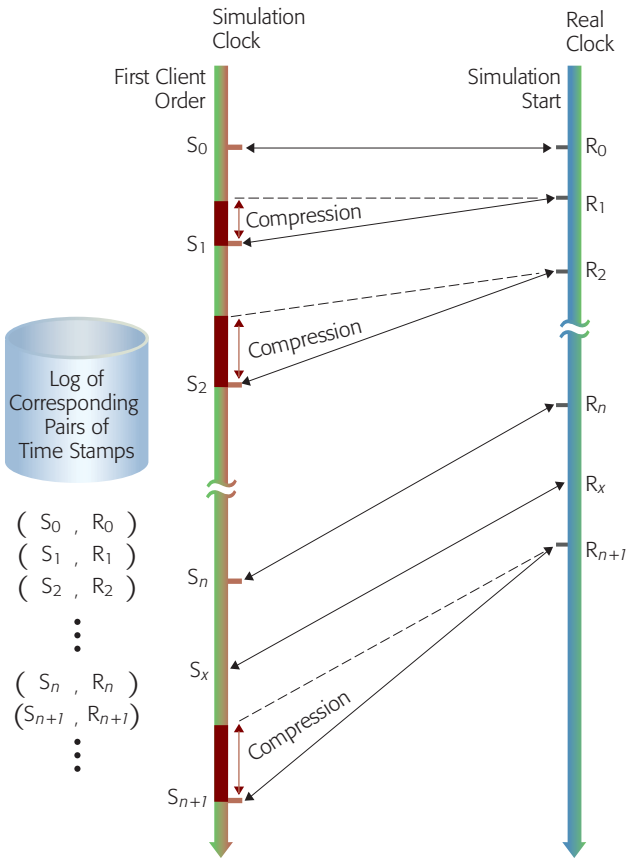

Figure 12
Corresponding pairs of time stamps for simulation and real clocks

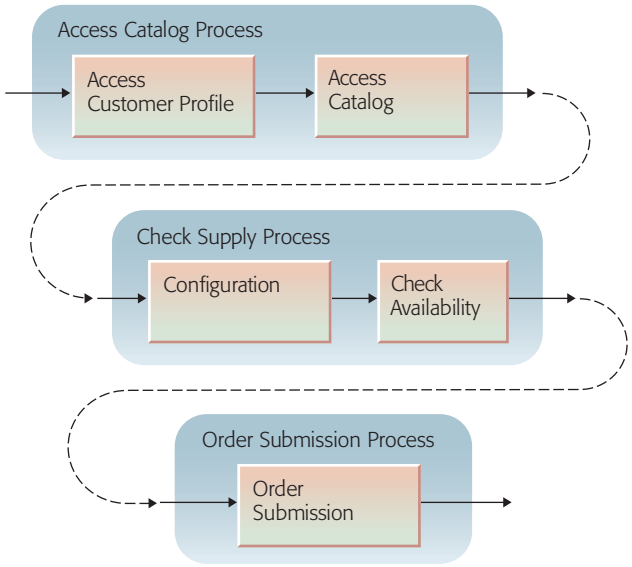

Figure 13
A sample process for Web-based PC shopping that illustrates PSM simulation

```
Time=2003-10-22 13:00:16,type=EMEA,CustClass=Others,Item#=4
Time=2003-10-22 13:00:20,type=AP,CustClass=Tier1,Item#=4
Time=2003-10-22 13:00:28,type=AMERICAS,CustClass=Tier1,Item#=1
Time=2003-10-22 13:00:48,type=EMEA,CustClass=Tier1,Item#=1
Time=2003-10-22 13:00:52,type=AP,CustClass=Others,Item#=2
Time=2003-10-22 13:00:55,type=EMEA,CustClass=Tier1,Item#=2
Time=2003-10-22 13:01:01,type=AMERICAS,CustClass=Others,Item#=3
Time=2003-10-22 13:01:18,type=AMERICAS,CustClass=Tier1,Item#=4
Time=2003-10-22 13:01:24,type=AMERICAS,CustClass=Tier1,Item#=2
Time=2003-10-22 13:01:28,type=AP,CustClass=Others,Item#=3
```

Figure 14
Sample client orders

The simulation framework allows the user to execute two types of testing: functional testing and performance testing. Functional testing is used for identifying the locations of application failures if they happen. Performance testing is used to provide estimates for hardware sizing and middleware configuration.

1. Functional testing
   In situations where the interaction between a workflow process and applications is stateless, this framework is useful for functional testing. By detecting time-outs in adaptive entities, failures in subprocesses can be identified. Then, the Simulation Management component examines the processes where the failures occur and finally identifies the locations of application failures by using the APIs of the Flow Engine. ***Table 6*** shows sample statistics for a simulation done in the functional-testing mode. To show how failures can be detected, a pseudo failure was set in the Check Availability task. Table 6 lists several client orders and the arrival time and the completion time of each order. Table 6 also lists whether a defect was identified, and if so, the state corresponding to the identified defect. For instance, in BaseRequest1, the client order arrived on October 22, 2003 at 13:00:16, and the order was completed on October 22, 2003 at 13:00.32. Alternatively, in BaseRequest3, the client order was placed on October 22, 2003 at 13:00:28, but the client order was not completed, resulting in a time-out. In this example, the statistics suggest that there is a defect and that the defect occurred in the Check Availability task.
2. Performance testing
   In the performance-testing mode, the gathered statistics are similar to those available from traditional business process simulations (such as WBI Workbench Version 4.2.4[22]). Table 6 describes customer order statistics (arrival, completion times) used for checking whether the system meets the cycle time requirements. From the queue statistics of each task (average queue size, maximum queue size) listed in ***Table 7***, the locations of any bottlenecks can be identified. In addition, from resource utilization (i.e., WCBE CPU = 63.7% and WAS CPU = 24.1%), insights can be obtained for hardware sizing and middleware configuration. Note that statistics of both simulated and real resources are reported. In this experiment, all the solution components were assumed to be deployed on a simulated WCBE server, and the Flow Engine and the Adaptive Entity Engine were run on a real Web application server (WebSphere Application Server). Furthermore, the acceleration method enabled shortening the testing time for simulating the execution of 10 client orders illustrated in Figure 14 by 47 percent, compared to a naïve synchronization of real and simulation clocks. Although the extent of shortening testing time will depend on the specific details of business integration

Table 5 Sample simulation data for WCBE-CPU resource

Task Name	Mean (sec)	Std Dev (sec)
Access Customer Profile	1.0	0.3
Access Catalog	2.0	0.7
Configuration	3.0	1.0
Availability Check	2.0	0.3
Order Submit	5.0	1.7

Table 6 Customer-order statistics

Customer Order ID	Arrival Time	Completion Time	Identified Problem
BaseRequest1	2003–10–22 13:00:16	2003–10–22 13:00:32	n/a
BaseRequest2	2003–10–22 13:00:20	2003–10–22 13:00:38	n/a
BaseRequest3	2003–10–22 13:00:28	n/a	Invoke Check Availability
BaseRequest4	2003–10–22 13:00:48	2003–10–22 13:01:07	n/a
BaseRequest5	2003–10–22 13:00:52	2003–10–22 13:01:08	n/a
BaseRequest6	2003–10–22 13:00:55	2003–10–22 13:01:12	n/a
BaseRequest7	2003–10–22 13:01:01	n/a	Invoke Check Availability
BaseRequest8	2003–10–22 13:01:18	n/a	Invoke Check Availability
BaseRequest9	2003–10–22 13:01:24	2003–10–22 13:01:41	n/a
BaseRequest10	2003–10–22 13:01:28	2003–10–22 13:01:42	n/a

solutions and the progress of the solution development, this result would be a good example illustrating the capability of the acceleration method.

CONCLUDING REMARKS

In this paper, we have discussed the role of analysis and simulation of business solutions in a SOA in realizing the business value inherent in an SOA; namely, flexibility, cost, and time to adapt business process deployments. In particular, we presented methodologies that can be used to analyze business solutions during the design and the development, testing, and deployment of the business solution. There is considerable scope for future research in this area. In order for the solution design-time analysis to be used widely, workload and middleware models that can be applied across different business solutions have to be developed and calibrated. Also, more accurate and fast quantitative-analysis techniques need to be further developed. The PSM simulation capabilities need to be further developed and integrated with deployment testing and analysis tools.

Table 7 Queue statistics

Task Name	Max Queue Size	Average Queue Size
Access Customer Profile	1	0.171
Access Catalog	2	0.382
Configuration	1	0.248
Check Availability	2	0.274
Order Submission	2	0.625

ACKNOWLEDGMENTS

We wish to acknowledge Jay Benayon, Kumar Bhaskaran, Steve Buckley, Henry Chang, David Gamarnik, Ying Huang, Santhosh Kumaran, Young Lee, Yingdong Lu, Mark Squillante, Vincent Szaloky and Frederick Wu, who have contributed to different aspects of this work.

CITED REFERENCES

1. M. Endrei, J. Ang, A. Arsanjani, S. Chua, P. Comte, P. Krogdahl, M. Luo, and T. Newling, *Patterns: Service Oriented Architecture and Web Services*, IBM Redbook, April 2004, ISBN: 073845317X, http://www.redbooks.ibm.com.
2. K. Gottschalk, S. Graham, H. Kreger, and J. Snell, "Introduction to Web Services Architecture," *IBM Systems Journal* **41**, No. 2, 170–177 (2002).
3. "MDA Guide, Version 1.0.1," *The Object Management Group*, June 2003, http://www.omg.org/mda.
4. S. Kumaran, "Model Driven Enterprise," presented at *Global EAI (Enterprise Application Integration) Summit 2004*, Banff, Canada, pp. 166–180.
5. J. D. Sterman, *Business Dynamics: Systems Thinking and Modeling for a Complex World*, McGraw-Hill Publishers (2000).

6. W. Grey, K. Katircioglu, S. Bagchi, D. Shi, G. Gallego, D. Seybold, and S. Stefanis, "An Analytic Approach for Quantifying the Value of e-Business Initiatives," *IBM Systems Journal* **42**, No. 3, pp. 484–497 (2003).
7. M. Laguna and J. Marklund, *Business Process Modeling, Simulation and Design*, Prentice Hall (2005).
8. D. Gamarnik, N. Jengte, Y. Lu, B. Ramachandran, M. Squillante, A. Radovanovic, J. Benayon, and V. Szaloky, "Analysis of Business Processes Using Queuing Analytics," Submitted to *BPM 2005 Conference* (2005).
9. Websphere Business Integration Monitor, http://www-306.ibm.com/software/integration/wbimonitor/.
10. *UML Profile for Schedulability, Performance, and Time Specification*, Object Management Group, http://www.omg.org/docs/ptc/02-03-02.pdf.
11. T.-K. Liu, A. Behroozi, and S. Kumaran, "A Performance Model for a Business Process Integration Middleware," *Proceedings of IEEE Conference on eCommerce,* 2003, pp. 191–198.
12. *Common Event Infrastructure*, International Business Machines Corp., http://www-306.ibm.com/software/tivoli/features/cei/.
13. *IBM System p5, eServer p5, pSeries, OpenPower, and IBM RS/6000 Performance Report*, October 5, 2005, http://www-03.ibm.com/systems/p/hardware/system_perf.pdf.
14. B. Appukuttan, T. Clark, S. Reddy, L. Tratt, and R. Venkatesh, "A Model Driven Approach to Building Implementable Model Transformations," *Proceedings of Workshop in Software Model Engineering, Sixth International Conference on the Unified Modeling Language*, 2003, http://www.metamodel.com/wisme-2003/04.pdf.
15. K. Czarnecki and S. Helsen, "Classification of Model Transformation Approaches," *Proceedings of OOPSLA 2003, Workshop in Generative Techniques in the Context of Model Driven Architecture*, 2003, http://www.softmetaware.com/oopsla2003/czarnecki.pdf.
16. M. deMiguel, D. Exertier, and S. Salicki, "Specification of Model Transformations Based on MetaTemplates," *Proceedings of Workshop in Software Model Engineering, Fifth International Conference on the Unified Modeling Language*, 2002, http://www.metamodel.com/wisme-2002/papers/deMiguel.pdf.
17. W. Witthawaskul and R. Johnson, "Specifying Persistence in Platform Independent Models," *Proceedings of Workshop in Software Model Engineering, Sixth International Conference on the Unified Modeling Language*, 2003, http://www.metamodel.com/wisme-2003/10.pdf.
18. T. Ziadi, B. Traverson, and J. Jezequel, "From a UML Platform Independent Component Model to Platform Specific Component Models," *Proceedings of Workshop in Software Model Engineering, Fifth International Conference on the Unified Modeling Language*, 2002, http://www.metamodel.com/wisme-2002/papers/ziadi.pdf.
19. Business Process Execution Language for Web Services (BPEL4WS), Version 1.1, May 2003, http://www-128.ibm.com/developerworks/library/specification/ws-bpel/.
20. P. Nandi, S. Kumaran, and K. Bhaskaran, "Method and System for Process Brokering and Content Integration for Collaborative Business Process Management," Pending United States Patent Application, 20030187743AI, 2002.
21. *Working with WebSphere Business Integration Server Foundation Process Choreographer*, International Business Machines Corp., http://www-106.ibm.com/developerworks/websphere/zones/was/wpc.html.
22. *WebSphere Business Integration Workbench V4.2.4*, International Business Machines Corp., http://www-306.ibm.com/software/integration/wbimodeler/workbench/.
23. P. Lendermann, N. Julka, L. Chan, and B. Gan, "Integration of Discrete Event Simulation Models with Framework-Based Business Applications," *Proceedings of 2003 Winter Simulation Conference*, IEEE, pp. 1797–1804.

Accepted for publication April 13, 2005.
Published online October 18, 2005.

Makoto Kano
IBM Research Division, Tokyo Research Laboratory 1623-14, Shimotsuruma, Yamato, Kanagawa, Japan (mkano@jp.ibm.com). Mr. Kano is a research staff member at the IBM Tokyo Research Laboratory. He received a Master's degree in mechanical engineering from the University of Tokyo in 2002. He is currently focusing on developing analysis techniques for business performance and information management. His research interests include bio-informatics and information visualization techniques.

Akio Koide
IBM Research Division, Tokyo Research Laboratory 1623-14, Shimotsuruma, Yamato, Kanagawa, Japan (e03097@jp.ibm.com). Dr. Koide leads the research group on Business Performance and Information Management at the IBM Tokyo Research Laboratory. He received a Ph.D. degree in theoretical physics in 1975 from the University of Tokyo. He joined in IBM in 1981 and has worked on the design of integrated application systems such as drug and chemical design systems and on medical imaging and database systems. More recently, he has applied his experience in system design combined with his background in theoretical physics to the performance analysis of business and its infrastructure.

Te-Kai Liu
IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598 (tekailiu@us.ibm.com). Dr. Liu is a research staff member at the IBM Watson Research Center. He is currently focusing on applying performance engineering disciplines to model-driven development of business-performance-management and business-process-integration solutions. His research interests include software performance engineering, performance modeling, wireless communication networks, radio frequency identification, intelligent vehicle highway systems, and pervasive computing. He is the author of 30 journal and conference papers and holds eight United States patents. He received a Ph.D. degree in computer engineering from the University of Southern California and is a member of IEEE and ACM.

Bala Ramachandran
IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598 (rbala@us.ibm.com). Dr. Ramachandran is a research staff member in the Mathematical Sciences department at the IBM Watson Research Center. He received a B.Tech. degree in Chemical Engineering from the Indian Institute of Technology, Madras, India in 1991, and a Ph.D. degree from Purdue University in 1996, specializing in operations research. At IBM, he has focused on developing analysis techniques for business process management and risk management. ■