Datenbank-basierte Modellierung und Simulation von Produktionsabläufen als Basis für die Feinplanung

Bernhard Kabelka[1], Felix Breitenecker[1], Wolfgang Stöcher[2], Markus Vorderwinkler[2]
bernhard@kabelka.net, fbreiten@osiris.tuwien.ac.at,
wolfgang.stoecher@profactor.at, markus.vorderwinkler@profactor.at
[1]Technische Universität Wien
Wiedner Hauptstraße 8-10, A-1040 Wien, Austria
[2]PROFACTOR Produktionsforschungs GmbH
Im Stadtgut A2, A-4407 Steyr-Gleink, Austria

Kurzfassung

Heutzutage wird in vielen Bereichen verstärkt der Computer (insbesondere die Simulation) für die Produktionsfeinplanung eingesetzt. Die in diesem Bereich oft geforderten kurzen Antwortzeiten erfordern sowohl eine wohlüberlegte Programmierung als auch eine gute Wahl des Simulators. Darüber hinaus ist die Grundstruktur derartiger Prozesse oft sehr ähnlich, sodass es nahe liegt, ein Modell eines (in gewisser Weise abstrahierten) Produktionsablaufes zu erstellen, das nur noch kleine Änderungen erfahren muss, um an ähnliche Prozesse angepasst zu werden.

In der vorliegenden Arbeit wurde daher eine Simulation einer allgemeinen Produktionsanlage entwickelt, die über die Daten in einer Datenbank konfiguriert wird. Darüber hinaus wurden drei verschiedene Simulatoren auf deren Eignung in diesem Bereich untersucht. Dabei zeigte sich, dass sich für den Einsatz der Simulation für die Feinplanung eines Produktionsprozesses der etwas höhere Aufwand einer Implementierung in einer höheren allgemeinen Programmiersprache durchaus lohnt.

1 Einleitung

Zum Zwecke der Feinplanung einer Werkstättenfertigung muss für jede Maschine eine exakte Reihenfolge der durchzuführenden Arbeitsschritte festgelegt werden. Um nun unterschiedliche Arbeitspläne eines komplexen Produktionsprozesses bewerten zu können (unter Berücksichtigung von Umständen wie produkt-spezifischer Rüstung der Maschinen u.Ä.), bedient man sich heutzutage immer öfter eines Simulationsmodells der betreffenden Produktionsanlage. Hierfür werden zahlreiche Arbeitssequenzen bestimmt, entweder ad hoc mit Hilfe von Prioritätsregeln, oder vorab unter Verwendung eines Plans (engl. schedule). Mit Hilfe des Simulationsmodells können diese Reihenfolgen nun in Bezug auf Schlüsselgrößen wie Durchsatz, Termintreue und Auslastung untersucht werden. Auf diese Weise kann die Güte unterschiedlicher Prioritätsregeln bestimmt bzw. ein Produktionsplan in Bezug auf gewisse Schlüsselkriterien optimiert werden.

Eine Schedule-Optimierung (z.B. mit Hilfe von Simulated Annealing oder jeder anderen Metaheuristik) erfordert also die Berechnung von unzähligen verschiedenen Arbeitsreihenfolgen, die anschließend miteinander verglichen werden müssen. Daher sollte ein einzelner Simulationslauf eine möglichst kurze Laufzeit besitzen.

Mit anderen Worten, beim Einsatz der Simulation im Bereich der Feinplanung sind gründliche Überlegungen betreffend der Wahl des Simulators erforderlich, da eben nicht nur Funktionalität und Benutzerfreundlichkeit des verwendeten Programms, sondern vielmehr Geschwindigkeitsüberlegungen eine entscheidende Rolle spielen.

In weiterer Folge soll nun ein weitgehend abstrahiertes Modell einer allgemeinen Produktionsanlage entworfen werden, das die Verwendung der entwickelten Simulation (mit nur geringfügigen Modifikationen) in vielen ähnlichen Situationen gestatten soll. Dabei soll das erstellte Simulationsprogramm dazu in der Lage sein, einerseits mit Hilfe von Prioritätsregeln einen Zeitplan für die Produktion zu erstellen, und andererseits auch eine durch einen Schedule vorgegebene Abarbeitungsreihenfolge einfach durchzuspielen.

Das Modell wurde in drei verschiedenen Simulatoren implementiert. Die Wahl fiel mit eM-Plant, AnyLogic und C++SIM auf einen Simulator für Produktion und Logistik, einen hybriden General-Purpose-Simulator und eine diskrete Simulationsbibliothek für eine allgemeine Programmiersprache. Diese drei Simulatoren sollen in Bezug auf Funktionalität, Benutzerfreundlichkeit und Laufzeit miteinander verglichen werden.

2 Modellierung

Wie bereits erwähnt, handelt es sich bei den in der vorliegenden Studie untersuchten Produktionsprozessen nicht um Fließband-, sondern um Werkstättenfertigung. Mit anderen Worten, die Reihenfolge der Arbeitsschritte wird nicht durch das Modell-Layout, sondern vielmehr durch eine geordnete Liste von Operationen u.Ä. bestimmt. Da jedoch ein weitgehend abstrahiertes Modell zum Einsatz kommen soll, sollten weder Stamm- noch Betriebsdaten im Modell selbst abgelegt werden. Allerdings sind gerade diese Daten für eine Simulation der betreffenden Produktionsprozesse unerlässlich. Aus diesem Grund werden diese Daten in einer Datenbank abgelegt, und von dort vom Simulationsprogramm eingelesen. Um dabei eine weitgehend redundanzfreie Speicherung der Daten ohne eminente Gefahr von Anomalien zu gewährleisten, wurde, wie in [1] genauer dargelegt, für die benötigte Datenbank ein Datenmodell im Sinne der Datenbanktheorie erstellt.

Die eben geschilderte Vorgehensweise hat außerdem den Vorteil, dass beispielsweise Änderungen im Produktspektrum oder betreffend der Anzahl oder des Typs der verwendeten Maschinen nur in der Datenbank vorzunehmen sind, und weitgehend automatisiert ins Modell übernommen werden können: Dank diesem Zugang kann das Simulationsmodell nämlich – ähnlich wie in [2] – in der Initialisierungsphase weitgehend automatisiert aufgebaut werden, indem gemäß den Daten in der Datenbank die Maschinen erzeugt und konfiguriert werden.

Nach diesem anfänglichen Aufbau können beliebig viele Simulationsläufe durchgeführt werden. Unmittelbar zu Beginn jedes einzelnen Laufes werden auch die Informationen zu den aktuell zu produzierenden Produkten (sowie zum Schedule, sofern vorhanden) von der Datenbank importiert. Nach dem Simulationslauf werden dann die Resultate betreffend Losverspätungen, Durchlaufzeiten u.Ä. wieder in die Datenbank geschrieben. Diese Information kann schließlich von einem externen Optimierungsprogramm verwendet werden, um den Schedule zu verbessern, und anschließend die Simulation erneut zu starten, womit der Kreislauf von neuem beginnt.

Für die Simulation an sich werden die zu produzierenden Güter in Losen durch die Maschinen geschleust, wobei sich ein Los für jeden Arbeitsschritt jeweils bei allen

möglichen Maschinen zur Bearbeitung anmeldet. Dabei wird auch eine maschinen-spezifische Losbewertung berechnet, die der Priorität des Loses entspricht. Sobald eine der Maschinen verfügbar ist, wird aus der Gesamtheit aller an dieser Maschine angemeldeten Lose das „beste“ (gemäß der im Vorfeld berechneten Bewertung) zur Bearbeitung herangezogen.

Um der Forderung zu entsprechen, beide Varianten für die Reihenfolge-Generierung (Prioritätsregeln bzw. Scheduling) zu kombinieren, wurden die Bewertungen so gewählt, dass bei Abarbeitung der Lose nach aufsteigender Bewertung entweder gewisse Prioritätsregeln erfüllt, oder der vorgegebene Plan eingehalten wird.

In ersterem Fall wurden jedoch nur einige Standard-Regeln, wie zum Beispiel FIFO (First In, First Out) oder SPTF (Shortest Processing Time First), implementiert. Eine Erweiterung des Modells zwecks vergleichender Untersuchungen unterschiedlicher Prioritätsregeln (an einem konkreten Fallbeispiel) wären zwar mit nur geringfügigen Modell-Modifikationen möglich, stehen aber noch aus.

Bei der Simulation wurden auch Einflüsse wie produkt-spezifisches Umrüsten, begrenzte Verfügbarkeit oder regelmäßige Wartung der Maschinen in Betracht gezogen.

Das Modell wurde schließlich, wie bereits oben dargelegt, in drei Simulatoren umgesetzt. Die Vorgehensweise war in allen drei Fällen weitgehend dieselbe – das Herz der Simulation bildete eine Maschinenklasse, die eine einzelne Maschine modellierte – sodass in weiterer Folge nur auf die Unterschiede zwischen den einzelnen Simulatoren eingegangen werden soll.

3 Simulatoren

Die Software **eM-Plant 7.0** ist ein objektorientierter Simulator für Produktion und Logistik. Mit ihrer graphischen Benutzeroberfläche sowie einer Vielzahl von vordefinierten Komponenten (wie zum Beispiel Quelle, Senke, Warteschlange, oder Server) erleichtert sie dem Benutzer die Arbeit sehr. Eine weitere Konfiguration der Bausteine bzw. das Entwerfen eigener Bausteine ist mit der Software-eigenen Sprache „SimTalk“ möglich. Leider ist die Sprache nicht so mächtig wie andere objektorientierte Programmiersprachen; so wäre beispielsweise im vorliegenden Fall ein AVL-Baum recht nützlich gewesen, der jedoch nicht zur Verfügung steht. Auch eine Syntaxüberprüfung ist, da SimTalk einerseits eine interpretierte Sprache ist, und andererseits nur wenige Datentypen kennt, nur eingeschränkt möglich.

Dafür konnten – dank der Datenstruktur „TableFile“ und der implementierten ODBC-Schnittstelle – ganze Tabellen aus der Datenbank in die Modelldatei importiert werden, und mussten daher nicht jedes Mal neu eingelesen werden. Auch der umfangreiche Debugger bietet dem Benutzer einigen Komfort.

Der Java-basierte Simulator **AnyLogic 5.2.0** unterstützt sowohl kontinuierliche, diskrete, als auch hybride Simulation. Wie in eM-Plant wird der Modellaufbau durch eine graphische Benutzeroberfläche und vordefinierte Bausteine vereinfacht. Zwar weisen letztere eine etwas geringere Funktionalität auf als jene in eM-Plant, sind jedoch Open-Source, sodass sie problemlos an Benutzerwünsche angepasst werden können, wobei dafür der volle Umfang von Java zur Verfügung steht. Dank der Basis UML-RT sind auch Komponenten wie Statecharts zur Konfiguration der Bausteine verfügbar.

Für die Datenbankanbindung konnte die ODBC-Schnittstelle aus Performancegründen leider nicht verwendet werden, doch es stand ein nativer JDBC-Treiber für MySQL zur Verfügung. Die Verwaltung der Abfrage-Ergebnisse obliegt in diesem Fall dem Benutzer, wobei dafür in diesem Fall auch Datenstrukturen wie AVL-Bäume gewinnbringend eingesetzt werden konnten.

Eine Schwäche des Programms liegt beim Ausführen von mehreren aufeinanderfolgenden Simulationsläufen: Beim Neustart der Simulation werden alle Objekte außer der Wurzel zerstört und erneut erstellt. Das hat zur Folge, dass jedwede Konfiguration dieser Objekte vor jedem einzelnen Lauf wiederholt werden muss, was klarerweise in längeren Laufzeiten resultiert.

Das Paket **C++SIM 1.7.4** ist eine objektorientierte Simulationsbibliothek für diskrete, prozess-orientierte Simulation in C++ und wurde von Mitgliedern des Arjuna-Projektes an der Universität von Newcastle upon Tyne entwickelt.

Es werden Betriebssystem-Threads verwendet, um die einzelnen Prozesse zu simulieren, wobei die Programmierung anscheinend nicht völlig fehlerfrei gelungen ist: Gelegentlich kann es (zumindest bei Verwendung von *Microsoft Visual C++ 6.0*) vorkommen, dass das Programm bei der Beendigung der verwendeten Betriebssystem-Threads aus unerfindlichen Gründen stecken bleibt. Bei Verwendung eines Debug-Programms war der Fehler jedoch nicht reproduzierbar, sodass dessen genaue Ursache ungeklärt bleibt.

Nachdem es sich bei C++SIM nur um eine Simulationsbibliothek für C++ handelt, ist keine graphische Benutzeroberfläche verfügbar. Auch Basisbausteine sind nicht vordefiniert. Der Benutzer muss daher selbst derartige Bausteine entwerfen, was zu Beginn eher aufwändig ist. Einmal erstelle Komponenten können jedoch – dank der Objektorientierung – immer wieder verwendet werden.

Auch die Frage der Datenbankanbindung bleibt dem einzelnen Benutzer überlassen. Im vorliegenden Fall wurde mit MySQL gearbeitet (das in C++ geschrieben ist), und die zugehörige MySQL-Bibliothek benützt, was ohne viel Aufwand den direkten Zugriff auf die Datenbank ermöglichte.

4 Fallstudien

Für Laufzeituntersuchungen wurden zwei Fallstudien herangezogen, bei denen jeweils eine Feinplanung mit möglichst kurzen Antwortzeiten eine entscheidende Rolle spielt. Die zugehörigen Simulationsmodelle wurden jeweils einige tausend Mal ausgeführt.

Die erste Probleminstanz betrifft den Aluminium-Magnesium-Druckguss. Es wurde eine Fabrik mit acht Maschinen, die zur Produktion von 49 verschiedenen Produkten (mit je zwei Arbeitsschritten: Druckguss und Entgraten) verwendet werden, modelliert. Es wurde ein Zeitraum von einem Jahr mit 700 Aufträgen (bei einem durchschnittlichen Volumen von etwa 14500 Stück) betrachtet.

Die zweite Testinstanz stammt aus der Halbleiter-Herstellung und ist Teil jenes Prozesses, der auch in [3] beschrieben wird. Dabei wurde ein Abschnitt der Produktion mit 185 Maschinen herausgegriffen. Die 868 betrachteten Produkte besitzen zwischen vier und 45 Arbeitsschritte. Der Simulationszeitraum erstreckt sich über fünf Monate, und umfasst 816 Aufträge mit einem Gesamtvolumen von 30159 Wafern.

Eine detailliertere Beschreibung der beiden Prozesse ist in [1] zu finden.

Es zeigt sich, dass C++SIM den beiden Simulatoren eM-Plant und AnyLogic in Bezug auf die Laufzeiteffizienz überlegen ist: Bereits in der Initialisierungsphase können letztere nicht mit C++SIM mithalten. Allerdings muss die Initialisierung nur einmal für eine beliebige Anzahl von Simulationsläufen durchgeführt werden, weshalb Laufzeitunterschiede in diesem Bereich nicht überbewertet werden sollten.

Aussagekräftiger ist hier schon die Laufzeit der Simulation selbst, wenn Einlesen von bzw. Schreiben in die Datenbank nicht berücksichtigt wird:

Fallstudie		eM-Plant	AnyLogic	C++SIM
#1	Dispatching	926 ms	209 ms	32 ms
	Scheduling	628 ms	229 ms	30 ms
#2	Dispatching	10910 ms	15630 ms	425 ms
	Scheduling	7030 ms	15280 ms	302 ms

Tabelle 1: Durchschnittliche Dauer der Simulation an sich

Wie Tabelle 1 zeigt, ist im Fall der Simulation selbst die Überlegenheit von C++SIM in Bezug auf die Laufzeit am deutlichsten: Die mit C++SIM erstellte Simulation ist 6.6 Mal bis 50 Mal so schnell wie die Simulation mit eM-Plant bzw. AnyLogic.

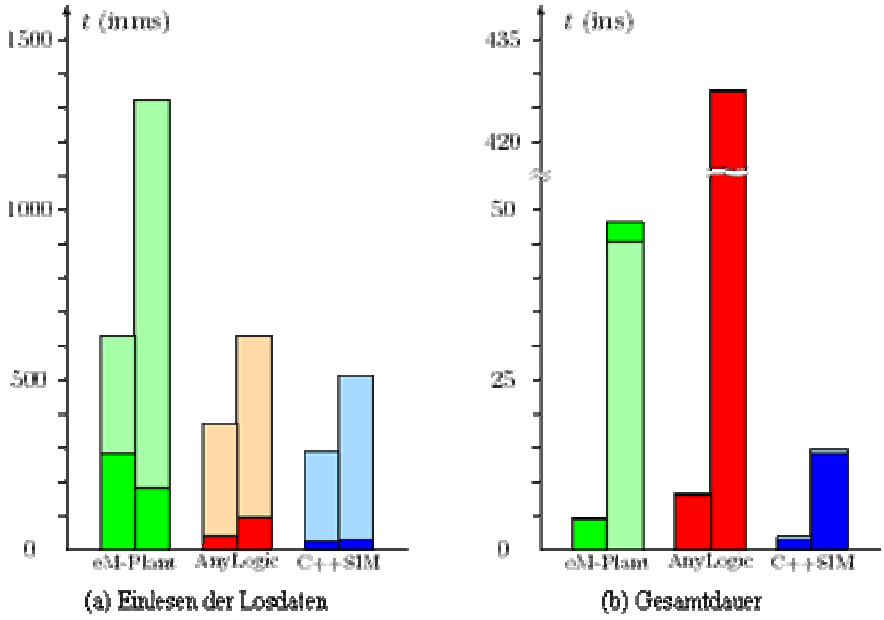

Bild 2: Die Diagramme zeigen die Laufzeit ausgewählter Phasen der Simulation, wobei der linke bzw. rechte Balken jedes Paares mit der Laufzeit der ersten bzw. zweiten Testinstanz korrespondiert. Des weiteren entsprechen die dunkleren Balken der Laufzeit im Fall von Dispatching, die helleren jener im Fall von Scheduling.

In anderen Belangen (zum Beispiel den Datenbankzugriff betreffend; siehe Bild 2(a)) sind die Laufzeitunterschiede zwar weniger gravierend, aber doch bemerkenswert.

Schließlich zeigt die Abbildung in Bild 2(b) die Gesamtdauer eines kompletten Laufes (einschließlich Initialisierung, Einlesen der Daten und Resultatausgabe). Hierbei ist C++SIM eindeutig am schnellsten, doch auch die anderen Simulatoren erreichen meist recht ansprechende Laufzeiten.

5 Fazit

Es wurde ein Modell eines allgemeinen Produktionsprozesses entworfen. Die Implementierung desselben erfolgte mit drei verschiedenen Simulatoren, deren Eignung für eine derartige Simulation ermitteln werden sollte. Bei der Wahl des bestgeeignetsten Simulators gilt es einige unterschiedliche Aspekte zu beachten, und leider ist keine der drei Varianten in allen Belangen tadellos:

Die beiden Simulatoren eM-Plant und AnyLogic sind dank einer graphischen Benutzeroberfläche und vieler vordefinierter Bausteine deutlich benutzerfreundlicher gestaltet als C++SIM. Im Fall von AnyLogic und C++SIM ist wiederum die Vielfalt von Java bzw. C++ begrüßenswert, und führt zu einer hohen Flexibilität. Im Bereich des Debugging sind die drei Werkzeuge ungefähr gleichwertig, sofern für Java bzw. C++ Debug-Programme von Fremdanbietern herangezogen werden. Schließlich hat die Simulation mit C++SIM einen klaren Laufzeitvorteil gegenüber jener mit eM-Plant bzw. AnyLogic.

Wenn es also um die Abbildung eines Produktionsprozesses für die Feinplanung geht, wo

- keine Animation erforderlich ist,
- eine komplexe Logik die Verwendung der graphischen Elemente vieler höherer Simulatoren erschwert oder gar unmöglich macht (und daher ein Ausprogrammieren derselben erforderlich ist),
- und nicht zuletzt Geschwindigkeitsüberlegungen eine entscheidende Rolle spielen,

dann lohnt sich der etwas größere Aufwand einer Implementierung in einer allgemeinen höheren Programmiersprache.

6 Literatur

[1] *Kabelka, B.*: Datenbank-basierte Modellierung und Simulation von Produktionsabläufen als Basis für die Feinplanung. Diplomarbeit, TU Wien, 2005.

[2] *Tauböck, S.; Wartha, C.; Steiner, M.; Pirkner, G.; Breitenecker, F.*: Manufacturing 2: Creation of a Self Adaptive Simulation for Radex Heraklit Industries. 34th Winter Simulation Conference, San Diego, CA (2002), S. 1026–1029.

[3] *Schickmair, M.; Graml, M.; Pichler, C.; Laure, W.*: Simulation-Based Synthesis of Composite Dispatching Rules. Proceedings of the ESS'04 – 16th European Simulation Symposium, 2004