Simulation as a cloud service for short-run high throughput industrial print production using a service broker architecture

Sunil Kothari *, Thomas Peck, Jun Zeng, Francisco Oblea, Anabelle Eseo Votaw, Gary Dispoto

Hewlett-Packard Company, 1501 Page Mill Road, Palo Alto, USA

Abstract

Evaluating end-to-end systems is uniquely challenging in industrial/commercial printing due to a large number of equipment combinations and customization needed for each customer and application. Moreover any mismatch in capacities may render multi-million dollar investments to zero returns on investment. Simulation can help foresee changes on the shop floor when demand changes. Providing a library of components that can be assembled together is the usual approach used by many simulation vendors which still leaves a simulation engineer in the loop to make it usable. We detail our experiences on implementing a prototype (private) cloud service using service broker architecture and a dynamic model generator. The service broker handles the heterogeneity associated with demand and equipment configurations whereas the dynamic model generator customizes a generic model based on inputs from the user. This helps avoiding rewiring of simulation models on each engagement. The schema and the necessary front and back-end codes all reside in the cloud and, therefore, users pay on a per use basis without worrying about the upgrade/update of software at their end. The service supports multi-tenancy which results in low costs per user and provides sharing of resource information yet restricting access to proprietary workflows and policies. A typical run costs a very small amount, which is affordable for even small-sized PSPs. We show the utility of our work in the context of educational book publishing to evaluate equipment changes needed when the current lumpy order demand stream changes to a highly fragmented demand stream. We also discuss how our work can be extended to several other domains such healthcare, transportation, 3D printing.

1. General introduction

Printing is one of the ways in which digital information can be converted into a physical format. When a printing facility serves multiple customers with multiple print products then the facility becomes a print service provider (PSP). Though many print facilities and operations can be termed as PSPs, in this paper we restrict ourselves to high throughput industrial print facilities, which are characterized by very high volume demand (thousands of printed products per day), high throughput equipment (more than 40,000 feet of paper per hour), worker shifts all day and night and multiple million dollar equipment. A typical workflow in a PSP is summarized below and detailed steps are also shown in Fig. 1 [41]:
1. Gather order information and customer supplied content (Steps 1 and 2).
2. Pre-process content (convert all PDFs to v1.4) and do various checks such as color, paper layout, and registration (Steps 3–7).
3. Create a proof (Steps 8–10) and ask customer for feedback.
4. If all OK, proceed to Step 11. If not OK, make necessary changes and restart the prepress process (go to Step 3).
5. Add production information (batching information) and rasterize the content (Step 12).
6. Print the rasterized content (Analog workflow will have Steps 13–18; digital workflow will only have Step 18).
7. Do post-press steps such as cutting, folding, and stitching (Steps 19–21).
8. Gather, package and ship print products (Steps 22–24).

The above steps are illustrated for digital commercial/industrial print. For analog commercial/industrial print, Steps 13–17, which are costly and time consuming, are completely eliminated from the workflow. Moreover, over the last two decades, digital printing has slowly, but surely, replaced analog printing for many applications due to advancement in printing quality, color accuracy, and cost effectiveness for the shorter runs. This does not means that analog printing is going to go away. For certain applications, analog printing will continue to be cost effective.

The demand profile has also played a key role in this transition. For example, Precision Printing, a PSP based in UK, has reported an astounding more than 200% increase in number orders per day but a 99% decrease in average value of the order as it moved from analog to a mix of analog and digital printing as shown in Table 1. The data for this table has been compiled by the author from an article published in [1].

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Before 2005</th>
<th>After 2005</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration</td>
<td>All Analog Presses</td>
<td>Digital + Analog Presses</td>
</tr>
<tr>
<td>Turnover</td>
<td>$8M</td>
<td>$19.2M</td>
</tr>
<tr>
<td>Average orders/day</td>
<td>45</td>
<td>10,000</td>
</tr>
<tr>
<td>Avg. order value</td>
<td>$795</td>
<td>$3.68</td>
</tr>
</tbody>
</table>

Fig. 1. Analog and digital workflow.
Below is an excerpt from a case study published by PRIMIR [35].

“Serving the needs of Websites such as Tinyprints and Frecklebox revealed two major issues: a significant increase in the number of orders (over 3000 jobs a day), and a decrease in the quantity and price of each ordered item (Progressive’s minimum order value is $1.44)”

2. Problem statement

To speed-up transition of Print Service Providers (PSPs) from analog to digital printing presses, we believe the PSPs should have tools which help them guide this conversion. In many cases, it boils down to answering various what-if questions that a PSP shop floor manager and senior management might have. Previously, these questions could be answered by the shop floor manager just based on his experience and knowledge of the factory. But with the increase in number of jobs (due to customization, personalization of content) and decreasing run lengths (due to print on demand, personalization), the volume, velocity and variety of answering what-if questions cannot be answered the traditional way.

A particular motivation of our current work involves answering questions involving educational book production and handling demands which are very short-run. This is precipitated by a change in the way elementary and high school (K-12) educational book market is changing as shown in Table 2. The main printed product here is booklet apart from other accessories which get packed with booklets such as examination sheets and teacher’s keys. Fig. 2 shows the typical process flow in producing a booklet, which starts with content being downloaded and then is branched into two concurrent process flows one deals with cover while the other deals with inside page or a book block. After printing both come together and are bound and then trimmed at the sides. An optional three-hole drill may occasionally be used if a number of booklets are put together in a folder.

The driver for this change of demand is the decision by various schools to get a personalized book that meets the needs of individual class/student rather than a region as a whole. Eventually, each student will be able to order get a personalized book with a lead time of few days. The volume of demand remains the same but the demand gets fragmented to smaller and smaller orders along with more frequent orders (with the expectation that delivery will be faster). Moreover there is a change in the business model too. Prior to 2014, publishers would put a large order for books based on anticipated volumes. In 2014 and beyond each class each student will be able to order his own books. These changes will permeate to how the books are packed and shipped. However, in this paper, we only consider the changes to the production of these booklets.

The strain on the production floor will come up in terms of managing the changed demand – a shorter lead time yet maintaining the high throughput requirements for better return on investment (ROI). As seen from Fig. 1, there is a lot of equipment that go before and after the printing press. While the artifacts produced by the upstream equipment (from printing press) deal only with electronic objects, the downstream equipment all deal with physical products. The upstream equipment can almost change instantaneously to change in resolution, color profiles, change in book dimensions, etc. without paying a penalty in set up times (although they may face increasing processing times). The downstream equipment face either wasted cycles or non-trivial set up times that range from a few minutes to few hours. Interestingly, in traditional simulations of analog printing factories, processing times are significantly more than set up times. With the change in order profile to high-mix demands set up times become comparable or even significantly more than the processing times. Therefore, accounting of setup times becomes crucial. That means that we really have to pay attention to each copy and simulate at the copy level. This is irrespective of manual, automated or semi-automated setups needed across the equipment set.

3. Proposed approach

The what-if questions are primarily about changing the existing flow of processes (What if a single printing press is used to print the entire book and not just the cover of a book?), adding/replacement of a set of equipment (What will be the gain in production time if I add a new printing press to an existing setup?), changes in operating policies (What if we change scheduling of jobs from first in first out to earliest due first?) or changes in demand (What if each class/students orders his own books?). The effects of these changes are quantified in terms of operating costs, equipment utilization, cycle times and return on investment. While simulation tools can certainly help in answering such questions, our aim is to empower solution architects and factory managers who may not be entirely familiar with the simulation technology and its

<table>
<thead>
<tr>
<th>Table 2</th>
<th>Changes in K-12 educational market and their effect on demand generation and fulfillment.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vectors</td>
<td>Until 2013</td>
</tr>
<tr>
<td>Demand mix</td>
<td>Each school gets the same booklets for each class</td>
</tr>
<tr>
<td>Demand fulfillment</td>
<td>Anticipated demand from publisher guides production runs</td>
</tr>
<tr>
<td>Packaging and logistics</td>
<td>Shipped as a pallet to each school per semester</td>
</tr>
</tbody>
</table>
terminology. Senior management often have doubts about using simulation but such questions were settled earlier in our previous work, where we showed that there is 8% error in end-to-end lead time between the simulation data and the audit data from a real print factory [2].

A typical workflow for a simulation as a cloud service is shown in Fig. 3 as a six step process. We enumerate each of the steps next.

Step 1. The solution architect interviews a factory manager to gather the minimal domain inputs required for the simulation to run.
Step 2. User inputs such as demand profiles, production plans, operating policies, equipment cost and performance characteristics are then used to simulate a production run for a fixed period in the cloud.
Step 3. Current state of the print factory is determined by computing financial and production performance metrics.
Step 4. Future configuration state of the print factory is specified and may involve a change of demand, a change in number of equipment or a change in operating policy or a combination of aforementioned changes.
Step 5. Multiple scenarios are executed concurrently in the cloud.
Step 6. Future state is determined from the performance and financial metrics gained from the simulated scenarios. Users can do the scenario by scenario comparison for all the scenarios.

The Service Broker [4] module forms a critical component of our work and hides a lot of decision making in the proposed architecture. It breaks orders into products and products into tasks and routes each task to the respective equipment while following the order defined in the production plans. In addition, there are server groups, which represent a collection of machines that can fulfill a given process. Each individual machine is referred to as a server. We allow heterogeneous server in server groups. With each server group, we associate a dispatcher that schedules and forwards tasks to individual equipment based on certain policies. Together, they form the core pieces of Service Broker architecture. Throughout this paper, we will use the terminology “Service Broker actor” to refer to the actor which does this complicated routing of jobs, and we will use “Service Broker architecture” to refer to a collection of Server Groups, Servers, Dispatchers and Service

![Fig. 2. Production plan for a booklet in BPMN.](image-url)
Broker Actor in a certain fashion that promotes loose coupling and reuse of Ptolemy models. Fig. 9 provides a conceptual overview of above concepts.

The ideas described in this paper have been translated into a prototype/cloud service named Production Designer, based on the open source electronic design automation toolkit Ptolemy II [2], and is targeted toward solution architects and PSP shop floor managers as a tool/service to aid in decision making. It provides a quantitative insight to intuitive and non-intuitive results. The service currently runs on HP Helion [9] as well as HP Labs private cloud, based on OpenStack [8], and exploits elastic computing paradigm for running, gathering and analysis of concurrent simulation experiments.

Our main contributions are:

1. Using service broker architecture to take out the heterogeneity and provide loose coupling in modeling print production instances to promote reuse of simulation models.
2. Abstracting the domain parameters and dynamically specializing a generic Ptolemy schema based on domain-specific user inputs.
3. A novel architecture to enable Simulation-as-a-Service.
4. Concrete implementation of the ideas in an open source simulation framework.
5. Case study to illustrate how the same generic schematic in Ptolemy is specialized for two scenarios.

Section 4 describes the related work in the area. Section 5 describes the modeling of commercial/industrial printing as a 4-dimensional entity and our modeling and simulation environment Ptolemy II. Section 6 illustrates the Service Broker architecture and its implementation. Section 7 discusses the cloud architecture. Section 8 illustrates some sample simulation runs and their outputs and the user interfaces in Production Designer. Section 9 concludes the paper with some pointers to future work.

4. Related work

Spreadsheet-like static estimation tools are used in commercial/industrial printing for answering what-if questions. These tools fail to account for factory dynamics. Desktop tools based on stochastic discrete event simulations exist in IT management, hospital services, electronic manufacturing services, etc. They rely on either large amount of event log data or weeks of simulation experts’ time to construct custom-made simulation models.
The service oriented architecture (SOA) and the role of service broker has been discussed in several papers in the context of grid computing, web services and cloud computing [4,15,16,30–32,34]. We discuss the relevant work in each of these areas. The use of SOA-compliant DEVS (SOAD) is described for simulation of service-based systems is described in [28] and is orthogonal to our work. Kanacilo et al. [23] and our work share similar goals that we want to make simulation within easy reach of business people. They look into displaying historical information, collaboration on rail design and are able to use different simulation software. Clearly, they do not focus on model reuse.

A number of web-based simulation and supporting tools are discussed in [20], including the use of grid computing in simulation. Many of the concepts in grid computing also apply to cloud computing [29]. The authors mention (citing [30]) that grid computing never really took off for the industrial application of simulation, especially the commercial, off-the-shelf (COTS) simulation packages. In [31], Venugopal et al. describe a GridBus broker which shares a number of functionalities that we defined for our service broker although we keep our service broker quite light weight. For example, it does not do job scheduling.

Web services are an integral component of many service oriented architecture (SOA) implementations. A vast amount of literature is available on various aspects of SOA. We provide a summary of papers which discuss service broker in the context of web services and service oriented architecture. Web services are used in many distributed simulation applications. Eisen and Stein [27] discusses offering YANOS simulator as a web service. Models are uploaded to the server and can be instantiated by providing values. Clearly, they do not account for a generic model like we do. Tsai et al. [19] provide a comprehensive framework to rapidly develop simulation models and associated infrastructure to run them. While very powerful but it requires technical knowhow to operate the system. Moreover, it is not clear from their description whether it is cloud enabled. Wang et al. [21] describe a methodology to study the three different kind of service-oriented simulation frameworks. Wang et al. [22] improved the earlier paper [21] to describe the service broker in the context of service-oriented modeling and simulation framework as the role of a web search. Clearly the service broker we have does not fulfill that functionality but instead is confined within a single system and promotes loose coupling so that model reuse is possible. Huang et al. [24] describe a SOA system where a simulation manager dynamically chooses a simulation engine and dynamically configures it. This is similar to how we instantiate a generic schema. Clearly, their work involves dealing with a large number of different simulation technologies. Kim et al. [26] describe the rube framework which uses XML throughout the modeling and simulation process for dynamically generating custom models. They also compare their work with the MoML representation used in Ptolemy II. It is unclear from the paper how they can reuse a simulation model without implementing a service broker architecture.

The use of simulation in manufacturing is a major research area. In [42], authors cite several areas based on a study of 290 papers published (which was selected from a pool of 12,000 papers containing ‘manufacturing’ and ‘simulation’ keywords) from 2002 to 2013. The papers were classified into three broad areas: manufacturing system design, manufacturing system operation, simulation language/package development. Our work falls into the first and the last category. In the manufacturing system operation category, several works related to facility layout, material handling, cellular manufacturing system design and flexible manufacturing system design are discussed. In the simulation language/package development, contributions related to generic simulation models, simulation metamodeling and optimization are discussed. In particular, they cite paper by Fowler and Rose [48] on the general purpose architectures for model reuse and plug-and-play interoperability. Interestingly, the cloud deployment of simulation platforms is not discussed in the paper. In [43], the authors discuss the personalization and customization challenges in designing manufacturing networks and the role of simulation.

Simulation as a cloud service and composability of multiple simulation services was described by the author in [11] in the context of wildfire spread simulation. Problems such as maintaining “simulation time”, mismatches, and scalability arise when multiple simulation services are composed together. The author propose a framework that has five layers: graphical user interface, simulation experiment, simulation service, simulation execution and infrastructure layers. The “simulation time” problem is solved by introducing a coordinator service. The mismatch problem is handled by providing specifications at the experiment and service layers. For the scalability problem, two spatial partitioning algorithms are proposed to parallelize the simulations. The user still has to manually compose services and some of the concepts that we describe here can definitely be used to automate the creation of composite services.

Cayirci cites a number of concepts and issues related to offering modeling and simulation as a cloud service (MSaaS) [33]. Those issues include various cloud architectures such as standalone, federated standalone, composed, and automatically composed MSaaS. Cayirci also discussed security, privacy and risk issues with MSaaS. In our case, we are running our service in a stand-alone private cloud; therefore, many of the issues such as security and privacy are not relevant for us.

Various aspects of simulation in the cloud are dealt in the more recent published research papers. In the next few paragraphs we summarize these papers. Tao et al. propose the cloud manufacturing (CMfg) platform [44] which encompasses the advancement in computing side such as the cloud computing, Internet of Things (IoT), service-oriented technologies, and the advanced manufacturing concepts such as virtual manufacturing, lean manufacturing, digital manufacturing to create a platform for on-demand, high utilization use of manufacturing resources. Simulation is proposed one of the cloud service (SIMaaS), which could be run as a public, private, community or hybrid CMfg service but implementation details such as architecture for the service, domain-specific or domain neutral, and open source or commercial packages are not mentioned in the paper. Li et al. [45] describe two important and relevant contributions. First, they describe individuation virtual desktop which helps dynamically create VMs based on user inputs. In our case, we hide the configuration and creation of VMs from the user since our aim is to give business users an interface that demands minimal technology inputs from the users.
Second, they create a more dynamic way of dispatching simulations to the cloud based on the type of simulation. In our prototype, we currently only have one simulation software (i.e., Ptolemy II) and a fixed virtual machine configuration. Third, the fault tolerant mechanism proposed in their work is different from ours. They support a migration when a VM crashes or becomes too slow whereas we re-spawn a VM when a crash is detected and the resources of the crashed VM are returned to the cloud after storing the debug information for future analysis.

Chen and Lin [46] describe ways to estimate simulation workloads for load balancing in the context of CMfg. They identify four metrics to estimate workloads: file size, number of job types, frequency of releasing jobs to a factory, total number of operations, etc. Their approach is useful if we have to weigh running in a private cloud vs. public cloud or a hybrid cloud or if the number of different simulation systems are to be run in the same cloud. Currently, we are at the prototype stage and many of the work cited here will be relevant as we scale our approach to multiple cloud systems and multiple simulation systems.

Zacharewicz et al. [47] present an approach to combine G-DEVS, an extension to DEVS formalism [49], and the high-level architecture (HLA) [50] to bring distributed simulation capabilities to workflows. Our work can be extended with their approach to have a distributed service broker actor which manages several of the dispatchers and servers.

There are some other works which do not deal with SOA, but relate to the print production domain. Kuehne [25] discusses the networked print production simulation (using JavaSim) with Job Description Format (JDF) and Job Messaging Format (JMF), which are XML representation of workflow models and data exchange. Many of the print equipment are JDF/JMF capable; therefore, by having JDF/JMF capability, real production modules can be replaced with simulated ones. Kuehne claims that with JDF/JMF interface, simulation can become an operational planning and control. For our work, we do not need JDF/JMF capability but it can be integrated with Ptolemy II simulation engine. We have also developed a prototype implementation where the rasterization of a PDF is done inside Ptolemy and we certainly believe Ptolemy has the potential to be an operational planning and control tool.

Many of the research efforts mentioned in this section deal with the SOA and service broker at the system level rather than at the model level. While many of the approaches can be scaled down at the model level we think our contributions show that it opens up interesting (some domain-specific) design choices and brings extra complexity but it also brings extra de-coupling that is needed so the models can be re-used in a majority of the cases. Additionally, hosting in the cloud creates additional issues which again depend on the domain and the intended users.

5. Ptolemy II and modeling commercial/industrial print as a cyber-physical system

Cyber physical systems [36] have been a key research area for funding by National Science Foundation [37]. Growth in cyber physical systems will help us put computational and physical elements together that will have a tremendous impact on various domains such as manufacturing, healthcare, and transportation. We need tools in system design, system verification, real-time control and adaptation, and in manufacturing to design, analyze and optimize a wide range of cyber physical systems [37]. To design and analyze cyber physical systems, each domain has to be virtualized and control logic separated from the underlying physical infrastructure such as machine and labor. This initially has to be done for each domain but, over time, considerable vocabulary and knowledge would carry over to several domains. GE has termed such highly networked, intelligent systems as Industrial Internet [39].

Ptolemy II [2], an open source object oriented electronic design automation toolkit from UC Berkeley, has been extensively used for modeling and simulation of cyber physical systems [12,13] where compute, communication and physical resources come together for a specific purpose. The core enablers of a CPS are ubiquitous sensing, just-in-time computing and communication. Through integrating computation and physical processes, a CPS monitors, coordinates and controls its operations and physical states. In a CPS, sensors are deployed to collect information about the physical operations of the system (“nerve”), the sensing data are communicated in real-time to the computing (“brain”) for analyses and decision making, and the control signals are transmitted to coordinate and drive the system operation (“muscle”). Industrial and commercial printing is a specific instance of cyber physical system [3,14,38], where compute (rasterization engine), communication (sending color data to cloud for analysis, signals for servo motor controls) and physical resources (printing press, finishing devices, sensors) collaborate to produce physical outputs such as photobooks and calendars.

Modeling and simulation of CPS systems is challenging due to scale, heterogeneity and dynamic nature of CPS systems [5]. Although in the current work we have only used the stochastic discrete event domain but in the future we plan to integrate other domains such as continuous domain for modeling ink flow, dryer capacity in a printing press, and, in future, look at combining such domains in the same simulation environment. We believe Ptolemy II platform can help us achieve that. In fact, Zeng et al. [40] have done some related work in modeling print-head design, which deals with ink flow.

In this paper we focus on discrete event simulation.

Our initial engagements with modeling specific customers involved two different workflows, one for a large format print service provider and the other for a photobook. Fig. 4 shows the schema used for a poster workflow with the main actors numbered. We summarize them below:

1. Initialize date and time use for simulation and set up different tables.
2. Define the different mix of product that will be used in the simulation.
3. Orders are converted into products and products into parts.
4. Various prepress operations are performed to make sure that the job is ready for printing.
5. Jobs are queued and dispatched in a pre-determined batch size so that demand fluctuations are minimized on the shop floor.
6. If the job is not ready, it may be routed back for more prepress operations. Otherwise, the job takes different routes based on the value stream. There are three value streams: (1) Custom Book, (2) Standard Book, and (3) Softcover Book.
7. Covers are printed.
8. The inside pages of a photobook are printed.
9. Both the cover and inside pages (aka bookblock) are brought together by scanning the bar codes on both the parts.
10. The cover and insides pages are glued and bound together. If there are defects, they are routed to Step 4.
11. Multiple photobooks (that form an order) are brought together.
12. Completed order is shipped.

Fig. 5 shows another modeling exercise and involves a wide format press which prints directly on poster sheets. Print quality is rigorously controlled at various points in the control points. The numbered steps in Fig. 5 are described further below:

Step 1. Simulation time is initialized.
Step 2. Various tables are created or emptied (if already available) in the database. Orders are released to the shop floor based on pre-determined policies.
Step 3. Check the incoming PDF document for compatibility with the RIP (a process called pre-flight), assemble images in the layout, do a soft-proof and if it is error-free then go to next step else change and pre-flight again.
Step 4. Various color transformations are then done to ensure the colors are accurately reproduced.
Step 5. The contents are converted from an input format (PDF) to a format that press understands.
Step 6. For each job, various set up costs are determined. One of the costs is for switching the media. Orderid’s of two consecutive jobs are compared to determine whether a change of media is required.
Step 7. The actual cost and time are computed here.
Steps 8 and 9. Posters are printed, only if their hard copy passes the check.
Step 10. Printed posters are checked for quality. Rework is sent back to Step 6.
Step 11. Finishing may involve any of the following, mounting, cutting, gluing, etc.
Step 12. Finished posters are checked for quality and rework is sent back to Step 6.
Steps 13–16. Posters are sorted, kitted (adding additional accessories to each poster), shipped and installed once they pass quality checks.
While both the schemas work for their respective workflows, at the system level there are multiple similarities:

(a) Rework is involved in both the schemas.
(b) Equipment is measured on the same metric such as utilization and cycle time.
(c) Equipment takes inputs and transforms it to outputs and passes the output to the next step in the workflow.

There are dissimilarities too:

(a) Some workflows may have different rework destination nodes.
(b) Workflows differ in the number of processes and in how processes are organized.
(c) A process can be fulfilled by multiple machines.

Both the schemas require a simulation engineer to understand the process flows, several hours of consultative engagements spread over multiple days, yet the models are customer specific and not amenable to change in process flows or change in equipment configurations. Moreover, their use and installation require deep technical knowledge.

Additionally, there are hardware and software issues such as number of servers, number of CPU’s per server, software updates to each server, and operating system upgrades to each server and clients. Thus, not only a simulation engineer but also an IT engineer is occasionally needed to maintain the simulation hardware and software infrastructure. We believe this is a major hurdle in why simulation is unreachable to solution architects and factory managers.

We envisioned a tool that can take minimal user inputs and make simulation reachable to a wider audience. Since many of our use cases involved operating a few times a year, it cannot be a capital expenditure. A cloud service will not turn the expense into operational expenditure (pay per use) but will also avoid hardware issues related to upgrade/update of components. By adopting service broker architecture, we ensure that the heterogeneity of workflow and resource configurations should be accounted for in a schema.

That raises the rather difficult question of how generic a schema do we want. A very general schema is theoretically possible but our experience suggests that it will be very big and very complicated but that will cover all domains. A less general schema could be vertical specific (such as retail, finance, and healthcare) and would be easier to manage but would still be complicated and will require a lot of compute resources. A step lower is to focus on schemas which are in a domain but focus on a specific industry, say commercial and industrial printing in our case. At this level, we cannot guarantee 100% coverage (within a given vertical domain) but we can certainly cover 80% of the cases and still be bound by reasonable compute guarantees.

To enable this generic schema we need a different kind of architecture to handle wide diversity of order demands and their corresponding fulfillment. Fortunately, service broker [4,17] is a proven concept to take into account such heterogeneity. But introduction of service broker in the schema provides us with a number of design and implementation choices (in the
printing domain) which further guide the overall implementation. This paper not only suggests a possible architecture for modeling distributed printing fulfillment processes but also suggests the design alternatives.

The simulation engine running inside the Production Designer is based on Ptolemy II. We have made several enhancements to the original Ptolemy II and, as a result, what we have is a much customized version of an open-source tool. More specifically, we have added a library of actors (entities in discrete events) including: Demand Generator models that use either real factory order data or statistical order profiles to drive production simulation, family of Resource models (e.g., equipment and workers) and other modules that provide dynamic payload estimate, job sequencing, routing and assignments, and factory schematic that enables automated scripting of factory model eliminating the need of manual placement-and-routing.

5.1. Modeling production plans

An order comes with multiple demands and each demand is associated with a product. Each product is associated with a production plan. This structure is described in Fig. 6. We treat a production plan for a product as a directed graph. The additional demand structure helps us generate orders coming from high-mix low copy count and low-mix high copy count for the same order profile. This is an essential element of our case scenario described in Section 4.

If a rework for a node is not defined, then that edge is treated as a self-loop, i.e., it points to itself. Thus, each process is represented as a node in this graph. We ignore back edges generated due to rework since their effect on production and financial metric is negligible. When process is defined we assign a default payload which can then be customized to specific applications, i.e., when processes are actually used in a process flow.

In Table 3, we tag the various processes involved in the booklet production into the three categories shown in Fig. 1. This categorization represents meta-information about process which is then exploited to signal transitions (elaborated in Section 3). This becomes all the more important when we allow users to enter new processes. Slight misspellings of process names (for example, print to print) will spoil the transitions from prepress to press if the names are used for transitions.

5.2. Modeling orders

The order generation module generates orders from the order profile by sampling from a Gaussian distribution on the demand profile to generate the actual demand. The following attributes are generated for an order:

Order attributes:

(a) order arrival time; (b) order due date; and (c) order id (an auto-generated number).

Demand attributes:

![Fig. 6. Order, demand, product relationships and their corresponding attributes.](image-url)
(a) Demand id and (b) demand name.

**Product attributes:**

(a) Product id; (b) product type; (c) average resolution of images; (d) number of images; and (e) length and width of final product.

We do not consider separate shipments of products. We assume that all products will be collected and shipped together.

### 5.3. Modeling operating policies

We consider the various rules for prioritizing jobs and assigning of tasks to the machines. Jobs can be prioritized based on the earliest delivery date, minimum slack, minimum lead time, etc. For assigning tasks to machines, various policies such as FIFO and round-robin are currently built-in.

### 5.4. Modeling resources

We consider equipment (and their consumables) and workers as resources. For equipment, we consider the cost characteristics such as equipment cost, maintenance costs, working space, and number of operators and for time characteristics we consider the processing modes and speeds, set up times, dependence of set up times on incoming and outgoing products. We consider both the substrates for printing presses, i.e., sheets and rolls. We generate the actual setup and processing times based on the inputs assuming a Gaussian distribution. Thus, each server has a little variability even within the same server group.

In our case determination of set up time is crucial. We use the mapping in Table 4 to guide whether a setup is required when an incoming product dimension changes. A similar mapping is also done for outgoing product and the effect of change in dimensions. Note that for prepress processes (which deal mostly with electronic objects) there is zero effect on set up of these resources as the incoming/outgoing dimension changes. The changes to incoming/outgoing dimensions can be either done in parallel or sequentially. Fig. 7 shows a partial decision tree to evaluate the set up time from the individual set up times and equipment mechanism for dealing with the changes (see Table 7).

Note that the mapping in Table 4 is for products which are within the specifications provided by the equipment manufacturer. For example, if a Laminator only accepts a 10–21 cm range covers then it is implicit in Table 4 that all products that hit the laminator are within that range. We do a separate analysis to figure out which equipment combination is best for a given substrate and a given product line.

### Table 3

Process categories.

<table>
<thead>
<tr>
<th>Process name</th>
<th>Process category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Download</td>
<td>Prepress</td>
</tr>
<tr>
<td>Rasterize Cover</td>
<td>Prepress</td>
</tr>
<tr>
<td>Rasterize BookBlock</td>
<td>Prepress</td>
</tr>
<tr>
<td>Print Cover</td>
<td>Press</td>
</tr>
<tr>
<td>Print BookBlock</td>
<td>Press</td>
</tr>
<tr>
<td>Laminate</td>
<td>Post-Press</td>
</tr>
<tr>
<td>Perfect Bind</td>
<td>Post-Press</td>
</tr>
<tr>
<td>Three-Knife Drill</td>
<td>Post-Press</td>
</tr>
<tr>
<td>Three-Hole Drill</td>
<td>Post-Press</td>
</tr>
</tbody>
</table>

### Table 4

Mapping the equipment and the set up needed for incoming and outgoing products.

<table>
<thead>
<tr>
<th>Equipment</th>
<th>Process</th>
<th>Brand</th>
<th>Change of input dimension</th>
<th>Change of output dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Length</td>
<td>Width</td>
</tr>
<tr>
<td>Machine 1</td>
<td>RIP</td>
<td>Brand A</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Machine 2</td>
<td>PrintCover</td>
<td>Brand B</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Machine 3</td>
<td>PrintBlock</td>
<td>Brand C</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Machine 4</td>
<td>Laminator</td>
<td>Brand D</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Machine 5</td>
<td>CutFoldBlock</td>
<td>Brand E</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Moreover, for many of the press and postpress equipment, the change in set ups may mean a lot of paper wastage since equipment requires calibration, ramp up/down to/from production speeds. Additionally, for roll-fed machines, changing rolls is time consuming (a few hours), and so, they are setup for minimizing setup times rather than minimizing paper waste. We believe factory managers will be interested in these kinds of pareto-optimal design spaces and tools that can guide them in taking such decisions.

Next we describe how Service Broker interfaces the demand and its fulfillment by breaking an order into products, products into tasks and attaching a payload estimate to each task.

6. The service broker architecture and implementation

Fig. 8 illustrates the generic model/schema needed to model a commercial PSP’s operations. Fig. 9 shows Service Broker and other components in the generic schema in Ptolemy II.

The service broker takes into account the diversity of demand and presents a uniform interface to supply. For example, different demands and workflows are translated into a set of tasks with an associated payload. But supply side can vary too. For instance, a small PSP may have 2 binders, 3 cutters and 1 printer whereas a big PSP might have 16 printers, 6 binders, 9 cutters. This diversity of resource-mix somehow has to be handled uniformly in a simulation schema. So we employ a generic simulation schema that assumes a maximum of 20 different process groups and each process group can have up to 20 different servers with the limitation that total number of servers in all server groups must not exceed 20. This is our design decision and, of course, there are other ways of allocating in on a per-case basis, but it gives us what we want and keeps the schema simple.

Service Broker architecture comprises of the service broker module, which consists of Order Prioritizer, Task Router and Payload Analyzer, and a number of resource groups. We will describe the three service broker sub modules a little later. The service broker module routes job to a dispatcher which maintains information about the current state of each resource in that particular resource group as shown in Fig. 9. A resource group is a collection of resources (all different kinds of perfect binding machines) which satisfy a common purpose. On completion of a task, the machine sends a message to the service broker which then identifies the next action to be performed for that task based on the process flow. If two processes use the same piece of equipment, it will appear in each resource group. The machines themselves are our custom built actors and they serve the following purpose:

Fig. 7. Decision tree to evaluate set up times based on changes in incoming product dimensions.
(a) Handle interrupts caused by machine breakdown and worker breaks.
(b) Compute the machine roll change if it is a piece of equipment that needs a substrate such as a printing press.
(c) Update task queues on job arrival and departure.
(d) Store previous job dimensions to calculate set up times needed for current job.

Fig. 8. Top level Ptolemy II schema showing service broker and server group actors.

Fig. 9. Service broker architecture for commercial/industrial print.
There are several advantages with this architecture:

1. Each machine has a task queue which takes into account any buffering of tasks.
2. Service broker takes into account the parent children relationship as in fork and join of nodes, i.e., all parents have to finish before a child node is executed.

When orders are generated form a demand profile, we also calculate for a given order the amount of paper (sheets/web length) needed to fulfill that order. This information is passed on to the machines and serves as a payload for a machine.

The entire abstraction of a print factory is captured in the following arrays in Ptolemy II:

- DispatchPolicy – Contains one element per each process for the policy used by the dispatcher for job scheduling.
- Production Planning – Working modes for machines, batching and ganging instructions.
- Order Profiles – Stores the order profiles used to generate orders.
- Servers – Stores various machine attributes such as set up times, workers, price, and failure rates.
- ServerGroups – Stores the various process related info such as name, process category, subcategory, and server indexes.
- ProductionPlans – Stores the process flows and the associated payloads.
- Costing – Stores the accounting information.
- PEqDimChangeTable – Captures the incoming/outgoing changes and their effect on setup times.

The next several sections are devoted to describing the Service Broker functionalities below.

6.1. Order prioritizer

When new jobs come we need to prioritize which orders go first which go next. Current prioritization rules are: first in first out, earliest due date first and minimum slack time.

6.2. Task router

In Production Designer, new orders are admitted at every TAKT time interval. When and how many orders to admit is a user-defined value. When old tasks are finished, the Service Broker acknowledges the receipt of old tasks and updates the current state. The next steps in the execution are based on the status of the current task, i.e., whether it failed or executed successfully, and process category of the finished task (i.e., prepress, print and postpress). We envisage other categories such as shipping and installation but the current categories are enough to model the production within the four walls of a PSP. Specifically,

- If the finished node is marked for rework then we look up for rework destination and dispatch the node accordingly. In the current version failed tasks are re-sent but we allow rework policies to be specified for each process node such that any ancestor node and subsequent nodes can be sent. Of course, this calls for updating the state in the Service broker actor which can be done easily since Service Broker maintains additional data structures to know the current state.
- If the category of all the successors is other than “prepress”, and the current node is not marked for rework then we explode the current node into a set of independent nodes such that each copy of a product is represented in the simulation. We also updated the data structures related to tracking and concurrent dispatch of nodes. The actual algorithms follow a little later in this section. We also check if the completion of this node marks the completion of an order. If so, we also update the order completion status.
- If the successors are all categorized as “prepress” then we do not explode the current node into individual copy nodes but, if possible, do a concurrent dispatch of its successor nodes. At this point, we might check for order completion, but it is an unrealistic workflow where all the operations are categorized as prepress.

We track each node by a string of the format “a#b#c#d#e”, where a stands for PSP ID, b stands for demand ID, c stands for product id, d stands for copy ID and e for a node ID. Each ID is a positive integer.

Thus, we have visibility at the copy level of an individual order, and we can check for an order completion when the last copy of each product is completed.

The algorithm that tracks order completion and also handles the concurrent dispatch of nodes is outlined below:

Inputs: Incoming task node t, a persistent list L – each member of the list is a string of the form b#c#d#e where b denotes an order number, c a production plan for a particular product in b, d a copy no and e a node in the production plan c, an order collection C and a persistent data structure R that is designed for near constant time query of order/copy completion status.
Main Steps:
1. Initialize O to [],
   Let k = EXTRACT_TRACKID(t); Add k to L.
2. Let p = extract_productID(k)
   Let G(V,E) = build_workflow_graph(p) /* p uniquely indexes a production plan represented in XML */
3. Check the status of the task t.
   a. If the task failed, add k to O. Go to Step 4. /* Task needs rework */
   b. If the task is a new order task, go to Step 5.
   c. If the task is an old order task, go to Step 5.
4. Let V_r be the successors of the reworked node v_r /* set the state and execute the node as a normal node */
   a. Remove all the nodes reachable from v_r in L
   b. Add the rework destination node to O.
   c. Go to Step 6.
5. Let z = EXTRACT_PROCESS_CATEGORY(t),
   let P = successors(t, G)
   a. If z is prepress,
      then if all the members of P are also prepress
         then go to Step 5
      else if all the successors are categorized as non-prepress
         then remove k from L /* Non-prepress operations will be done per copy */
         let n be the copies needed for the product p
         Let J = EXPLODE(k, n)
         For each j in J
         Add j to L
         Go to Step 5
      else raise exception ("Process categories of successors cannot be prepress and non-prepress")
6. For each v in V
   a. If task(v) == t
      Then v_g = v /* found the desired vertex */
      Break
7. Let Vcg = childrens(v_g, G) and Va = {v | v in V and |childrens(v, G)| == 0}
   If |Vcg| == 0 /* check v_g is a leaf node */
   Then If EXTRACT_TRACKIDS(Va) ∈ L /* All leaf nodes of G are in L */
      Then go to Step 7
   Else do nothing
   Else for each child v_c of V_g
   a. Let V' = parents(v_c)
   b. If EXTRACT_TRACKIDS(V') ∈ L
      Then add task(v_c) to O
   8. If tasks(V) ∈ L Then L = L – tasks(V)
9. For each element o in O
   a. send o
   EXPLODE(k, C)
1. Let V be the set of all the predecessors of task(k).
2. Let J be an empty list.
3. For v in V
   a. Let k' = task(v) /* get the tracking id */ from the current task
   b. Let k'' = transform(k') Change the tracking id to include the copy id starting from 1 to m (where c is the number of copies)
   c. Add k'' to J
4. Return J

Fig. 10 shows the algorithm used for concurrent dispatch of tasks on a sample workflow. A detailed listing of each step is provided in Table 5.

6.3. Payload analyzer

Previously we mentioned that each task is associated with a payload that determines how much “work” each machine has to do to process that task. This is necessary since we want to be consistent how we measure the work done for a task
between the orders and for tasks within an order (assuming they have same process on two different paths). We associate a payload expression with each task. For example, payload for a cut process \( i \Rightarrow \text{cut} \) is:

\[
\text{ceil} \left( \frac{\text{pages}}{\text{copies}} \right) = \frac{\text{capability_cutter}}{3}
\]

where pages, copies are order attributes and capability_cutter is a resource attribute and \( \text{ceil} \) is a mathematical function defined in the underlying SQL (see Section Software Used for the exact specification of this implementation). At runtime, the payload expression is evaluated for that particular task and results in a value.

The above formula is an expression of the form \( A \Rightarrow B \) where \( A \) servers as a guard and is a Boolean expression, i.e., it always evaluates to true and false, whereas \( B \) serves as a compute expression and always evaluates to a decimal number. This formalism is expressive enough to encode an entire decision tree as a collection of expressions of the form \( A \Rightarrow B \), where each path in the decision tree is represented by a member of this collection. The entire idea of estimating payload and then using data as expressions is a novel feature in our application.

Table 5
A sample working of the routing algorithm.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Excitation task label</th>
<th>State at the beginning</th>
<th>State at the end</th>
<th>Tasks released</th>
<th>Tasks in progress</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>New Order</td>
<td>[]</td>
<td>[]</td>
<td>1A</td>
<td>None</td>
</tr>
<tr>
<td>2</td>
<td>1A</td>
<td>[]</td>
<td>[1A]</td>
<td>2A</td>
<td>3A, 4A</td>
</tr>
<tr>
<td>3</td>
<td>2A</td>
<td>[1A]</td>
<td>[1A, 2A]</td>
<td>None</td>
<td>3A, 4A</td>
</tr>
<tr>
<td>4</td>
<td>New Order</td>
<td>[1A, 2A]</td>
<td>[1A, 2A]</td>
<td>1B</td>
<td>3A, 4A</td>
</tr>
<tr>
<td>5</td>
<td>1B</td>
<td>[1A, 2A]</td>
<td>[1A, 2A, 1B]</td>
<td>2B, 3B</td>
<td>3A, 4A</td>
</tr>
<tr>
<td>6</td>
<td>4A</td>
<td>[1A, 2A, 1B]</td>
<td>[1A, 2A, 1B, 4A]</td>
<td>6A</td>
<td>3A, 2B, 3B</td>
</tr>
<tr>
<td>7</td>
<td>3B</td>
<td>[1A, 2A, 1B, 4A]</td>
<td>[1A, 2A, 1B, 4A, 3B]</td>
<td>5B</td>
<td>3A, 2B, 6A</td>
</tr>
<tr>
<td>8</td>
<td>3A</td>
<td>[1A, 2A, 1B, 4A, 3B]</td>
<td>[1A, 2A, 1B, 4A, 3B, 3A]</td>
<td>5A</td>
<td>2B, 6A, 5A, 5A</td>
</tr>
<tr>
<td>9</td>
<td>2B</td>
<td>[1A, 2A, 1B, 4A, 3B, 3A]</td>
<td>[1A, 2A, 1B, 4A, 3B, 3A, 2B]</td>
<td>4B</td>
<td>6A, 5A, 4B</td>
</tr>
<tr>
<td>10</td>
<td>6A</td>
<td>[1A, 2A, 1B, 4A, 3B, 3A, 2B]</td>
<td>[1A, 2A, 1B, 4A, 3B, 3A, 2B, 6A]</td>
<td>None</td>
<td>5A, 4B</td>
</tr>
<tr>
<td>11</td>
<td>5B</td>
<td>[1A, 2A, 1B, 4A, 3B, 3A, 2B, 6A]</td>
<td>[1A, 2A, 1B, 4A, 3B, 3A, 2B, 6A, 5B, 4B]</td>
<td>7B</td>
<td>5A, 4B, 7B</td>
</tr>
<tr>
<td>12</td>
<td>4B</td>
<td>[1A, 2A, 1B, 4A, 3B, 3A, 2B, 6A, 5B, 4B]</td>
<td>[1A, 2A, 1B, 4A, 3B, 3A, 2B, 6A, 5B, 4B, 5A]</td>
<td>7B</td>
<td>7B, 6B</td>
</tr>
<tr>
<td>13</td>
<td>5A</td>
<td>[1A, 2A, 1B, 4A, 3B, 3A, 2B, 6A, 5B, 4B, 5A]</td>
<td>[1A, 2A, 1B, 4A, 3B, 3A, 2B, 6A, 5B, 4B, 5A, 7B]</td>
<td>None</td>
<td>6B, 7A</td>
</tr>
<tr>
<td>14</td>
<td>7B</td>
<td>[1A, 2A, 1B, 4A, 3B, 3A, 2B, 6A, 5B, 4B, 5A, 7B]</td>
<td>[1B, 3B, 2B, 5B, 4B]</td>
<td>None</td>
<td>6B</td>
</tr>
<tr>
<td>15</td>
<td>7A</td>
<td>[1B, 3B, 2B, 5B, 4B]</td>
<td>[]</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

Fig. 10. A visual representation of each iteration step.
We exploit the underlying SQL parser and math libraries to parse these expressions and avoid writing a parser, although other libraries can be used. Thus, we support complex mathematical expressions as payload expressions.

Since the above computation is generic to all processes and a process can have multiple expressions attributed to it (as different paths in the decision tree), we need a systematic way to discover and compute relevant expressions for a given process. The general outline is as follows:

1. Initialize the variables used in the expression with relevant order profile data.
2. Given a process, find the relevant expression(s).
3. From the expressions in Step 2.
   a. Evaluate guard.
   b. If the guard is false go to step 2 and choose remaining expressions else go to step 4.
4. Compute the sub-expression identified in Step 3b.

This payload value is shown in Fig. 11 for different processes. These values are dimensionless values and each server group will interpret it as the amount of work that it has to do. It is here individual equipment performance characteristics comes into play in the sense that two servers within a server group may take different times to process that workload because of individual variation in servers, i.e., a better server will process the same workload faster than a slower server.

6.4. Accounting for cost and time

We account for cost and time for various resources. By resources, we mean the actual equipment, their consumables and the workers. The simulation engine generates a multitude of events which are captured in the Jobtraces table. Table 6 shows the different events and the usefulness of various events.

Since many of these events such as WIP, EXITS happen on a per process basis, and then on a per TAKT basis, we will end up a lot of time in querying when data is to be shown as charts. To ease this burden, we pre-process the results in TAKT intervals, i.e., at time interval t, we query the result from the interval [t-TAKT, t] and add to the values accumulated for the interval [0, t-TAKT]. This leads to the interesting question of what happens if the simulation stops in between the time TAKT time intervals. This is actually a binary problem. Whenever the simulation stops, it is either going to be at some TAKT interval or at some other time which is a little less than the TAKT time interval. If it is the latter, we do an extra pre-processing for that small amount of time so that it does not go unaccounted. In Ptolemy II, the DE Director controls the post processing when a simulation is terminated.

Table 8 shows the kind of events we need to remember the results from previous TAKT intervals. The choice of events is heavily dependent upon the kind of charts we want to draw for the user and also on intermediate computation values.

Fig. 11. Conceptual view of Payload Analyzer.
Table 6
Generated events and their use.

<table>
<thead>
<tr>
<th>Event</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORKERBREAK</td>
<td>Generates at the start of a worker break</td>
</tr>
<tr>
<td>WIP</td>
<td>Captures the queue size in front of a machine-generated when a task comes to</td>
</tr>
<tr>
<td></td>
<td>a machine and leaves a machine</td>
</tr>
<tr>
<td>SERVICE TIME</td>
<td>Captures the time taken to process a task</td>
</tr>
<tr>
<td>RELEASED</td>
<td>Captures the task exit from the ServiceBroker component</td>
</tr>
<tr>
<td>MACHINE_UPTIME</td>
<td>Captures the time machine has been doing work, excludes any breakdowns.</td>
</tr>
<tr>
<td>FAILED</td>
<td>Generated when a task fails and needs a rework</td>
</tr>
<tr>
<td>EXITS</td>
<td>Generated when a task leaves a machine after it has finished execution</td>
</tr>
<tr>
<td>ENTERING</td>
<td>Generated when the task enters the waiting queue for a machine</td>
</tr>
<tr>
<td>ENDSIMULATION</td>
<td>Generated when the simulation has ended</td>
</tr>
<tr>
<td>COMPLETED</td>
<td>Generated on completion of an order</td>
</tr>
<tr>
<td>ADMITTED</td>
<td>Generated on order admission</td>
</tr>
</tbody>
</table>

Table 7
Postprocessing events and functionality.

<table>
<thead>
<tr>
<th>EVENT</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG_CYCLE_TIME</td>
<td>Captures the previous average cycle time computations</td>
</tr>
<tr>
<td>AVG_WIP</td>
<td>Captures the average WIP from previous computations</td>
</tr>
<tr>
<td>CAPITAL_EQUIPMENT_COST</td>
<td>Captures the capital equipment cost (a one-time event)</td>
</tr>
<tr>
<td>CYCLE_TIME</td>
<td>Captures the cycle time for orders</td>
</tr>
<tr>
<td>GM_COST</td>
<td>Captures the previous general manufacturing cost</td>
</tr>
<tr>
<td>LATE_CHARGES</td>
<td>Captures the late charges associated with an order so far.</td>
</tr>
<tr>
<td>MACHINE_UPTIME</td>
<td>Captures the machine uptime so far</td>
</tr>
<tr>
<td>MAX_WIP</td>
<td>Captures the maximum WIP seen so far</td>
</tr>
<tr>
<td>NUM_CYCLE_TIME_READINGS</td>
<td>Used to count the number of values averaged so far</td>
</tr>
<tr>
<td>NUM_WIP_READINGS</td>
<td>Used to count the number of WIP readings so far</td>
</tr>
<tr>
<td>ORDER_COMPLETED_BEFORE_DUETIME</td>
<td>Captures orders completed before the due time</td>
</tr>
<tr>
<td>ORDER_COST</td>
<td>Captures the order cost for the orders so far</td>
</tr>
<tr>
<td>ORDER_TOTAL_COMPLETED</td>
<td>Captures total orders completed so far</td>
</tr>
<tr>
<td>PROCESS_COST</td>
<td>Captures the process cost so far</td>
</tr>
<tr>
<td>PROCESS_TOTAL_CYCLE_TIME</td>
<td>Captures the total cycle time for each process</td>
</tr>
<tr>
<td>PRODUCTION_TIME</td>
<td>Captures the production time for each machine</td>
</tr>
<tr>
<td>PRODUCT_COST</td>
<td>Captures the production cost for each product</td>
</tr>
<tr>
<td>REVENUE</td>
<td>Captures the cost to the order producer</td>
</tr>
<tr>
<td>SLACK</td>
<td>Captures the slack in orders (for completed orders)</td>
</tr>
<tr>
<td>TOTAL_PROCESS_COST</td>
<td>Captures the total process cost</td>
</tr>
</tbody>
</table>

Table 8
Equipment configuration.

<table>
<thead>
<tr>
<th>Equipment</th>
<th>Low-mix, high-copy count scenario</th>
<th>High-mix, low copy count scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIP Servers</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Printing Press for Cover</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Printing Presses for Making Book Blocks</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>PerfectBinders</td>
<td>1 (of Brand A)</td>
<td>1 (of Brand A), 2 (of Brand B)</td>
</tr>
<tr>
<td>Laminators</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Three Knife Trimmers</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Three Hole Drills</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ShrinkWrap</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Kutting, Shipping</td>
<td>Mostly labor intensive</td>
<td>Mostly labor intensive</td>
</tr>
</tbody>
</table>

7. Cloud architecture and platform

Fig. 12 shows an end-to-end flow of information. Our current prototype provides end-to-end functionality, including creating and documenting interviews, executing and monitoring the simulation cluster, and generating and retrieving analyses. We describe below some of the important modules:

Account Manager – handles user credentials and authenticates users sessions.
Model Manager – serves as a container for scenarios.
Scenario Manager – handles multiple runs, maps the MoML files to individual scenarios.
Ptolemy MoML generator – Instantiates the template with the interview data, and sends it to the Model Manager.
Simulation Launcher – Map a particular scenario to a VM. Requests cloud resources to launch the simulation. Also, keeps track of the simulation status.

Analysis Scheduler – Schedules all the queries for the display of chart and table data. Determines from Simulation Launcher if a scenario is still running it goes to the cloud to get the data. If not, then it goes to the simulation data manager to retrieve the data from the archive.

The interview data is stored in a private database accessible only by this PSP. The shared databases including production plan catalog, printing process catalog and equipment catalog to translate the interview data into factory system schematics that serve as virtualPSP inputs. It populates the interview data into multiple (N) scenarios covering all possible meaningful combinations of demand trends and resource compositions.

All (N) scenarios from one interview form a simulation cluster; their concurrent execution requires (2N + 1) virtual machines (VMs) of different custom images. These VMs are partitioned into three tiers (Fig. 13): one VM as the cluster’s web-based access gateway that hosts the analyzer web application that synthesizes outputs of all scenarios; each scenario includes a VM that executes virtualPSP and another VM that hosts the input job database and output production database that virtualPSP reads from and writes to. All (N) scenarios are executed concurrently by the Cluster Executor module (Fig. 12) which aims to minimize, primarily, the time to complete all simulations and also the retention of VMs.

The Cluster Executor module includes scripts to: (1) provision necessary VMs on demand based on specifications provided by Scenario Factory to create the simulation cluster; (2) construct the cluster so that all involved VMs preserve the interconnection topology as shown in Fig. 13, and monitor the simulation progress and VM health; (3) dynamically
terminate and release VMs once the simulations are completed; and (4) download input database from and upload analyses and recommendations to the Scalable Persistent Store.

In our experiments, we encountered various failure situations, for instance, time taken to generate a new VM varies sizably, and a VM fails to acquire proper public/private IP address, or fails to open necessary listener ports for both Tomcat Application servers and database servers. To address these failures, we have incorporated additional failure recovery features including, for example, over-provisioning and server pooling. We define an over-provision factor \( \alpha \) for VM acquisition. Instead of terminating all idled VMs, server pooling and recycling is implemented to retain a small number (\( \beta \)) of freshly idled VMs (of different EMIs in proportion) to supplement the creation of the next simulation cluster. Default value for \( \alpha \) is set to 0.1 and \( \beta \) is optimized between timely establishment of next simulation cluster and the cost incurred by retaining idle VMs.

Initially, our prototype was deployed in a datacenter in HP Labs running an open-source cloud computing system Eucalyptus (v1.62) [7,10]. As the business needs arose, we successfully migrated the cloud application to HPCS public cloud and then to an OpenStack [6] private cloud, again hosted at HP Labs. A technical report describing our experience with the migration is documented in [4].

The largest simulation cluster that we have successfully executed has \( N = 59 \) using 119 VMs with 2.36 GB input order database per scenario. A typical Production Designer run involves 2 demand what-ifs and 2 resource variations which requires a simulation cluster of \( N = 16 \), that is, 33 VMs and \( \sim 40 \) GB total storage with \( \sim 30 \) min lifespan. A small monthly subscription fee can cover the infrastructure cost required for 100+ planning experiments (1800+ simulations) annually. Such service subscription model can be very affordable for even the small-sized PSPs.

8. Experiments and results

We ran two scenarios involving a changing a long-run demand to a short-run demand reflecting where an industrial printer would like to go in future. We used the following equipment for both the demands although extra equipment can be added to meet the future state requirements.

8.1. Current scenario: Low-mix, high copy count

We ran the current model for a 7-day production period for 3 runs with each run averaging around 49 orders. Each order was due after 80 h from admission to the factory floor. A mean total of around 4000 booklets were produced and the overall production metric and financial metric in Table 9. The warm up for this scenario was estimated by visual inspection of utilization over three different processes: RIP, PrintBlock and Shipping over a 7 day production period. The warmup period was detected around 90,000 seconds as shown in Fig. 14.

The order stream in this scenario consists of orders arriving by school for each subject for different classes. Fig. 15 shows the distribution of copy counts for different orders for one particular run. It was assumed that each class has 3–5 sections each consisting of 20–40 students. Orders for each class and for each subject varied in number of pages but not in the dimension of the final printed product. The price of booklets was based on the page intervals – with higher pages having less per unit page cost than booklets with less number of pages. The results from the 3 runs are summarized in Table 9.

<table>
<thead>
<tr>
<th># Orders</th>
<th>Throughput</th>
<th>Avg. cycle time</th>
<th>% Paper waste</th>
<th>Revenue (in dollars)</th>
<th>Production cost (in dollars)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run1</td>
<td>42</td>
<td>3611</td>
<td>25,324</td>
<td>3.09</td>
<td>74,343</td>
</tr>
<tr>
<td>Run2</td>
<td>63</td>
<td>5377</td>
<td>50,604</td>
<td>2.96</td>
<td>116,996</td>
</tr>
<tr>
<td>Run3</td>
<td>42</td>
<td>3754</td>
<td>18,304</td>
<td>2.95</td>
<td>86,303</td>
</tr>
</tbody>
</table>

Fig. 14. Low-mix, high copy count scenario: utilization and warm-up period estimation
8.2. Future scenario: High-mix, low copy count

The simulations for this scenario involved changing a few attributes in the order profiles. Each order was assigned a due date of 36 h after its arrival. The warm up for this scenario was determined to be around 90,000 seconds. Fig. 16 shows the utilization of the three process varying with time. Fig. 17 shows the copy distribution over all the orders generated for this scenario for one particular run. Table 10 shows the various metrics generated for this scenario. In this paper, we assume that book thickness varies and the length and width do not vary. But even with these assumptions, we can see the dramatic effect on the system performance.

We outline final thoughts on the two scenarios. The number of scenarios which can be investigated is huge. We can even vary workflows without rewiring simulation schemas. What we have shown in the above scenarios is the flexibility that generic schema provides. It is clear from the tables and figures that the future scenario involves a significant investment in both prepress and postpress equipment. Of course, the results shown here are highly dependent on the kind of demand met by the system and how we operate the plant. But the results clearly indicate where a factory manager would like to focus on – for low-mix demand on printing press and for high-mix demand on shipping and RIP process.

8.3. User interface to capture inputs

We currently have a running cloud service where user inputs are capture in the UIs shown in Figs. 18a–18d. The user interface is developed in Apache Flex and the back-end uses a combination of technologies to write the user inputs in a form of XML that Ptolemy takes as input as shown in Fig. 12.
Table 10
High-mix, low copy count scenario: production and financial metric.

<table>
<thead>
<tr>
<th></th>
<th># Orders</th>
<th>Throughput</th>
<th>Avg. cycle time (s)</th>
<th>% Paper waste</th>
<th>Revenue (in dollars)</th>
<th>Production cost (in dollars)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run1</td>
<td>2377</td>
<td>1270</td>
<td>322,730</td>
<td>2.83</td>
<td>40,912</td>
<td>32,268,622</td>
</tr>
<tr>
<td>Run2</td>
<td>2187</td>
<td>1259</td>
<td>325,002</td>
<td>3.21</td>
<td>40,093</td>
<td>34,067,289</td>
</tr>
<tr>
<td>Run3</td>
<td>2302</td>
<td>1353</td>
<td>316,294</td>
<td>3.43</td>
<td>43,302</td>
<td>33,484,167</td>
</tr>
</tbody>
</table>

Fig. 18a. Equipment inventory, fixed cost parameters, cost and performance characteristics of a resource.

Fig. 18b. Substrate and worker breaks.
8.3.1. Scalability

Currently, the generic model is designed to support the following:

- Up to 20 different order profiles.
- Each order profile can have a variable (max up to 5) different/similar products.
- Up to 20 different processes across multiple workflows.
- Each process could be executed by 20 similar/different equipment.
- Each production plan can have at most 1 sheet-fed machine and at most 1 roll-fed machine.

Fig. 18c. Production plan input.

Fig. 18d. Job profiles and factory operating policies.
9. Conclusions and future work

We showed a print production as a 4-dimensional space over which we can map any print production operation using a Service Broker architecture and simulation as a cloud service to bring simulation closer to domain experts. High throughput of industrial printing coupled with short run demands meant that we need to give copy semantics to our fulfillment model. We showed how we can reason about individual demands and equipment configurations that show both the production and equipment performance over a certain period. Several other powerful features have been implemented but were not discussed in this paper including targeted dispatching of tasks, i.e., a certain task only goes to a certain set of equipment. We showed a case study where equipment change and demand changes both can be handled within the same model.

A solution architect often has to propose an end-to-end solution to meet unseen demands will have to ask a simulation engineer to construct a new model or modify an existing model. Creating a model takes time and the effort is not reusable since the printing operations, equipment configurations vary from one customer to another. However, if we abstract enough and have a service broker able to intelligently handle the flow of tokens, we can reuse the same model with minor changes. This holds true not only for print factories but also other manufacturing, healthcare, transportation domains. This leads to an interesting question:

Can a service broker actor be domain agnostic?

The way we implement Service broker actor and other architectural pieces such as server groups and dispatchers hold the answer to this question. If the service broker actor is just tasked with routing tasks, we believe then it can be domain independent. In our implementation, service broker actor also computes the amount of substrate used for each printing task. Such details are specific to printing.

The implementation of service broker architecture brought a lot of complexity to the Service Broker module, since it becomes a gateway to routing tasks and quantifying effort needed for tasks. At several places, we had design choices which we hope to elaborate in our future publications. Despite the complexity, the service broker architecture helps in dealing with a variety of demand and equipment configuration without changing simulation schema – effectively eliminating the need of a simulation engineer to rewire the components with the changes in simulation scenario.

There are several commercial and enterprise cloud services by various CAD/CAM, manufacturing vendors where finding services is either through integration with other services or are standalone services linked to a portal. The current prototype and the approach demonstrated here should scale to standalone cloud service and private, public computing cloud/manufacturing cloud deployments.

We are currently developing a model and a vocabulary to describe heterogeneous equipment and reason about them automatically and plan to integrate it to have an end-to-end system cloud service that brings out both the dynamic and static aspects of system design. We also plan to automate the warm-up period detection based on various approaches mentioned in [18]. We also plan to leverage the current prototype for HP’s 3D printing efforts.

Acknowledgments

The efforts described in this paper have come over a long period of time involving efforts by people other than the authors. We acknowledge the contributions of following people from Hewlett-Packard Company: I-Jong Lin, Frank Droggo, Michael Reasoner, Jun Li, Eric Hoarau, Eviatar Halevi, Shay Maoz and Gene McDaniel.

References
