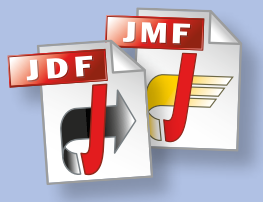
Thomas Hoffmann-Walbeck
Sebastian Riegel

# Der JDF-Workflow

Lehrbuch zur Automatisierung
in der grafischen Industrie




Verlag Beruf und Schule

**Prof. Dr. Thomas Hoffmann-Walbeck**

Mathematiker, Software-Entwickler, Projektmanager. Seit 1998 Professor an der Hochschule der Medien im Studien-gang Druck- und Medientechnologie. Lehrgebiete: Prepress, Angewandte Informatik, JDF-Vernetzung.

**Dipl.-Ing. Sebastian Riegel**

Diplom-Abschluss an der Hochschule der Medien in Stuttgart. Seit 2003 wissenschaft-licher Mitarbeiter im Studiengang Druck- und Medientechnologie, technisch verantwortlich für das CtP-Labor und für die JDF-Integration der Hochschule.

Das Buch **Der JDF-Workflow** führt in die JDF/JMF-Technologie ein. Das „Job Definition Format“ und das „Job Messaging Format“ basieren auf dem XML-Standard und stellen eine der wichtigsten Innovationen der letzten Jahre zur Automatisierung der Print-Produktion dar. Anhand vieler Beispiele aus den Bereichen Auftragsmanagementsysteme, Druckvorstufe, Druck und Druckweiterverarbeitung werden typische Abläufe der grafischen Produktion und ihre Implementierungen in dem JDF-Modell vorgestellt. Auch einige spezielle Belange aus dem Verpackungsdruck werden diskutiert.

Als Leserkreis für dieses Buch ist gedacht an Auszubildende in der grafischen Industrie nach der Zwischenprüfung, an Studierende der Druck- und Medientechnologie, an Praktiker in Druckereien oder bei Herstellern und schließlich auch an Medieninformatiker.

ISBN 978-3-88013-675-5

Der JDF-Workflow

Thomas Hoffmann-Walbeck
Sebastian Riegel

# Der JDF-Workflow

Lehrbuch zur Automatisierung
in der grafischen Industrie

Verlag Beruf und Schule

Thomas Hoffmann-Walbeck, Sebastian Riegel
jdf@hdm-stuttgart.de


Postfach 2008, D-25524 Itzehoe
E-Mail-Adresse: vbus@online.de
www.vbus.de

Druck: AALEXX Druck GmbH, D-30938 Großburgwedel

ISBN 978-3-88013-675-5

# Vorwort

Warum ein Buch über JDF-Workflow? Vor allem, warum JDF? Ist das nicht einfach nur ein Datenformat, das hinter den Kulissen dafür sorgt, dass Maschinen in einer Druckerei die richtigen Daten bekommen? Wen interessiert schon beim Telefonieren, wie im Hintergrund die Signalisierungsprotokolle ablaufen? Und kann man nicht auch Auto fahren ohne die technischen Details der Kolbenprofile zu kennen?

In der Tat sind diese Argumente nicht ganz von der Hand zu weisen. Trotzdem meinen wir, dass die Beschäftigung mit JDF nicht nur für Medieninformatiker interessant sein kann, denn

- das Job Definition Format (JDF) beschreibt die meisten Prozesse in der Druckindustrie und stellt damit ein wichtiges Modell für die grafische Industrie dar,
- die Technologie ist noch recht neu und die Bedienung funktioniert noch nicht selbsterklärend wie ein Telefon oder ein Auto (beim Telefon allerdings immer weniger). Einen JDF-Workflow zu projektieren, einzurichten, zu testen und möglicherweise von Fehlern zu befreien benötigt durchaus gewisse Grundkenntnisse des Formates.

Aber wir beabsichtigen auch nicht, uns nur auf den Niederungen des Datenformates zu bewegen. Wichtiger ist in der Tat der Workflow selber, der mit Hilfe des JDFs aufgebaut werden kann. Deswegen werden wir beispielsweise in vielen Kapiteln zuerst die Prozesse völlig unabhängig vom JDF beschreiben, um erst anschließend die Modellierung im JDF zu erklären. Wir haben versucht, beide Teile zu trennen. So können Sie sich nur mit einem der beiden Dinge auseinander setzen, wenn Sie möchten.

Als Leserkreis für dieses Buch ist gedacht an Auszubildende in der grafischen Industrie nach der Zwischenprüfung, an Studierende der Druck- und Medientechnologie, an Praktiker in Druckereien oder bei Herstellern und schließlich auch an Medieninformatiker.

Wir setzen in diesem Buch keinerlei Kenntnisse über das JDF-Format voraus, erwarten aber, dass die grundlegenden Produktionsprozesse zur Herstellung eines Druckproduktes bekannt sind. Im Glossar haben wir allerdings noch einige dieser Prozesse definiert, um den Lesern das Verständnis gegebenenfalls zu erleichtern.

Natürlich können wir in einem solchen Buch nicht alle Prozessschritte zur Herstellung eines grafischen Produktes bis ins mikroskopische Detail beschreiben. Dafür gibt es spezielle Fachbücher über Bildbearbeitung, Computer-to-Plate, Offsetdruck usw. Wir werden hier zum Teil nur mit Beispielen in die technische Tiefe gehen können. Andererseits möchten wir dieses durchaus tun und uns nicht mit dem Philosophieren über den allgemeinen Sinn von Prozessvernetzung begnügen.

Und noch eine Warnung: Wir wollen die grundsätzlichen Prinzipien von JDF-Workflows erhellen und keine Tips & Tricks zu einzelnen Workflow-Produkten geben. Wir werden nicht einmal die Workflowsysteme der verschiedenen Hersteller groß vorstellen – dafür sind unserer Ansicht nach die Fachzeitschriften zuständig.

Wir unterrichten seit mehreren Jahren Prepress-Workflow an der Hochschule der Medien in Stuttgart, sowohl theoretisch in Vorlesungen als auch praktisch in Übungen. Hierbei haben wir festgestellt, dass es allgemein in dem Workflow-Bereich wenig geeignete Literatur gibt. Auf der einen Seite gibt es natürlich die über 1000-seitige Spezifikation des JDF-Formats [13], die sich aber in erster Linie an Informatiker richtet und ansonsten eher abschreckend ist, auf der anderen Seite werden allgemeinere Beschreibungen des wirtschaftlichen Nutzen einer JDF-Automatisierung veröffentlicht (z.B. in [32] oder [33]). Natürlich gibt es von den Herstellern von JDF-Workflow umfangreiche Handbücher über die Benutzung ihrer Systeme. Herstellerübergreifende Bücher explizit über „Workflow“ sind ziemlich rar (siehe z.B. [16] ) und meist auch eingeschränkt auf einen Teilbereich der Produktion (siehe z.B. [10] und [22]).

Wir sind der festen Überzeugung, dass mit dem JDF-Workflow eine neue Produktionsform in der grafischen Industrie entstanden ist und auch noch weiterentwickelt wird, die für manche faszinierend und für andere auch erschreckend ist. Und so glauben wir, dass sich die nähere Beschäftigung mit diesem Thema auf jeden Fall lohnt. Es gilt also immer noch (oder auch schon wieder) das Zitat aus dem Klimschs Jahrbuch von 1924/25 (Seite 109):

*„Unaufhaltsam und rücksichtslos geht die wirtschaftliche Entwicklung ihren Weg. Das Alte stürzt, und neues Leben sprießt aus den Ruinen. Ein Rückblick zeigt die fast sprunghafte Entwicklung, die im letzten Jahrzehnt die grafische Technik genommen hat“*

In der Einleitung (Kapitel 1) werden wir die allgemeinen Eigenschaften und Erwartungen an einen JDF-Workflow diskutieren. Dabei wird auch kuez auf die Entwicklung eingegangen, die zu diesem Format geführt hat. Wer bereits seit einer gewissen Zeit die Fachartikel über das JDF-Thema verfolgt hat, kann dieses Kapitel getrost überschlagen.

Im zweiten Kapitel möchten wir drei Szenarien von Druckproduktionen beschreiben. Alle drei Druckereien befinden sich auf dem derzeitigen Stand der Technik, jedoch nur zwei setzen JDF-Technologien ein. Wir möchten damit Lesern die Unterschiede verdeutlichen, die damit einhergehen können. Außerdem werden in Abschnitt 4 von diesem Kapitel noch Begriffe wie Workflow, Jobtickets etc. definiert und in Abschnitt 5 allgemeine Merkmale von Workflows vorgestellt.

Das Prozess-Ressourcen-Modell bzw. das Produzent-Konsument-Modell ist Thema in Kapitel 3. Hier werden die Grundzüge und Vorteile dieser Sichtweise mit Beispielen aus den Bereichen Auftragsmanagement, Vorstufe, Druck und Weiterverarbeitung demonstriert.

In Kapitel 5 wird eine Kurzeinführung in XML, die *Extensible Markup Language*, gegeben, soweit sie für das Verständnis von JDF-Code vonnöten ist.

Im darauf folgenden Kapitel 6 werden die wichtigsten JDF-Strukturen vorgestellt. Kapitel 7 beschäftigt sich mit dem Job Messaging Format (JMF), dem „SMS der Druckindustrie“. Es ist ein Datenformat und ein Protokoll zur Kommunikation in einer JDF-Umgebung. Beide bilden die Grundlagen für die Kapitel 8 bis 12, in denen es um Workflow-Details und deren JDF-Entsprechungen aus den Bereichen Auftragsmanagement, Vorstufe, Druck, Weiterverarbeitung und Verpackungsdruck geht.

In Kapitel 13 werden zwei mögliche JDF-Projekte diskutiert, welche die Leser interessieren könnten. Im ersten Abschnitt geht es um die Implementierung von JDF-Workflows in Druckereien. Der zweite Abschnitt gibt eine kurze Einführung in die JAVA-Programmierung von JDF-Applikationen. Hierbei geht es nur darum, eine erste Hilfestellung zu leisten, wie beispielsweise Bibliotheken einzubinden sind.

Am Ende mancher Kapitel haben wir noch Übungen angefügt, die helfen sollen, den Stoff weiter zu durchdringen.

Die JDF-Terminologie ist englisch. Wir haben die Fachbegriffe auch nur dann ins Deutsche übersetzt, wenn es uns sinnvoll

erschien. In diesen Fällen haben wir die englischen Originalbegriffe *kursiv* in Klammern gesetzt. In vielen Fällen haben wir aber auch gleich die englische Notation beibehalten und dann bestenfalls die deutschen Übersetzungen in Klammern angefügt. Außerdem haben wir am Schluss des Buches noch einer englisch-deutsche Übersetzungstabelle der JDF-Prozessnamen angefügt.

Die Autoren möchten sich für die wertvolle Unterstützung beim Verfassen dieses Buches bei den folgenden Personen ganz herzlich bedanken:

Herrn Dieter Adam (MB Bäuerle), Herrn Jan Breithold (HELL Gravure Systems), Herrn Ruben Cagnie (EskoArtwork), Frau Anja Dannhorn (Fujifilm), Herrn Gottfried Grasl (Heidelberger Druckmaschinen), Herrn Stefan Kopec (Druckerei Mack), Frau Ulrike Kurz (MBO), Herrn Bernd Laubengaier (Druckerei Laubengaier), Frau Prof. Dr. Christa Neß (Hochschule der Medien), Herrn Lieven Plettnick (EskoArtwork), Frau Ulrike Seethaler (Heidelberger Druckmaschinen), Herrn Matthias Siegel (MB Bäuerle), Herrn Klaus Stocklossa (MBO).

Über Anregungen, Korrekturen, Bemerkungen oder Ergänzungen zu diesem Buch würden wir uns sehr freuen.

Thomas Hoffmann-Walbeck
Sebastian Riegel
c/o Hochschule der Medien
Nobelstraße 10
D-70569 Stuttgart
0711-89232128 oder 0711-89232115
jdf@hdm-stuttgart.de

# Inhaltsverzeichnis

# 1 Einleitung

Unter einem „JDF-Workflow“ wird im Allgemeinen eine Prozessvernetzung in der grafischen Industrie verstanden, die auf den beiden offenen standardisierten Datenformaten *Job Definition Format* (JDF) und *Job Messaging Format* (JMF) basiert. Erklärtes Ziel der Vernetzung ist die Automatisierung der Prozessschritte durch Integration unterschiedlicher Anwendungen und Systeme. Die zugrunde liegende Idee ist dabei ganz einfach: Es geht nämlich im Wesentlichen nur darum, Informationen über einen Druckjob zusammenzufassen und an die Stellen weiterzugeben, die sie benötigen. Man bedenke z.B. nur einmal, wo überall die Größe des Druckbogens manuell eingetragen werden muss: bei der Kalkulation, bei der digitalen Montage, am Leitstand der Druckmaschine, am Planschneider... Hier lässt sich offenbar Aufwand reduzieren.

Aber ein JDF-Workflow hat noch weitere Ziele, wie wir sehen werden. Es sind hier vor allem Fehlerreduzierung, Zeitersparnis und Kostentransparenz zu nennen.

## 1.1 Entwicklung des JDF

Bereits kurz vor der DRUPA 2000 wurden diese Datenformate von den Initiatoren Heidelberger Druckmaschinen, manroland, Agfa und Adobe angekündigt und auf der Drupa selber dann präsentiert. Noch im gleichen Jahr (September) wurde die Weiterentwicklung dieser Formate der „*International Cooperation for the Integration of Processes in Prepress, Press and Postpress*“ (CIP4) übergeben. Der Verein hat derzeit über 300 Mitglieder, meist Zulieferer, Anwender, Berater und Institute der grafischen Industrie.

Die CIP4-Organisation ist aus dem bereits 1995 gegründeten CIP3-Konsortium hervorgegangen, wobei CIP3 ein Kürzel für „*International Cooperation for Integration of Prepress, Press and Postpress*“ ist. Auch das vom CIP3-Konsortium propagierte „*Print Production Format*“ (PPF) wurde von CIP4 übernommen. Seinen Sitz hat das CIP4-Konsortium in Zürich (Schweiz).

Sowohl das PPF als auch das JDF/JMF sind Schnittstellenformate für Vernetzungslösungen. Beide Formate dienen dem Prinzip der industriellen Produktion von Druckprodukten, das im Gegensatz zur herkömmlichen, eher handwerklich orientierten

Abbildung 1.1
Geschichte des JDFs und CIP4-Logo

1993 Konzeptentwicklung vom PPF
1995 Öffentliche Vorstellung der PPF Version 1.0
Gründung der CIP3-Organisation
1996 PPF Version 2.0
1998 PPF Version 3.0
2000 Öffentliche Vorstellung vom JDF. Gründung des CIP4-Konsortiums
2001 JDF Version 1.0
2002 JDF Version 1.1
2004 JDF Version 1.3
2008 JDF Version 1.4

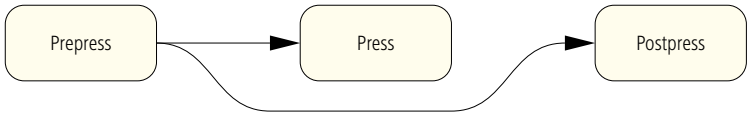


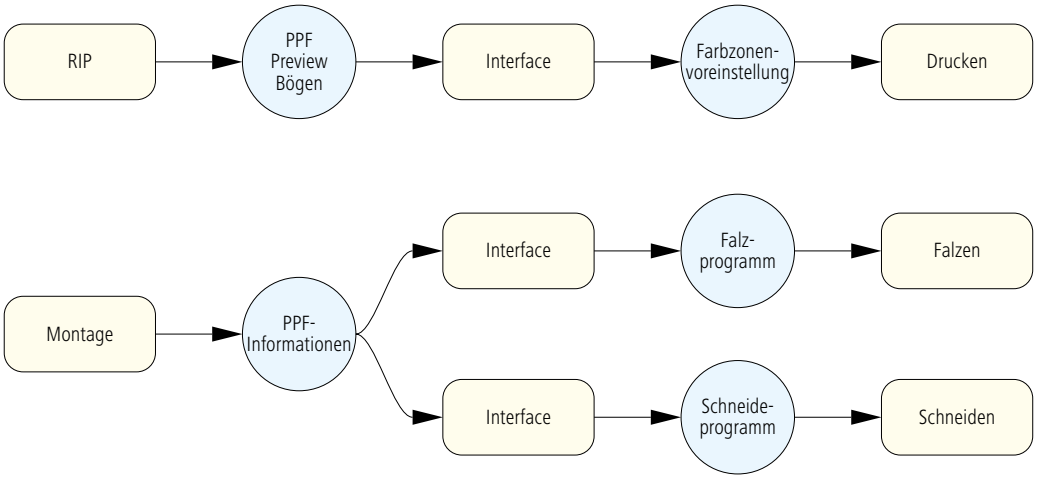

Abbildung 1.2
Verteilung der PPF-Informationen

Produktionsweise gesehen wird. In diesem Zusammenhang fällt häufig das Schlagwort *Computer Integrated Manufacturing* (CIM).

Das PPF, das in der Industrie weithin auch als „CIP3-Format" bekannt ist, ermöglicht den Transport von technischen Daten, meist von der Vorstufe zum Drucksaal oder von der Vorstufe zur Weiterverarbeitung (Abbildung 1.2). Die am weitesten verbreitete Anwendung ist die Speicherung eines Vorschaubildes (*Preview*) eines Druckbogens in einer PPF-Datei, eine Aufgabe, die ein RIP übernimmt, der einem Plattenbelichter vorgeschaltet ist. Dieses Vorschaubild wird dann einer Software übergeben, die daraus Farbzonenvoreinstellungen für eine Offsetdruckmaschine berechnen kann (Abbildung 1.3). Ein weiteres Beispiel für den PPF-Workflow ist die Weitergabe der Schneid- und Falzinformationen eines Bogens von der digitalen Montage an die Weiterverarbeitung. In der Montage werden die entsprechenden Marken gesetzt und anschließend die Informationen in geeigneter Weise in eine PPF-Datei geschrieben. Eine PPF-interpretierende Software kann dann hieraus für einen Planschneider oder für eine Falzmaschine ein Schneide- beziehungsweise Falzprogramm in einem herstellerspezifischen Datenformat generieren. In PPF werden also keine Maschinensteuerungsdaten übergeben, sondern abstraktere Informationen, aus denen Maschinensteuerungsdaten erzeugt werden können.

Abbildung 1.3
detaillierter PPF-Workflow

Zusammenfassend bietet der PPF-Workflow folgende Vorteile



gegenüber einer Produktionsweise ohne PPF-Vernetzung:

- Technische Daten, speziell für Maschinenvoreinstellungen, können über Abteilungsgrenzen hinaus weitergereicht werden.
- Mehrfacherfassung von Daten (wie z.B. Bogengröße) bei unterschiedlichen Aggregaten können entfallen.
- Gewisse Prozessschritte können durch das Weitereichen der technischen Daten sogar komplett wegfallen, wie z.B. der früher übliche Plattenscan zur Herstellung einer Farbzonenvoreinstellung.
- Durch die Offenlegung und Standardisierung des *Print Production Formats* können PPF-fähige Module unterschiedlicher Hersteller miteinander kommunizieren.

Weitere Details zu dem Print Production Format findet man in Abschnitt 4.2.

## 1.2 Hauptmerkmale des Job Definition Formats

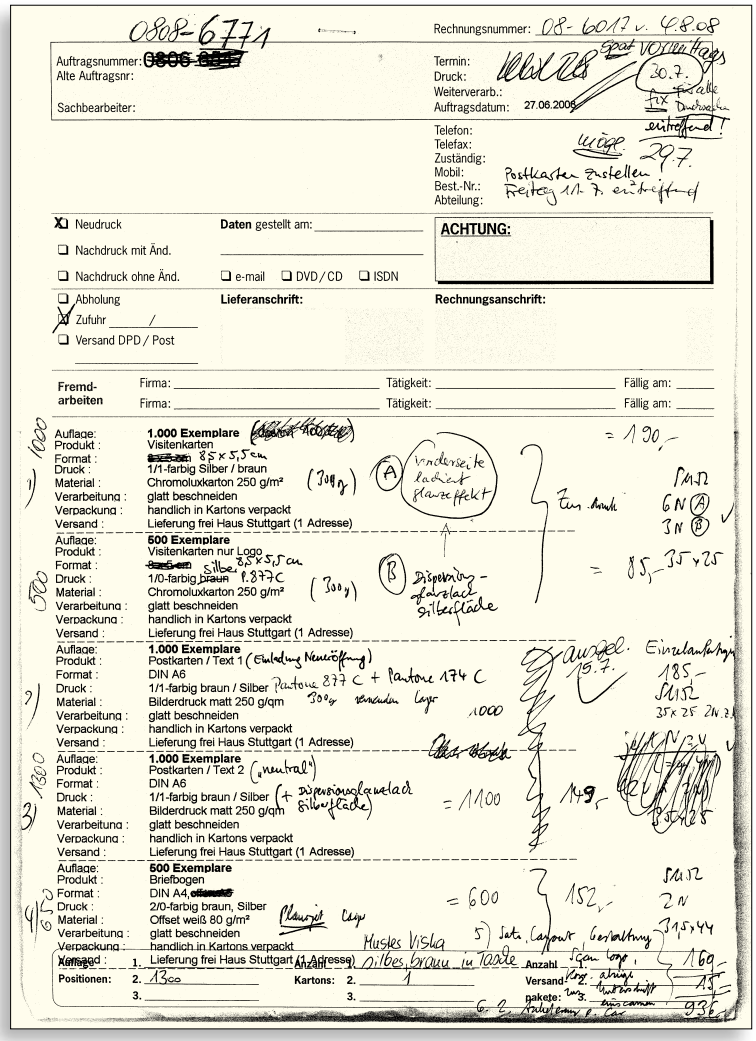
Manchmal wird das JDF als „elektronische Auftragstasche“ bezeichnet (vergleiche Abbildung 1.4). Doch es ist eigentlich viel mehr. Denn es wäre eine „Auftragstasche“, die beispielsweise automatisch Workflows steuert, Maschinen voreinstellt und die Betriebsdaten protokollieren kann.

Das JDF bietet hauptsächlich Unterstützung:

- bei der Weitergabe von Auftragsdaten,
- bei der Weitergabe von Voreinstellungsdaten,
- bei der Betriebs- und Maschinendatenerfassung,
- bei der Disposition und
- bei der Auftragsverfolgung.

Die JDF-Spezifikation umfasst die Funktionalität des PPF, geht aber noch weit darüber hinaus. Folgende zusätzlichen Möglichkeiten sind bei JDF/JM-Workflows gegeben:

- Unterstützung der Schnittstellen zwischen Kalkulationsprogrammen bzw. Management-Informations-Systemen (MIS) und Produktion. Mit JDF lassen sich auch „weichere“ Informationen definieren, wie sie zur Zeit der Produktbeschreibung bereits vorliegen können, so z.B. eine ungefähre Rasterfrequenz. Im weiteren Produktionsablauf müssen natürlich diese Parameter irgendwann eindeutig festgelegt werden.




- Maschinen- und Betriebsdatenerfassung, die sowohl die Jobverfolgung (*„Job Tracking“*) als auch die Nachkalkulation vom jeweiligen Auftrag ermöglichen.
- Ein Protokoll, das Jobs und Produktionsanlagen in einem gewissen Grad auch steuern kann.
- Unterstützung von *eCommerce*-Verbindungen zwischen Druckprodukteinkäufer und Druckproduktehersteller. „Business-Objekte“ wie Angebotsanfragen, Angebote, Auftragsbestätigungen etc. können definiert werden.
- Ansätze für ein *Plug-and-play*-Verfahren zur Integration neuer JDF/JMF-Software in eine entsprechende Produktionsumgebung.

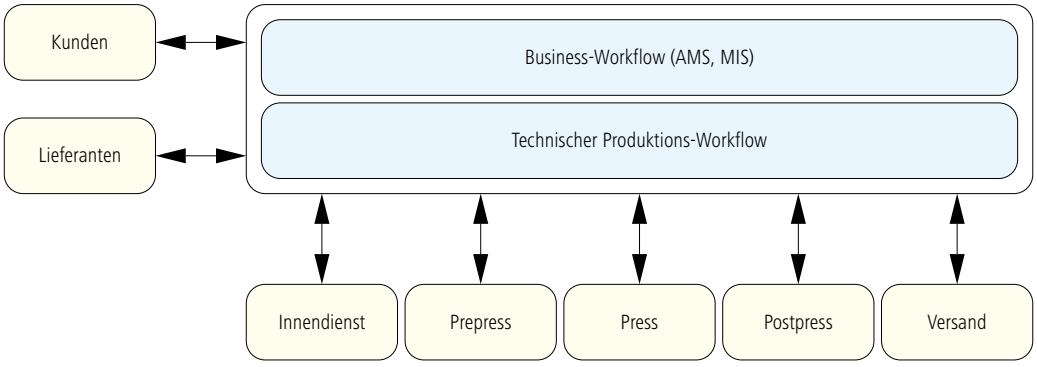
Abbildung 1.4
Eine Auftragstasche, die bei laufender Produktion mit Zusatzinformationen versehen wurde.


Abbildung 1.5
Veränderung eines JDF-Dokumentes, wenn verschiedene „Dienstleister“ an der Bewältigung der Aufgaben beteiligt sind.

- Das Erstellen eines Produktionsprotokolls für die einzelnen Druckjobs.
- Verfahren, die es erlauben, Teilstücke von JDF-Dateien herauszuschneiden und weiterzureichen, sei es an einen Dienstleister, wie z.B. einen Plattenlieferanten, oder auch nur an verschiedene Software-Module innerhalb einer Druckerei. Nach Beenden der Teilaufgabe können dann die veränderten JDF-Teile wieder in die originale JDF-Datei eingefügt werden. (Abb. 1.5)

Die Details zu diesen Punkten werden in späteren Kapiteln ausführlich diskutiert und sind vielleicht auch dann erst richtig ver-

Abbildung 1.6
Kommunikationswege im JDF-Workflow



ständlich. Doch lässt sich vielleicht jetzt schon erkennen, dass der JDF-Workflow im Vergleich zum PPF-Workflow vor allem die Auftragsabwicklung optimiert (Abbildung 1.6).

Doch warum wird dieser Aspekt immer wichtiger? Die Antwort ist einfach: Die Verringerung der durchschnittlichen Auflagenhöhen und die Erhöhung der durchschnittlichen Produktionsgeschwindigkeiten bzw. die Reduktion der Rüstzeiten von Druck- und Weiterverarbeitungsmaschinen lassen die Anzahl der zu bearbeitenden Jobs pro Stunde im Betrieb steigen. Damit fallen die Kosten, die unabhängig von der Auflagenhöhe sind, immer mehr ins Gewicht. Und das sind nun einmal vornehmlich die des Auftragshandlings und der Vorstufenarbeiten.

## 1.3 Implementierungen von JDF-Workflows

JDF ist nur ein Datenformat und kein Workflow. Die Spezifikation enthält keinerlei Anleitungen für Workflow-Hersteller, wie sie ihre Module entwerfen sollen. Analog steht auch nicht in der PDF-Spezifikation von Adobe geschrieben, welche Funktionen ein Layout-Programm haben soll, das in der Regel PDF exportieren und manchmal auch importieren kann.

Trotzdem gibt es einige allgemein gültige Dinge, die für den JDF-Workflow gelten.

Die JDF-Daten wandern nicht frei von einer Software zur nächsten oder von einer Maschine zur nächsten. Das wäre zwar theoretisch denkbar, vermutlich aber ein Albtraum für Software-Entwickler und Anwender gleichermaßen. Stattdessen gibt es eine zentrale Stelle mit einer zentralen Datenablage / Datenbank, bei der die JDF-Daten für die einzelnen Jobs gesammelt werden. Die Zentrale kann dem Management-Informations-System (MIS) zugeordnet sein oder auch einem übergreifenden Produktionssystem.

JDF/JMF-Implementierungen sind nicht flächendeckend im Produktionsprozess von Print-Produkten vertreten, zumindest nicht zurzeit. Häufig sind einige Produktionsmaschinen, vor allem aus der Weiterverarbeitung, überhaupt nicht an das JDF/JMF-Netz angeschlossen, oder aber Informationen werden auch über andere Protokolle weitergegeben, sei es über das PPF oder über herstellerspezifische. Die klassischen Workflowsysteme für Computer-to-Plate-Anlagen waren in vielen Fällen die Keimzellen von allgemeineren JDF-Workflows. Dort findet man dann

auch häufig die Vernetzung zum MIS einerseits und zum Drucksaal andererseits. Allerdings gilt auch hier die Einschränkung, dass manche Informationen am JDF/JMF vorbei geschleust werden. In der Layoutabteilung einer Werbeagentur wird man meist vergeblich nach JDF-basierenden Arbeitsvorgängen suchen.

Die JDF-Spezifikation definiert nur die möglichen Inhalte, die zwischen den Workflow-Komponenten ausgetauscht werden können. Sie sagt aber nicht, welche Komponenten welche Information zur Verfügung stellen und wer sie dann auswerten muss. Es ist ein bisschen so, als ob zwar vereinbart wäre, dass Daten über Flughöhe, Flug- und Sinkfluggeschwindigkeit zwischen Fluglotsen und Piloten ausgetauscht werden können, nicht jedoch, wer dafür verantwortlich ist, wenn überhaupt jemand. Das würde sicherlich nicht lange gut gehen. Und so gibt es auch für den JDF-Workflow zusätzlich Vereinbarungen, welche die Kommunikation unterschiedlicher Modulklassen betreffen, wie z.B. die Schnittstelle zwischen MIS und Druckvorstufe oder zwischen MIS und Druckmaschine. Das CIP4-Konsortium stellt hierfür einen ganzen Satz unterschiedlicher Papiere bereit [12], die unter dem Sammelbegriff *Interoperability Conformance Specifications* (ICS) laufen. Näheres hierzu findet man in Abschnitt 6.7.

Es hat einige Jahre gedauert, bis nach der Veröffentlichung der JDF-Spezifikation 1.0 im Jahr 2001 die ersten JDF-kompatiblen Applikationen auf den Markt kamen. Inzwischen (seit 2008) gibt es die Version 1.4 der JDF-Spezifikation, aber vor allem auch sehr viele JDF-kompatible Anwendungen. Das CIP4-Konsortium veröffentlicht regelmäßig einen Katalog über JDF-Applikationen und Dienstleistungen unter dem Titel *„The JDF Marketplace“* [14]. In dem im Juni 2008 herausgegebenen Katalog waren mehrere hundert Produkte und Dienstleistungen gelistet.

Außerdem kann JDF-Software zertifiziert werden, das heißt, sie wird auf die ICS-Anforderungen hin überprüft. Eine Liste der zertifizierten Produkte findet man ebenfalls auf der Website der CIP4-Organisation.

Doch weder die schlichte Existenz der ICS-Papiere noch die Zertifizierungen lassen den Schluss zu, dass man nun wahllos Software-Module unterschiedlicher Hersteller blind und ohne zu testen zusammenstöpseln könne. Die Tücken liegen sehr im Detail. Die ICS geben immer nur Mindeststandards vor, die für spezielle Druckprodukte nicht unbedingt ausreichen.

Und deswegen ist der Aufbau oder die Erweiterung einer JDF-

Integration in einer Druckerei immer ein Projekt, bei dem Hersteller und Anwender im Vorfeld noch viele Dinge klären und testen müssen. Checkpunkte und Tipps zur praktischen Umsetzung eines solchen Projekts findet man in [35], eine etwas allgemeinere Diskussion in Kapitel 13.

---

**Übung:**

- Machen Sie sich mit den offiziellen Web-Seiten der CIP4-Organisation vertraut (www.cip4.org). Lesen Sie dabei insbesondere die *Introduction* der aktuellen Spezifikation [13].

# 2 Grundlagen Workflow

In diesem Kapitel sollen zwei recht entgegengesetzte Dinge behandelt werden. Zunächst werden in den ersten Abschnitten die Arbeitsweisen von drei Druckereien vorgestellt, die wir X, Y und Z nennen. Die Firma X hat zwar einen Workflow eingerichtet, der völlig up-to-date ist, jedoch keine speziellen JDF-Merkmale enthält. Die Firma Y hingegen hat einige Komponenten implementiert, die JDF nutzen. Die Firma Z ist weitgehend JDF/JMF-vernetzt.

Nach diesen eher recht konkreten Beschreibungen geht es umso abstrakter weiter. In Abschnitt 2.4 werden grundlegende Begriffe wie Workflow, Workflow-Managementsystem und Jobticket definiert. Ein paar allgemeine Eigenschaften von Workflows möchten wir dann in 2.5 behandeln, wobei wir dort zum Schluss noch eine Workflow-Klassifikation vorstellen, die wieder konkreter auf Druckereien bezogen ist.

## 2.1 Beispielfirma ohne JDF-Workflow

Der vollstufige Betrieb X hat nur 12 Mitarbeiter. In der Vorstufenabteilung ist seit mehreren Jahren ein modernes Workflow-Managementsystem (WMS) installiert mit einem Formproofer und einem Plattenbelichter als Ausgabemöglichkeit. Das WMS kann auch PPF-Daten für die Farbzonenvoreinstellung erstellen, diese Funktionalität wird derzeit aber noch nicht genutzt. Im Drucksaal wird an drei Offsetmaschinen gearbeitet, einer Fünf-, einer Vier- und einer Zweifarbenmaschine im Format bis zu 50x70 cm. Außerdem gibt es noch eine Digitaldruckmaschine mit Inline-Weiterverarbeitung. In der Druckweiterverarbeitung befinden sich neben Planschneider und Falzmaschine auch ein Buchdruckzylinder, auf dem gestanzt und geprägt wird. Circa 50 Aufträge werden pro Woche bearbeitet, wobei viele Aufträge aus mehreren Teilprodukten bestehen.

Angebotsanfragen kommen meist über E-Mail oder Telefon, manchmal noch über Fax ins Haus. Ein/e Sachbearbeiter/in gibt die wichtigsten Angaben zu den Anfragen in ein Auftrags-Managementsystem (AMS) ein. Die Kalkulation übernimmt häufig der Inhaber selbst, wobei er nur eine Preistabelle und einen Taschenrechner benutzt. In etwa einem Drittel aller Fälle fehlen Angaben, die zur Kalkulation notwendig sind, und es müssen Rückfragen bei den Anfragenden gestellt werden. Die Anzahl

der Angebotsvarianten, die vom potentiellen Auftraggeber gefordert werden, hat in den letzten Jahren stark zugenommen. Vor allem Werbeagenturen wünschen für ihren Auftrag häufig mehrere leicht modifizierte Angebote.

Das Verhältnis zwischen Auftragserteilung und Angebotserstellung liegt unter 10 %. Mit der Auftragserteilung werden meist auch gleich die Druckdaten als Mail-Anhang mitgeschickt, davon 80 % als PDF-Datei, der Rest als offene Dateien (InDesign, QuarkXPress, OfficeSuite). Die Sachbearbeiter bestellen dann das Papier, bestimmen das Ausschießschema, teilen den Job einer Druckmaschine zu und drucken aus dem Auftrags-Managementsystem heraus eine Lauftasche für die Produktion aus (siehe Abbildung 1.6). Bei Aufträgen über 500 € schicken sie auch noch eine Auftragsbestätigung an den Kunden.

Die Vorstufenabteilung der Firma X untersucht dann mit einem Preflight-Programm die angelieferten Dateien, wobei etwa zwei Drittel fehlerhaft sind und korrigiert werden müssen. Alle Fehler, die innerhalb 10 Minuten korrigiert werden können, werden ohne Rücksprache mit den Auftraggebern und auch ohne Berechnung sofort behoben. Nach der digitalen Montage wird ein Formproof ausgegeben, der per Post oder mit eigenem Fahrdienst zum Kunden gebracht wird. Erst wenn dieser den Plot abgezeichnet hat, werden die Platten ausgegeben. Nur wenn der Auftrag sehr eilig und der Kunde seit Jahren bekannt ist, wird anstatt eines Plots ein Voransichtsbild des Druckbogens im PDF-Format erstellt und dem Kunden per Mail zur Kontrolle zugeschickt. Wenn der Kunde explizit einen farbverbindlichen Proof verlangt, wird dieser bei einer anderen Firma erstellt.

Die Platten gehen dann zum Drucksaal. Als Plantafel dient ein großer Wandkalender, in dem die Endtermine der Aufträge eingetragen sind. Leider klappt das nicht immer, da manchmal die Sachbearbeiter vergessen, ihre Termine auf den in der Produktion aufgehängten Kalender zu übertragen.

Während der Produktion gibt es häufig noch Änderungen. Wenn der Kunde anruft, um noch eine Änderung durchzugeben, sucht der Sachbearbeiter die Lauftasche im Haus und trägt dann die Änderung dort ein.

Die Fakturierung und das Mahnwesen werden von dem eingangs erwähnten Auftrags-Managementsystem unterstützt. Auch gibt es eine Online-Schnittstelle zu einem Steuerbüro.

Eine Nachkalkulation der Aufträge findet nur sporadisch und

dann manuell statt. Auch die Tageszettel werden nur periodisch ausgefüllt, um die interne Preistabelle für die Kalkulation von Zeit zu Zeit zu überprüfen.

Die Auftragsabwicklung für den Offsetdruck und für den Digitaldruck ist übrigens identisch.

Der Inhaber meint, dass in seiner Firma auf der einen Seite hochproduktive und automatisierte Produktionsmittel eingesetzt würden, auf der anderen Seite aber die Auftragsabwicklung nicht denselben Automatisierungsstand erreicht habe.

In der Tat scheint uns das eine sehr typische Situation in kleineren Druckerein in Deutschland zu sein.

## 2.2 Beispielfirma mit teilweiser JDF-Vernetzung

Der Druckdienstleister Y druckt auf seinen zwei großen Rollendruckmaschinen vornehmlich Kataloge, Magazine und auch Zeitschriftenbeilagen und auf seinen Bogenoffsetmaschinen im IIIB-Format meist Umschläge für diese Produkte. Angebotsanfragen erhält die Firma entweder von Bestandskunden oder von Außendienstmitarbeitern, die bei Neukunden akquirieren. Für die Außendienstler gibt es hierzu ein papierbasiertes Anfrageformular, in dem Angaben über Seitenumfang, Auflage, Papier, Farbigkeit und so weiter eingetragen werden können. Diese Anfragen werden vom Innendienst in einem Auftrags-Managementsystem (AMS) erfasst; bei Neukunden werden auch die Kundenstammdaten angelegt. Kalkuliert wird ebenfalls mit dem AMS. Die Angebote werden an die potentiellen Auftraggeber versendet, die Außendienstmitarbeiter verfolgen das Angebot, führen ggf. Nachverhandlungen und stellen Ergänzungsangebote aus.

Bei Auftragerteilung werden aus dem AMS heraus Informationen wie benötigte Druckkapazität, Dateneingangstermin und Liefertermin für die Abteilung „Produktionsplanung und -steuerung“ (PPS) ausgedruckt. Die Mitarbeiter dort planen den Job ein und stecken entsprechende Karten in die Plantafel. Die Planungsdaten werden nicht mehr zurück an das AMS gegeben, so dass beispielsweise Terminüberschreitungen nicht automatisch an das AMS gemeldet werden.

Die Papierbestellung wird nach Abstimmung mit dem Papierein-

kauf ebenfalls vom Innendienst aus angestoßen. Die Bestellung geht dann automatisiert per E-Mail an den Papierlieferanten.

Natürlich wird über das AMS dem Kunden parallel eine Auftragsbestätigung zugeschickt.

Der Bogenplan (Falz- und Druckbogen sowie die Paginierung) wird im AMS spezifiziert. Auch werden dort Produktionsdetails eingetragen, die eigentlich für die Kalkulation nicht notwendig sind, wie beispielsweise die Angaben zum Raster. Das AMS erzeugt dann pro Job eine elektronische Auftragstasche als HTML-Dokument, das unternehmensweit allen als Information zur Verfügung steht. Für die Druckvorstufe wird separat auch ein Papierdokument ausgedruckt, in dem die Vorgänge aufgelistet sind und nach Durchführung von den Mitarbeitern abgezeichnet werden. Informationen werden mittels JDF an das Prepress-Workflowsystem übergeben, so dass viele der im AMS eingegebenen Werte automatisch übernommen werden können. Von dort werden auch Daten an die Druckmaschinen übergeben, hauptsächlich Farbzonenvoreinstellungen im PPF-Format (vergleiche Abschnitt 4.2), aber auch Voransichtsbilder für den farbverbindlichen Softproof, der zur Farbabstimmung verwendet wird.

Die Druckfreigabe durch den Kunden erfolgt entweder über einen klassischen Hardcopy-Proof oder aber auch – vor allem bei weniger umfangreichen Produkten – über ein Internetportal. Dort kann der Kunde des Druckdienstleisters seine Korrekturwünsche oder seine Freigaben vornehmen, die direkt in das Workflowsystem der Druckvorstufe weitergereicht werden.

Die Betriebsdatenerfassung (BDE) erfolgt über BDF-Terminals, in denen die Mitarbeiter Arbeitszeiten, Kostenstellen, Rüstzeiten und dergleichen eingeben. Die Druckmaschinen geben Statusinformationen an ein vernetztes Maschinenautomatisierungssystem, das allerdings keine Verbindung zurück an das AMS hat. Hier müssen Eckwerte manuell übertragen werden. Das AMS ermittelt dann die Nachkalkulation, also die Abweichungen zwischen Soll der vorkalkulierten Kosten und dem Ist-Zustand.

Dieses Beispiel macht zwei Dinge deutlich: Einerseits zeigt es die Verbesserungen der Produktion, die durch ein AMS und entsprechende JDF-Schnittstellen erreicht werden. Andererseits zeigt es auch die typische Zwickmühle von JDF: Viele Schnittstellen, wie die Online-Druckfreigabe oder die zwischen AMS und Produktionsplanung, werden teilweise mit proprietären Datenformaten realisiert oder werden noch manuell bedient.

## 2.3 Firma mit weitreichender JDF/JMF-Vernetzung

Der vollstufige Betrieb Z mit seinen 150 Mitarbeitern ist ein typischer Universaldrucker, der Werksatz (Bücher, Broschuren, Magazine), aber auch Geschäftspapiere und Visitenkarten druckt. Die Aufträge erhält er üblicherweise über E-Mail und Fax. Die Kalkulation erfolgt in einem MIS. Dieses hat eine Online-Anbindung zum Papierlieferanten, und jede Nacht werden die tagesaktuellen Preise automatisch dem MIS überspielt. Auch die Papierbestellung geht nach Überprüfung der eigenen Bestände im Hochregallager direkt über das Internet an den Papierlieferanten. Die Schnittstelle wird über ein herstellerspezifisches Nachrichtenprotokoll (XML) abgewickelt.

Wenn aus dem Angebot ein Auftrag wird, wird bei der Druckerei Z bei jedem Aufruf direkt aus der Datenbank eine HTML-Auftragstasche generiert, die unternehmensweit jedem zur Verfügung steht.

Das MIS generiert pro Auftrag eine JDF-Datei, die an einen Workflow-Server in die Produktion weitergegeben wird. Dort stehen dann alle relevanten Produktionsdaten zur Verfügung. Mit Hilfe dieser JDF-Daten wird dann einerseits die elektronische Produktionsplanung vorgenommen, andererseits werden die Jobs in der Vorstufe bearbeitet. Letzteres heißt, dass das Ausschießschema und weitere Produktionsdaten von der JDF-Datei übernommen und gegebenenfalls aber auch noch modifiziert werden können.

Die Kunden laden vielfach die Druckdaten über das Internet auf einen Server der Druckerei hoch. Das Produktionssystem der Druckerei erhält die Nachricht, dass Daten eingetroffen sind und verarbeitet sie. Die Resultate der Datenprüfung werden einerseits in der zugehörigen JDF-Datei protokolliert und andererseits dem Kunden zugestellt.

Vier der sechs Druckmaschinen erhalten über JDF/JMF die notwendigen Daten wie Bogengröße, Farbzonenvoreinstellung und Farbigkeit. Umgekehrt schicken diese auch die Gerätezustanddaten und Informationen über die Aufträge zurück an den Workflow-Server. Zwei weitere Druckmaschinen älteren Baujahrs sind offline über Terminals angebunden. Der Workflow-Server sendet seinerseits einen Teil der Betriebs- und Maschinendaten, die er von der Vorstufe und den Druckmaschinen erhält, zu-

rück an das Auftrags-Managementsystem, so beispielsweise den Druckplattenverbrauch oder auch bestimmte auftragsbezogene „Meilensteine“ wie die Fertigstellung der Druckplatten oder das Beenden des Druckvorgangs.

Die Weiterverarbeitungsmaschinen sind zurzeit noch nicht direkt vernetzt – die BDE erfolgt ausschließlich über BDE-Terminals.

Nach Produktionsabschluss druckt das AMS einen Lieferschein aus. Eine Datenanbindung zu Logistikunternehmen wurde jedoch nicht realisiert, da der Versand durch zu viele unterschiedliche Firmen erfolgt; häufig auch über kleine lokale Speditionen, die keine Systemanbindung zur Verfügung stellen.

Auch bei dieser Firma ist deutlich, dass, trotz des sehr hohen Vernetzungsgrades viele Bereiche noch nicht über JDF/JMF integriert sind. Dazu zählt beispielsweise die Weiterverarbeitung, bei der es zwar schon seit langem herstellerspezifische Workflow-Lösungen gibt. Bedingt durch die in diesem Bereich üblichen langen Investitionszyklen, braucht es viele Jahre, bis neue Technologien eingesetzt werden.

## 2.4 Definitionen

Ein Workflow ist ein „Arbeitsablauf“ oder „Arbeitsfluss“ und ein Workflow-Managementsystem ist ein System zur Verwaltung eben dieses Arbeitsablaufs. Und das Wasser, das den Arbeitsfluss im Jahr 1555 angetrieben hat, wird heute durch ein Job-

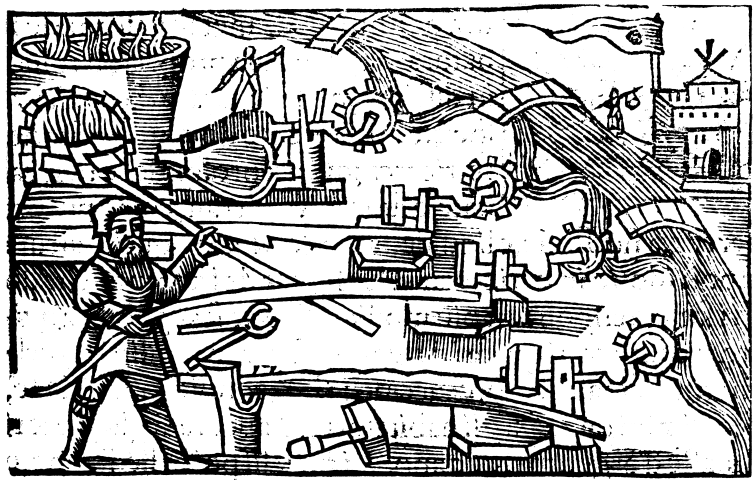


Abbildung 2.1
Olaus Magnus
Historia de gentibus septentrionalibus, 1555
Forges de Dalécarlie
Bibliothèque Sainte-Geneviève

ticket ersetzt (siehe Abbildung. 2.1). Im Grunde ist das schon alles, dennoch wollen wir die Begriffe noch ein klein wenig präziser erläutern, ohne uns allerdings in den unendlichen Weiten der Begriffsdefinition zu verlieren.

Fangen wir mit dem Begriff *Computer Supported Cooperative Work* (CSCW) an. Dabei handelt es sich um das interdisziplinäre Forschungsgebiet, dass sich mit Gruppenarbeit befasst, insbesondere mit den hierfür benötigten Kommunikationstechnologien. Die verschiedenen Sichtweisen bei CSCW werden zum Beispiel in [41] ausführlich wiedergegeben. Eine *Groupware* ist eine CSCW-Applikation, also eine Soft- und Hardwarelösung zu CSCW. *Groupware*-Produkte gibt es viele und von unterschiedlicher Art. Ein E-Mail-System wäre als ein wichtiges Beispiel zu nennen, aber auch Konferenzsysteme, Planungssysteme, Gruppeneditoren, Informationssysteme.

Als *Workflow* verstehen wir eine Folge von definierten Arbeitsschritten, wobei die Aktivitätsfolgen durch Ereignisse ausgelöst, gesteuert und beendet werden. Ein Workflow muss zunächst nicht computerunterstützt sein. „Kochen“ wäre zum Beispiel auch ein Workflow, so wie das Drucken an einer Druckmaschine: Grundrüsten, Plattenwechsel, Einrichten, Bogen ziehen, Passer und Farbe überprüfen, OK-Bogen definieren, Fortdruck überwachen und so weiter.

Ein *Workflow-Management* umfasst die Aufgabe zur Definition (Modellierung), Verwaltung, Ausführung, Steuerung und manchmal auch die Simulation von Workflows. Um bei den eben genannten Beispielen zu bleiben: Die Organisation einer Großküche oder auch die Disposition der Druckjobs mit einer Plantafel könnte man als Workflow-Management bezeichnen.

Schließlich ist ein *Workflow-Managementsystem* (WMS) eine *Groupware* zur Unterstützung eines Workflow Managements. Damit ist also ein WMS eine rechner-basierende Applikation, mit deren Hilfe man Folgen von Arbeitsschritten definiert, verwaltet und steuert.

Wen dieses Thematik genauer interessiert, sei auf [31] und besonders auf die Web-Seiten der *Workflow Management Coalition* (WfMC) [43] verwiesen. Deren Definition von Workflow und Workflow-Managementsystem möchten wir nicht verschweigen:

- „*Workflow*: The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action,

according to a set of procedural rules."
(Die Automatisierung eines Geschäftsprozesses, insgesamt oder partiell, wobei Dokumente, Informationen oder Aufgaben von einem Teilnehmer zum nächsten weitergeleitet werden, um weitere Aktivitäten auszulösen, die durch ein mittels Regeln festgelegtes Verfahren ablaufen.)

- „*Workflow-Managementsystem*: A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications."
(Ein System, das die Ausführung von Workflows mittels Software definiert, erzeugt und verwaltet, wobei ein oder mehrere Software-Module einbezogen sind. Diese können die Prozessdefinitionen interpretieren, mit den Workflow-Beteiligten interagieren und, wenn nötig, IT-Tools und Anwendung aufrufen.)

So ist der Begriff „JDF-Workflow" eigentlich nicht ganz korrekt und man sollte stattdessen von einem „JDF-basierenden Workflow-Managementsystem" reden. Wegen der Langatmigkeit dieser Bezeichnung wird aber allgemein von „JDF-Workflow" gesprochen und wir schließen uns dem an.

*Jobtickets* können nun als elektronische Informationseinheiten zur Unterstützung von Workflow-Managementsystemen in der grafischen Industrie definiert werden. Ein Jobticket kann zum Beispiel eine Arbeitsanweisung oder ein Parametersatz zur Steuerung des Druckauftrags sein. Insbesondere können natürlich Jobtickets mit dem Job Definition Format geschrieben werden. Mit anderen Worten: Das JDF ist ein Beispiel für ein Jobticket-Format.

Ein WMS besteht aus mehreren *Workflow-Engines*, auch *Jobticket-Prozessoren* genannt. Das sind Softwaremodule (wie eine *Trapping-Engine*, ein *Color Transformation Modul* oder die Leitstand-Software einer Druckmaschine), die Jobtickets erzeugen, interpretieren und ausführen können.

Jobtickets stellen in der Druckvorstufe eine Form von „Metadaten" dar, also Daten, die Informationen über andere Daten enthalten. Die Metadaten stehen im Gegensatz und gleichzeitig im Zusammenhang mit den *Content*-Daten. Letztere sind die Druckdaten, also offene Applikationsdaten von InDesign, Quark

usw. oder die geschlossenen Austauschformate wie PDF. Metadaten lassen sich noch einmal unterteilen in „Objektbeschreibungen“ und „Anweisungen“. Eine Objektbeschreibung ist beispielsweise die Aussage, dass ein Bild eine Auflösung von 400 ppi hat. Eine Anweisung hingegen wäre, dass dieses Bild auf 300 ppi heruntergerechnet werden soll. Metadaten können zusammen mit den Content-Daten in einer Datei vorliegen (siehe XMP in Abschnitt 4.1) oder separat wie bei JDF. Wenn die Metadaten außerhalb der Content-Daten liegen, müssen die Content-Daten referenziert werden, das heißt, es wird ein Bezug zum Beispiel über den Dateinamen hergestellt.

Auch wenn Metadaten und Content-Daten getrennt vorliegen, wird die Grenze zwischen beiden häufig verwischt. Schon allein im Dateinamen der Content-Daten liegt eine Meta-Information, die auch durchaus von WMSen zur Workflow-Steuerung eingesetzt wird.

## 2.5 Workflow-Klassifikation

Nachdem nun der Begriff *Workflow-Managementsystem* genauer definiert worden ist, wollen wir ihn jetzt für die grafische Industrie in drei Stufen einteilen:

- WMS innerhalb einer Abteilung,
- abteilungsübergreifendes WMS innerhalb eines Betriebes,
- Betriebsübergreifende WMS mit webbasierender Anbindung („Online-Portale“) zu Kunden und/oder Lieferanten.

Die abteilungsübergreifenden Workflow-Managementsysteme werden aus pragmatischen Gesichtpunkten häufig noch einmal in

- WMS mit MIS-Anbindung
- WMS ohne MIS-Anbindung

unterteilt, denn in der Tat spielt die MIS-Anbindung eine entscheidende Rolle. Man spricht dann auch von einer „Workflow-Integration“.

Es ist selbstverständlich, dass die beschriebenen Kategorien in der Realität nicht immer so klar voneinander abzugrenzen sind. Ein WMS bindet meist nur einen Teil aller Aktivitäten ein, das heißt neben dem WMS innerhalb einer Abteilung gibt es in dieser Abteilung Aktivitäten, die nicht vom WMS unterstützt werden.

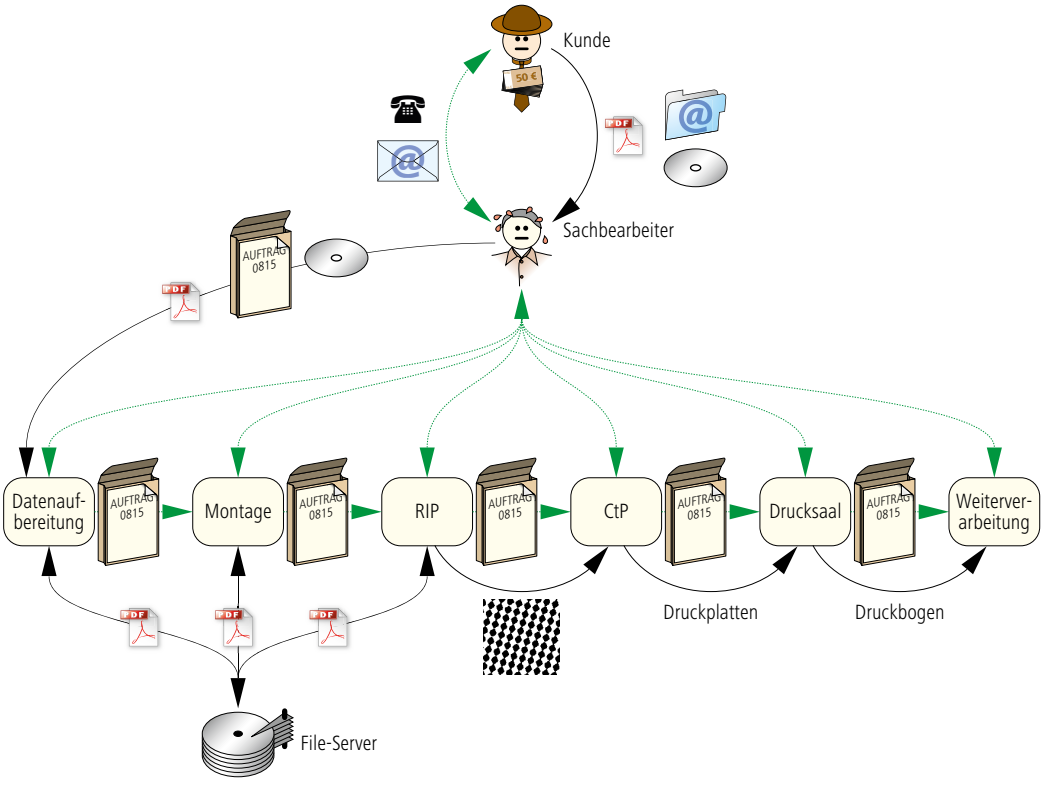



Abbildung 2.2
Modell eines klassischen Workflows in der Druckvorstufe

Oder es gibt bei einem abteilungsübergreifenden WMS ganze Abteilungen, die nicht in das WMS integriert sind. Es stellt sich also immer auch die Frage der WMS-Reichweite. Ein RIP, das auch Daten zur Berechnung der Farbzonenvoreinstellung der Druckmaschine liefert, stellt bereits ein abteilungsübergreifendes WMS dar – genauso eine Druckerei, die mehrere Abteilungen über JDF integriert hat!

Im Folgenden sollen die drei Kategorien nun näher erläutert werden.

Der klassische Workflow ist in der Abbildung 2.2 skizziert. Der Kunde bzw. der Auftraggeber kommuniziert einerseits Metadaten (grüner Pfeil) und andererseits Content-Daten (schwarzer Pfeil) an das Auftragsmanagement einer Druckerei. Metadaten werden häufig telefonisch oder per E-Mail an die Druckerei geschickt, Content-Daten über Internet, als Mail-Anhang oder vereinzelt noch auf einer CD/DVD. Nachdem der Sachbearbeiter den Auftrag in ein Auftrags-Managementsystem eingegeben und eine Auftragstasche ausgedruckt hat, gibt er die Daten zur Datenprüfung und Datenaufbereitung an die Produktion weiter.

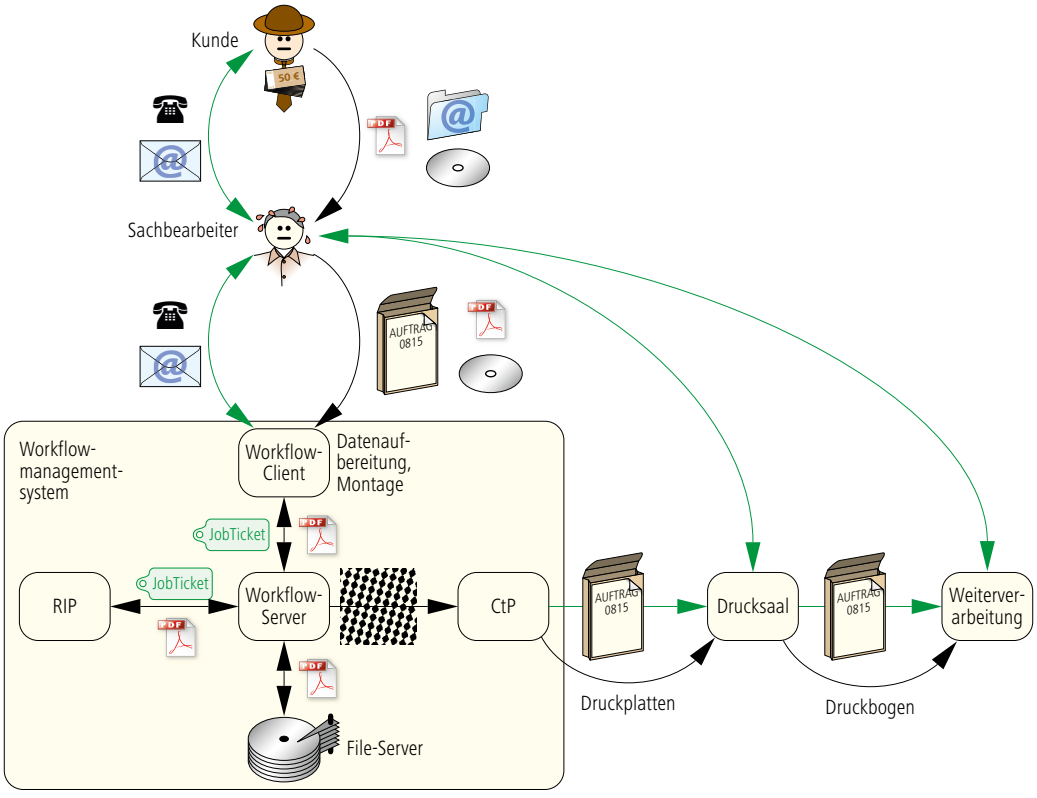



Abbildung 2.3
Client-Server-basierendes WMS

In dem hier gezeigten Einzelkomponenten-Workflow werden die Druckdaten von unterschiedlichen Softwarekomponenten verarbeitet, wobei „Montage“ und „RIP“ nur als Beispiel stehen. Sind mehrere Personen und mehrere Rechner an diesem Produktionsvorgang beteiligt, werden die Daten auf einem Fileserver zwischengespeichert, die Auftragstasche wird jedoch manuell weitergegeben. Zum Schluss wird eine Druckform ausgegeben und die zusammen mit der Auftragstasche an den Drucksaal weitergeleitet. Für die Auftragsverfolgung muss das Auftragsmanagement die einzelnen Stationen des Produktionsablaufs kontaktieren (telefonisch oder persönlich).

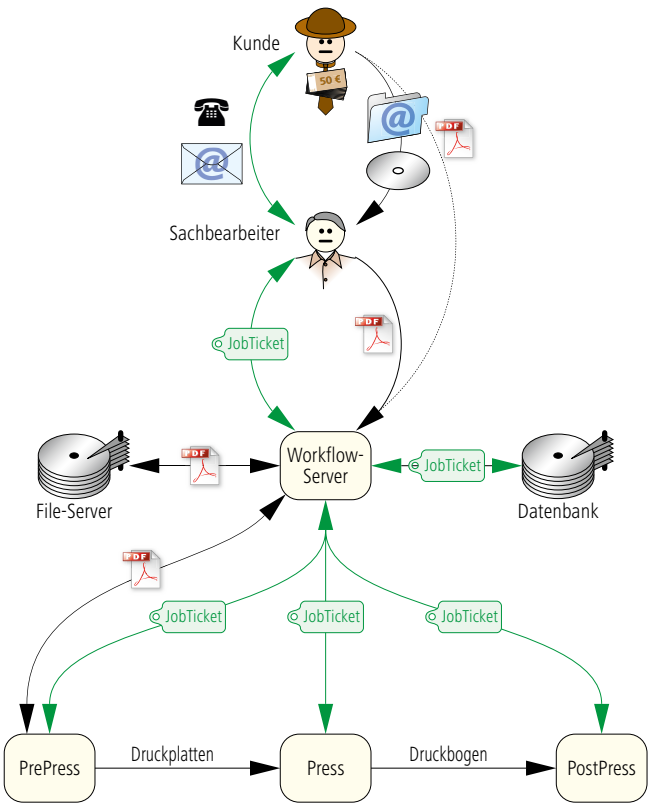
Was sind die offensichtlichen Schwachpunkte dieses Workflows? Es gibt mehrere Dinge zu nennen:

- Die Content-Daten und die Auftragsdaten laufen über unterschiedliche Kommunikationskanäle, was die Produktionssicherheit verringert.
- Die Betriebsdatenerfassung findet entweder gar nicht oder

nur über die berüchtigten Stundenzettel unabhängig von den Arbeitsprozessen statt.
- Die Auftragsverfolgung (*Job-Tracking*) ist aufwändig.
- Voreinstellungsdaten von Maschinen müssen immer wieder neu eingegeben werden.

Abbildung 2.4 vollständig integriertes WMS

Diese Arbeitsweise umfasst also streng nach der Definition kein wirkliches Workflow-Managementsystem. Ein solches hat man erst in Abbildung 2.3. Durch ein Client-Server-Workflowsystem kann die Produktion dort wesentlich besser automatisiert werden.




Als Client kann der (die) Anwender(in) die nötigen Jobtickets aufrufen, die an die serverbasierten Jobticketprozessoren weitergeleitet werden. Nur der Systemadministrator kann die Jobtickets generell ändern, zum Beispiel also festlegen, dass standardmäßig für die Plattenausgabe ein 70er Raster erzeugt wird. Ein normaler Prepress-Operator darf dann nur individuell für seinen Druckauftrag diesen Eintrag verändern.

Durch diese fest definierten Jobtickets, die aber dennoch individuell angepasst werden können, erlangt man in der Produktion ein hohes Maß sowohl an Sicherheit als auch an Flexibilität.

Die schwarzen Pfeile in dem gelb hinterlegten WMS in 2.3 transportieren also sowohl Content- als auch Metadaten. Hier wird also die strikte Trennung der beiden Kommunikationskanäle überwunden.

Das Konzept stellt eine wesentliche Verbesserung gegenüber dem Einzelkomponenten-Workflow dar, hat aber immer noch einige Beschränkungen:

- kein übergreifendes Auftrags- und Produktionsmanagement

- keine übergreifende Betriebsdatenerfassung (BDE)
- keine abteilungsübergreifende Übernahme von Voreinstellungen für Maschinen.

Mit einem solchen WMS ist aber bereits heute die abteilungsübergreifende, elektronische Weitergabe von Information durchaus üblich. So werden zum Beispiel Daten, die als Grundlage zur Berechnung der Farbzonenvoreinstellung herangezogen werden, von der Druckvorstufe an den Drucksaal weitergeleitet.

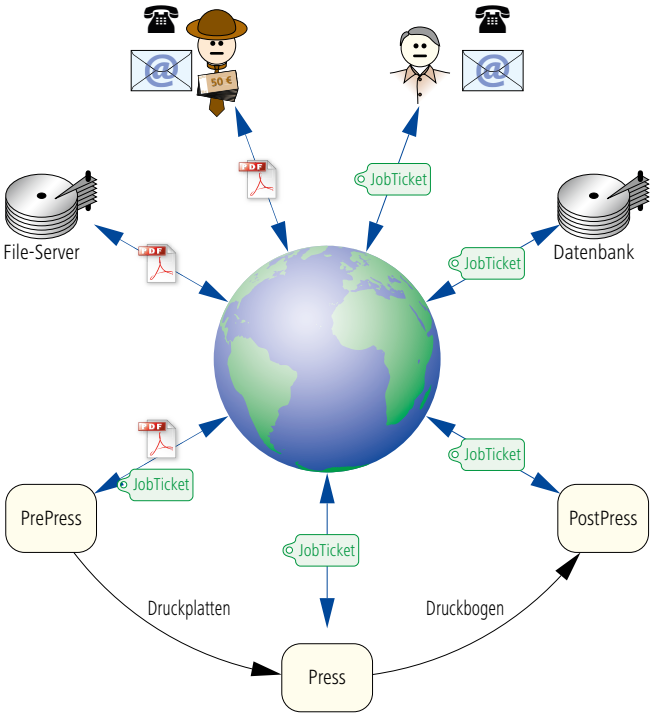
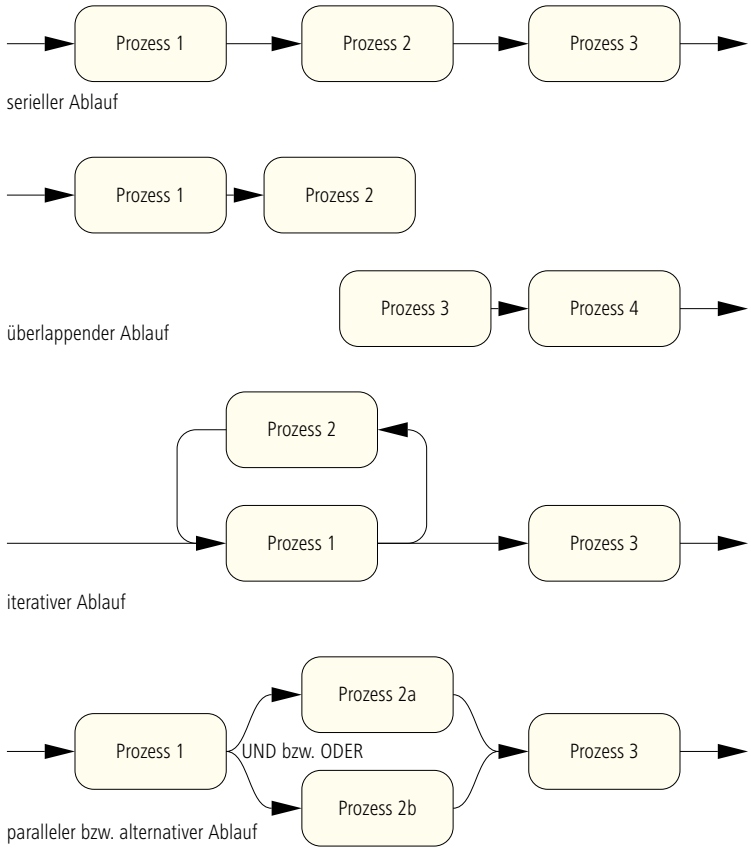



Abbildung 2.5
Internet-basiertes WMS
(Die Standorte der beteiligten „Dienstleister“ spielen nahezu keine Rolle).

Ein vollständiges integriertes und abteilungsübergreifendes WMS innerhalb eines Betriebes ist in Abbildung 2.4 skizziert. Hier werden Metadaten über das lokale Netzwerk an die Aggregate der verschiedenen Abteilungen transportiert und auch wieder zurück. Die Content-Daten vom Auftraggeber werden natürlich nur bis zur Druckvorstufe geleitet. Es soll noch einmal angemerkt werden, dass eine solche Skizze nicht die Wirklichkeit widerspiegelt, da die Vernetzung zurzeit in den Betrieben immer nur partiell ist.

In Abbildung 2.5 schließlich findet der Datenaustausch nur noch über das Internet statt. Auftragsmanagement, Druckvorstufe und die Druckformherstellung können an beliebigen Orten auf der Welt liegen und zu verschiedenen Firmen gehören. Erst wenn physische Dinge wie Druckplatten oder Druckbogen transportiert werden müssen, ist geografische Nähe zwingend. Zwar erscheint eine solche Skizze noch recht realitätsfremd oder übertrieben zu sein, doch wenn man sich die in den letzen Jahren entwickelten Beispiele von internet-basiertem Workflow zwischen Auftraggeber und Druckerei ansieht, ist man nicht mehr so weit davon entfernt.

Abbildung 2.6 verschiedene Ablaufmöglichkeiten von Geschäfts- und Produktionsvorgängen




## 2.6 Eigenschaften von WMS und Jobticket-Formaten

Folgende drei Forderungen an Workflow-Managementsysteme beziehungsweise an den zugrunde liegenden Jobticket-Datenformaten möchten wir genauer diskutieren:

- Anpassbarkeit
- Standardisierung
- Erweiterbarkeit

Diese Schlagworte klingen auf den ersten Blick unmittelbar einleuchtend, sind aber beim näheren Hinsehen nicht völlig unproblematisch.

### Anpassbarkeit

Das WMS-Modell muss sich den Arbeitsabläufen eines Betriebes anpassen und nicht umgekehrt. Dieser anscheinend richtige Satz hat aber seine Grenzen: Einen völlig chaotischen Arbeitsablauf will man gar nicht modellieren. Die Forderung

muss also besser heißen, dass ein WMS beliebige, aber wohldefinierte Produktionswege abbilden soll. In der Tat ist die Analyse und die Festlegung der Produktionswege eine wichtige und schwierige Aufgabe bei der Einführung eines JDF-Workflows in einem Betrieb.

Geschäfts- und auch Produktionsvorgänge laufen im Allgemeinen zeitlich nicht immer nur seriell ab, sondern auch parallel bzw. alternativ, überlappend oder auch iterativ, also sich wiederholend (siehe Abbildung 2.6). Bei der Herstellung von Druckprodukten gibt es für diese Situationen viele Beispiele:

- seriell: erst drucken, dann falzen, … ,
- parallel: Texte und Bilder für ein Seitenlayout erstellen,
- überlappend: Platten (mehrere Formen) belichten und drucken,
- alternativ: Job auf Maschine A oder Maschine B drucken,
- iterativ: Proof, Korrektur, Proof, Abnahme.

Ein WMS muss diese Gegebenheiten modellieren können. Dabei stellt sich die Frage, was davon im Jobticket-Datenformat verankert sein muss und was in der WMS-Software, die dieses Jobticket-Format verwendet. Im Job Definition Format kann man zum Beispiel überlappende Vorgänge modellieren. Hierzu wird der aus der Informatik stammende Begriff *Pipe* verwendet, also eine Röhre. Das Prinzip ist, dass ein Vorgang einen Output produziert (wie z.B. die Druckplatten erzeugen) und diesen in die *Pipe* einspeist und überlappend dazu ein zweiter Vorgang (das Drucken) diesen Output als Input verwendet, also quasi aus der *Pipe* holt. Und während der Druckvorgang abläuft, kann die Plattenbelichtung weitere Platten in die *Pipe* ablegen. Mit anderen Worten: Mit der Struktur *Pipe* lässt sich eine Lagerhaltung (wie ein Silo) beschreiben (Abbildung 2.7).

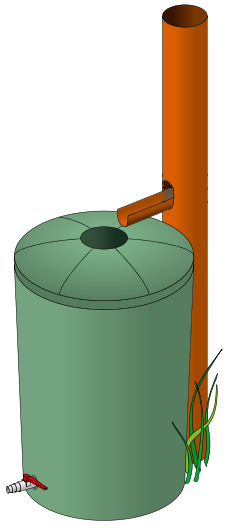


Abbildung 2.7
Das Pipe-Prinzip kennt jeder Hobby-Gärtner von seiner Regenwassertonne: 1. Der Output des Regens ist der Input des Gießens. 2. Während des Regens kann gleichzeitig gegossen werden, solange Wasser in der Tonne (=Pipe) ist.

Bei der Herstellung eines Druckproduktes ist es auch recht typisch, dass erst im Laufe der Produktion viele technische und auch organisatorische Details geklärt werden. Das bedeutet, dass die Jobtickets dynamisch immer neue Informationen aufnehmen können müssen. Die Farbzonenvoreinstellung zum Beispiel weiß man halt nicht gleich bei der Auftragsannahme. Außerdem wird vielleicht am Produktionsbeginn zwar bereits das Druckprodukt beschrieben, aber noch recht ungenau. Wie bereits vorher erwähnt, ist zum Beispiel am Anfang vielleicht die ungefähre Rasterfrequenz bekannt, die verwendet werden soll, aber erst im weiteren Produktionsablauf werden diese Angaben

präzisiert. Auch diesem Sachverhalt müssen WMS und Jobticket-Format Rechnung tragen.

Selbst Änderungen der Produktionsparameter während der Jobbearbeitung (z.B. ausgelöst durch den Auftraggeber) sind durchaus üblich, aber leider sehr schwer in einem WMS abzubilden. Hier ist noch viel Entwicklungsarbeit nötig.

**Standardisierung**

Um herstellerübergreifende Integration von *WMS-Engines* zu ermöglichen, ist es erforderlich, dass die zugrunde liegende Kommunikation standardisiert ist. Doch auch die Standardisierung hat ihre Grenzen. Denn was würde sie in letzter Konsequenz bedeuten?

Zunächst müsste ein Jobticket-Format vollständig sein, das heißt es müsste alle denkbaren Kommunikationsinhalte für die Herstellung grafischer Produkte abdecken. Wäre dem nicht so, würden einige WMS-Hersteller wegen der fehlenden Funktionalität für ihre speziellen Anforderungen das standardisierte Jobticket-Format nicht verwenden können. Denn die Alternative, dass sich die WMS-Hersteller in der Funktionalität auf das einschränken müssten, was im Jobticket-Format definiert werden kann, will sicherlich keiner: Der Fortschritt wäre damit völlig blockiert und Firmen könnten sich in ihren Produkten nur schwer differenzieren. Schließlich ist eine solche Standardisierung auch nur eine freiwillige Vereinbarung zwischen Firmen und keine staatliche Verordnung! Insofern ist eine Beschränkung auch völlig unrealistisch. Eine nicht vollständige Standardisierung wäre also zum Scheitern verurteilt. Andererseits wird es eine vollständige Spezifikation niemals geben können, dazu sind die Produktionswege viel zu komplex.

Doch wie löst man dieses Dilemma? Die Antwort ist, dass die Standardisierung nur zu einem gewissen Teil erfolgen kann und Jobtickets mit privaten, nicht standardisierten Information ergänzt werden. Das widerspricht zwar eigentlich der Idee der Standardisierung und kann auch durchaus zu Inkompatibilitäten zwischen *Workflow-Engines* führen, ist aber der einzig verbleibende Ausweg. Wir werden auf die so genannte „Erweiterbarkeit“ von Jobticket-Formaten gleich noch näher eingehen.

Es genügt auch nicht, bei der Spezifikation von Jobticket-Formaten nur die Kommunikationsinhalte – also das „Was“ – zu definieren. Es muss auch das „Wer“ festgelegt werden. Denn eine

Kommunikation zwischen mehreren *Workflow-Engines* kann erst dann richtig funktionieren, wenn klar ist, wer eine gewisse Information zur Verfügung stellen muss und wer auf der anderen Seite sich auf diese Bereitstellung der Information verlassen kann. Im JDF/JMF-Umfeld gibt es hierzu die *Interoperability Conformance Specifications* (ICS), die in Abschnitt 6.7 behandelt werden.

Ein ganz pragmatischer Gesichtpunkt stellt noch eine Einschränkung bei der Standardisierung dar. Workflow-Managementsysteme werden nicht oder zumindest nur selten von Grund auf neu programmiert, sondern entwickeln sich über Jahre. So müssen häufig „Altlasten" berücksichtigt werden, und Umstellungen auf neue Jobticket-Formate finden nur nach und nach statt. Deswegen findet man auch in den neuesten JDF/JMF-basierenden Workflow-Managementsystemen immer wieder *Workflow-Engines*, deren Kommunikationskanäle auf proprietären (herstellerspezifischen) oder älteren Jobticket-Formaten beruhen.

Schließlich noch eine letzte, mehr oder minder triviale Bemerkung: Standards verändern sich! Die JDF-Spezifikation gibt es bereits in vier Versionen, und damit wächst natürlich die Gefahr, dass Workflow-Engines unterschiedliche Versionen „sprechen" und „verstehen".

**Erweiterbarkeit**

Den WMS-Herstellern muss die Möglichkeit eingeräumt werden, trotz eines standardisierten Modells eigene spezielle Erweiterungen hinzuzufügen. Diese Forderung auf „Erweiterbarkeit" gilt dabei ganz besonders für das Jobticket-Datenformat. Doch was bedeutet eigentlich eine solche Eigenschaft für Datenformate?

Jobticket-Information werden gemäß den Regeln und Strukturen (der „Syntax") anderer Sprachen bzw. Dokumentstrukturen zusammengestellt. Diese sind in der Praxis:

- PostScript (PS)
- Portabe Definition Format (PDF)
- Extensible Markup Language (XML).

In Kapitel 4 werden die Jobticket-Formate *Print Production Format* (PPF) und *Portabel Jobticket Format* (PJTF) genauer analysiert. An dieser Stelle sei nur darauf hingewiesen, dass das PPF in PS und das PJTF in PDF kodiert ist. Das Job Definition

Format hat eine XML-Dokumentenstruktur. Die Erweiterbarkeit bei XML wird in Abschnitt 5.2 erklärt.

PS und PDF sind als Seitenbeschreibungssprache beziehungsweise als Dokumentenstruktur zur Seitenbeschreibung in der grafischen Industrie wohlbekannt ([4] und [5]). Beide können in ihrem Sprachumfang erweitert werden, das heißt, private Schlüsselwörter können frei gewählt und optional bei der Firma Adobe Systems Incorporated registriert werden (zur Vermeidung von Namenskonflikten). Wir wollen an dieser Stelle die Erweiterungsmöglichkeiten von PS anhand eines Beispiels vorstellen. Das Verfahren beim objektorientierten PDF verläuft ganz analog.

In PS können Variablennamen definiert und mit Werten vorbesetzt werden. Der Variablenname ist eine Zeichenkette, die durch einen vorgestellten Schrägstrich (/) gekennzeichnet wird. Das Schlüsselwort *def* weist einer Variablen einen Wert zu. Zum Beispiel definiert der PS-Code

*/hev /Helvetica-Bold def*

eine neue Variable mit dem Namen *hev*, die den Wert *Helvetica-Bold* hat. In diesem Fall dient die neue Variable einfach nur als Kürzel für den längeren Originalnamen. Genauso kann man aber auch die Zeile

*/CIP3PreviewImageWidth 1425 def*

schreiben. Der Variablenname *CIP3PreviewImageWidth* ist in der PPF-Spezifikation definiert und gibt vereinbarungsgemäß die Breite eines Voransichtsbildes in Pixeln an. Es können aber genauso eigene Variablennamen mit eigenen Werten erfunden werden, wie zum Beispiel

*/MeineErweiterung /ganz_toll def*

Das ist gültiger PS-Code mit dem natürlich keiner etwas anfangen kann. Aber Firmen können das PPF mit eigenen Variablen und weiteren PS-Strukturen bestücken, so dass sie innerhalb ihres eigenen Workflows Zusatzinformation transportieren können, welche die CIP3-Organisation im PPF nicht vorgesehen hat.

Diese privaten Erweiterungen stellen keine „theoretische Möglichkeit“ dar, sondern werden in der Praxis vielfach verwendet, sowohl im PPF, im PJTF als auch im JDF! Damit ist natürlich unter Umständen die Kompatibilität zwischen den Systemen eingeschränkt. Deswegen sollten diese privaten Einträge auch nur ergänzend sein und nicht die standardisierten Informationen ersetzen. Wenn *Workflow-Engines* auf private Erweiterungen stoßen und sie diese nicht verstehen, sind sie angehalten, sie zu ignorieren.

Zusammenfassend kann man sagen, dass sich die drei Begriffe Anpassbarkeit, Standardisierung und Erweiterbarkeit teilweise gegenseitig widersprechen und dass man versuchen muss, bei der Spezifikation von Jobticket-Formaten, aber vor allem auch in der Implementierung von Workflow-Managementsystemen, eine Art Kompromiss zwischen diesen drei Eigenschaften zu finden.

# 3 Print-Workflow-Modelle

In diesem Kapitel werden die vier folgenden Workflow-Modelle, die im Zusammenhang von Print-Workflows vielfach eingesetzt werden, anhand von einigen Beispielen beschrieben:

- Aktivitätenlisten
- Zustandsübergangsdiagramme und Aktivitätendiagramme
- Flussdiagramme
- Produzent-Konsument-Modell beziehungsweise Prozess-Ressourcen-Modell

Weitere Modellierungsansätze für allgemeine Geschäftsprozesse findet man in [18].

Die Aktivitätenliste wird häufig auch als *ToDo*-Liste bezeichnet und stellt eine Sammlung von Teilaufgaben dar, wie beispielsweise eine übliche Einkaufsliste. Eine erweiterte Aktivitätenliste kann vielleicht auch noch die Verantwortlichkeit und die Anfangs- und Endtermine der Teilaufgaben auflisten. Es geht also nur darum, wer macht was, wann und womit. Nicht aufgeführt in den Aktivitätenlisten sind Abhängigkeiten zwischen den Teilaufgaben oder auch explizit aufgeführte Entscheidungskriterien bei möglichen Verzweigungen von Arbeitsabläufen.

Letzteres ist ein wichtiges Merkmal von Zustandsübergangsdiagrammen. Dort werden Zustände als Rechtecke oder Kreise und Zustandsübergänge mit Pfeilen zwischen den betroffenen Zuständen gezeichnet. Ein klassisches Beispiel stellten die Zustände einer Maschine dar, wie *ruhend*, *beschäftigt*, *unterbrochen* und *defekt* (Abbildung 3.1). Im Fehlerfall gibt es zum Beispiel dann einen Zustandsübergang von *ruhend* oder *beschäftigt* auf *defekt*. Bei der Workflow-Beschreibung hat man

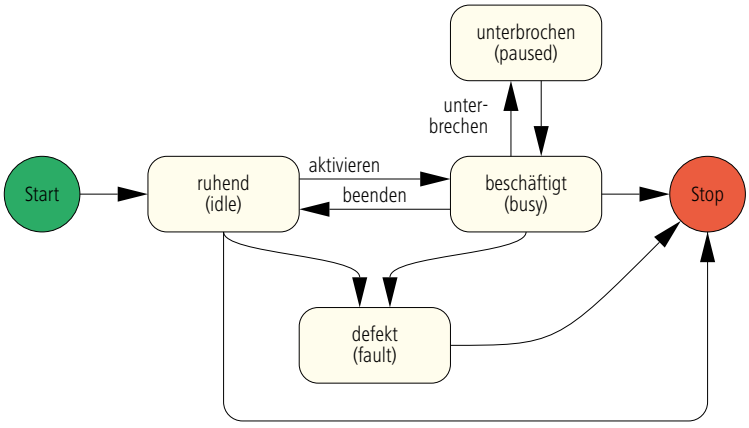



Abbildung 3.1
Zustände, welche eine Maschine in der Produktion einnehmen kann.

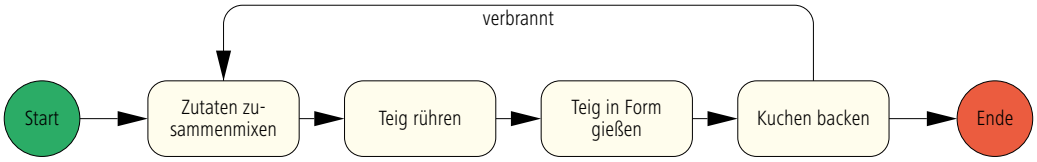



Abbildung 3.2 Aktivitätendiagramm für das Backen eines Kuchens

die gleiche Art von Diagrammen, nur werden hier Aktivitäten anstatt Zustände dargestellt. Die Pfeile sind folglich die Übergänge zwischen den Aktivitäten und wir werden in diesem Buch solche Diagramme Aktivitätendiagramme nennen. Auch Backrezepte lassen sich in dieser Weise sehr übersichtlich darstellen (Abbildung 3.2).

Die Flussdiagramme dienen in erster Linie zur Visualisierung von Programmabläufen bei der Software-Entwicklung. Sie finden aber auch in allgemeineren Modellen zu Geschäfts- und Produktionsprozessen Anwendung oder können sogar bei der Lebensplanung für den Alltag sehr hilfreich sein, wie man in Abbildung 3.3 sieht. Der grüne bzw. rote Kreis markieren den Anfang und das Ende des Diagramms, das gelbe Parallelogramm eine Ein- oder Ausgabe, die orange Raute eine Abfrage, die aber nur *ja* oder *nein* als Antwort zulässt. Das blaue Rechteck stellt eine allgemeine Operation dar, das graue mit den doppelten, vertikalen Linien ein Unterprogramm, das in der Regel mit einem neuen Flussplan genauer spezifiziert wird. All die geometrischen Elemente sind in ihrer Bedeutung genormt, nicht jedoch die Farben, die wir ihnen gegeben haben.

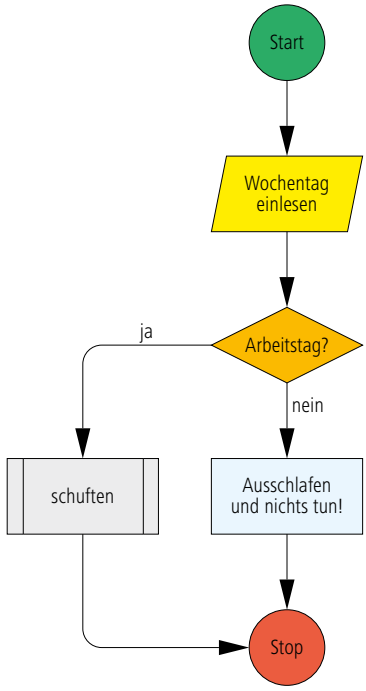



Abbildung 3.3 Flussdiagramm mit unterschiedlichen Operationen

Das Produzent-Konsument-Modell, das auch Erzeuger-Verbraucher-Modell genannt wird, beschreibt genau das, was man erwarten würde: Ein Produzent erzeugt ein Produkt und stellt es in einen Pufferspeicher, wie z.B in ein Regal in einem Super-

markt. Der Konsument entnimmt ein Stück aus dem Puffer und verbraucht es. Beide Vorgänge sind zeitlich von einander entkoppelt, nur muss der Produzent natürlich erst einmal etwas produzieren, bevor der Konsument es konsumieren kann.

Im Print-Workflow ist dieses Modell sehr weit verbreitet, nur dass man anstatt von Produzenten und Konsumenten schlicht von Prozessen redet und anstatt von Produkten von Ressourcen. Ein Plattenbelichtungsprozess erzeugt beispielsweise als Ressource die belichteten Druckplatten, die von dem Druckprozess gewissermaßen konsumiert werden. Der Druckprozess erzeugt seinerseits wieder eine Ressource, nämlich einen Stapel Druckbogen, der von einem Weiterverarbeitungsprozess verwendet wird und so weiter (Abbildung 3.4).

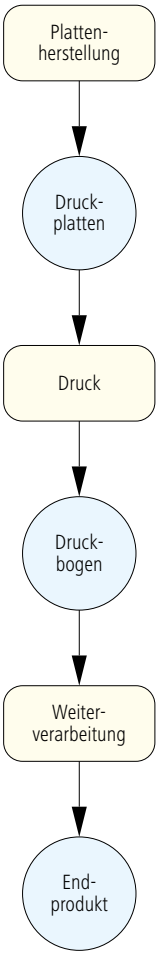



Abbildung 3.4
Produzent-Konsument-Modell im Print-Workflow

Auch das Kuchenbacken, das wir in Abbildung 3.2 mit einem Zustandsübergangsdiagramm modelliert haben, lässt sich alternativ sehr schön in einem Produzent-Konsument-Modell darstellen. Die Zutaten zu mixen, zu verrühren usw. sind die Prozesse, die Zutaten selber, also Mehl, Eier, Butter und Milch, sowie die Zwischenergebnisse, wie der Rührteig oder die gefüllte Backform, sind die Ressourcen. Dabei gilt die Faustregel: Die Prozesse sind die Verben, die Ressourcen sind die Substantive.

Das JDF-Modell beruht auf dem Prozesse-Ressourcen-Prinzip und folglich wird es uns über das gesamte Buch begleiten.

Wir werden nun diese drei Modelle auf die Teilbereiche Auftragsmanagement, Prepress, Press und Postpress anwenden.

## 3.1 Auftragsmanagement

Die Aktivitäten des Auftragsmanagement bis zur Auftragserteilung lassen sich vereinfacht in folgender Liste zusammenstellen:

- Angebot anfordern (Kunde zur Druckerei)
- Angebot abgeben (Druckerei zum Kunden)
- Auftrag erteilen (Kunde zur Druckerei)

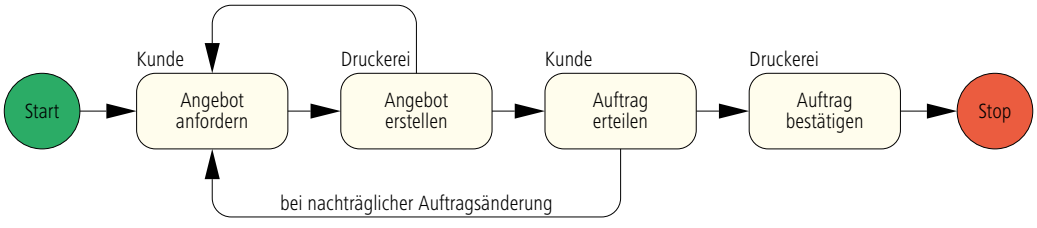



Abbildung 3.5
Aktivitäten des Auftragsmanagements

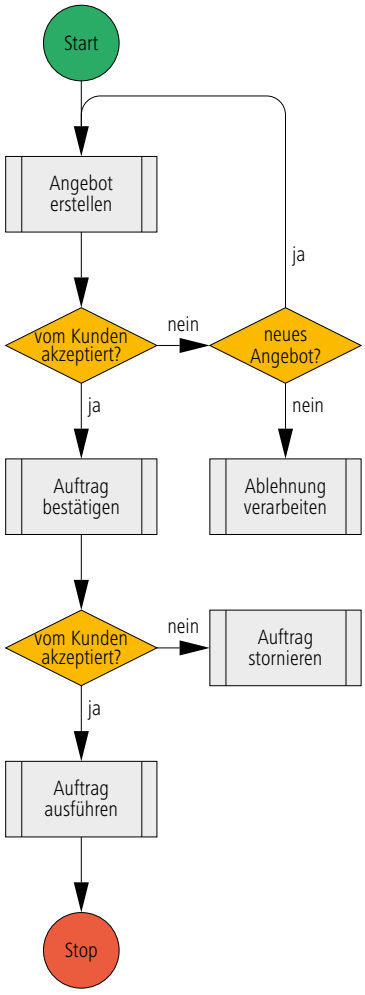



Abbildung 3.6
Flußdiagramm mit Aktionen einer Druckerei

- Auftrag bestätigen (Druckerei zum Kunden)

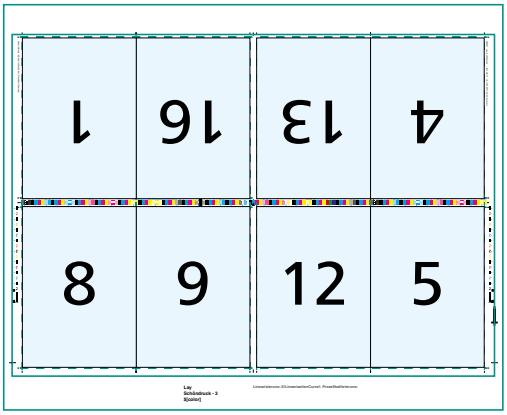
Offenbar gibt es hier eine zeitliche Abhängigkeit zwischen den Teilaufgaben und es liegt nahe, diese dann mit einem Aktivitätendiagramm wie in Abbildung 3.5 zu beschreiben.

In der Regel kann man einen Workflow mit einem Flussdiagramm noch detaillierter darstellen. Denn jede Aktivität und auch jeder Übergang zwischen den Aktivitäten in einem Aktivitätendiagramm kann selbst wieder eine Menge von Abfragen und Operationen gemäß der Flussdiagramm-Terminologie sein. In der Abbildung 3.6 sind die Druckerei-Aktionen aus dem Diagramm 3.5 noch einmal ausführlich als Flussdiagramm dargestellt (siehe auch Figur 3 in [40]).

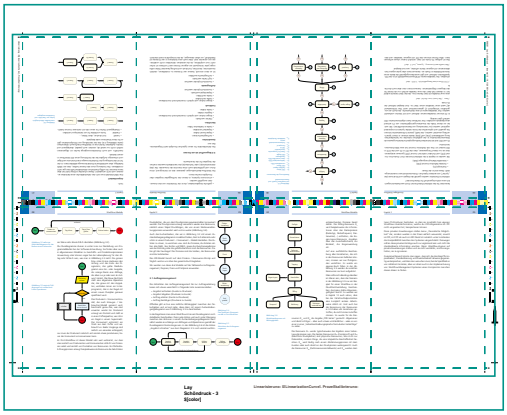
## 3.2 Ausgabe-Workflow in der Vorstufe

Eine Druckformherstellungsabteilung einer Druckerei, die fertige PDF-Seiten angeliefert bekommt, hat typischerweise folgende Aktivitäten (zur Klärung der Begriffe siehe das Glossar am Ende des Buches):

- Preflight durchführen, Daten ggf. korrigieren
- Normalisieren der PDF-Dateien
- Farbraumtransformationen durchführen
- Trapping
- Einen oder mehrere Standbogen auswählen bzw. erstellen




Bogen-Layout



ausgeschossener Bogen mit platzierten Inhalten

Abbildung 3.7
die Begriffe „Bogen-Layout erzeugen“ und „Ausschießen“

(Bogen-Layout)
- Ausschießen
- RIPing der Daten zur Ausgabe von Formproof(s) auf einem Plotter
- Druckfreigabe
- RIPing der Daten, um Voransichtsbilder zu erzeugen
- RIPing der Daten zur Belichtung, Plattenbelichtung und Entwicklung von Druckplatten
- Farbzonenvoreinstellungswerte berechnen.

Wir wollen nur kurz auf den Unterschied zwischen „Bogen-Layout erzeugen“ (*stripping*) und „Ausschießen“ (*imposition*) eingehen, da diese beiden Begriffe in der Praxis häufig nicht akkurat getrennt werden. Bei der Erzeugung eines Bogen-Layouts geht es zunächst darum, die Größe des Druckbogens und den Papierbeginn (= Position des Druckbogens auf der Druckplatte) festzulegen. Anschließend werden die Seiten oder Nutzen auf den (virtuellen) Druckbogen in ihrer Größe definiert und positioniert, im Akzidenzdruck häufig mit Hilfe eines Ausschießschemas. Hierzu müssen auch die Bindeart und die Art des Rückseitendrucks sowie der Greiferrand und weitere Ränder wie Vorfalz, Nachfalz oder der Fräsrand bei Klebebindungen berücksichtigt werden. Die Seiten beziehungsweise Nutzen werden als Platzhalter (Templates) definiert, also ohne Inhalt. Die Seitenplatzhalter werden auch Musterseiten genannt. Schließlich werden noch Marken auf den Bogen gesetzt, wie Druckkontrollstreifen, Schneide- und Falzmarken, Registerzeichen und dergleichen. Das Ergebnis ist ein Standbogen oder ein Bogen-Layout.

Beim Ausschießen werden die Seiten oder Nutzen, die typischerweise im PDF-Format vorliegen und manuell oder auto-

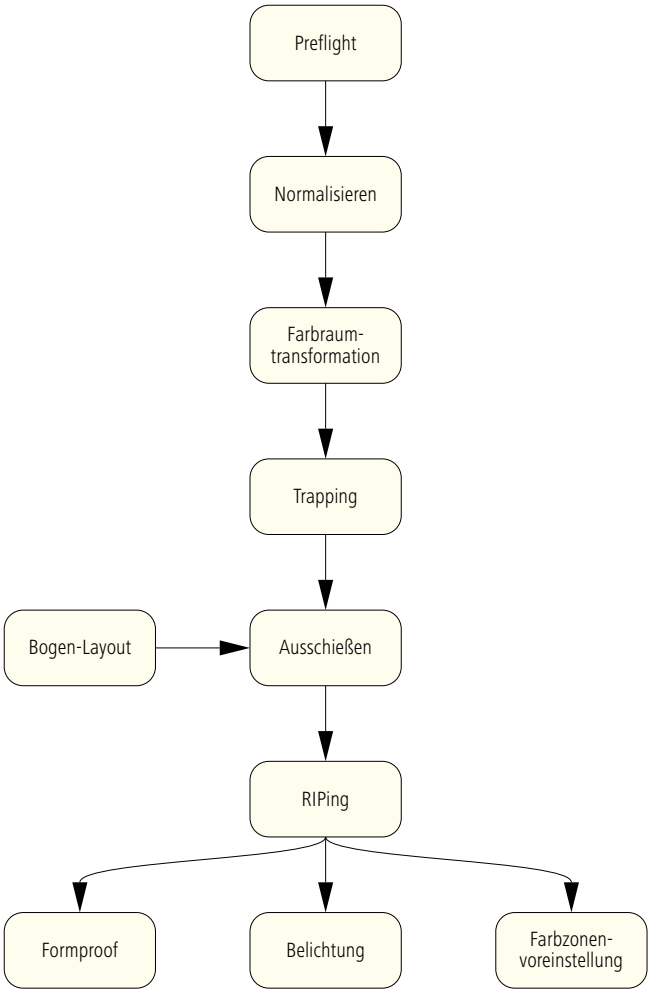



Abbildung 3.8
Aktivitätendiagramm mit Abhängigkeiten

matisch dem Standbogen zugeordnet werden, zusammen mit den Marken zu einer Struktur zusammengerechnet. Die Musterseiten werden dabei mit Content gefüllt und es entstehen Dateien, welche die Bogen beschreiben (Abbildung 3.7), beispielsweise im PDF-Format.

Früher wurden beide Prozesse innerhalb eines Programms, dem Ausschieß- oder Montageprogramm, durchgeführt. Heute werden sie häufig getrennt: Das Erstellen des Bogen-Layouts, das (noch) unter Kontrolle eines Operators passiert, geschieht häufig mit einer stand-alone Software auf dem Desktop-Rechner des Operators, während der automatische Prozess des Ausschießens auf einem Server im Hintergrund durchgeführt wird.

Das Aktivitätendiagramm, in dem dann auch die Abhängigkeiten zu erkennen sind, sieht beispielsweise wie in Abbildung 3.8 aus.

Viel genauer wird dann das entsprechende Prozess-/Ressourcenmodell, aber auch deutlich komplizierter (in Abbildung 3.9). Die Prozesse sind durch abgerundete Rechtecke symbolisiert, die Ressourcen durch Kreise, wobei wir hier zwischen den blauen Ressourcen $R_1$ bis $R_{10}$ und den violette $R_{11}$ bis $R_{24}$ differenzieren. Denn die blauen Ressourcen sind reine Input-Ressourcen für die Prozesse, während die violetten Ressourcen (auch) Output-Ressourcen darstellen. Die blauen Ressourcen sind grundsätzlich Parameter-Ressourcen, die Detailinformationen für die entsprechenden Prozesse bereitstellen. Die RIPing-Ressource $R_7$ wird beispielsweise die Informationen über den Rasterprozess (Rastertyp, Rasterfrequenz, Rasterwinkel...) enthalten, die Bogen-Layout-Ressource $R_5$ Angaben über das Aus-

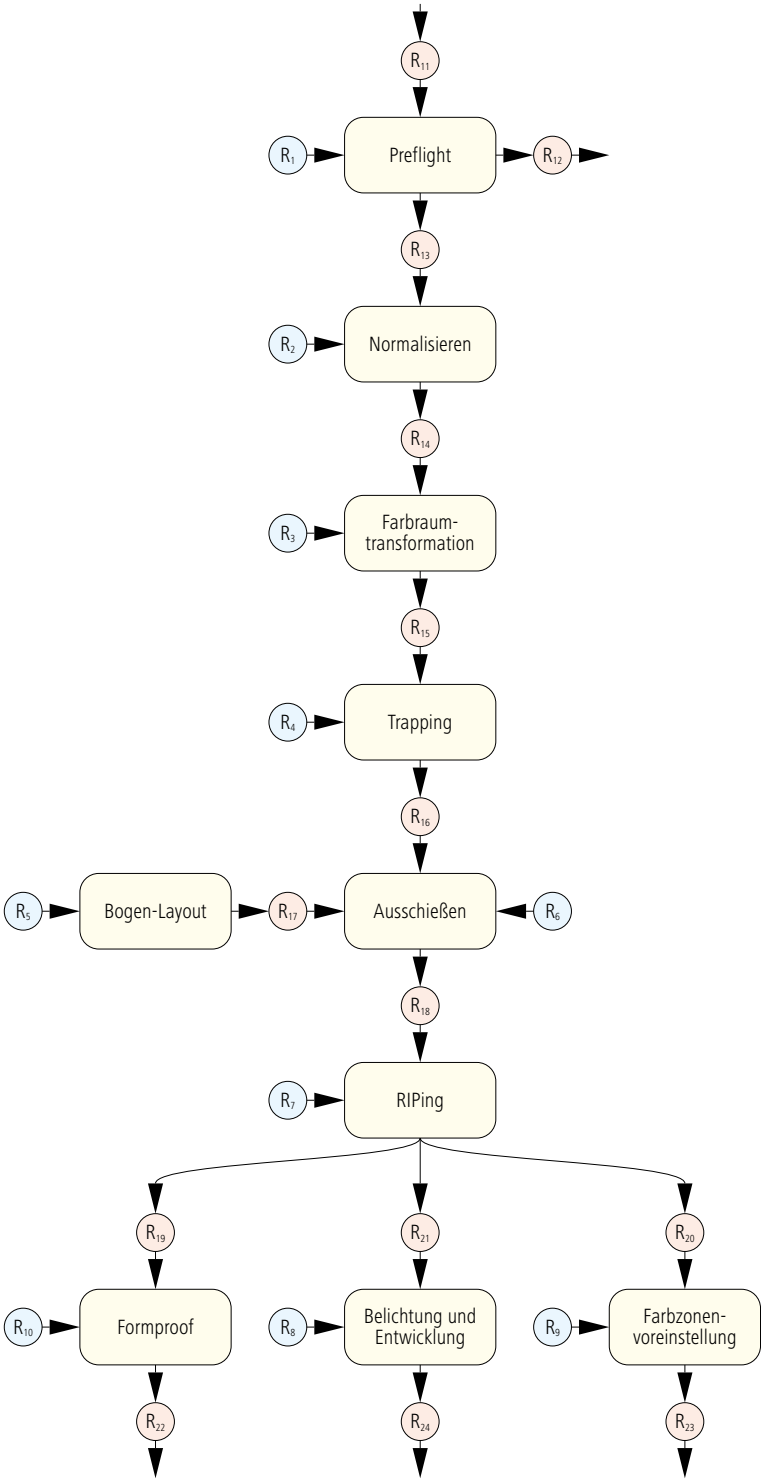



Abbildung 3.9
ein komplexes Prozess-Ressourcen-Modell

Bezeichnung der Ressourcen:

$R_1$ bis $R_{10}$ Eingabeparameter des jeweiligen Prozesses
$R_{11}$ PDF-Seiten
$R_{12}$ Preflight-Report
$R_{13}$ PDF-Seiten
$R_{14}$ normalisierte PDFs
$R_{15}$ farbkonvertierte PDFs
$R_{16}$ über-/unterfüllte PDFs
$R_{17}$ Standbogen
$R_{18}$ PDF-Bogen
$R_{19}$ Datei für Formproofer
$R_{20}$ Vorschaubild
$R_{21}$ TIFF-B
$R_{22}$ Formproof
$R_{23}$ Farbzonenvoreinstell-werte
$R_{24}$ bebilderte Druckplatte

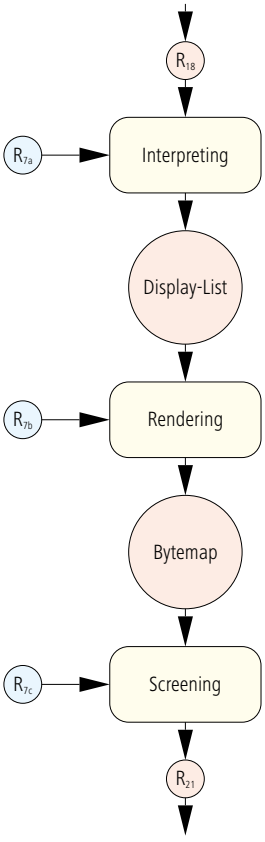



Abbildung 3.10
RIPing-Prozess

schießschema, die Bindeart, die Bogenwendung etc.

Auf eine ausführliche Beschreibung aller Einzelheiten, die sich in den Ressourcen befinden können, müssen wir aus Platzgründen verzichten. Es würde ein eigenes Buch füllen. In der Abbildung 3.9 werden die violetten Ressourcen nur kurz aufgelistet.

Man sollte sich allerdings darüber im Klaren sein, dass das Szenario in der Abbildung 3.9 nur ein Beispiel für einen Workflow in der Druckformherstellung beschreibt, das nicht strikt allgemeingültig ist. So werden wir in Kapitel 12 auch sehen, dass bei der Faltschachtelproduktion eine komplett andere Arbeitsweise üblich ist. Und auch bei der Benennung der Ressourcen in 3.9 haben wir Annahmen getroffen, die nicht immer zutreffen müssen. So wurde für die Ressourcen $R_{11}$ und $R_{13}$ die Angabe „PDF-Seiten“ gemacht. Allgemeiner und damit richtiger, aber auch etwas umständlicher, wäre es, von einer in einer „Seitenbeschreibungssprache formulierten Seitenfolge“ zu reden.

Die Ressource $R_{11}$ würde typischerweise das Ergebnis eines Seitenlayout-Prozesses sein. Die beiden Ressourcen $R_{22}$ (Formproof) und $R_{24}$ (bebilderte Druckplatten) sind physische Ressourcen, also nicht nur Datensätze, sondern Dinge, die eine körperliche Beschaffenheit haben. $R_{22}$ wird häufig nach einem Abstimmungsprozess mit dem Kunden oder auch direkt an den Druckprozess weitergereicht. Auch die Ressourcen $R_{23}$ (Farbzonenvoreinstellwerte) und $R_{24}$ werden dem Druckprozess als Input-Ressourcen dienen.

Am Modell 3.9 kann man auch gut erkennen, dass das Prozess-/Ressourcenmodell zwar Abhängigkeiten zwischen den Prozessen, nicht aber eine strikte Reihenfolge beschreibt. So wird nicht angegeben, ob zuerst ein Formproof ausgegeben wird und dann eine Druckplatte oder umgekehrt. Letzteres wäre natürlich widersinnig. In Kapitel 9 werden Methoden vorgestellt, die garantieren, dass Proofs rechtzeitig in der Produktionskette gefertigt werden (Approval-Prozess).

Wie fein abgestuft sollte nun der Prozess sein? Man kann ja auch viel gröbere Einheiten definieren, wie es in der Abbildung 3.4 zu sehen ist. Auf der anderen Seite kann man natürlich genau so gut das Workflow-Modell 3.9 noch viel feiner zerlegen. So könnte man den Prozess „Belichtung und Entwicklung“ aufspalten (bei einer entsprechenden Platte und einem entsprechenden Belichter) in: Platten belichten, stanzen, vorwärmen, chemisch entwickeln, abwaschen, gummieren, einbrennen. Ja, denkbar wäre sogar, dass man den Belichtungsvorgang sel-

ber noch einmal auftrennt in: Platte aus Stapel vereinzeln, der Belichtungstrommel zuführen, fixieren, belichten, aus dem Belichter ausführen. Und so könnte man immer mehr ins Detail gehen, bis zum Schluss jeder Impuls an einem Schrittmotor einen Prozess darstellen würde.

Das wäre sicherlich nicht vernünftig. Richtig aber ist, jeden Vorgang, der in der Produktion unter Umständen einzeln für sich angestoßen wird und auch nur von einem Gerät oder einer Software ausgeführt wird, als einen Prozess zu definieren. So wird niemand nur eine Platte auf eine Belichtungstrommel positionieren, wenn sie nicht auch fixiert und belichtet wird. Also wäre eine Aufspaltung des Belichtungsprozesses unsinnig. Einen Belichtungsprozess und einen Entwicklungsprozess zu definieren kann, demnach aber schon wieder sinnvoll sein. Umgekehrt wäre die Prozessdefinition von Abbildung 3.4 ebenfalls nicht zweckmäßig, da bei der Plattenherstellung und in der Weiterverarbeitung mehrere unabhängige Geräte und Applikationen beteiligt sind.

Das JDF-Modell sieht in der Tat recht ähnlich wie in Abbildung 3.9 aus. Doch der Prozess RIPing wird beispielsweise noch feiner aufgeschlüsselt, und zwar so, wie in Abbildung 3.10 skizziert. Es ist leicht zu erkennen, dass der Vorgang RIPing in drei Prozesse aufgespalten wurde: *Interpreting*, *Rendering* und *Screening*. Demzufolge wurde auch $R_7$ auf drei Ressourcen aufgeteilt: in die $R_{7a}$-Ressource *InterpretingParams*, in die $R_{7b}$-Ressource *RenderingParams* und in die $R_{7c}$-Ressource *ScreeningParams*.

Wir werden die drei Begriffe nur kurz erläutern (siehe Abbildung 3.11): Der Interpretationsprozess analysiert die Seitenbeschreibungssprache und vereinfacht die Datenstrukturen. In den *InterpretingParams* könnte dann beispielsweise ein Eintrag darüber stehen, ob die Seite noch skaliert bzw. an die Größe des Ausgabemediums angepasst werden soll. Das Ergebnis des Interpretationsprozesses ist eine (nicht standardisierte) Datenstruktur, die *Display-Liste* genannt wird. Der *Rendering*-Prozess hat als nächstes die Hauptaufgabe, die in der Seitenbeschreibungssprache mathematisch definierten geschlossenen Konturen zu füllen. Das Ergebnis ist eine Pixelstruktur, auch *Bytemap* genannt. Die Auflösung dieser Pixelstruktur ist einer der Einträge in den *RenderingParams*. Die so erhaltenen Pixelstrukturen haben noch Farbtiefe, das heißt die Pixel können noch unterschiedliche Farbschattierungen aufweisen. Beim *Screening*-Prozess werden diese Pixel in ein Raster (Bitmap) umge-

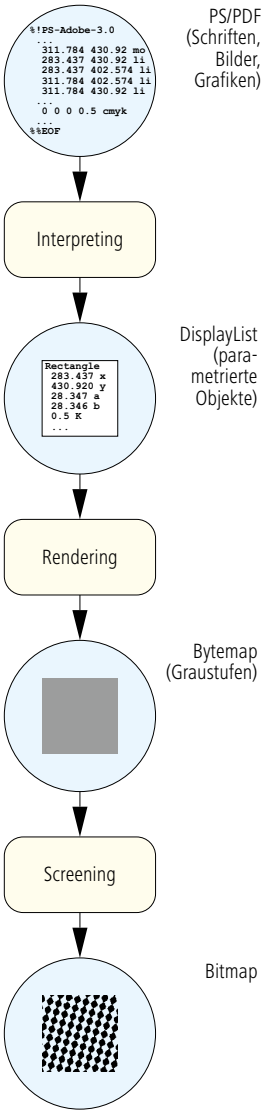



Abbildung 3.11
Rendering- und Screeningprozess

wandelt, sei es in ein periodisches Raster (AM-Raster) oder ein nicht-periodisches Raster (FM-Raster). Die Informationen, die für den Rasterprozess wichtig sind (Typ, Frequenz...) sind in den *ScreeningParams* eingetragen. Genauere Erläuterungen zu diesem Prozess findet man zum Beispiel in [22] oder in [29]. Doch warum wird nun das Riping in die drei Prozesse aufgeteilt? Als Begründung mag genügen, dass der Screening-Prozess nich immer durchgeführt werden muss. Denn nur für die Plattenbebilderung oder für einen Rasterproof muss gerastert werden, nicht jedoch für einen üblichen Digitaldruck.

Wir wollen noch einmal darauf hinweisen, dass es sich hierbei um ein Workflow-Modell handelt, nicht um den Workflow selber. Eine Output-Ressource enthält also nur die Beschreibung der belichteten Platten, nicht die Platten selbst. Trotzdem spricht man dabei von einer *physischen Ressource*. Um jedoch sprachlich den Text nicht zu überlasten, wird in diesem Buch häufig nicht zwischen Modell und Wirklichkeit unterschieden. Wir werden also beispielsweise weiterhin schlicht von einer *Plattenressource* sprechen.

Selbst die Parameter-Ressourcen müssen nicht alle unbedingt direkt in dem Workflow-Modell enthalten sein. So wird zum Beispiel eine PDF-Datei, welche die zu druckenden Elemente von Seiten oder Bogen festlegt, nicht in einer JDF-Datei eingebettet sein. Stattdessen steht dort nur, wo sich die PDF-Dateien im Dateisystem befinden. In der JDF-Datei befindet sich also nur die Referenz auf die PDF-Datei.

Vielleicht fragen Sie sich nun, wer eigentlich dieses Workflow-Modell zusammenstellen muss. Ist es denn so, dass ein(e) Operator(in) für jeden Druckauftrag eine solche Workflow-Beschreibung vorbereitet, damit anschließend die Produktion mehr oder minder automatisch ablaufen kann? Wenn ja, muss er (sie) dann wirklich alle Ressourcen genauestens festlegen? Und wenn das so wäre, ist das nicht ein fürchterlicher Aufwand?

Nun, in der Tat, das funktioniert so nicht. Aus welchen Quellen wird dann aber die Workflow-Beschreibung für einen Druckjob gespeist? Die Antwort lautet, dass Informationen aus den folgenden Bereichen zum Teil automatisch extrahiert werden:

- Kalkulation / Auftrags-Managementsystem,
- Druckdaten der Auftraggeber,
- voreingestellte Produktionsparameter (Grundwerte, Default-Werte),
- grafische Eingabemasken, die in der Produktion von den

Anwendern ausgefüllt werden,
- *eCommerce*-Systeme, in denen der Auftraggeber Angaben über Druckjobs machen kann.

Die Workflow-Beschreibungen passieren also im Hintergrund ohne Zutun und größtenteils auch ohne Kenntnis der Anwender. Der (die) Administrator(in) oder Techniker(in) der Hersteller/Anwender machen die Vorgaben für die Grundwerte.

## 3.3 Der Prozess Bogenoffset

Die Aktivitätenliste für einen typischen Druckjob sieht im Bogenoffset folgendermaßen aus:

### Grundrüsten

- Auftragstasche lesen
- Papier bereitstellen
- Papierlauf einstellen
- Farbe in Farbkasten füllen

### Einrichten

- Platten wechseln
- drucken, Bogen ziehen und visuell/messtechnisch kontrollieren
- Register stellen
- Farbe stellen

### Fortdruck

- drucken, Bogen ziehen und visuell/messtechnisch kontrollieren
- Farbe nachstellen
- Farbe nachfüllen
- Papierstapel wechseln
- Gummituchzylinder waschen

### Auftragsende

- Gummituchzylinder waschen
- ggf. Farbe wechseln
- Auftragstasche ausfüllen

Abbildung 3.12
JDF-Modell für den Offsetdruck

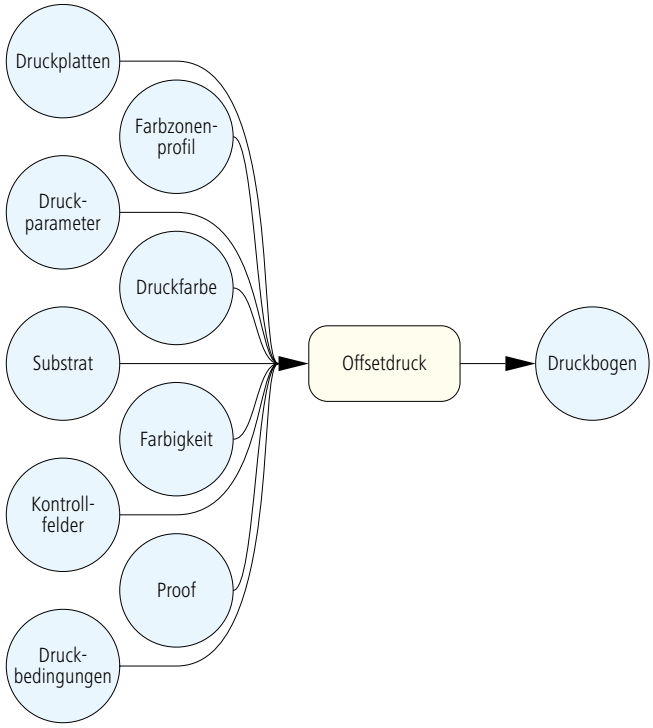
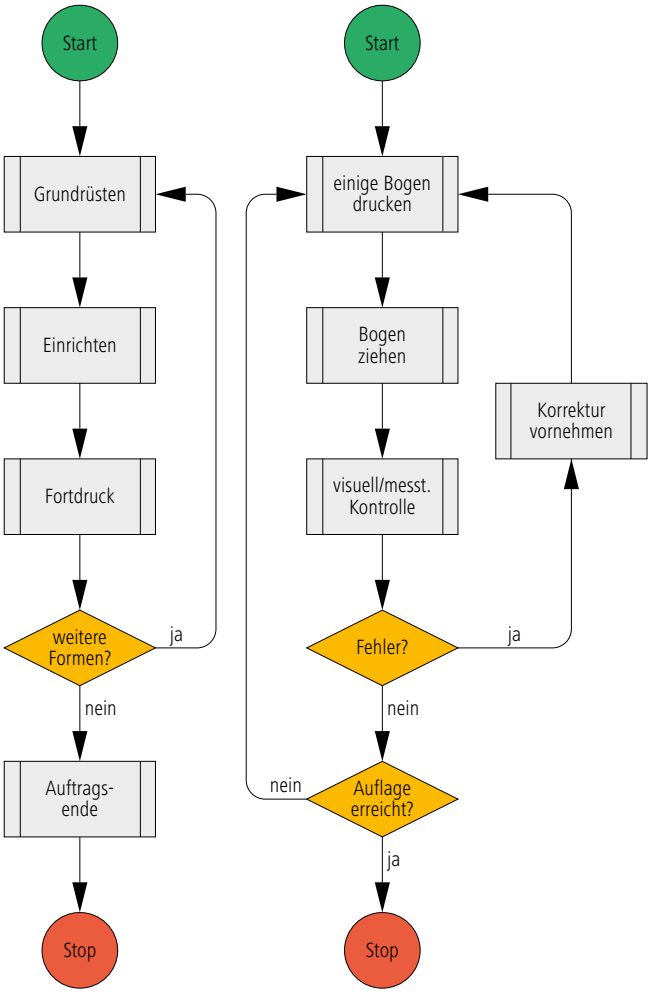


Abbildung 3.13
Flussdiagramm, das den Bogenoffset beschreibt




Ist es denn sinnvoll, hieraus vier Prozesse zu modellieren, nämlich „Grundrüsten“, „Einrichten“, „Fortdruck“ und „Auftrag beenden“? Oder sollte sogar jeder Unterpunkt ein eigener Prozess sein? Letzteres ist sicher nicht zu empfehlen, da die einzelnen Aktivitäten nicht unabhängig von einander sind. Aber auch eine Einteilung in vier Prozesse ist problematisch, vor allem, weil die Unterpunkte meist identisch oder zumindest ähnlich sind. Die Tätigkeiten beim Einrichten oder beim Fortdruck sind eigentlich gleich, nur das Ergebnis ist unterschiedlich: einmal Makulatur und einmal Gutbogen. Zu der Auftragliste ist noch zu bemerken, dass natürlich das Schema recht starr ist. So wird in der Praxis eher „verzahnt“ gearbeitet, das heißt, während ein Produktionsjob läuft, liest der Drucker bereits die Auftragstasche für den nächsten Job. Auch sind nicht alle Aktivitäten zwingend erforderlich. Wenn beispielsweise ein Papier in einer Druckerei bekannt ist, entfällt in der Regel dann beim Grundrüsten der Punkt „Papierlauf einstellen“.

Insofern hat sich auch die CIP4-Organisation für ein sehr einfaches Prozess-Ressourcen-Modell beim Offsetdruck entschieden, wie in Abbildung 3.12 zu sehen ist. Um dennoch zwischen den einzelnen Aktivitäten beim Offsetdruck unterscheiden zu können, spricht man dann von „Zuständen“, in welchem sich der Druckprozess befindet.

Beim Digitaloffset (die Druckplatten werden hierbei in der Druckmaschine bebildert) und beim Inline-Finishing, insbesondere also beim Rollenoffset, finden dann in der Druckmaschine

mehrere Prozesse statt, also nicht nur der Druck, sondern auch Belichten, Schneiden, Falzen und so weiter.

In der Zeichnung 3.12 ist zu erkennen, dass nur manche Input-Ressourcen vorhanden sein müssen, andere – wie beispielsweise ein Proof – sind nur optional. Man sieht auch, dass dieses Modell im Prinzip nur wenig über den eigentlichen Druckprozess aussagt. Möchte man diesen genauer darstellen, bietet sich ein Flussdiagramm an.

Die Abbildung 3.13 zeigt links die Hauptebenen der oben definierten Aktivitätenliste als Flussdiagramm, wobei jede als Unterprogramm gezeichnet ist. Das Unterprogramm „Fortdruck“ ist dann noch einmal auf der rechten Seite der Abbildung gezeigt. Auch hier findet man wieder Unterprogramme und man könnte wieder neue Flussdiagramme für „Bogen ziehen“ und „optische und/oder messtechnische Kontrolle“ entwerfen. So kann man immer tiefer gehen und den Grad der Detailbeschreibung immer weiter erhöhen.

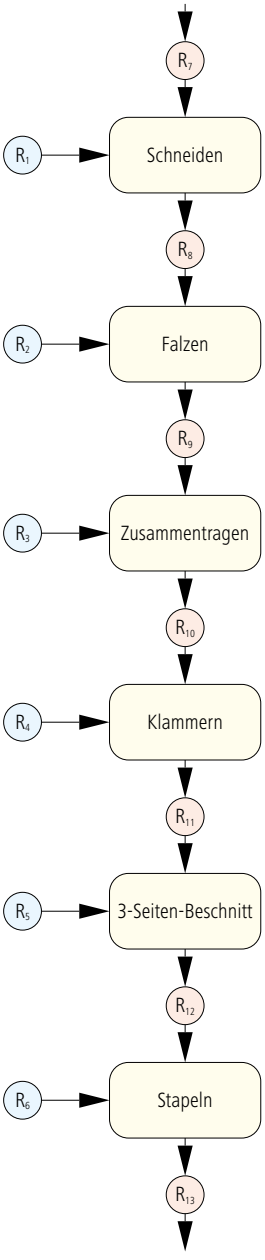



Abbildung 3.14
Prozess-Ressourcen-Modell für die Produktion einer geklammerten Broschur

## 3.4 Modell eines Postpress-Beispiels

Die Druckweiterverarbeitung ist zu vielfältig, um ein gesamtheitliches Modell zu entwickeln. Auch eine allgemeine Aktivitätenliste wäre lang und immer unvollständig, so dass wir es nicht für sinnvoll halten, eine solche zusammenzustellen. Stattdessen wollen wir nur das Prozess-Ressourcen-Modell eines Beispiels anschauen, nämlich das einer geklammerten Broschur. Die Ressource $R_7$ stellt dabei den Input der Weiterverarbeitung dar, also die Druckbogen, wie sie aus dem Drucksaal kommen. Die Ressourcen $R_8$ bis $R_{13}$ sind Beschreibungen von Zwischenprodukten, die während der Produktion in der Weiterverarbeitung entstehen:

$R_7$ Druckbogen

$R_8$ Falzbogen

$R_9$ gefalzte Bogen

$R_{10}$ Blöcke

$R_{11}$ geklammerten Blöcke

$R_{12}$ beschnittene Blöcke

$R_{13}$ gestapelter Produkte

Die Ressourcen $R_1$ bis $R_6$ sind Parameter-Ressourcen, welche die Informationen für die jeweiligen Prozesse bereithalten. Wir wollen für jede dieser Ressourcen ein paar typische Einträge nennen:

$R_1$ Position der Schneidböcke / Schneidmarken, Anlage

$R_2$ Falzreihenfolge, Falzposition, Anlage

$R_3$ Reihenfolge der zu sammelnden Komponenten

$R_4$ Anzahl, Position, Breite, Winkel,
Form und Querschnitt der Klammern

$R_5$ Breite und Höhe des Endproduktes

$R_6$ Maximale Höhe und Gewicht, Anzahl der Schichten pro Stapel,
Anzahl der Printprodukte pro Schicht

Die Größe des Druckbogens wird beim JDF übrigens nicht in $R_1$, sondern in $R_7$ eingetragen. Auch die Papiereigenschaften, die ja insbesondere wichtig für den Falzprozess sind, wurden nicht in $R_2$ eingetragen, sondern in der Ressource $R_8$.

Wir haben nun aus den Bereichen Prepress, Press und Postpress Beispiele für die Workflow-Beschreibung in Form von Aktivitätenlisten, Aktivitätendiagrammen, Flussdiagrammen und Prozess-Ressourcen-Modellen gesehen. Wir meinen, dass das Prozess-Ressourcen-Modell besonders geeignet ist, komplexere Prozessabfolgen klar und einfach zu visualisieren. Ein Flussdiagramm kann hingegen zur Darstellung der Handlungsstränge innerhalb eines Prozesses sehr hilfreich sein. Die Aktivitätenliste ist vielleicht am einfachsten zu erstellen, lässt aber am wenigsten Informationen erkennen. Außerdem verleitet sie dazu, Einträge wild zusammenzuwürfeln, so dass zum Schluss ein unstrukturiertes *Brainstorming*-Ergebnis übrig bleibt.

---

**Übungen:**

- Planen Sie das nächste Wochenende mit Hilfe eines Flussdiagramms. Definieren Sie Alternativen für Aktivitäten, wenn Bedingungen nicht erfüllt werden können.
- Beschreiben Sie die Zubereitung Ihres Lieblingsgerichts im Detail als Prozess-Ressourcen-Modell – und vergessen Sie nicht eine Vor- und Nachspeise!

# 4 Klassische Metadaten und deren Anwendung

Nachdem bereits in Abschnitt 2.4 der Unterschied zwischen *Content*-Daten (Druckdaten) und Metadaten diskutiert und erläutert wurde, sollen hier nun einige Beispiele mit entsprechenden Anwendungen vorgestellt werden.

Wir wollen in Abschnitt 4.1 mit dem Exif-Format beginnen, welches erlaubt, technische Werte und Einstellung bei der Aufnahme mit Digitalkameras mitzuschreiben. Auch den IPTC-Standard für Fotos werden wir präsentieren.

XMP steht für *Extensible Metadata Platform* und ist von der Firma Adobe für unterschiedliche Bild- und Dokumentenformate spezifiziert. Die Idee hinter dem Konzept ist die Weitergabe von Informationen über die Applikationsgrenze hinaus. So werden beispielsweise Metadaten, die in eine TIFF-Bilddatei geschrieben werden, nicht nur in das InDesign-Dokument, in das dieses Bild platziert wird, weitergereicht, sondern auch in die PDF-Datei, die aus InDesign heraus exportiert wird. So findet man vielleicht Copyright-Informationen über Fotos in einer PDF-Ausgabedatei, auch Jahre später beim Nachdruck, wenn die Bilddatei schon lange nicht mehr zur Verfügung steht.

In Abschnitt 4.2 soll das *Print Production Format* ausführlicher behandelt werden, nachdem es bereits in Kapitel 1 als wichtiges Vorgängerformat von dem Job Definition Format kurz vorgestellt wurde.

Im letzten Abschnitt dieses Kapitels geht es um einen weiteren Vorläufer des JDFs, das *Portable Job Ticket Format* (PJTF) von der Firma Adobe. Dieses Format ist eng verbunden mit der mittlerweile nicht mehr aktuellen *Extreme-RIP*-Architektur von Adobe. Diese RIP-Architektur wurde aber über viele Jahre an viele Hersteller von CtP-Workflowsystemen lizenziert und ist im Markt noch vielfach anzutreffen.

Die Inhalte dieses Kapitels sind nicht direkt für das Verständnis der weiteren JDF-bezogenen Kapitel wichtig und können auch überschlagen werden. Da aber viele Workflow-Installationen zurzeit (noch) Metadaten in einer babylonischen Sprachenvielfalt verwenden, vor allem JDF/JMF, PPF und PJTF, haben wir die Grundzüge der klassischen Metadaten mit in das Buch aufgenommen.

Außerdem ist es durchaus denkbar, dass XMP und JDF in Zukunft vermehrt zusammen eine Workflow-Implementierung realisieren.

## 4.1 Metadaten für Fotos und Dokumente

Es gibt im Wesentlich drei Gründe, warum man Fotos mit Metadaten versieht:

- Fotos können in großen Archiven leichter identifiziert und aufgefunden werden,
- Copyright-Verletzungen aus Unkenntnis werden durch Copyright-Einträge in den Fotos verhindert,
- eine Beschreibung der technischen Details der Fotos ermöglicht dem Fotografen im Nachhinein die Auswertung der Kameraeinstellung und gibt außerdem einen schnellen Überblick über Eignung der Fotos für unterschiedliche Anwendungen.

Wichtigste Fragen beim Design von Foto-Metadaten sind einerseits die Festlegungen, welche Informationen über die Bilder gespeichert werden sollten und andererseits das Beibehalten dieser Daten bei Formatkonvertierungen.

### Exif und IPTC

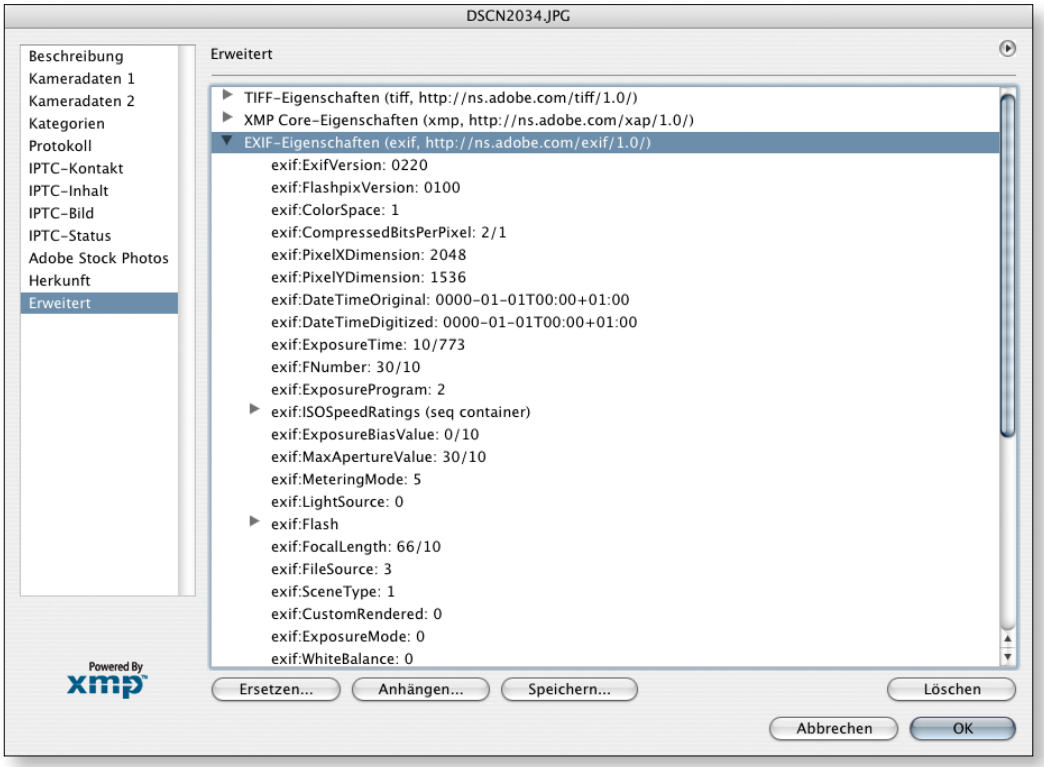
Exif steht für *Extended Interchange Format* und wurde von der *Japan Electronics and Information Technology Industries Association* (JEITA) [27] speziell für Digitalkameras entworfen. Aber auch Scan-Programme stellen einige dieser Metadaten zur Verfügung. Für jedes Foto, das mit einer Digitalkamera aufgenommen wird, werden Details über die Kamera, über den Aufnahmezeitpunkt und über die Einstellungen der Kamera während des Aufnahmezeitpunkts (Belichtungszeit, Auflösung, Blendeneinstellung...) in die Bilddatei gespeichert. Mit diversen Applikationsprogrammen können diese Daten dann angezeigt und auch modifiziert und ausgewertet werden. In Abbildung 4.1 sieht man die Exif-Daten, die automatisch von der Digitalkamera bereitgestellt wurden.

Die Exif-Spezifikation Version 2.2 [28] definiert sehr viel mehr Felder als in der Abbildung gezeigt, wie zum Beispiel auch Copyright, Kommentar oder GPS-Informationen. Sie beschreibt außerdem die Exif-Datenstruktur sowohl für JPEG- als auch für

TIFF-Bilddateien. Weitere Formate werden allerdings nicht unterstützt, auch nicht PDF. Um dennoch die Exif-Einträge in dieses Format hinüberzuretten, werden die Informationen in andere Strukturen umgebaut, wie im Absatz über XMP gezeigt wird.

Die Exif-Informationen können in TIFF und JPEG eingebaut werden, weil beide Datenformate erweiterbar sind. Das „T“ bei TIFF steht für *Tag* (oder *Tagged*), was im Deutschen so viel wie „Anhänger“ oder „Schildchen“ bedeutet. Faktisch sagt dies, dass jede Informationseinheit über das gespeicherte Bild (Bildhöhe, Bildbreite...) mit einer definierten Kennung in Form einer positiven ganzen Zahl versehen wird. Insgesamt sind $2^{16}$ (= 65536) unterschiedliche Tags möglich, und da nur wenige bisher belegt sind, können noch viele zusätzliche *Tags* festgelegt werden [3]. Bei JPEG-Bildern ist die Erweiterbarkeit durch eine ähnliche Technik realisiert. Die nach der JPEG-Norm komprimierten Bilder werden normalerweise gemäß dem Dateiformat JPEG File Interchange Format (JFIF) gespeichert [20], das wie bei TIFF die einzelnen Informationseinheiten mit Kennungen versieht, die hier allerdings *Segments* und nicht *Tags* genannt werden. Die Nummer für Exif-Daten ist beispielsweise 225.

Abbildung 4.1
Exif-Daten eines JPG-Bildes



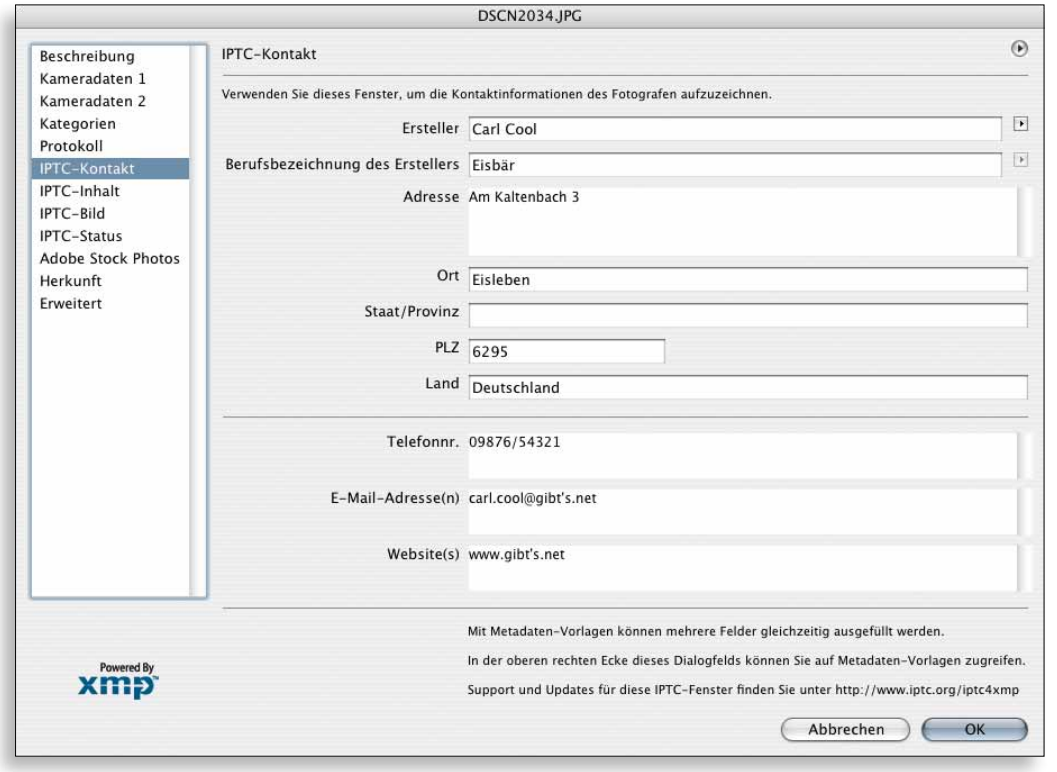



Abbildung 4.2
IPCT-Daten in einer JPG-Datei

Einige Digitalkameras speichern ihre Fotos in herstellerspezifischen RAW-Formaten ab. In diesem Fall werden dann die Exif-Daten typischerweise in separate Textdateien abgelegt.

Der *International Press Telecommunication Council* [26] ist ein weltweiter Verband, in dem Nachrichtenagenturen und nationale Zeitungsverbände vertreten sind. Bereits seit den frühen 90ern propagierte dieser Metadaten zur Beschreibung von Bildern, die in vielen Punkten zum Exif ähnlich sind und die allgemein unter dem Begriff *Information Interchange Modell* (IIM) bekannt sind. Auch hier können Eigentumsrechte vermerkt, aber vor allem die Bilder mit Schlüsselwörtern beschrieben werden (Abbildung 4.2).

### XMP

Die Firma Adobe hat 2004 mit der Spezifikation der *Extensible Metadata Platform* (XMP) eine deutlich weitreichendere Spezifikation [2] herausgebracht, die nicht nur Metadaten für Bilder, sondern auch für Dokumente unterschiedlicher Art definiert. Idee dieser Spezifikation ist, dass einmal eingegebene

Metadaten auch bei Formatwandlungen erhalten bleiben. Voraussetzung ist natürlich, dass einerseits die Einbettung der Metadaten in ein spezielles Datenformat gemäß der XMP-Spezifikation überhaupt möglich ist (wie bei TIFF, JPEG JPEG 2000, GIF, PNG, HTML, PDF, AI, SVG/XML, PSD, PostScript und EPS) und natürlich auch, dass die Applikationen, welche die Formattransformationen vornehmen, die XMP-Daten auch mitwandeln.

Tatsächlich können XMP-Informationen auch separat und außerhalb der Applikationsdateien abgespeichert werden. Doch das ist eher untypisch und wird höchstens bei Datenbankeinträgen gemacht.

Die XMP-Daten können sich auf ein gesamtes Dokument beziehen, aber auch auf einzelne Komponenten eines Dokuments. Eine solche Situation erlaubt es beispielsweise, dass den Grafiken und Bildern, die in einem PDF-Dokument eingebettet sind, eigene XMP-Metadaten assoziiert sind. Abbildung 4.3 gibt das Prinzip wieder, während Abbildung 4.4 eine Seite dieses Buches zeigt, bei der die XMP-Informationen von Komponenten zu sehen sind. In der XMP-Sprache wird das Wort „Ressource“ verwendet,

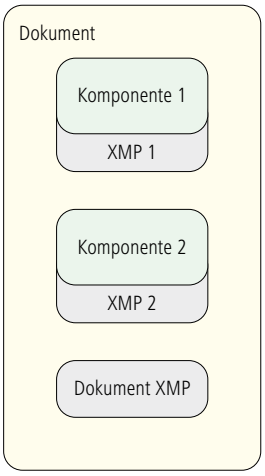



Abbildung 4.3
Prinzip für XMP von Komponenten eines Dokumentes

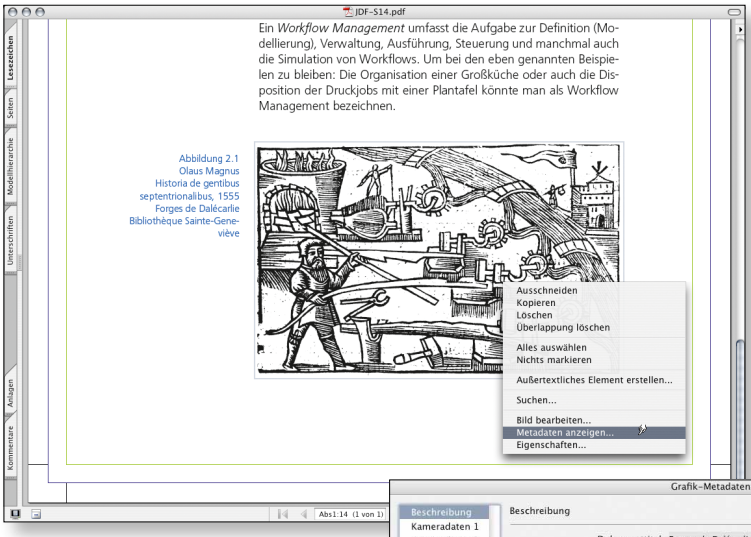
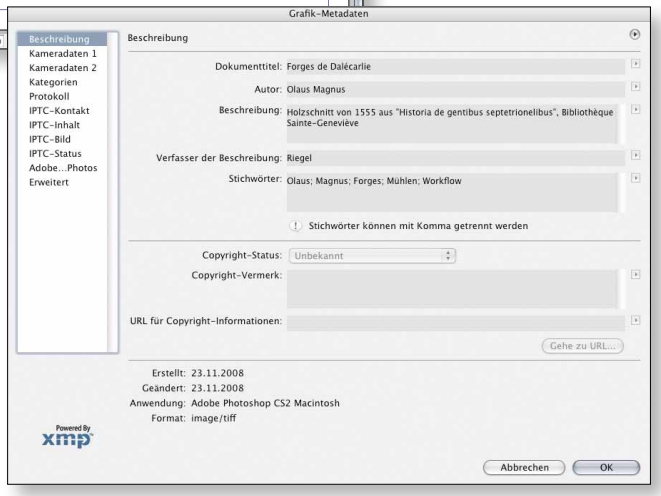



Abbildung 4.4
XMP-Informationen eines Bildes innerhalb einer PDF-Datei



die durch XMP beschrieben werden kann. Eine Ressource ist also entweder ein Dokument oder eine (sinnvolle) Komponente eines Dokuments.

Die XMP-Metadaten können in XML-Syntax gespeichert werden. Da wir XML erst im nächsten Kapitel behandeln wollen, sei hier nur so viel vorweggenommen: XML ist sehr leicht erweiterbar, indem man neue Schemata definiert. Man kann dabei gleichzeitig mehrere Schemata benutzen. Die Firma Adobe hat einige Schemata vordefiniert, wie:

- Dublin Core Schema,
- XMP Basic Schema,
- XMP Rights Management Schema,
- XMP Paged-Text Schema,
- EXIF Schema for EXIF-specific Properties.

Die vollständige Liste findet man in [2]. Hier werden nur einige Beispiele von vielen möglichen Einträgen aufgeführt, die durch diese Schemata realisierbar sind. Der Titel (*title*) und die Sprache (*language*) einer Ressource sind Eigenschaften, die im *Dublin Core Schema* definiert sind. Das *XMP Basic Schema* ermöglicht Informationen über Erstellungsdatum (*CreateDate*) oder Applikationsnamen (*CreatorTool*), mit dem die Ressource hergestellt wurde. Das *XMP Rights Management Schema* zeigt die Besitzerrechte an, während man mit dem *XMP Page-Text Schema* beispielsweise die Seitenzahlen (*Npages*) eines Dokuments speichern kann. In dem EXIF-Schema werden die meisten EXIF-Eigenschaften noch einmal unter XMP definiert. Damit können die Metainformationen der Digitalfotos auch in die XMP-Strukturen integriert werden.

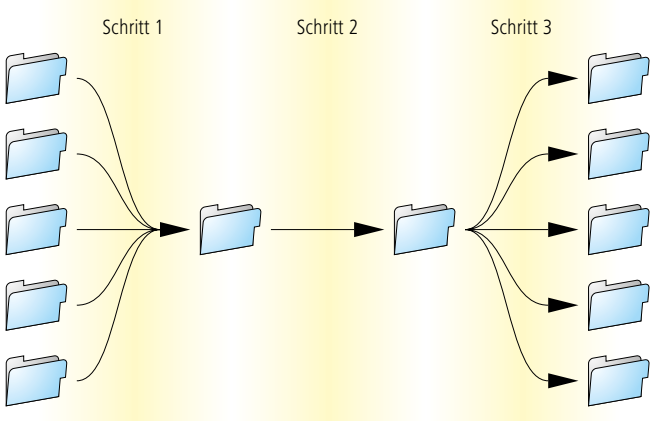
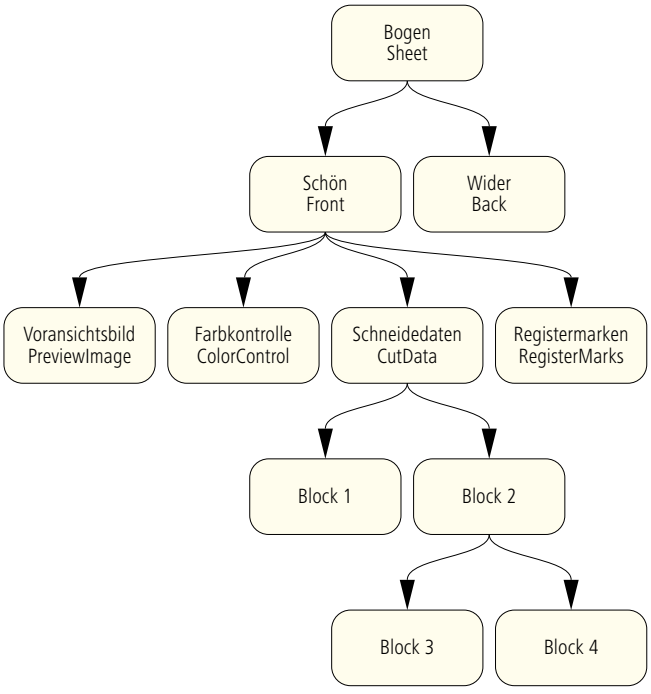



Abbildung 4.5
Fremdsoftware innerhalb eines WMS

Natürlich können nicht nur Adobe, sondern auch andere Firmen solche Schemata definieren und dann eigene, herstellerspezifische Informationen in die Daten einbauen. So kann ein WMS-Hersteller die XMP-Möglichkeiten nutzen, um Metadaten zu transportieren und seinen Workflow zu steuern. Ein weiteres Beispiel, welche Möglichkeiten XMP bietet, zeigt Abbildung

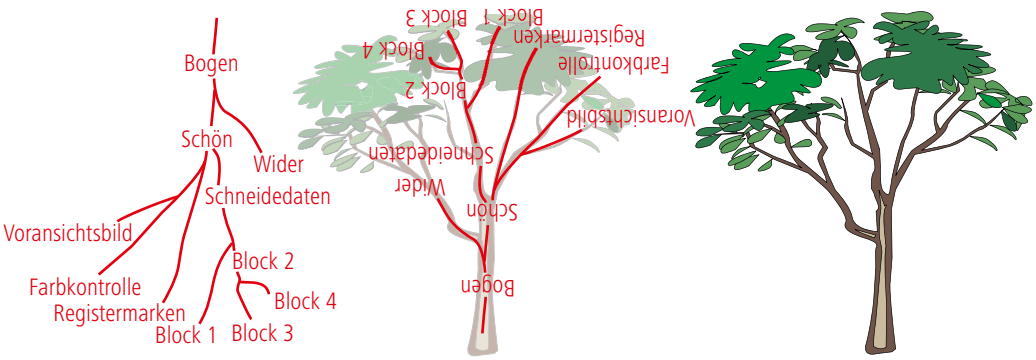
4.5. Hier wird skizziert, wie sich ein Software-Hersteller mit seiner Applikation in ein vorhandenes WMS hineinmogeln kann: In Schritt 1 entnimmt die Applikation beispielsweise PDF-Daten aus den Jobordnern des WMS und schreibt den Speicherort in einen XMP-Eintrag jeder Datei. Im zweiten Schritt führt sie ihre Arbeiten durch und modifiziert damit die PDF-Dateien. Zuletzt werden dann die Daten gemäß der XMP-Einträge wieder zurück in die Originalordner des WMS kopiert.

Abbildung 4.6
Struktur der PPF-Daten




Zusammenfassend lässt sich über die vorgestellten Metadaten Exif, IPCT und XMP sagen, dass sie deskriptiv sind, das heißt sie beschreiben Eigenschaften von digitalen Ressourcen. Ihre Kerneigenschaften sind recht beschränkt, aber Erweiterungsmöglichkeiten sind zumindest bei XMP gegeben. Diese Art von Metadaten kann keine Prozesse beschreiben, welche nötig sind, um die Daten zu modifizieren. Produktionssoftware kann aber natürlich die Metadaten nutzen, um über vorgegebene Ausführungsalternativen zu entscheiden. Da die Metadaten typischerweise zusammen mit den eigentlichen Ressourcendaten in einer Datei integriert sind, sind sie naturgemäß auch auf die Produktionsbereiche beschränkt, die mit digitalen Dokumenten umgehen, also nur auf die Druckvorstufe. Die beiden

Abbildung 4.7
Baumstruktur



Metadatenformate, die wir in den nächsten beiden Abschnitten behandeln, sind da vom Grundsatz her völlig anders.

## 4.2 Print Production Format (PPF)

In der Einleitung haben wir bereits die beiden wichtigsten Anwendungen des PPF vorgestellt: Übergabe von Daten aus der Vorstufe zur Errechnung von Farbzonenvoreinstellungen für eine Offsetdruckmaschine sowie zur Berechnung von Schneid- und Falzprogrammen für die Weiterverarbeitung. Im Abschnitt „Erweiterbarkeit“ in Kapitel 2.6 wurde schon erwähnt, dass PPF in PostScript kodiert ist und es wurde dort auch ein sehr kurzes Beispiel vorgestellt.

Abbildung 4.8
PPF-Beispiel

```
CIP3BeginSheet
    CIP3BeginFront
        CIP3BeginPreviewImage ...
        CIP3EndPreviewImage
        CIP3BeginColorControl ...
        CIP3EndColorControl
        CIP3BeginCutData ...
            CIP3BeginCutBlock ...
            CIP3EndCutBlock
            CIP3BeginCutBlock ...
                CIP3BeginCutBlock ...
                CIP3EndCutBlock
                CIP3BeginCutBlock ...
                CIP3EndCutBlock
            CIP3EndCutBlock
        CIP3EndCutData
        CIP3BeginRegisterMarks ...
        CIP3EndRegisterMarks
    CIP3EndFront
    CIP3BeginBack
    CIP3EndBack
CIP3EndSheet
```

Die Abbildung 4.6 zeigt anhand der Beschreibung eines Druckbogens, dass PPF-Informationen in ihrer logischen Strukturierung baumartig aufgebaut sind. Wie man sich erklären kann, dass von Baumstrukturen die Rede ist, zeigt Abbildung 4.7, in der die Wandlung unserer Bogenbeschreibung in einen Baum hineingezaubert wurde. Der Bogen bildet das Wurzelelement (*root element*), alle Verzweigungen sind Äste und alle Informationseinheiten, die keine weiteren Verzweigungen haben, sind die Blätter. Anstatt von Blättern und Ästen spricht man allerdings lieber allgemeiner von Knoten. Jeder Knoten, bis auf das Wurzelelement, ist ein Kind eines Elternknotens; alle Knoten bis auf die Blätter sind auch Elternknoten. Da in einer Baumstruktur jeder Knoten nur einen (alleinerziehenden) Elternteil hat, benutzt man manchmal auch das Kunstwort „Elter“ anstatt „Eltern“.

Eine wichtige Eigenschaft bei der PPF-Baumstruktur (und bei vielen anderen Baumstrukturen auch) ist die der Vererbung. Kinder erben alle Eigenschaften vom Elter, können sie aber mit neuen Werten versehen. Darüber hinaus können sie aber auch

```
%!PS-Adobe-3.0
...
CIP3BeginSheet
    /CIP3AdmArtist (Carl Cool) def
    /CIP3AdmJobCode (8) def
    /CIP3AdmJobName (8 Zustaende) def
    /CIP3AdmPaperExtent [ 1984.251968 1417.32283 ] def
    /CIP3AdmSheetName (FB 002) def
    /CIP3AdmTypeOfScreen (amplitude modulated) def
    /CIP3AdmPaperGrammage 100.0 def
    /CIP3AdmPaperThickness  0.035 mm def
    /CIP3AdmPaperColor [ 93.0  0.0 -3.0 ] def
    /CIP3AdmCreationTime (Mon Dec 11 18:29:57 2006) def
     ...
CIP3EndSheet
%%CIP3EndOfFile
```

Abbildung 4.9
Details einer PPF-Datei

neue Eigenschaften besitzen. Ein Beispiel: Die Knoten „Schön“ und „Wider“ sind Kinder vom „Bogen“. Die Eigenschaften vom Bogen, wie beispielsweise die Größe (*CIP3AdmPaperExtent*) und die Grammatur (*CIP3AdmPaperGrammage*), erben sie von ihm. Vielleicht erben sie auch die Eigenschaft Rasterart (*CIP3AdmTypeOfScreen*), aber sowohl Schön oder Wider könnten den Rastertyp auch überschreiben. Schließlich könnte aber auch die Rasterart überhaupt nicht im Bogen definiert sein, sondern nur in den beiden Kindern.

Natürlich können auch mehrere Bogen innerhalb einer PPF beschrieben werden, dann sind sie Kinder eines anderen Wurzelementes, das *PPF Directory* heißt.

Die Baumstruktur wird im PS-Code übrigens durch eine Um-

Abbildung 4.10
Definition eines Voransichtsbildes in PPF

```
CIP3BeginPreviewImage
    %%Page: 1
    %%PlateColor: Cyan
    CIP3BeginSeparation
        /CIP3PreviewImageWidth 1490 def
        /CIP3PreviewImageHeight 1210 def
        /CIP3PreviewImageBitsPerComp 8 def
        /CIP3PreviewImageComponents 1 def
        /CIP3PreviewImageMatrix [1490 0 0 -1210 0 1210] def
        /CIP3PreviewImageResolution [ 50.800 50.800 ] def
        /CIP3PreviewImageEncoding /Binary def
        /CIP3PreviewImageCompression /RunLengthDecode  def
        /CIP3PreviewImageDataSize 515348 def
        CIP3PreviewImage...die Bilddaten
        CIP3EndPreviewImage
    CIP3EndSeparation
    analog für die Separationen Magenta, Yellow und Black
CIP3EndPreviewImage
```

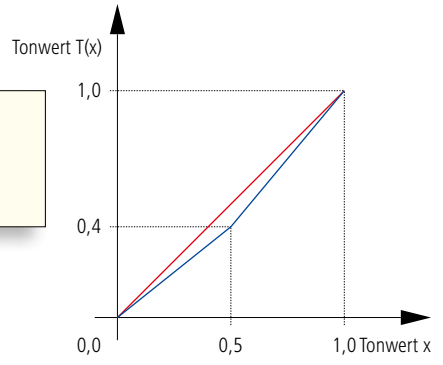
klammerung realisiert. Abbildung 4.8 zeigt die Baumstruktur von 4.6 in dieser Weise.

In Abbildung 4.9 sieht man noch einmal einen etwas ausführlicheren Code-Ausschnitt einer PPF-Datei. Die erste und die letzte Zeile beginnen jeweils mit einem Prozentzeichen, das bei PostScript ein Kommentarzeichen ist. Das heißt, die Zeilen sind nur PS-Kommentare und werden nicht vom PS-Interpreter interpretiert. Die folgenden Zeilen sind mehr oder minder selbsterklärend. Das Papierformat ist in DTP-Punkt angegeben, es hat also die Breite von 1984,251968·2,54/72 = 70 cm und die Höhe von 1417,32283·2,54/72 = 50 cm. Die Grammatur ist wie üblich in g/m² definiert, die Papierfarbe in CIE-L*a*b*.

Die bekannteste Anwendung für PPF ist die Farbzonenvoreinstellung für Offsetdruckmaschinen. Das PPF nimmt dabei eigentlich aber eine relativ bescheidene Rolle ein, denn es werden nur die Voransichtsbilder (*Preview*) der separierten Montagebogen und einige weitere Informationen im PPF gespeichert, die es einer speziellen Interface-Software ermöglichen, hieraus die Zonenwerte zu errechnen (siehe Abbildung 1.3). Die Zonenvoreinstellungswerte sind also nicht Teil des PPF-Standards. Abbildung 4.10 zeigt einen Ausschnitt eines solchen Preview-Bildes als PPF-Struktur. Die Bilddaten selber

Abbildung 4.11
Transferkurven in PPF

```
A  /CIP3TransferPlateCurveData
     [0.0 0.0 1.0 1.0] def
B  /CIP3TransferPlateCurveData
     [0.0 0.0 0.5 0.4 1.0 1.0] def
```

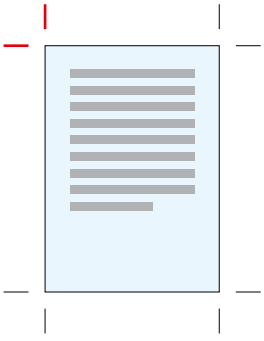



sind lauflängenkomprimiert und in ASCII85-kodiert. Die Breite (*CIP3PreviewImageWidth*) und die Höhe (*CIP3PreviewImageHeight*) ist in Pixeln, die Auflösung (*CIP3PreviewImageResolution*) ist in Pixel per Inch (ppi) angegeben. Letztere beträgt 50,8 ppi. Die *CIP3PreviewImageMatrix* gibt die Pixellaufrichtung im Bild an, hier ist sie von links nach rechts und von oben nach unten definiert. Die Definition von Matrizenrechnung findet man in der PS- und in der PDF-Spezifikation beschrieben (siehe auch die Übung am Endes des neunten Kapitels).

Tatsächlich genügt es nicht, nur die Preview-Daten an die Berechnungssoftware für die Farbzonenvoreinstellungswerte zu geben. Denn normalerweise findet noch nach der Berechnung des Preview-Bildes für den Montagebogen eine Veränderung der Tonwerte statt. Und schließlich sind es ja die Tonwerte, die für die Berechnung der Farbzonenvoreinstellungswerte wichtig sind. Doch wer oder was verändert die Tonwerte? Für die Herstellung von Offsetdruckplatten muss das Bild, das ja noch eine Farbtiefe hat, gerastert werden (im Englischen: *Screening*). Doch beim Rastern wird meist noch eine Tonwertmodifikation vorgenommen. Die Tonwertmodifikation soll beispielsweise die Tonwerte einem standardisierten Tonwertzuwachs im Druck anpassen. Im Sprachgebrauch von PS und PDF nennen sich diese Kurven *Transferkurven*, ansonsten auch abhängig vom Zweck Linearisierungskurven, Prozesskalibrierungskurven, Tonwertkompensationskurven oder im Flexodruck auch Bump-up-Kurven. In 4.11 sind zwei Transferkurven und deren Beschreibung in PPF dargestellt. Es wird immer ein Satz von Koordinatenpaaren aufgelistet (nämlich x und $T_{(x)}$), zwischen denen interpoliert wird, um die restlichen Funktionswerte zu definieren. Die Kurve

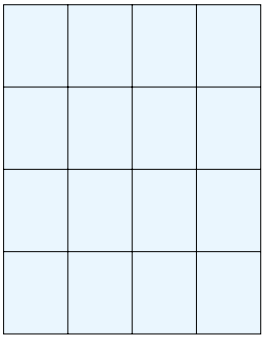
Abbildung 4.12
alternative Möglichkeiten zur Angabe von Schneidinformationen in PPF

Angaben der Schnittlinien über Schneidemarkenposition



```
29.4 cm 5.0 cm /TopVerticalCutMark CIP3PlaceCutMark
29.4 cm 5.0 cm /LeftHorizontalCutMark CIP3PlaceCutMark
```

oder alternativ über Schneideblöcke



```
/CIP3BeginCutBlock
    /CIP3BlockTrf [1 0 0 1 1 4 cm 4 cm] def
    /CIP3BlockSize [29.9 cm 68.7 cm] def
    /CIP3BlockElementSize[10.1 cm 6.02 cm] def
    /CIP3BlockSubdivision [4 4] def
    /CIP3BlockType /CutBlock def
    /CIP3BlockElementType /Unknown def
    /CIP3BlockName (Block1) def
/CIP3EndCutBlock
```

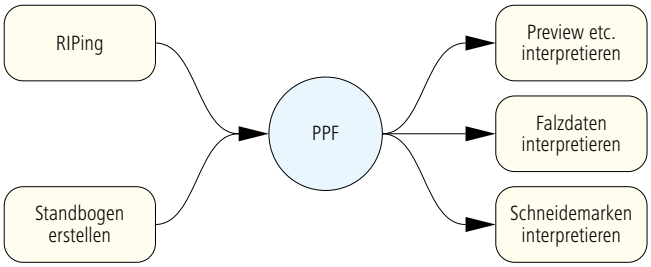



Abbildung 4.13
PPF-Produzent und PPF-Konsument

A ist eine Transferkurve, die keine Tonwertveränderungen vornimmt, während Kurve B die Tonwerte in den Mitteltönen reduziert.

Ein Preview-Bild und die Transferkurven können von einem RIP bereitgestellt werden, Schneide-, Register-, Farb- und Falzmarken hingegen nicht, da diese beim RIP nur noch als bedeutungslose, allgemeine Grafiken ankommen. Nur die Montagesoftware kennt den Sinn dieser Marken und kann entsprechende PPF-Informationen zusammenstellen. In die PPF-Datei werden dabei für Register-, Farb-, und Schneidemarken die Positionen auf dem Bogen angegeben. Anstatt Schneidemarken können jedoch auch Schneideblöcke in eine PPF-Datei eingetragen werden. Hierbei werden keine Schneidreihenfolgen definiert, sondern nur einzelne Nutzengrößen, die geschnitten werden müssen, wie in Abbildung 4.12 zu sehen ist. Bei den Falzdaten kann hingegen auch die Falzsequenz festgelegt werden.

Bisher bezogen sich alle PPF-Eigenschaften auf Druckbogen. Hierauf war in der Tat der Wirkungsbereich des Formats bis zur Version 2.1 auch beschränkt. Mit der Version 3.0, die 1998 herauskam und die letzte Version dieses Jobticket-Formats darstellt, wurden weiterreichende Produktdefinitionen möglich. Es konnten dann sogar mehrere Teilprodukte eines Gesamtproduktes in einer PPF-Datei beschrieben werden. Insgesamt folgte man mit der Einführung von „Produktoperationen“ (/*CIP3ProductOperation*) und „Produktparameter“ (/*CIP3ProductParams*) dem Prozess-Ressourcen-Modell, das in Kapitel 3 vorgestellt wurde. Eine Reihe von Produktoperationen, wie Zusammentragen, Klammern, Fadenheften oder die Durchführung eines Drei-Seiten-Beschnitts, sind in Version 3.0 definiert.

Bei den vorgestellten Beispielen hatten wir zwei PPF-Produzenten kennengelernt, nämlich den RIP und die Montage-Software sowie drei PPF-Konsumenten, nämlich die jeweiligen PPF-Interface-Module für die Druckmaschine, die Falzmaschine und den Planschneider. Damit ergibt sich in der Theorie das Strukturbild, das in 4.13 zu sehen ist. Leider ist es in der Praxis nicht so einfach, für unterschiedliche PPF-Produzenten zusammen eine PPF-Datei zu erzeugen. Die PPF-Spezifikation gibt hier auch keine Vorgaben, wie das geschehen soll, und

es gibt nur herstellerspezifische Lösungen. Somit hat man es in der Praxis bei einem herstellerübergreifenden Workflow eher mit mehreren PPF-Dateien pro Druckauftrag zu tun, vorausgesetzt, es sind mehrere PPF-Produzenten im Spiel. Da aber auch ein PPF-Konsument unter Umständen von unterschiedlichen PPF-Produzenten Informationen benötigt, wird die Situation etwas unübersichtlich. Ein mögliches Beispiel hierfür ist das PPF-Druckmaschinen-Interface, das einerseits Preview-Bilder vom RIP, andererseits Positionsangaben über die Registermarken von der Montage-Software erhält. Damit wird der PPF-Workflow bei steigender Funktionalität komplex und schwer zu warten: Es müssen diverse Hotfolder eingerichtet und im Fehlerfall kontrolliert werden. Das Archivieren der verstreuten PPF-Dateien ist schwerfällig.

Insgesamt ist es vielleicht deswegen nicht verwunderlich, dass der PPF-Workflow sich häufig ausschließlich auf die Weitergabe der RIP-Daten für die Farbzonenvoreinstellung beschränkt, weil da der Nutzen (Verringerung der Makulatur und Rüstzeit) in Relation zum Aufwand sehr hoch ist.

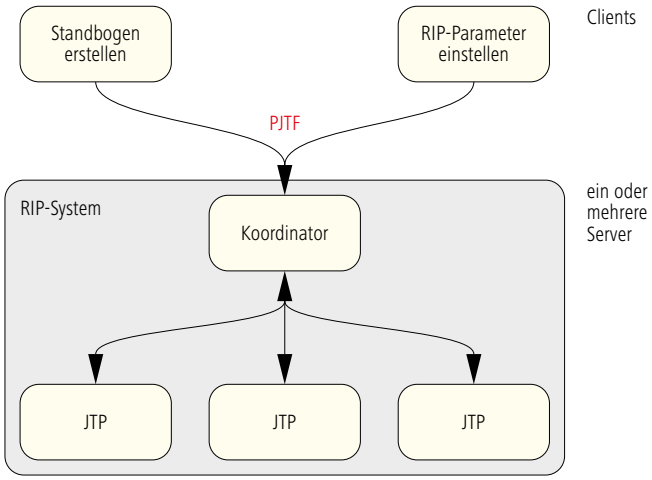



Abbildung 4.14
Architektur des Extreme-RIPs

## 4.3 Portable Jobticket Format (PJTF)

Das *Portable Jobticket Format* [6] von Adobe Systems Inc. ist ebenfalls ein Vorgängerformat vom JDF. Dieses Format ist eng gekoppelt an die *Extreme-RIP*-Architektur, die, wie auch das PDF, von Adobe entwickelt wurde.

Die Idee der *Extreme-RIP*-Architektur besteht darin, die im RIP angesiedelten Funktionen als unabhängige Module zu definieren, welche über PJTF mit den nötigen Metadaten versorgt werden. Die Module verarbeiten also Jobtickets und werden deswegen auch „Jobticket-Prozessoren“ (JTP) genannt. JTP-Beispiele sind einerseits die klassischen Aufgaben eines RIPs, also *Interpreting*, *Rendering* und *Screening*, aber auch erweiterte Funktionen wie Normalisieren, Farbraumtransformationen, Trapping, Ausschießen, Berechnung der Proof-Daten oder Erstellung einer PPF-Datei zur Farbzonenvoreinstellung. Die JTPs

können auf einem Server, aber auch auf unterschiedlichen Servern in einem verteilten System ablaufen. Vielfach wird so das Trapping-JTP auf einen eigenen Server ausgelagert, da es ein recht zeitaufwändiger Vorgang ist, der viel CPU- und Arbeitsspeicherkapazität benötigt. Auch mehrere Instanzen von einem JTP auf einem oder mehreren Servern werden häufig eingerichtet, um den Durchsatz insgesamt zu verbessern. Die *Extreme-RIP*-Architektur war über viele Jahre die Grundlage für die meisten Workflow-Managementsysteme im CtP-Bereich und wird erst seit 2006 durch die neue Adobe *Print Engine* Technologie nach und nach abgelöst.

Abbildung 4.15
ein PJTF-Objekt

```
13 0 obj
<<
    /BCL 0.95
    /ITO true
    /TW 0.2
    ...
>>
endobj
```

Die JTPs werden von einem Koordinator-Modul gesteuert, das einerseits die Jobtickets von außen empfängt und andererseits die Information an die JTPs weiter gibt. In der Regel werden die Standbogenbeschreibungen sowie die spezifischen Einstellungen für die unterschiedlichen JTPs von einer Client-Software aus an den Koordinator übertragen. Ein Anbieter eines Extreme-RIP-Systems kann eigene JPTs in das System integrieren oder auch auf die von Adobe zurückgreifen. In jedem Fall muss der Koordinator von Adobe lizenziert werden. Der Anwender kann keine fremden JTPs (beispielsweise von anderen Herstellern) in ein vorhandenes System integrieren, da die Schnittstellen nicht offen gelegt sind: Nur die Inhalte der PJTF-Daten sind festgelegt, nicht jedoch, wie diese zu den JTPs gelangen (Abbildung 4.14).

Folgende Dinge können im PJTF beschrieben werden:

- Daten zum Auftragsmanagement (Auslieferungsadresse, Drucktermin...)
- Beschreibung des Standbogens
- Einstellungswerte für Prozesse im Bereich Ausgabe-Workflow (Preflight, Trapping, Rastern...)
- Einstellungswerte für Prozesse der Druckverarbeitung.

Der Fokus liegt aber eindeutig auf der Standbogenbeschreibung und dem CtP-Ausgabe-Workflow; alle anderen Bereiche sind in der Spezifikation eher vernachlässigt worden.

In den vorherigen Abschnitten wurde ausgeführt, dass das XMP-Format in XML und das PPF in PostScript kodiert wird. Das PJTF basiert nun auf PDF, das heißt eine PJTF-Datei ist von der Struktur her eine PDF-Datei, nur dass PJTF-spezifische Schlüsselwörter Verwendung finden. Diese sind allerdings dem

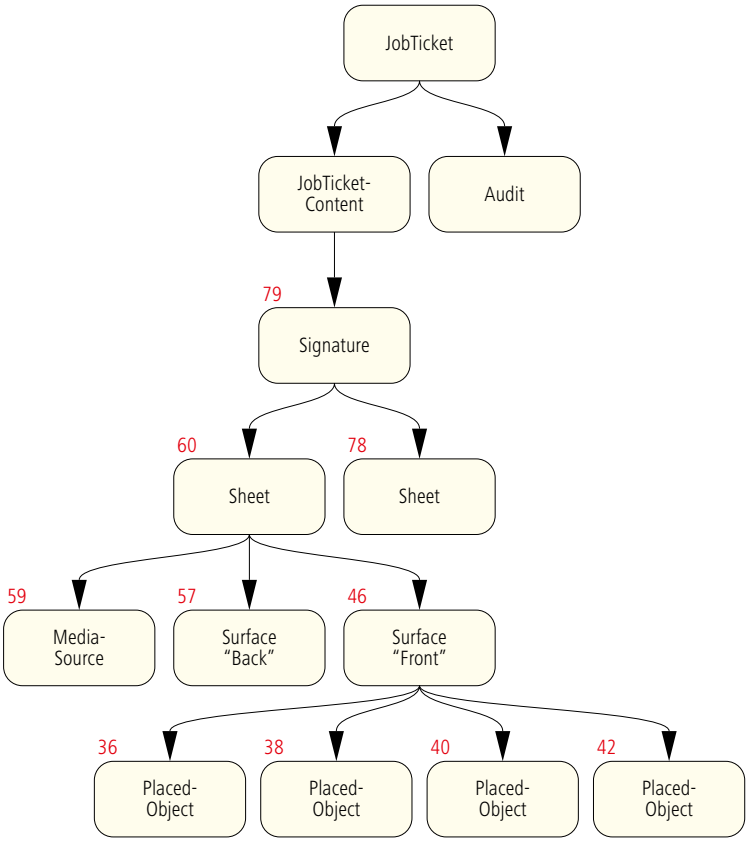



Abbildung 4.16
Aufbau eines PJTF-Dokumentes

Programm Acrobat (von Adobe) nicht bekannt, so dass deren Werte nicht angezeigt werden können. Eine PDF-Datei besteht generell aus einer Sammlung von nummerierten Objekten und einer *Crossreference*-Tabelle, in der eingetragen ist, wo genau die einzelnen Objekte in der Datei zu finden sind. Abbildung 4.15 zeigt ein PJTF-Objekt mit der Nummer 13. Das Objekt enthält ähnlich wie beim PostScript-kodierten PPF eine Art Tabelle, in der pro Zeile immer der gleiche Aufbau vorliegt: links ist eine Zeichenkette und rechts der Wert dieser Zeichenkette. Diese Struktur lässt sich auch als Schlüsselwort (oder Variablenname) und Wert interpretieren oder auch als Eintrag in einem (ziemlich seltsamen) Wörterbuch. Tatsächlich nennt Adobe diese Struktur auch *Dictionary*, also Wörterbuch. Jedes *Dictionary* wird durch doppelte spitze Klammern eingerahmt (<< und >>). Das gilt für PostScript genauso wie für PDF.

Das Objekt 13 in Abbildung 4.15 definiert einige Details zum Trapping; die Schlüsselwörter sind in der PJTF-Spezifikation definiert. So steht

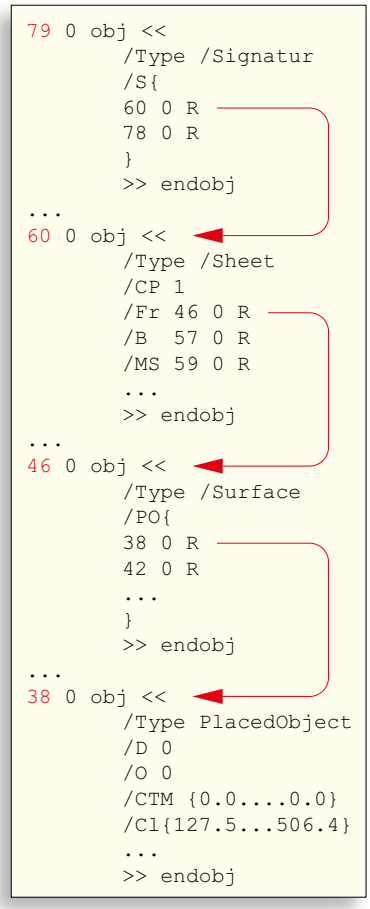



Abbildung 4.17
Verweise zwischen den Objekten

/BLC für *Black Color Limit* und der Wert von 0.95 besagt, dass jeder Tonwert der Farbe Schwarz über 95% wie die Volltonfarbe Schwarz überfüllt/unterfüllt werden soll. Und der Schlüssel /ITO und der Wert *true* besagten, dass Bilder zu anderen Objekten überfüllt oder unterfüllt werden sollen (*ImageToObjectTrapping*). Die Trap-Breite, im Englischen *TrapWidth*, wird mit /TW gekennzeichnet und in $^{1}/_{72}$ Inch angegeben.

Wie XMP und PPF ist auch das PJTF in einer Baumstruktur aufgebaut. Die Referenz zwischen den Knoten wird durch eine Referenz von einem Objekt zu einem anderen realisiert. Die Referenz wird durch ein „R“ gekennzeichnet, wobei davor die Objektnummer des Objekts steht, auf das verwiesen wird.

Wir wollen nun einen Ast genauer untersuchen, der einen Standbogen beschreibt. Jedes Rechteck in Abbildung 4.16 stellt in dem PJTF-Beispiel ein Objekt dar, jeder Pfeil symbolisiert den Verweis auf ein anderes Objekt. Die IDs der Objekte sind oberhalb der Rechtecke rot vermerkt.

Das Wurzelelement in PJTF ist das Objekt mit dem Namen *JobTicket*. Es verweist auf zwei Objekte, den *JobTicketContent* und das *Audit*-Objekt. Im *Audit*-Objekt werden alle Änderungen in der PJTF-Datei protokolliert, die im Laufe der Produktion vorgenommen werden. Das *JobTicketContent*-Objekt verweist unter anderem auf das *Signature*-Objekt, in dem Bogen mit gleichem Ausschießschema zusammengefasst werden können. In der Abbildung nicht zu sehen sind weitere Referenzen des *JobTicketContent*-Objekts zum Beispiel auf Objekte, die angeben, an welchem Speicherort im Dateisystem sich diejenigen PDF-Dateien befinden, in denen die Marken für den Standbogen plat-

ziert sind. Das *Signature*-Objekt 79 besteht aus zwei *Sheet*-Objekten 60 und 78. Das heißt, in dem Beispiel umfasst die Signatur zwei Druckbogen. Das Objekt *Sheet* mit der Nummer 60 referenziert seinerseits wieder auf zwei Objekte vom Typ *Surface* (Oberflächen), nämlich einmal für *Front* (Schön) und einmal für *Back* (Wider), die durch die Objekte 57 und 46 repräsentiert werden. Außerdem gibt es einen Verweis auf das *MediaSource*-Objekt mit der Kennzeichnung 59, unter dem sich Angaben über Bogengröße, Bogenfarbe und dergleichen befinden. Auch hier ist der Baum nicht restlos dargestellt. Das Objekt 46 zeigt auf die Objekte vom Typ *PlacedObject*, die Eigenschaften der Seitenplatzhalter wie Position, Größe und Beschnitt beinhalten.

Der tatsächliche Code der Objekte 79, 60, 46 und 38 ist etwas verkürzt in Abbildung 4.17 zu sehen, wobei die Referenzen noch mit Pfeilen grafisch hervorgehoben sind. In der ersten Zeile jedes Objekts ist seine ID definiert, außerdem wird das Dictionary mit << geöffnet. In der zweiten Zeile wird angegeben, um welchen Typ von Objekt es sich handelt, also *Signature*, *Sheet*, *Surface* oder *PlacedObject*. Alle sonstigen Einträge in den jeweiligen Dictionaries sind objektspezifisch und hier nicht weiter ausgeführt. Als Beispiel soll nur der Eintrag /CTM{0.0...0.0} in Objekt 38 dienen, der die C*urrent Transformation Matrix* definiert, welche die Größe und Position der Seitenplatzhalter auf dem Bogen angibt. Jedes Dictionary wird mit >> und jedes Objekt mit dem Schlüsselwort *endobj* geschlossen.

Zusammenfassend lässt sich sagen, dass das PJTF sicherlich in der Zukunft komplett vom JDF abgelöst wird, zurzeit aber noch einige Workflow-Managementsysteme beispielsweise PJTF-Standbogen importieren.

---

**Übungen**

- Öffnen Sie ein Digitalfoto in Photoshop und füllen Sie die Dateiinformationen aus.
- Platzieren Sie das Foto in InDesign.
- Exportieren Sie das InDesign-Dokument in PDF.
- Untersuchen Sie die PDF-Datei in Acrobat hinsichtlich XMP-Daten. Selektieren Sie hierzu das Bild und lassen Sie sich im Kontextmenü die Bildeigenschaften anzeigen.
- Untersuchen Sie die PDF-Datei mit einem geeigneten Texteditor, wie z.B. WordPad, MFC Anwendung. Suchen Sie nach „xmp“ und untersuchen Sie die dortigen Einträge.

# 5 Kurzeinführung in XML

Die *Extensible Markup Language* (XML) – zu deutsch: erweiterbare Auszeichnungssprache – ist eigentlich keine Sprache, sondern vielmehr ein System, Sprachen zu definieren, also eine Metasprache. XML ist auch ein Standard zur Definition von Dokumententypen, die zwischen Programmen ausgetauscht werden sollen. Ein Beispiel für einen solchen XML-Dokumententyp ist das Job Definition Format. Das ist der einzige Grund, warum wir uns hier mit XML beschäftigen. Es geht also in diesem Kapitel nicht darum, das Thema XML erschöpfend zu behandeln, sondern nur die nötige Terminologie vorzustellen, damit man JDF leichter versteht. Wir werden auch nicht Vorzüge und Nachteile von XML im Vergleich zu anderen Dokumentenstrukturen (wie PDF oder PostScript) behandeln, das Verhältnis zu der *Standard Generalized Markup Language* (SGML) oder die Unterschiede zur *Hypertext Markup Language* (HTML) diskutieren. Eine etwas genauere, aber immer noch sehr komprimierte Einführung findet man zum Beispiel in [42].

Wir werden also in Abschnitt 5.1 nur sehr pragmatisch den Aufbau eines XML-Dokuments erläutern und Begriffe wie Elemente, Attribute und Werte erklären. In 5.2 geht es um die Möglichkeiten, das „Vokabular“ von XML-Dateien zu bestimmen, die in Namensräumen zusammengefasst werden. Schließlich wird in Abschnitt 5.3 die *commerce eXtensible Markup Language* (cXML) vorgestellt, ein XML-basierender Standard zum elektronischen Austausch von Handelsdaten. Die cXML wiederum bildet die Grundlage für Teile der JDF-Spezifikation.

## 5.1 Aufbau eines XML-Dokuments

Allgemein beginnt eine XML-Datei mit der sogenannten Deklaration, ein Prolog vor den eigentlichen XML-Elementen.

```
<?xml version="1.0" encoding="UTF-8" ?>
```

Abbildung 5.1
Deklaration eines XML-Dokumentes

Diese Deklaration ist daran zu erkennen, dass innerhalb der spitzen Klammern (< und >) noch Fragezeichen (?) stehen. In Abbildung 5.1 steht dann noch in der Deklaration die Angabe der Versionsnummer von XML, nämlich 1.0. Seit Februar 2004 ist zwar die Version 1.1 freigegeben, trotzdem wird vielfach noch die Version 1.0 benutzt. Die Versionsnummer ist durch ein Attribut dokumentiert, das aus einem Attributnamen (hier *version*) und dem in Gänsefüßchen eingeklammerten Attributwert

(hier „1.0“) besteht. Der Attributwert wird durch ein Gleichheitszeichen = dem Attributnamen zugeordnet. Man nennt das auch „Wertzuweisung“. Häufig spricht man aber auch von einem Attribut und nennt dabei nur den Attributnamen – redet also beispielsweise von dem Attribut *version*. Die einzelnen Attribute sind durch ein Leerzeichen voneinander getrennt.

Das Attribut *version* ist obligatorisch, muss also immer in einer XML-Deklaration stehen, alle weiteren Attribute sind optional. In dem Beispiel findet man das optionale Attribut *encoding*, das Informationen über die Kodierungsart des vorliegenden XML-Dokuments bereitstellt. UTF-8 ist eine internationale Kodierung auf Basis der ISO/IEC-10646-Norm mit mindestens 8 Bit Zeichenbreite. UTF steht übrigens für „UCS Transformation Format“ und UCS für „Universal Multiple-Octet Coded Character Set“. Der UCS-Zeichensatz ist angelehnt an den besser bekannten Unicode-Zeichensatz.

Meist kommt nach der Deklaration ein XML-Element, das Wurzelelement (*root element*) der XML-Datenstruktur. Bei JDF ist das Wurzelelement immer ein Element mit dem Namen JDF, wie wir im nächsten Kapitel sehen werden. Jetzt wollen wir uns jedoch nicht um JDF kümmern und erläutern stattdessen in Beispiel 5.2 ein anderes Wurzelelement mit dem Namen *Contact*. Die Elemente sind gewissermaßen die Bausteine der XML-Struktur. Jedes Element hat einen Namen, wie beispielsweise *Contact*, *Person* oder *ComChannel*. Elemente werden durch *Start-Tags* und *End-Tags* begrenzt, wobei die *Tags* durch spitze Klammern gekennzeichnet sind. Der *Start-Tag* des Elements *Contact* ist also `<Contact>`, das *End-Tag* ist `</Contact>`. Das *End-Tag* erkennt man an dem Schrägstrich /. Elemente können Unterelemente enthalten, so ist das Element *Person* beispielsweise ein Unterelement – auch Kindelement genannt – vom Elternelement

Abbildung 5.2
XML-Elemente

```
<Contact>
    <Person FirstName="Carl" FamilyName="Cool" NamePrefix="Herr">
        <ComChannel Locator="03475/101010" ChannelType="Phone"
         ChannelUsage="Private"  ChannelTypeDetails="Landline" />
        <ComChannel Locator="03475/101011" ChannelType="Phone"
         ChannelUsage="Business" ChannelTypeDetails="Landline" />
        <ComChannel Locator="carl.cool@frisch.de
         ChannelType="E-Mail" ChannelUsage="BusinessPrivate" />
    </Person>
    <Company OrganizationName="Frisch GmbH"/>
    <Address City="Eisleben" Street="Am Kaltenbach 3" Country="Deutschland"
     PostalCode="06295" />
    <!--Das ist eine einfache XML-Struktur, die zufälligerweise sogar JDF-Code
     ist -->
</Contact>
```

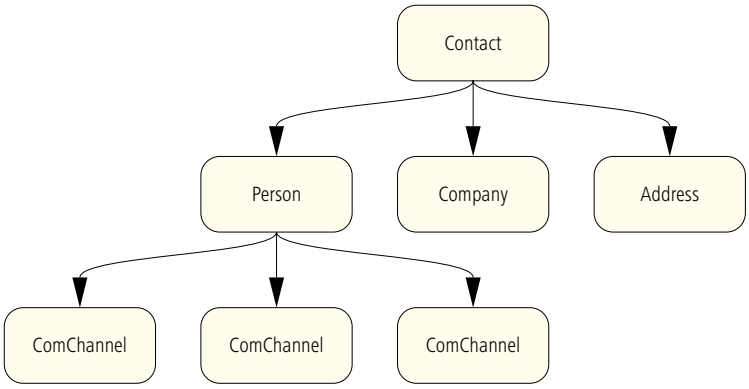



Abbildung 5.3
Baumstruktur der XML-Elemente

*Contact*. Die drei *ComChannel*-Elemente werden als Kinder von *Person* als „Geschwister“ bezeichnet, genauso wie auch *Person*, *Company* und *Address* Geschwisterelemente sind. Die Baumstruktur des Beispiels in Abbildung 5.2 ist deutlich in Abbildung 5.3 zu erkennen.

Nicht alle Elemente haben explizite *Start*- und *End-Tags*. Das *Company*-Element enthält keine weiteren Unterelemente und darf deswegen ohne *End-Tag* geschrieben werden. Allerdings muss vor der abschließenden spitzen Klammer ein Schrägstrich (/) stehen. Elemente, die keine weiteren Unterelemente haben, bezeichnet man übrigens als „leer“. Also nur leere Elemente können auch in der Form <Elementname.../> aufgeschrieben werden. Wie man in dem Beispiel erkennen kann, dürfen leere Elemente durchaus eigene Attribute enthalten.

Viele Elemente haben Attribute, jedoch nicht alle, wie man am Beispiel von *Contact* sieht. Welche Eigenschaft eines Elementes als Attribut geschrieben wird und welche in einem Unterelement definiert wird, muss von denen festgelegt werden, die einen Dokumententyp spezifizieren. So könnte man in unserem Beispiel natürlich auch die Attribute vom Element *Person* einfach als Attribute vom Element *Contact* definieren. Dann wäre das Element *Person* schlicht überflüssig und könnte entfernt werden.

Bei den Attributen und bei den Elementnamen wird zwischen Groß- und Kleinschreibung unterschieden. Auch sind nicht alle Zeichen in den Element- oder Attributnamen erlaubt, wie das Leerzeichen oder auch XML-eigene Zeichen wie die spitzen Klammern oder das Apostroph.

Es können Kommentare in ein XML-Dokument geschrieben werden, die nur für den Menschen zum Lesen bestimmt sind, nicht jedoch zur Verarbeitung durch XML-Software. Kommentare be-

ginnen mit <!-- und enden mit --> und können über mehrere Zeilen gehen.

## 5.2 XML-Namensräume

Datenaustausch mit XML-Dokumenten findet massenhafte Anwendungen in vielen Gebieten. Das geht von Beschreibungen chemischer Moleküle, der Seitenbeschreibungssprache XPS von Microsoft (*XML Paper Specification*) bis zu *Business-to-Business*-(B2B)-Schnittstellen für den Austausch von Geschäftsdaten wie Bestellungen, Rechnungen oder Produktkatalogen. Doch wer und wie werden die hierfür nötigen Strukturen, die Elementtypen und deren Attribute definiert? Dieses „Markup-Vokabular“ und die Regeln, wie diese zu verwenden sind, werden nicht von einer zentrale Stelle verwaltet, sondern jeder kann sein eigenes Modell definieren. Wenn XML-Dokumente zum Datenaustausch von unterschiedlichen Software-Modulen genutzt werden sollen, ist natürlich Einigung auf das Markup-Vokabular notwendig. Diese Namensvereinbarung wird durch die Deklaration eines „Namensraums“ (englisch: *name space*) im XML-Dokument getroffen. Nicht im jedem XML-Dokument muss jedoch ein Namensraum deklariert werden, zum Beispiel dann nicht, wenn es nur für betriebsinterne oder sogar für private Zwecke vorgesehen ist und keine Namenskonflikte durch doppelte Element- oder Attributnamen auftreten können. Bei JDF-Dokumenten werden aber meist sogar mehrere Namensräume deklariert: Einmal zur Bezeichnung und Struktur der allgemeinen JDF-Elemente und deren Attribute gemäß der CIP4-Spezifikation und aber auch zur Bezeichnung privater Ergänzungen, die Workflow-Hersteller vornehmen.

Ein XML-Namenraum wird durch das *xmlns*-Attribut deklariert. Der Wert des Attributs ist einfach ein Identifikator, genauer ein *Universal Resource Identifier* (URI). Dieser Identifikator, wie in Beispiel 5.4 gezeigt, sieht zwar in der Regel aus wie eine Internetadresse, bezeichnet aber eigentlich nur den Namen des

Abbildung 5.4
Namensräume in XML-Dokumenten

```
<?xml version="1.0" encoding="UTF-8" ?>
   <JDF xmlns="http://www.CIP4.org/JDFSchema_1_1"
    xmlns:HdM="www.hdm-stuttgart.de.com/schema/HdM"...>
      <ResourcePool...>
      ...
      </ResourcePool >
      <HdM:PrivateElement... />
   </JDF>
```

Namensraums. Hinter dem URI kann eine Ressource im Internet stehen, muss aber nicht. Mit anderen Worten: Der URI kann möglicherweise ein *Uniform Resource Locator* (URL) sein, im Allgemeinen ist er es aber nicht. In dem Beispiel ist, wie man sich leicht überzeugen kann, die erste Namensraumdeklaration ein URL, während die zweite nur ein URI ist (die Internetadresse existiert nicht).

In der zweiten Deklaration steht hinter dem Attributnamen *xmlns* durch einen Doppelpunkt getrennt noch der Präfix *HdM*, wobei *HdM* nur für „Hochschule der Medien“ steht. Mit Hilfe eines solchen Präfixes ist es dann möglich, Elementnamen oder Attributnamen verschiedenen Namensräumen zuzuordnen. Wenn ein Namensraum keinen Präfix enthält, ist er der Default-Namensraum. In dem Beispiel ist das Element *ResourcePool* ein Unterelement des Elementes JDF. Beide Elementnamen stammen aus dem Default-Namensraum, der ohne Präfix angegeben ist (http://www.CIP4.org/JDFSchema_1_1), während das Element *HdM:PrivateElement* mit dem zweiten Namensraum über das Präfix *HdM* verbunden ist.

Die Struktur eines XML-Dokumenttyps kann in einer *Document Type Definition* (DTD) oder in einem XML-Schema definiert werden, wobei das Schema moderner ist und weitreichendere Möglichkeiten bietet. Wir wollen uns jedoch nicht um die Einzelheiten kümmern, wie genau die Definition eines Dokumententyps vorgenommen wird. Es ist für das Verständnis der Zusammenhänge in diesem Buch auch nicht wichtig. XML Schemata werden manchmal auch mit XSD (*XML Schema Definition*) abgekürzt. Entsprechend heißt die Datei, die das JDF-Schema enthält, auch „JDF.xsd“ und kann von der CIP4-Webseite heruntergeladen werden. Da ein Schema selbst wieder in XML definiert und folglich als Text kodiert ist, kann es auch mit einem normalen Textverarbeitungsprogramm oder mit einem Browser gelesen und auch verändert werden.

In einem Schema wird üblicherweise festgelegt:

- welche Elemente in einem Dokument erlaubt sind,
- welche Attribute in diesen Elementen erlaubt sind,
- welche Attribute für ein Element obligatorisch und welche optional sind,
- die Elter-Kind-Beziehungen zwischen den Elementen,
- die Datentypen für Elemente und Attribute,
- Häufigkeiten von Elementen,
- Referenzierungen zwischen Elementen.

XML-Dokumente müssen *wohlgeformt* und *gültig* sein. Unter *Wohlgeformtheit* versteht man die Einhaltung der allgemeinen XML-Syntax, während eine Gültigkeit nur dann gegeben ist, wenn zusätzlich die Regeln, die in dem Schema (oder den Schemata) definiert sind, eingehalten werden. In dem XML-Dokument muss ausgewiesen werden, gemäß welchem Schema oder welchen Schemata es strukturiert ist.

Ein Programm, das XML lesen kann, wird *Parser* genannt, was mit „Analysierer" ins Deutsche übersetzt werden könnte: Ein Parser hat dabei nur die Aufgabe, den XML-Text zu analysieren und insbesondere auf seine Wohlgeformtheit zu untersuchen. Überprüft der Parser das XML-Dokument zusätzlich auf seine Gültigkeit, so wird er auch als „validierender Parser" bezeichnet. Der Vorteil von validierenden Parsern ist, dass XML-Dokumente, die nicht gemäß den angegebenen Schemata aufgebaut sind,

Abbildung 5.5
XMP in XML codiert

```
20 0 obj
...
<x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="XMP toolkit 2.9-9, framework
 1.6">
   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:x="http://ns.adobe.com/iX/1.0/">
      <rdf:Description rdf:about="" xmlns:pdf="http://ns.adobe.com/pdf/1.3/"
       pdf:Trapped="False" pdf:Producer="Acrobat Distiller 7.0 (Windows)"
       pdf:GTS_PDFXConformance="PDF/X-1a:2001"
       pdf:GTS_PDFXVersion="PDF/X-1:2001">
      </rdf:Description>
      <rdf:Description rdf:about="" xmlns:xap="http://ns.adobe.com/xap/1.0/"
       xap:CreateDate="2007-08-24T12:00:39+02:00"
       xap:CreatorTool="PScript5.dll Version 5.2"
       xap:ModifyDate="2007-08-24T12:00:39+02:00">
      </rdf:Description>
      <rdf:Description rdf:about=""
       xmlns:dc="http://purl.org/dc/elements/1.1/">
         <dc:creator>
            <rdf:Seq>
               <rdf:li>Carl Cool</rdf:li>
            </rdf:Seq>
         </dc:creator>
         <dc:title>
            <rdf:Alt>
               <rdf:li xml:lang="x-default">
                test.indd
               </rdf:li>
            </rdf:Alt>
         </dc:title>
      </rdf:Description>
   </rdf:RDF>
</x:xmpmeta>
...
endobj
```

sofort erkannt und ggf. ausgesondert werden können. Damit ist das Risiko reduziert, dass ein XML-Dokument nicht richtig verarbeitet werden kann. Ein validierender Parser ist also für XML so etwas wie ein Preflight-Check-Programm für PDF-Dateien.

### 5.3 Resource Description Framework (RDF) und XMP

Im vorherigen Kapitel wurde das XMP-Format vorgestellt und auch erwähnt, dass es in einer XML-Auszeichnungssprache geschrieben werden kann. Diese Sprache nennt sich *Resource Description Framework* (RDF) [44], ist also ein System zur Beschreibung von Ressourcen vor allem im Internet. Es geht hierbei meist um die Speicherung von Metadaten für Internetressourcen, also um zusätzliche Informationen, die im einfachsten Fall das Suchen im Web effektiver gestalten sollen. Darüber hinaus geht es aber um mehr, nämlich um das Konzept des *Semantischen Webs*, in dem die Meta-Informationen nicht nur von Suchmaschinen, sondern auch von anderen Programmen gelesen und ausgewertet werden können, von so genannten Web-Agenten.

Das RDF ist für JDF nicht wichtig – wir möchten es dennoch kurz vorstellen, um die Datenrepräsentierung von XMP beschreiben zu können. Wir wollen das Beispiel 5.5, das ein RDF/XMP-Objekt im PDF-Format zeigt, etwas genauer untersuchen: Das Element *<x:xmpmeta>* ist das Wurzelelement, das genau ein Kindelement *<rdf:RDF>* hat. Letzteres umschließt selber wiederum drei Kindelemente *<rdf:Description>*. In jedem *rdf:Description*-Element ist ein eigener Namensraum definiert: Beim ersten mit dem Präfix *pdf* werden Metadaten zur PDF-Datei abgelegt, beim zweiten mit dem Präfix *xap* geht es um das Erstellungs- bzw. Modifikationsdatum der Datei, beim dritten mit den Präfix *dc* um Urheber und Titel der Datei. „dc“ steht dabei für „Dublin-Core“, siehe Kapitel 4.

Ein paar Zeilen in dem Beispiel sind vielleicht noch unklar: Das Element *<rdf:Seq>* beschreibt eine geordnete Liste, jeder Listeneintrag ist ein *<rdf:li>*-Element. Die Liste besteht hier allerdings nur aus einem Eintrag. Das Element *<rdf:Alt>* gibt die Möglichkeit, alternative Werte anzugeben. Etwas merkwürdig erscheinen im ersten Moment die Attribute *rdf:about=““*. RDF wurde, wie gesagt, geschaffen, um hauptsächlich Internetressourcen zu beschreiben. Den Wert des Attributs *about* gibt dann der URI der entsprechenden Ressource im Internet an. Hier hat man je-

```
<cXML>
    <Header>
    Header Information
    </Header>

    <Request>
    Request Information
    </Request>
</cXML>
```

Abbildung 5.6
Aufbau eines cXML-Dokumentes

doch eine leere Zeichenkette, da es bei dem XMP-Beispiel nicht um Metadaten einer Internetressource geht, sondern um Metadaten einer PDF-Datei im lokalen Dateisystem, bei der sogar die XMP-Informationen innerhalb der PDF-Datei abgelegt sind.

## 5.4 Commerce Extensible Markup Language (cXML)

Produkt- oder Servicekataloge, Angebotsanforderungen, Angebote, Auftragserteilungen, Auftragsbestätigungen oder Rechnungen werden im Geschäftsleben zwischen den Geschäftspartner hin- und hergeschickt. Und das geschieht nicht nur über Briefe oder E-Mails, sondern zum Beispiel auch über Web-Portale oder andere E-Commerce-Systeme. In der grafischen Industrie sind die Geschäftspartner im einfachsten Fall der Druckeinkäufer und die Druckerei. Es gibt unterschiedliche Ansätze, diese geschäftlichen Transaktionen durch formale Festlegungen zu beschreiben. Die CIP4-Organisation, die für das Job Definition Format verantwortlich ist, propagiert hierfür eine XML-Auszeichnungssprache: *PrintTalk*. Die aktuelle Version von PrintTalk ist 1.3. Während man also die Einzelheiten eines Druckprodukts mit JDF beschreiben kann, können die kommerziellen Geschäftsvorgänge mit PrintTalk ausgedrückt werden. Genauer: In einer PrintTalk-Transaktion kann dann die JDF-Beschreibung des Druckproduktes integriert sein. Beide XML-Auszeichnungssprachen sind also eng verzahnt, wobei jede ihre speziellen Aufgaben hat. Damit ist es möglich, eBusiness-Modelle wie Web2Print zu implementieren. So können Kunden in einem Browser Druckprodukte in gewissen Variationen spezifizieren, Geschäftsdaten übermitteln und Aufträge erteilen.

In Abschnitt 8.4 werden wir PrintTalk genauer vorstellen. Da PrintTalk wiederum auf der allgemeineren *Commerce Extensible Markup Language* (cXML) [15] basiert, möchten wir deren Grundlagen in diesem Abschnitt kurz beschreiben.

cXML-Dokumente, enthalten kommerzielle Transaktionen und sind im Prinzip wie in Abbildung 5.6 aufgebaut. Das Root-Element ist immer ein cXML-Element, das typischerweise ein *Header*-Element und ein *Request*-Element enthält. Das *Header*-Element enthält Informationen zur Adressierung des Absenders und des Empfängers. Außerdem können Daten zur Authentifizierung (= Legitimierung oder Echtheitsprüfung) des Senders übermittelt werden – im Prinzip ein Passwort. Das Gerüst für einen *Header* ist in Abbildung 5.7 zu sehen. Die *From*- und

*To*-Elemente identifizieren den Erzeuger und den Empfänger des cXML-Dokuments. Die *From*- und die *Sender*–Instanz können, müssen aber nicht identisch sein. Denn das *From*-Element definiert den logischen Erzeuger des Dokuments, während das *Sender*-Element die Instanz, welche die http-Vebindung zum Empfänger aufbaut. Diese können unterschiedlich sein, wenn die cXML-Dokumente über große E-Commerce-Netzwerke geschleust werden. Beim *Sender*-Element wird dann auch ggf. das Passwort zur Authentifizierung eingetragen. Das *Request*-Element in dem Beispiel ist eine Bestellung eines Artikels. Auch diese hat zunächst einen allgemeinen Verwaltungsteil, den *OrderRequestHeader*, in dem beispielweise die Lieferadresse, die Rechnungsadresse, Informationen zur Steuer und zur Bezahlung eingetragen sind. Das *ItemOut*-Element enthält dann Details zu den Positionen, die bestellt werden.

```
<cXML>
    <Header>
        <From>
         ...
        </From>

        <To>
         ...
        </To>
        <Sender>
         ...
        </Sender>
    </Header>

    <Request>
        <OrderRequest>
            <OrderRequestHeader ...>
             ...
                <ShipTo>
                 ...
                </ShipTo>
                <BillTo>
                 ...
                </BillTo>
                <Tax>
                 ...
                </Tax>
                <Payment>
                 ...
                </Payment>
            </OrderRequestHeader>
            <ItemOut ...>
                <ItemID>
                 ...
                </ItemID>
            </ItemOut>
        </OrderRequest>
    </Request>
</cXML>
```

Abbildung 5.7
cXML-Dokument

Bei PrintTalk ist der Header identisch wie bei cXML. Auch das *Request*-Element findet man bei PrintTalk wieder. Dieses hat dann allerdings nicht mehr das Kindelement *OrderRequest*, sondern genau ein *BusinessObject*-Element. Ein solches Element beschreibt eines der oben genannten Transaktionen wie Angebotsanforderung, Angebot usw. Bei einigen *BusinessObjects*, wie z.B. bei der Angebotsanforderung, findet man dann auch ein JDF-Kindelement, in dem Informationen über das gewünschte Druckprodukt abgelegt sind.

# 6 Einführung in JDF

In den letzten Kapiteln haben Sie bereits einiges über JDF erfahren. Das Wichtigste sei hier noch einmal zusammengefasst:

- JDF ist ein Jobticketformat, in dem technische Informationen gespeichert werden können, das aber auch Funktionen des Auftragsmanagements unterstützt.
- JDF umfasst die Funktionalität von PPF und PJTF.
- JDF beruht auf dem Modell von Prozessen und Ressourcen.
- JDF basiert auf XML.

In diesem Kapitel wird in den ersten sechs Abschnitten der JDF-Code genauer vorgestellt. Sie stellen die Grundlage für alle weiteren Kapitel dar. Im letzten Abschnitt werden wir versuchen, den Schwerpunkt wieder mehr auf Workflows zu legen. Insbesondere werden wir die bereits in der Einleitung erwähnten *Interoperability Conformance Specifications* behandeln.

Diese JDF-Einführung soll einen Überblick über die Produktbeschreibungen und Festlegungen der Produktionsprozesse verschaffen, die mit JDF möglich sind. Am Ende des Kapitels sollte der Leser typische JDF-Dokumente lesen und verstehen können. Natürlich kann dieses Kapitel nicht die JDF-Spezifikation [13] ersetzen, die noch viel mehr Details enthält.

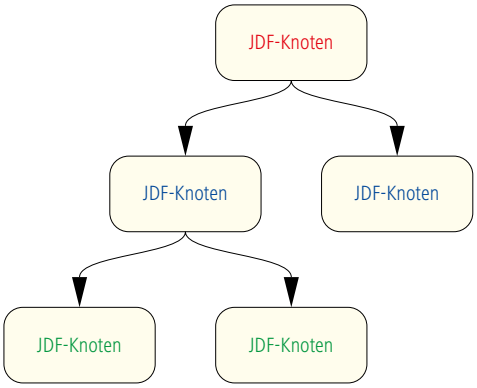
## 6.1 Aufbau eines JDF-Dokuments

Für jeden Druckjob gibt es im JDF-Workflow (mindestens) ein JDF-Dokument, das die Metadaten für den Druckjob enthält. Wie für ein XML-Dokument nicht anders zu erwarten, hat dieses JDF-Dokument eine Baumstruktur. Das Wurzelelement des Baums, aber auch jeder Ast und jedes Blatt, wird als JDF-Knoten (*JDF node*) bezeichnet. Normalerweise besteht ein Druckjob aus mehreren JDF-Knoten, wobei jeder Knoten einen definierten Teil des Jobs beschreibt. Jeder JDF-Knoten ist ein XML-Element, das selber wieder Unterelemente haben kann.

Natürlich müssen nicht wie in Abbildung 6.1 genau drei Hierarchiestufen vorliegen. Es können mal mehr oder auch mal weniger Stufen sein. Auf einer Stufe können beliebig viele andere JDF-Knoten liegen.

Nicht alle JDF-Knoten stellen Prozesse dar, sondern nur einige. In der Regel sind es die Blätter des Baumes. Insofern basiert

die JDF-Datenstruktur nur teilweise auf dem Prozess-/ Ressourcenmodell. Insgesamt gibt es nämlich vier unterschiedliche JDF-Knoten:




```
<JDF....>
   <JDF....>
      <JDF....>
      </JDF>

      <JDF....>
      </JDF>
      ...
   </JDF>

   <JDF....>
   </JDF>
   ...
</JDF>
```

Abbildung 6.1 zeigt verschachtelte JDF-Knoten

- Produktknoten (*procduct node* oder auch *product intent node*),
- Prozessgruppenknoten (*process group node*),
- kombinierte Prozesse (*combined processes*),
- Prozessknoten (*process node*).

Durch Produktknoten wird ein herzustellendes Druckprodukt mit seinen Teilen oder Teilprodukten beschrieben. Beispielsweise besteht ein Buch aus den Teilen Umschlag und Inhalt. Es gibt folglich drei JDF-Produktknoten: Buch, Umschlag und Inhalt.

Prozessknoten definieren einzelne Arbeitsschritte zur Herstellung des Druckproduktes oder eines seiner Teile. Wir haben bereits in Abschnitt 3.2 bis 3.4 die Prozesse *Interpreting*, *Rendering*, *Screening*, *Cutting* (Schneiden), *Folding* (Falzen), *Gathering* (Zusammentragen), *Stitching* (Klammern), *Trimming* (dreiseitiges Beschneiden), *Stacking* (Stapeln) kennen gelernt. Aber natürlich gibt es noch viele andere Prozesse und in den folgenden Kapiteln werden noch weitere vorgestellt.

Prozesse könne aus unterschiedlichen Gründen auch gruppiert oder auch kombiniert werden. Diese Art von JDF-Knoten nennt man dann Prozessgruppenknoten bzw. kombinierte Prozesse.

Das *RIPing* in Abbildung 3.8 könnte beispielsweise ein Prozessgruppenknoten oder auch ein kombinierter Prozess sein, der die drei Prozesse *Interpreting*, *Rendering*, *Screening* zusammenfasst (siehe Abb. 3.10). Für die genauen Unterschiede zwischen einem Prozessgruppenknoten und einem kombinierten

Prozess verweisen wir auf Abschnitt 6.4

*PlateMaking* (Plattenherstellung) in Abbildung 3.4 ist ein weiteres Beispiel für einen Prozessgruppenknoten, der unter anderem aus den Prozessen *Imposition* (ausschießen) und *ImageSetting* (belichten) besteht. Außerdem enthält er noch einen weiteren Prozessgruppenknoten, nämlich *RIPing*. Man sieht also, dass ein Prozessgruppenknoten sowohl Prozesse wie auch andere Prozessgruppen (die selber wieder Prozesse und Prozessgruppen zusammenfassen) enthalten kann. Dies ähnelt dem erneuten Gruppieren einzelner Grafikobjekte mit bereits gruppierten Grafikobjekten in Zeichenprogrammen.

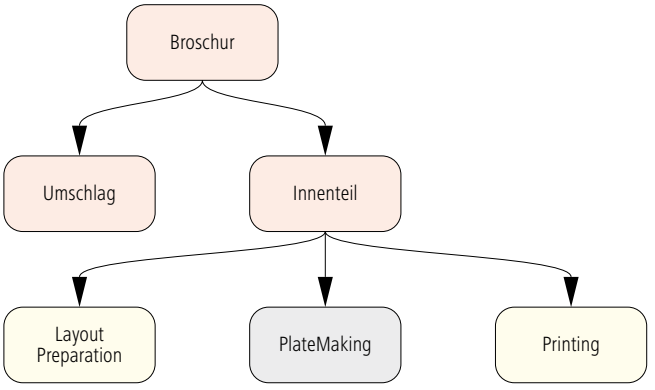
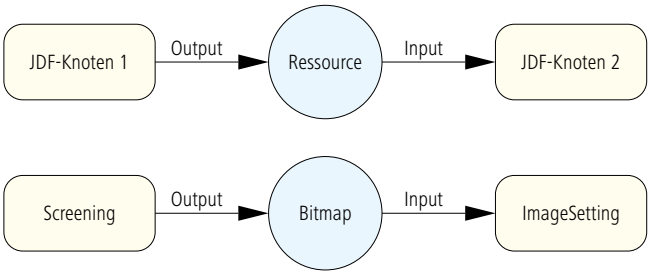



Abbildung 6.2 stellt den Zusammenhang der JDF-Knotentypen „Produktknoten“ (rot), „Prozessgruppenknoten“ (grau) und „Prozessknoten“ (gelb) dar.

Das Diagramm 6.2 zeigt den Zusammenhang zwischen den JDF-Knotentypen, wobei es in keiner Weise vollständig, sondern nur als Beispiel gedacht ist. Die roten JDF-Knoten stellen die Produktknoten dar, die grauen die Prozessgruppenknoten und die gelben die Prozessknoten. Ein kombinierter Prozess kommt in der Zeichnung nicht vor.

Es sei angemerkt, dass nicht in jedem JDF-Dokument diese drei JDF-Knotentypen vorkommen müssen. So kann es durchaus sein, dass ein JDF-Dokument vielleicht nur aus Produktknoten oder auch nur aus Produktknoten und Prozessgruppenknoten besteht. Ein JDF-Dokument eines kompletten Druckauftrages kann jedoch nicht ausschließlich aus Prozessknoten bestehen, da sich Prozesse immer auf Produkte oder Produktteile beziehen müssen. Mit anderen Worten: Das Wurzelelement ist in diesem Fall ein JDF-Produktknoten, der den gesamten Job repräsentiert.

Abbildung 6.3 Zusammenhang zwischen JDF-Knoten und Ressourcen




Während der Herstellung eines Druckproduktes wächst in einem JDF-Workflow das entsprechende JDF-Dokument. Wenn am Anfang die

Prozessknoten noch fehlen (weil die Prozesse noch nicht klar definiert sind), so werden sie nach und nach angefügt. Die Einträge werden, wie bereits in Abschnitt 3.2 beschrieben, zum Teil durch voreingestellte Defaultwerte der Produktion oder durch Eingaben der Anwender in der Produktion generiert.

Die JDF-Knoten existieren nicht für sich allein, sondern benötigen Ressourcen. Eine Ressource ist, wie wir bereits in Abschnitt 3.2 gesehen haben, ein physischer Gegenstand (Papier, Platte...) oder eine elektronische Einheit (Parametersatz, Datei...), die von einem Knoten verwendet (Input) oder erzeugt (Output) wird. Die Abbildung 6.3 zeigt noch einmal das Prinzip und ein sehr einfaches Beispiel. Wir haben aber in Kapitel 3 gesehen, dass normalerweise ein viel komplexeres Netz von Ressourcen und Knoten zur Beschreibung des Workflows vorliegt (siehe Abbildung 3.8). Doch nicht nur Prozessknoten, sondern auch Produktknoten haben Ressourcen. Sie beschreiben zum Beispiel Details über das für ein Teilprodukt vorgesehene Papier. Hier muss man sich also gedanklich von der anschaulichen Produzent-Konsument-Idee verabschieden, die bei Prozessen recht einleuchtend war. Man kann zwar noch die Vorstellung akzeptieren, dass beispielsweise beim Druckprozess Platten „aufgebraucht“ werden, nicht aber dass ein Produktknoten, wie ein Umschlag, seine Papierspezifikation konsumiert.

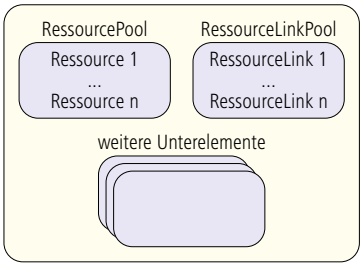



Abbildung 6.4 zeigt einen JDF-Knoten mit Unterelementen

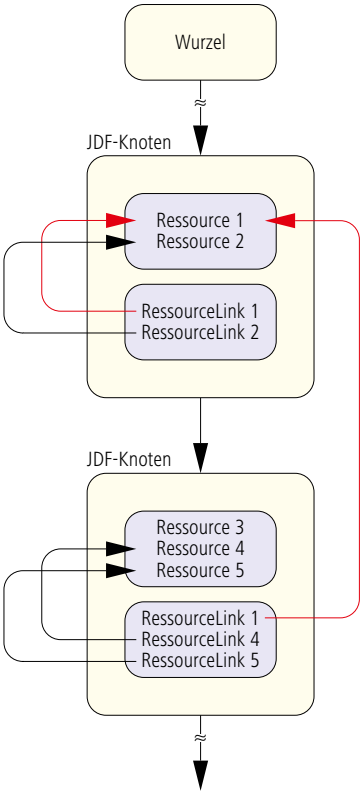



Abbildung 6.5, zwei JDF-Knoten greifen auf die selbe Ressource zu.

Doch wie stehen nun struktu-


Abbildung 6.6
Struktur eines JDFs mit einem Wurzel- und einem Unterknoten dar.

JDF-Knoten im Zusammenhang? Jeder JDF-Knoten hat nicht nur Attribute und deren Werte, sondern vor allem auch Unterelemente. Zwei Unterelemente sind hierbei besonders wichtig, der *ResourcePool* und der *ResourceLinkPool*. Der *ResourcePool* enthält Ressourcen für den JDF-Knoten, der *ResourceLinkPool* gibt an, welche Ressourcen für den JDF-Knoten verwendet werden und ob sie jeweils Input- oder Output-Ressourcen für den Knoten darstellen (Abbildung 6.4). Nicht jede Ressource, die von einem JDF-Knoten verwendet wird, muss im *ResourcePool* dieses Knotens stehen. Auch Ressourcen, die im Baum weiter oben – also näher zur Wurzel – in einem *RecourcePool* eines anderen JDF-Knotens enthalten sind, können benutzt werden. So können sich Prozesse auch eine Ressource teilen, ohne dass sie zweimal aufgeführt werden muss (Abbildung 6.5).

Es ist nicht einmal zwingend, dass jeder JDF-Knoten einen *ResourcePool* haben muss. Allerdings muss jedem JDF-Knoten ein *ResourceLinkPool* zugeordnet sein. Denn die *ResourceLinks* beziehen sich immer nur auf den JDF-Knoten, in dem sich der *ResourceLinkPool* befindet. Das heißt, hätte ein JDF-Knoten keinen *ResourceLinkPool*, so hätte er auch keinerlei Beziehung zu irgendeiner Ressource.

Abbildung 6.7
typische Attribute für ein JDF-Wurzelelement

```
<?xml version="1.0" encoding="UTF-8" ?>
<JDF ID="_4711" DescriptiveName="Frisch-Werbung" JobID="_42"
 Status="InProgress" Type="Product" Version="1.3"
 xmlns="http://www.CIP4.org/JDFSchema_1_1">
<!--  Generated by the CIP4 C++ open source JDF Library version
 CIP4 JDF Writer Java 1.3 -->
…
</JDF>
```

Die Struktur eines JDF-Dokumentes mit einem Wurzelelement und einem JDF-Unterknoten sieht typischerweise aus wie in Abbildung 6.6.

## 6.2 Beispiele für JDF-Knoten

Im Folgenden wird nun ein JDF-Dokument etwas genauer untersucht. Um jedoch diesen Abschnitt nicht zu überfrachten, haben wir viele Elemente und Attribute weggelassen und werden nur einige aufschlussreiche Ausschnitte präsentieren.

In Abbildung 6.7 ist in der ersten Zeile der XML-Prolog zu erkennen, wie er im letzten Kapitel vorgestellt wurde. Anschließend folgt das Wurzelelement mit dem Namen JDF, der rot markiert ist. Die Attributnamen sind in den Code-Beispielen immer grün eingefärbt.

Der JDF-Knoten hat mehrere Attribute, die kurz erläutert werden.

- Jeder JDF-Knoten muss innerhalb des Dokuments eine eindeutige *ID* haben.
- Der Wert für das optionale Attribut *DescriptiveName* ist eine für Menschen verständliche Bezeichnung des JDF-Knotens. In unserem Beispiel heißt der Knoten „Frisch-Werbung“, da das JDF-Dokument zu einem Druckjob gehört, bei dem es sich um eine Werbebroschur der Firma Frisch GmbH handelt.
- Die *JobID* ist die Bezeichnung des Jobs, wie sie das Auftrags-Managementsystem vergeben hat.
- Mit dem Attribut *Status* wird der aktuelle Zustand des JDF-Knotens wiedergegeben. Neben *InProgress* (in Berarbeitung) gibt es noch eine Reihe anderer Zustände, wie *Ready* (bereit), *Stopped* (angehalten), *Completed* (abgeschlossen), *Aborted* (abgebrochen) und weitere.
- Der JDF-Knoten ist vom Typ *Product*; es handelt sich

Abbildung 6.8
RessourcePool mit einer CustomerInfo-Ressource

```
<ResourcePool>
    <CustomerInfo ID="_4712"= CustomerID="_0815" Class= "Parameter"
     CustomerJobName="Frisch-Werbung" Status="Available" />
        <Contact ContactTypes="Customer">
            <Person FirstName="Carl" FamilyName="Cool" NamePrefix="Herr">
                <ComChannel Locator="03475/101010" ChannelType="Phone"
                 ChannelUsage="Private" ChannelTypeDetails="Landline" />
                 ...
            </Person>
            <Company OrganizationName="Frisch GmbH" />
            <Address City="Eisleben" Street="Am Kaltenbach. 3"
             Country="Deutschland" PostalCode="06295" />
        </Contact>
          ...
    </CustomerInfo>
     ...
</ResourcePool>
```

also um einen Produktknoten und somit nicht um einen Prozessknoten, einen Prozessgruppenknoten oder einen kombinierten Prozess. Ein Prozessgruppenknoten ist vom Typ *ProcessGroup*, ein kombinierter Prozess ist vom Typ *Combined*. Der Typ eines Prozessknotens jedoch bezeichnet den Prozess genauer, ist also *Interpreting*, *Cutting*, *Folding* oder dergleichen.

- Die *Version* gibt die Version der JDF-Spezifikation an, nach welcher der JDF-Knoten aufgebaut ist.
- Das Attribut *xmlns* definiert den XML-Namensraum (siehe Abschnitt 5.2).

Nach den Attributen und Werten folgt ein XML-Kommentar, der angibt, wie und mit welcher Computersprache (C++) und welcher Programmbibliothek (CIP4 JDF Writer Java 1.3) das Dokument verfasst wurde.

Wir wollen das Beispiel der Broschur Frisch-Werbung weiter verfolgen. Zunächst haben wir den *ResourcePool* und den *ResourceLinkPool* des Wurzelelements, wobei Abbildung 6.8 nur einen Ausschnitt zeigt. Als erste Ressource im Pool ist die *Customer-*

Abbildung 6.9 Ressource des Wurzelelementes

```
<Component ID="_4713" Class="Quantity" ComponentType="FinalProduct"
 DescriptiveName="Frisch-Werbung" Status="Unavailable" />
 ...
<DeliveryIntent Class="Intent" ID="_4714" Status="Available">
    <DropIntent>
        <DropItemIntent Amount="2000">
            <ComponentRef rRef="_4713" />
        </DropItemIntent>
    </DropIntent>
</DeliveryIntent>
```

*Info zu erkennen*, in der die Daten über den Auftraggeber gespeichert sind. Jede Ressource benötigt – genau wie auch jeder JDF-Knoten – eine eindeutige *ID*. Wir erkennen außerdem am Attribut *Class*, dass es sich um eine *Parameter*-Ressource handelt. Informationen über Menschen sind im JDF-Jargon nämlich von der Klasse *Parameter* und nicht physisch, wie man vielleicht erwarten könnte. Schließlich ist der *Status* der Ressource gleich *Available*, also benutzbar. Als Unterelement erkennen wir als Kontakt den bereits aus dem letzten Kapitel bekannten Carl Cool (Abbildung 5.2). Natürlich könnten dort weitere Kontaktpersonen vermerkt sein, z.B. wenn die Rechnungs- oder Lieferadresse abweicht.

Der *RessourcePool* enthält weitere Ressourcen. Die Abbildung

```
<ResourceLinkPool>
    <DeliveryIntentLink Usage="Input" rRef="_4714" />
    <ComponentLink Amount="2000.0" Usage="Output" rRef="_4713" />
    <ComponentLink Usage="Input" rRef="_4715" />
    <ComponentLink Usage="Input" rRef="_4716" />
</ResourceLinkPool>
```

Abbildung 6.10
ResourceLinkPool

6.9 zeigt die Ressourcen *Component* und *DeliveryIntent*. Die *Component*-Ressource hat eine ganz besondere Bedeutung und ist zwingend für jedes JDF-Wurzelelement vorgeschrieben. Denn diese Ressource repräsentiert das Endprodukt, das produziert werden soll. Als *Class* ist *Quantity* angegeben, was eine mögliche Unterart von physischen Ressourcen ist. Die Endprodukte sind abzählbar – im Gegensatz zu beispielsweise der physischen Ressource *Ink* (Farbe).

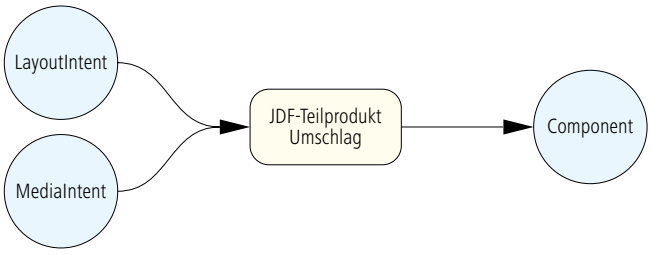
Die *DeliveryIntent*-Ressource beschreibt, wie und wann das Endprodukt zum Kunden ausgeliefert werden soll und gegebenenfalls auch wohin und womit. In unserem Beispiel ist nicht viel eingetragen. Es handelt sich nur um eine Lieferung, denn nur ein *DropIntent* ist angegeben. Es wird auch nur ein Produkt geliefert, da nur ein *DropItemIntent* verzeichnet ist, und davon 2000 Exemplare (*Amount*).

Die interessanteste Zeile ist jedoch die Referenz auf die *Component*. Es bedeutet, dass sich die Auslieferung beziehungsweise die Abholung auf das Endprodukt bezieht. Das Attribut *rRef* steht allgemein für eine Referenz auf eine Ressource, hier also auf die Ressource 4713, die *ID* des Endprodukts.

In Abbildung 6.10 ist der *ResourceLinkPool* zu sehen, also die Information, welche Ressourcen mit dem JDF-Knoten in Beziehung stehen und ob es sich um Input- oder Output-Ressourcen für den Knoten handelt.

Abbildung 6.11
separater JDF-Unterknoten „Umschlag“ mit eigenen ResourceLinks

Der Zusammenhang zwischen den einzelnen *ResourceLinks* und den Ressourcen selber ist einfach. Soll ein *ResourceLink* zu einer Ressource definiert werden, die den Namen *XYZ* und die *ID* gleich 333 hat, so wählt man *XYZLink* als Namen für den *ResourceLink* und gibt die Referenz mit *rRef* =“333“ an. In dem Beispiel wird so festgelegt, dass der Knoten die Input-Ressource *DeliveryIntent* und die Output-Ressource *Component* hat, was



unmittelbar einleuchtet.

In dem Beispiel sind noch zwei weitere Input-Referenzen (auf die Ressourcen 4715 und 4716), die wir noch nicht erklärt haben. Das JDF-Wurzelelement hat zwei JDF-Unterknoten, wobei der eine den Umschlag, der andere die Inhaltsseiten des Produktes repräsentiert. Beide Teilprodukte werden wieder durch *Component*-Ressourcen beschrieben. Da Umschlag und Inhaltsseiten benötigt werden, um das Endprodukt zu erhalten, sind die *Components* folgerichtig Input-Ressourcen für das Endprodukt.

Die JDF-Unterknoten haben jeweils wieder einen eigenen *Re-*

```
<JDF DescriptiveName="Umschlag" ID="_4715" Type="Product"   >
    <ResourcePool>
        <MediaIntent ID="_4717" Class="Intent" Status="Available"
        DescriptiveName="Papier für Cover">
            <Dimensions Actual="2437.7952755905512 1729.1338582677165"
             DataType="XYPairSpan"
             Preferred="2437.7952755905512 1729.1338582677165" />
            <Weight DataType="NumberSpan" Range="115 ~ 125" Preferred="120" />
        </MediaIntent>
    </ResourcePool>
<JDF>
```

Abbildung 6.12
MediaIntent-Ressource mit Informationen über den geplanten Bedruckstoff

*sourcePool* und einen eigenen *ResourceLinkPool*. Dort werden dann auch nähere Angaben über die Teilprodukte gemacht: Welches Papier soll eingesetzt werden? Wie steht es mit der Farbigkeit? Wie viele Seiten soll das Teilprodukt voraussichtlich umfassen?

Die *LayoutIntent*-Ressource gibt Dinge wie bevorzugte Größe des Endformats und Anzahl der Seiten an, die *MediaIntent*-Ressource beschreibt die Papiereigenschaften (Abbildung 6.11). Wenn der Sachbearbeiter oder die Sachbearbeiterin diese Informationen in ein MIS eingibt, sind häufig noch nicht alle Details restlos geklärt. Zum Beispiel könnte es sein, dass die genaue Seitenanzahl noch nicht feststeht oder die Grammatur der Papierbogen nur so ungefähr ausgemacht ist. Deswegen lassen diese Ressourcen auch gewisse Spannbreiten in den Angaben zu. *Intent* heißt im Englischen so viel wie „Absicht“ oder „Vorhaben“ und genau das wird in diesen *Intent*-Ressourcen auch beschrieben. *Intent*-Ressourcen findet man immer nur in Produkt-Knoten, die im Englischen auch häufig *Product Intent Nodes* genannt werden.

Es ist klar, dass diese beiden Ressourcen nicht für das Endprodukt, sondern – um in unserem Beispiel zu bleiben – für den Umschlag und den Innenteil separat definiert werden. Das muss auch so sein, da natürlich beide Teile in der Regel unterschiedliche Papiersorten verwenden, unterschiedliche Farbigkeit haben und so weiter.

Abbildung 6.12 zeigt die Ressource *MediaIntent* für den Umschlag. Die Ressource ist von der Klasse *Intent*. Zu beachten sind die beiden Unterelemente *Dimensions* und *Weight*.

Die Werte der beiden Attribute *Actual* und *Preferred* in dem Element *Dimensions* sind identisch. Es handelt sich um die Papiergröße in DTP-Punkten. Erst wenn man die Werte durch 72 dividiert und mit 2,54 multipliziert, kommt man zu den vertrauten Maßangaben in Zentimetern (hier: 86 x 61 cm). Unter *Preferred* ist der gewünschte Wert des Auftraggebers, unter *Actual* der beabsichtige Wert der Druckerei eingetragen. Beide sind hier offenbar gleich.

Im Attribut *Weight* ist die vom Auftraggeber gewünschte Grammatur in Gramm pro Quadratmeter vermerkt. Man erkennt, dass ein mögliches Intervall von 115 bis 125 g/m² genannt wird, wobei der bevorzugte (*Preferred*) Wert bei 120 g/m² liegt.

Einem XML-Parser muss natürlich mitgeteilt werden, dass es sich bei dem Wert „*115 ~ 125*“ um ein Zahlenintervall handelt. Dies geschieht dadurch, dass der Datentyp als *NumberSpan*, also als „Zahlenspanne“, definiert wird. Bei dem Element *Dimension* ist der Datentyp gleich *XYPairSpan*, das heißt, es kann ein Intervall von Zahlenpärchen definiert werden: Die Papiergröße von Breite1 x Höhe1 bis Breite2 x Höhe2. In Beispiel 6.12 wird davon allerdings gar nicht Gebrauch gemacht, sondern es wird nur eine feste Papiergröße direkt angegeben.

Neben dem *ResourcePool* und dem *ResourceLinkPool* kann je-

Abbildung 6.13
AuditPool

```
<AuditPool>
    <Created AgentName="SuperJDF" AgentVersion="1.0" Author=" Administrator"
     ID="_4717" TimeStamp="2008-10-22T17:09:47+01:00" />
</AuditPool>
```

der JDF-Knoten noch einen weiteren Pool aufweisen, den wir bis jetzt unterschlagen haben, nämlich den *AuditPool*. In diese Pools werden Protokolleinträge geschrieben, so dass dort nach Produktionsende die Herstellung eines Druckproduktes aufge-

zeichnet ist. Es werden einerseits die Modifikationen am JDF-Dokument selbst mitgeschrieben, andererseits aber auch die Ausführungen der JDF-Prozesse. So können dort beispielsweise auch die unterschiedlichen Vorgänge an der Druckmaschine (Plattenwechsel, Gummituch waschen, Gutproduktion ...) eingetragen werden, sofern die Druckmaschine diese Werte entsprechend meldet. Allgemeiner (aber immer noch unvollständig) gilt folgende Liste von Eintragungen (*Audit*-Elemente), die im *AuditPool* vorgenommen werden können:

- Generierung, Verändern oder Löschen eines Knotens,
- Prozesszeiten (Start, Ende...),
- Endestatus (*Completed*, *Aborted*, *Stopped*...),
- Fehler (*Warning*, *Fatal*, *Error*...),
- Verbrauchte oder fehlende Ressourcen.

Mit den Inhalten des *AuditPools* könnte dann ein MIS beispielsweise eine Nachkalkulation des Druckjobs vornehmen.

Die Abbildung 6.13 zeigt einen recht elementaren *AuditPool*, in dem nur der Eintrag vermerkt ist, wann und durch wen der Knoten erzeugt wurde. Hierbei ist

Abbildung 6.14
partitionierte Ressource

- *AgentName* der Name der Software, die den JDF-Knoten

```
<Component ID="_4713" Class="Quantity" ComponentType="FinalProduct"
 DescriptiveName="Frisch-Werbung" Status="Unavailable" PartIDKeys="Condition">
   <Component Condition="Good" IsWaste="false" />
   <Component Condition="Waste" IsWaste="true" />
</Component>
```

  erzeugt hat,
- *AgentVersion* die Version dieser Software,
- *Author* die Person (der PC-Benutzer), der den Knoten erzeugt hat,
- *TimeStamp* der Zeitpunkt bei der Generierung des Knotens.

Der Knoten wurde also am 22. Oktober 2008 um 17:09 Uhr kreiert. Die 47 gibt noch die Sekundenzahl an und die +01:00 die Zeitzone, also die Abweichung von der Greenwich Mean Time.

## 6.3 Partitionierte Ressourcen

Bei einigen Ressourcen gibt es ein Problem. Sind beispielsweise alle Druckplatten eines Druckjobs eine Ressource oder

ist jede Druckplatte für sich eine? Noch krasser wird die Situation bei Druckbogen. Einerseits macht es manchmal Sinn, alle Druckbogen eines Teilproduktes (wie Umschlag oder Innenseiten) insgesamt zu sehen. So zum Beispiel, wenn es um die Papierspezifikation geht. Andererseits müssen ab und zu auch einzelne Bogen gezählt werden, beispielsweise bei der Bestimmung der Anzahl der Makulaturbogen.

Jedem Druckbogen einzeln eine JDF-Ressource zuzuordnen würde riesige JDF-Dokumente erzeugen und wäre offensichtlich töricht. Also bleibt es besser bei einer Ressource für alle Druckbogen – aber eine Ressource, die man im Zweifelsfall auch wieder in einzelne Bestandteile zerlegen (= partitionieren) kann. Man benötigt also partitionierte (eigentlich genauer: partitionier-

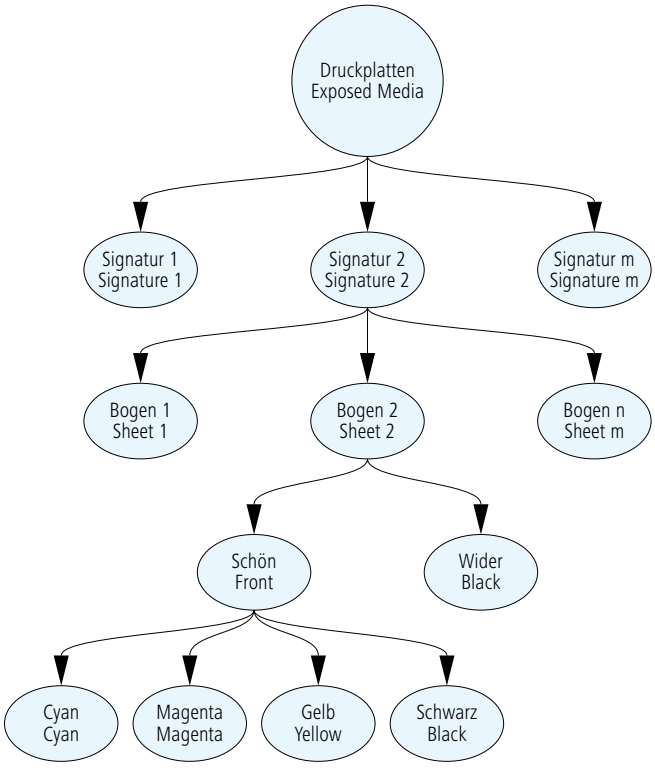



Abbildung 6.15
Componet-Ressourcen am Beispiel Druckplatten

Abbildung 6.16
partitionierte Ressource

```
<ExposedMedia Class="Handling" ID="_4718" PartIDKeys="SignatureName SheetName
 Side Separation" Status="Unavailable">
   <ExposedMedia SignatureName="Signature1">
      <ExposedMedia SheetName=" Sheet1" Status="Unavailable">
         <ExposedMedia Side="Front">
            <ExposedMedia Separation="Cyan" />
            <ExposedMedia Separation="Magenta" />
            <ExposedMedia Separation="Yellow" />
            <ExposedMedia Separation="Black" />
         </ExposedMedia>
         <ExposedMedia Side="Back">
            <ExposedMedia Separation="Black" />
         </ExposedMedia>
      </ExposedMedia>
      ...
   </ExposedMedia>
   ...
</ExposedMedia>
```

bare) Ressourcen.

Bei diesen Ressourcen muss natürlich angegeben werden, wie sie strukturiert werden. Im letzten Beispiel wurde die Ressource *Component* vorgestellt (siehe Abbildung 6.9). Sie soll nun als partionierte Ressource erweitert werden – auch wenn das Beispiel (Abbildung 6.14) zunächst ziemlich absurd erscheinen mag.

Neu hinzugekommen bei der Component-Ressource mit der ID 4713 ist nur das Attribut *PartIDKeys*. Der dazugehörige Wert stellt gewissermaßen die Zerlegungsvorschrift dar oder – anders ausgedrückt – einen Schlüssel zum Partionieren. In diesem Fall werden die Druckbogen in zwei Kategorien aufgeteilt wie die Akteure in Hollywood-Filmen: in gute und in schlechte. Die beiden Kinder-*Components*, also die Unterlemente der sie umschließenden Elter-*Component*, übernehmen dabei alle Elter-Attribute.

Im letzten Beispiel war die Klassifizierung nur einstufig. Doch in vielen Situationen muss man mehrstufig unterteilen. Nehmen wir als Beispiel die zu belichtenden Druckplatten. Die Gesamtheit aller Platten für einen Druckjob lassen sich, wie Abbildung 6.15 zeigt, in vier Ebenen aufgliedern. Die Druckplatten (*ExposedMedia*) können in mehrere Signaturen (*Signature*) aufgeteilt werden, jede Signatur wieder in mehrere Bogen (*Sheets*), jeder Bogen in Schön (*Front*) und Wider (*Back*), jede Bogenseite (*Side*) in eine oder mehrere Separation(en).

Die Abbildung 6.16 zeigt die so partitionierte Ressource. Im *PartIDKey* steht nun die Liste „*SignatureName SheetName Side Separation*“, also genau gemäß der Aufteilung in Abbildung 6.15. Die Reihenfolge der *PartIDKey*-Liste darf auch nicht verändert werden, sondern muss immer „von oben nach unten“ erfolgen. Genauer gesagt gilt die Regel: Je kürzer die Ahnenkette zum Wurzelelement ist, desto weiter links steht das Element in der Liste. Die gleiche Hierarchie muss natürlich auch bei der Verschachtelung der Unterelemente der partionierten Ressource eingehalten werden.

Im Beispiel 6.16 ist die Schönseite von Sheet1 CMYK, der Widerdruck nur Schwarz.

### 6.4 Gray Boxen und kombinierte Prozesse

Es gibt diverse Gründe, warum es wünschenswert sein kann, mehrere Prozesse zu einer neuen Struktur zusammenzuschließen:

- Gerade am Beginn einer Produktion können einzelne Prozesse noch nicht genau spezifiziert werden. Dann ist es hilfreich, wenn zunächst größere Produktionsstrecken insgesamt betrachtet werden können, ohne dass die Zwischenergebnisse zwischen den Prozessen innerhalb der Strecken angegeben werden müssen. Im Laufe des Produktionsfortschritts werden aber die Zwischenergebnisse durchaus festgelegt.
- Wenn eine Maschine oder eine Software mehrere Prozesse hintereinander ausführt, ohne dass jemand die Zwischenergebnisse bearbeitet, ist es unnötig, diese genau zu beschreiben. Dies kann beispielsweise bei einer integrierten

Digitaldruckmaschine der Fall sein, bei der die Prozesse Interpreting, Rendering, Screenig, Digital Printing, Stitching, Trimming aufeinander folgen. Aber auch in der Vorstufe für den Offsetdruck kann man die beiden Prozesse *PreviewGeneration* und *InkZoneCalculation* häufig zu einer Kombination zusammenschließen (siehe Abschnitt 10.1).

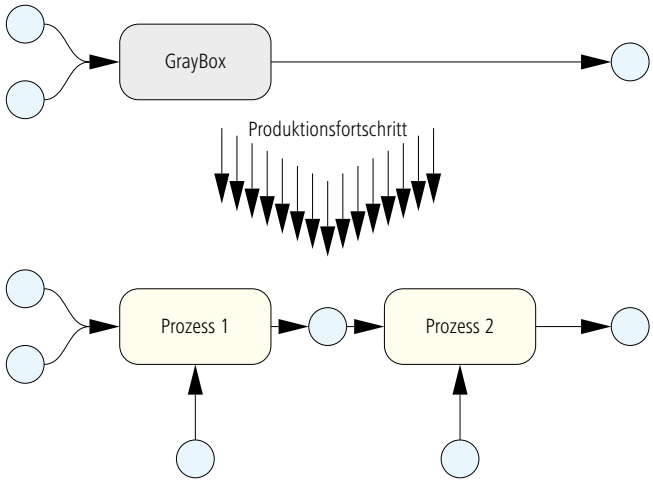



Abbildung 6.17
Eine GrayBox wird im Laufe der Produktion aufgelöst.

Der erste Fall führt uns zu dem Begriff *GrayBox*, der zweite zu einem kombinierten Prozess (*combined Process*).

Eine *GrayBox* ist ein Prozessgruppenknoten, in dem die Details noch nicht spezifiziert sind. Erst im Laufe der Produktion werden die Details hinzugefügt. Wenn alle Prozesse und Ressourcen vorhanden sind, wird die *GrayBox* schließlich aufgelöst, denn sie selber kann nicht ausgeführt werden wie die eigentlichen Prozesse. Typischerweise werden vom MIS eine oder mehrere *GrayBox*(en)

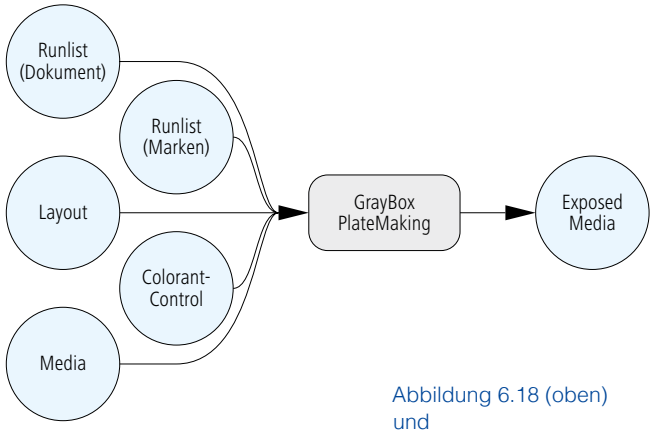



Abbildung 6.18 (oben) und
Abbildung 6.19 (unten)
Beispiel der GrayBox Offsetplattenherstellung

```
<JDF Type="ProcessGroup" Types="Imposition RIPing ImageSetting"
 DescriptiveName="GB PlateMaking" ...>
...
    <ResourceLinkPool>
        <RunListLink ProcessUsage="Document" Usage="Input" ... />
        <RunListLink ProcessUsage="Marks" Usage="Input" ... />
        <LayoutLink Usage="Input" ... />
        <ColorantControlLink Usage="Input" ... />
        <MediaLink Usage="Input" ... />
        <ExposedMediaLink Usage="Output" ... />
</JDF>
```

angelegt. Denn ein MIS kann in der Regel nicht alle nötigen Produktionsparameter angeben. Beispielsweise kann ein MIS zwar festlegen, wie viele Platten in welchem Format produziert werden müssen, nicht jedoch die einzelne Schritte, wie sie hergestellt werden. Ein MIS enthält meist keine genauen Trapping- oder Rasterinformationen, da sie für die Kalkulation unwichtig sind. Die fehlenden Einstellungen werden entweder durch voreingestellte Werte in den Workflow-Modulen, durch Festlegungen im Druckjob oder durch die Angaben eines Operators zugefügt.

In der schematischen Abbildung 6.17 sieht man, dass eine GrayBox durchaus Input- und Output Ressourcen haben kann. Die Output-Ressource(n) muss (müssen) sogar spezifiziert sein. Abbildung 6.18 zeigt das Beispiel der GrayBox, welche die Offestplattenherstellung (*PlateMaking*) festlegt. Sie weist fünf (oder mehr) Input-Ressourcen und eine (oder mehr) Output-Ressource(n) auf. Da wir die dort aufgeführten Ressourcen auch in den folgenden Kapiteln benötigen, seien sie hier kurz vorgestellt:

- Eine *RunList* (deutsch: Seitenfolge) ist eine geordnete Menge von Seitenbeschreibungen. So werden die Content-Daten als *RunList* geführt (in 6.18 mit dem Zusatz „Dokument“ versehen), aber auch die Markendaten, die für den Standbogen benötigt werden (in 6.18 mit dem Zusatz „Marken“).
- Die Ressource *Layout* beschreibt den oder die Standbogen.
- *ColorantControl* legt die Farbendefinitionen fest (welcher Farbraum, welche Farben...).

Abbildung 6.20
Beispiel des kombinierten Prozesses Offsetplattenherstellung

```
<JDF Type="Combined" Types="Imposition Interpreting Rendering"...>
   <ResourcePool>
      <RunList ID="_100"... />
      <Layout ID="_200""... />
      <InterpretingParams ID="_300" ... />
      <RenderingParams ID="_400" ... />
      ...
   </ResourcePool>
   <ResourceLinkPool>
      <RunListLink CombinedProcessIndex="0" Usage="Input" rRef="_100" />
      <LayoutLink CombinedProcessIndex="0" Usage="Input" rRef="_200" />
      <InterpretingParamsLink CombinedProcessIndex="1" Usage="Input"
       rRef="_300" />
      <RenderingParamsLink CombinedProcessIndex="2" Usage="Input"
       rRef="_400" />
      ...
   </ResourceLinkPool>
</JDF>
```

- *Media* charakterisiert die Druckplatten (Größe, Produktname...)
- *ExposedMedia* sind in diesem Fall die belichteten Platten.

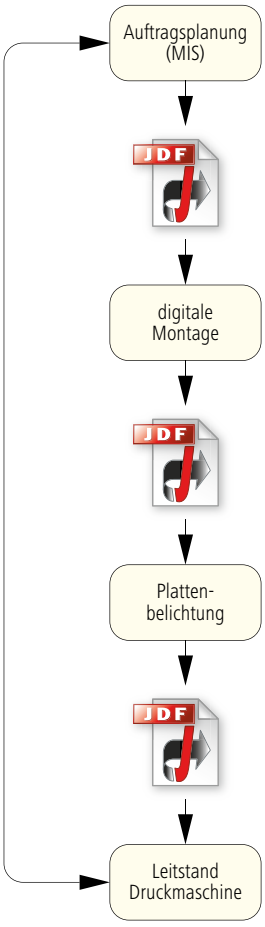
Der Code für *PlateMaking* ist in Abbildung 6.19 dargestellt. Prozessgruppenknoten vom Typ *GrayBox* sind immer an der Kombination der Attribute *Type* und *Types* zu erkennen. Der Wert des Attributs *Type* ist *ProcessGroup*, der Wert von *Types* ist eine Liste von Prozessen, welche von der *GrayBox* inbegriffen sind. Man beachte, dass *RIPing* eigentlich keinen Prozess darstellt, sondern selber wieder ein Prozessgruppenknoten ist.

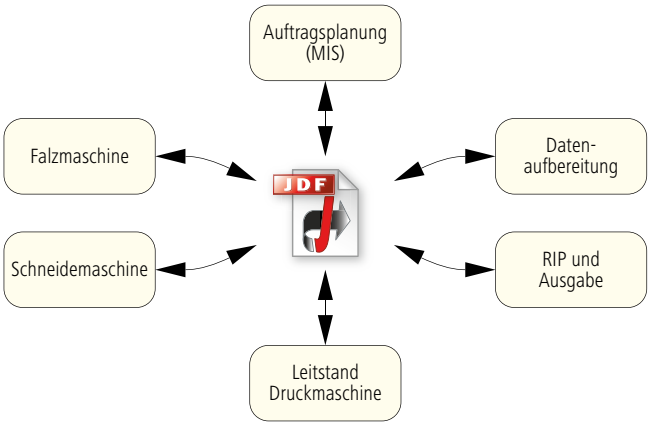
Es gibt übrigens auch Prozessgruppenknoten, die nicht die Funktion einer *GrayBox* haben. Dort fehlt dann auch konsequenterweise das *Types*-Attribut, und das *Type*-Attribut hat den Wert *ProcessGroup*. Diese Art von Prozessgruppenknoten enthalten Prozesse als JDF-Knoten.

Prozesse können aber nicht nur gruppiert, sondern auch kombiniert werden. Das Ergebnis einer solchen Prozesskombination (*Combined Process*) ist selber wieder ein Prozess. Er enthält eine Liste der Prozessnamen und die nötigen Ressourcen. In Abbildung 6.20 ist zu sehen, dass in diesem Fall der Wert des Attributs *Type* gleich *Combined* ist, während in dem Wert von *Types* die Prozesse aufgelistet werden – ähnlich wie bei einer *GrayBox*.

Ein kombinierter Prozess ähnelt vom Konzept her einer Prozessgruppe. Der Unterschied liegt darin, dass ein kombinierter Prozess von einem einzigen Gerät, eine Prozessgruppe im Allgemeinen von mehreren Geräten abgearbeitet wird. Aus diesem Grund müssen Ressourcen, die nur Zwischenergebnisse der Prozesse beschreiben, bei einem kombinierten Prozess nicht unbedingt aufgeführt werden.

Neu in Abbildung 6.20 ist eigentlich nur das Attribut *CombinedProcessIndex*. Mit diesen Werten wird festgelegt, zu welchen der unter *Types* aufgeführten Prozesse die Ressourcen in Beziehung stehen. Es wird von Null beginnend gezählt, das heißt der erste Prozess in der Liste, nämlich *Imposition*, hat den Index 0, der zweite (*Interpreting*) hat den




Abbildungen 6.21 und 6.22 sequentielle Weitergabe (**oben**), zentrale Verwaltung (**unten**)

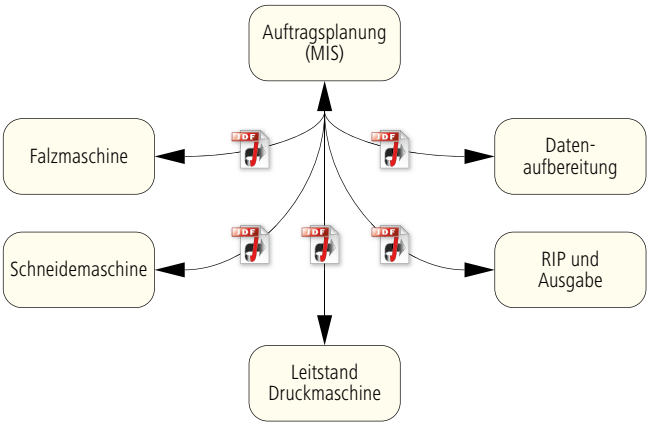



Abbildung 6.23
Das MIS ist für die JDF-Kommunikation verantwortlich.

Index 1 und der dritte (*Rendering*) den Index 2. Folglich sind die Ressourcen *RunList* und *Layout* Input-Ressourcen von *Imposition*, *InterpretingParams* Input-Ressource von *Interpreting* und die *RenderingParams* Input-Ressource von *Rendering*.

## 6.5 JDF-Workflow Architekturen

Nachdem wir nun gesehen haben, wie prinzipiell ein JDF-Dokument aufgebaut ist, wollen wir untersuchen, wie diese Dokumente zu den Workflow-Modulen gelangen.

Die einfachste Möglichkeit ist die sequenzielle Weitergabe der Information, die in Abbildung 6.21 skizziert ist. Dieses sehr simple Modell wird man aber in Wirklichkeit nicht antreffen oder bestenfalls nur in sehr einfachen Produktionsumgebungen, in denen zwei oder drei benachbarte Prozesse JDF-Information austauschen. Beispiel: Eine Montage-Software (*Stripping*) erzeugt ein JDF-Dokument für das Ausschießen (*Imposition*).

Ein realistischeres Modell ist die zentrale JDF-Datenablage, auf die unterschiedliche Workflow-Module Zugriff haben, wie in Abbildung 6.22 gezeigt. Diese Datenablage kann entweder eine Datenbank oder auch einfach nur eine Ordnerstruktur auf einem Dateiserver sein. Natürlich werden die einzelnen Workflow-Engines nicht völlig frei auf die JDF-Daten zugreifen können, sondern nur unter der Verwaltung einer speziellen, übergeordneten Software, welche die Konsistenz der Daten kontrolliert. Im Unterschied zur sequenziellen Situation liegen die Daten auf einem zentralen Server, was vorteilhaft auch in Hinsicht der Datensicherung ist. Die Abbildung 6.22 legt nahe, dass die JDF-Verwaltung Teil der Produktionssoftware ist. Es gibt aber auch das MIS-zentrische, zentrale Datenhandling, das Abbildung 6.23 darstellt. Es ist übrigens nicht zwingend, dass pro Auftrag nur ein einziges JDF-Dokument generiert wird. Denn anstatt immer dieses eine Dokument mit aktuellen Daten zu füllen, kann auch jedes Workflow-Modul neue Versionen vom JDF-Dokument erzeugen. So könnte man im Fehlerfall – zumindest theoretisch – wieder zu einem älteren Stand zurückkehren.

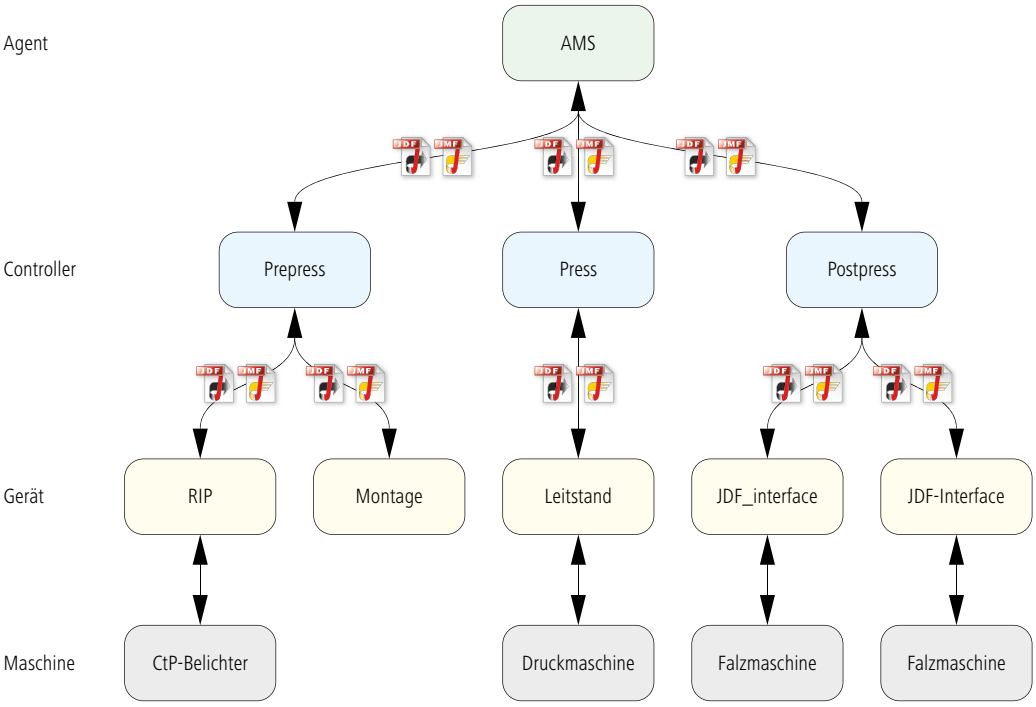



Abbildung 6.24
hierarchisch organisierte JDF-/JMF-Kommunikation

Doch um einem möglichen Missverständnis gleich noch vorzubeugen: Es ist natürlich nicht so, dass in einer JDF-Umgebung alle Daten für die Herstellung von Druckprodukten irgendwo zentral im JDF-Format gespeichert werden. Tatsächlich wird beispielsweise ein MIS weiterhin auf seiner eigenen Datenbank mit eigenen Tabellen und Datensätzen operieren, in denen auch all die Daten gespeichert sind, die nicht im JDF abbildbar sind (wie Kosten von Maschinen und Arbeitsgängen). Das JDF/JMF ist also nur eine Schnittstelle nach außen. Gleiches gilt auch für Prepress, Press und Postpress.

Die CIP4-Organisation schlägt ein weiteres, eher hierarchisches Modell vor, das in Abbildung 6.24 beispielhaft wiedergegeben ist. In diesem Modell geht es jedoch weniger um die Datenablage, sondern um die Funktionen, die JDF-Software einnehmen kann. Dabei gilt folgende Sprachregelung.

- Ein *Agent* kann ein JDF-Dokument generieren, einen JDF-Knoten erzeugen und modifizieren. Typischerweise übernimmt ein Auftrags-Managementsystem die Agentenrolle.
- Ein *Controller* hat die Aufgabe, JDF-Dokumente und auch

JMF-Nachrichten an die Geräte weiterzuleiten. Umgekehrt sollte er auch die Meldungen von den Geräten an den Agenten zurückmelden.

- Ein *Device* (Gerät) interpretiert die JDF-Knoten und initiiert entsprechende Aktionen in der angeschlossenen Maschine. Die Gerät-Maschine-Kommunikation basiert nicht auf JDF/JMF, sondern ist herstellerspezifisch. In der IT-Sprache würde man ein *Device* auch als JDF-Gerätetreiber bezeichnen.
- Eine *Machine* (Maschine) führt einen oder mehrere Prozesse aus ohne selbst JDF/JMF zu verstehen oder zu erzeugen.

Die Einteilung ist nicht immer absolut klar. So können auch viele *Controller* und *Devices* JDF-Knoten erzeugen und modifizieren, sie haben also Agentenfähigkeiten. Auch wird häufig ein *Agent* gleichzeitig auch ein *Controller* sein, wie das bei einigen Auftrags-Managementsystemen der Fall ist. Diese JDF-Architektur ist dann nur dreistufig, das heißt der *Agent* (das MIS) ist gleichzeitig der universale *Controller*. Die Einteilung in drei *Controller* in 6.24 ist auch nur ein Beispiel, es können mehr oder weniger *Controller* sein. Schließlich können *Controller* rekursiv arbeiten, ein *Controller* kann einen oder mehrere Untercontroller bedienen.

Abbildung 6.25
NodeInfo-Ressource

```
<ResourcePool>
    <NodeInfo FirstStart="2008-08-01T00:00:0+01:00"
     LastStart="2008-08-02T08:08:00+01:00"
     LastEnd="2008-08-02T15:00:00+01:00"... />
    ...
</ResourcePool>
```

Ein *Controller* kann dabei nach folgenden Methoden die Daten weiterleiten:

- Ein *Controller* schickt den kompletten JDF-Job der Reihe nach an alle Geräte.
- Der *Controller* ermittelt über JMF, welches Gerät welche Prozesse verarbeiten kann und schickt nur entsprechende Teile des JDF-Jobs an die Geräte.
- Wie im letzten Punkt, nur dass im *Controller* die Eigenschaften der Geräte „fest verdrahtet“, also fest definiert sind.

Bei den letzten beiden Möglichkeiten müssen Teile der JDF-Knoten herausgetrennt werden. Die Technik hierzu werden wir

im nächsten Abschnitt behandeln.

Die Kombinationen von Geräten und Maschinen wurden in den letzten Kapiteln „Jobticket-Prozessoren“ oder auch „Workflow-Engines“ genannt.

Die hier vorgestellten Modelle mögen zunächst alle recht komplex und auch sehr statisch wirken, und es stellt sich die Frage, wie Prozesse automatisch ablaufen können, so dass der Anwender auch einen Gewinn von einer solchen JDF-Installation hat. Tatsächlich können JDF-Geräte Abläufe auf einer Maschine automatisch anstoßen. Hierfür müssen dann folgende Bedingungen gelten:

- Ein im JDF beschriebener Prozess ist von der Maschine ausführbar.
- Der Status des Prozesses ist *Ready* oder *Waiting.*
- Alle Input-Ressourcen haben den Status *Available.*
- Die aktuelle Zeit liegt im definierten Produktionszeitenintervall.

Das Produktionszeitenintervall für einen Prozessknoten ist in einer eigenen Ressource mit dem Namen *NodeInfo* vermerkt. Allgemein werden in diese Ressource Informationen über Ausführung, Verantwortlichkeit für den Prozess und Konsequenzen bei Zeitüberschreitung abgelegt. In Beispiel 6.25 sollte der Prozess frühestens am 1. 8.2008 um 0.00 Uhr beginnen, spätestens jedoch am 2.8.2008 um 8:00 Uhr. Der Prozess muss am gleichen Tag spätestens um 15:00 Uhr beendet sein.

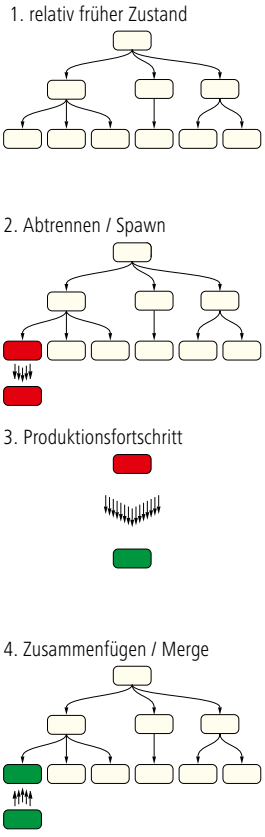



Abbildung 6.26
„Spawn“- und „Merge“-Verfahren, um JDF-Knoten abzutrennen bzw. wieder zusammenzuführen

## 6.6 Trennen und Zusammenführen

In Workflow-Managementsystemen können Dinge parallel ablaufen. Entsprechend müssen JDF-Informationen an mehreren JDF-Geräten gleichzeitig vorliegen. Außerdem kann es auch aus Gründen der Datenhaltung zweckmäßiger sein, nur diejenigen Informationen einem JDF-Gerät zu geben, die es benötigt und nicht das gesamte JDF-Dokument. In beiden Fällen muss man also Teile des JDF-Codes duplizieren und an andere

Abbildung 6.27
Originaldatei vor dem Trennen

```
<JDF DescriptiveName="Gesamtprodukt" JobID="_001"...>
    <JDF ...DescriptiveName="Umschlag" ID="_002 " JobPartID="_003"
     Status="Waiting" Type="Product">
     ...
    </JDF>
</JDF>
```

```
<JDF DescriptiveName="Gesamtprodukt" JobID="_001"...>
    <AuditPool>
        <Spawned NewSpawnID="_004" Status="Waiting"
         TimeStamp="2008-10-25T09:38:09+02:00"
         URL=file://file:/C:/GetrennteDatei.jdf
         jRef="002" rRefsROCopied="R1 R2 R3 R4 R5 R6 R7 R8 R9" />
    </AuditPool>
    <JDF DescriptiveName="Umschlag" ID="_002" JobPartID="_003"
     Status="Spawned" Type="Product">
    ...
    </JDF>
</JDF>
```

```
<JDF ... DescriptiveName="Umschlag" ID="_002" JobID="_001" JobPartID="_003"
     SpawnID="_004" Status="Waiting" Type="Product">
    </JDF>
    ...
    <AncestorPool>
        <Ancestor FileName="file:/C:/Originaldatei.jdf" JobID="_001"... />
         ...
        </Ancestor>
    </AncestorPool>
</JDF>
```

Abbildung 6.28 (ganz oben) Originaldatei nach dem Trennen
Abbildung 6.29 (oben) abgetrennte JDF-Datei

Geräte schicken. Da diese Geräte aber den kopierten Daten weitere JDF-Informationen anfügen können und zum Schluss natürlich das gesamte Produktionsprotokoll wieder in einem JDF-Dokument vorliegen sollte, muss es auch möglich sein, die kopierten und modifizierten JDF-Daten wieder in das ursprüngliche Dokument einzufügen. Damit dieser ganze Vorgang funktioniert, muss einiges beachtet werden. Vor allem dürfen nicht mehrere Stellen parallel den JDF-Code verändern, sondern es muss verabredet werden, wer nur lesenden (*read*) und wer lesenden und schreibenden (*read-write*) Zugriff hat. Kurz: Die

Abbildung 6.30 JDF-Datei nach der Zusammenführung

```
<JDF DescriptiveName="Gesamtprodukt" JobID="_001"...>
    <AuditPool>
...
        <Spawned NewSpawnID="_004" Status="Waiting"
         TimeStamp="2008-10-25T09:38:09+02:00"  URL="file://file:/C:/Test.jdf"
         jRef="002" rRefsROCopied="R1 R2 R3 R4 R5 R6 R7 R8 R9 />
        <Merged ... MergeID="_004" TimeStamp="2008-10-25T09:41:01+02:00"
         URL="file://file:/C:/GetrennteDatei.jdf" jRef="002" />
    </AuditPool>
    ...
    <JDF DescriptiveName="Umschlag" ID="_002" JobPartID="_003"
     Status="Waiting" Type="Product"...>
...
    </JDF>
</JDF>
```

Konsistenz der Daten muss gewährleistet werden.

Das Verfahren nennt sich im JDF-Jargon *Spawn* und *Merge*, was so viel heißt wie „laichen“ oder „hervorbringen“ und „zusammenfügen“. Anstatt vom „Laichen“ werden wir im Folgenden aber lieber vom „trennen“ reden – obwohl es eigentlich mehr ein Kopiervorgang ist.

In Abbildung 6.26 ist die Situation noch einmal grafisch dargestellt. Oben (1.) ist das originale Dokument dargestellt, von dem (in 2.) ein JDF-Knoten getrennt und ein neues Dokument erzeugt wird. Man sieht in (3.) deutlich, wie sich der getrennte Knoten unabhängig vom Originaldokument verändert (er wird giftgrün) und (in 4.) schließlich wieder mit dem ursprünglichen JDF-Baum zusammengeführt wird. Der abgetrennte Knoten kann durchaus auch weitere Kinderknoten erzeugen.

Natürlich können auch Ressourcen getrennt werden, ebenso partitionierte Ressourcen. Welche Ressourcen abgetrennt wurden, muss im Originaldokument vermerkt werden. Auch werden die *Spawn*- und *Merge*-Vorgänge im *AuditPool* mitprotokolliert.

Im Folgenden möchten wir ein Beispiel einer Trennung und der Zusammenführung zeigen.

In 6.27 sieht man einen JDF-Job noch vor dem Trennvorgang. Im nächsten Schritt wird das Teilprodukt „Umschlag“ getrennt (um es beispielsweise an einer anderen Produktionsstätte zu produzieren). In 6.28 sieht man, wie die Hauptdatei nach dem Trennen verändert wurde: Im *AuditPool* wird die *NewSpawnID* eingetragen, die sich auch in der getrennten Datei wiederfindet. Außerdem wird noch die *URL* der getrennten Datei vermerkt, unter *jRef* die ID des Knotens, der getrennt wurde, und schließlich auch in dem Attribut *rRefsROCopied* die Ressourcen (R1 bis R9), die getrennt wurden. Das *RO* in dem Attributnamen besagt, dass die Ressourcen *Read-Only* sind, also von den Prozessen, die den getrennten Teil bearbeiten, nur gelesen und nicht verändert werden dürfen. Es sei noch einmal darauf hingewiesen, dass sich die getrennten Teile des JDF-Baums – hier also der JDF-Knoten „Umschlag“ – weiterhin in der Originaldatei befinden. Abbildung 6.29 zeigt einen Ausschnitt der Datei mit den getrennten JDF-Daten. Alle IDs wie *JobID* und *JobPartID* ändern sich nicht; es wird nur zusätzlich die *SpawnID* definiert (die gleiche Nummer wie bei dem Attribut *NewSpawnID* zuvor). Zusätzlich werden noch unter dem *Ancestor*-Element Informationen über den ursprünglichen JDF-Job abgelegt, hier der Dateiname und der Speicherort sowie die *JobID*. Diese Informationen werden benötigt, um später die Daten wieder korrekt zusammenzuführen. In diesem Zustand können sich nun die Hauptdatei und die getrennte Datei unabhängig voneinander entwickeln, zumindest dann, wenn letztere auch Schreibrechte hat. Schließlich sollten aber irgendwann beide JDF-Bäume wieder zusammenwachsen. Das Ergebnis sieht man in 6.30, das eigentlich gleich ist zu dem vor dem Trennungsvorgang. Nur die Einträge im *AuditPool* beschreiben, dass zwischendurch Teile getrennt und später wieder zusammengeführt wurden.

Das ganze Verfahren ist rekursiv, das heißt, getrennte JDF-Elemente können abermals getrennt werden.

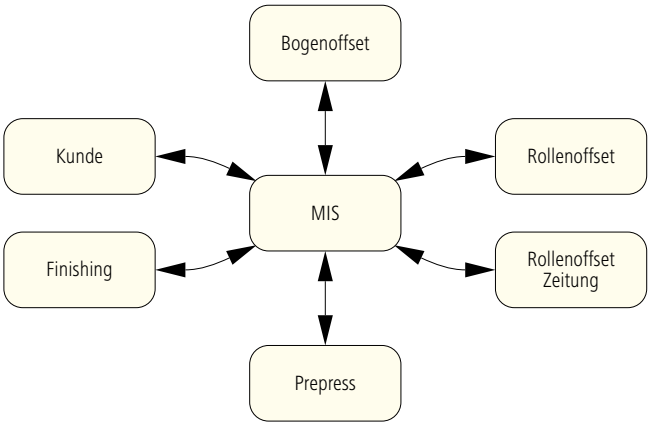


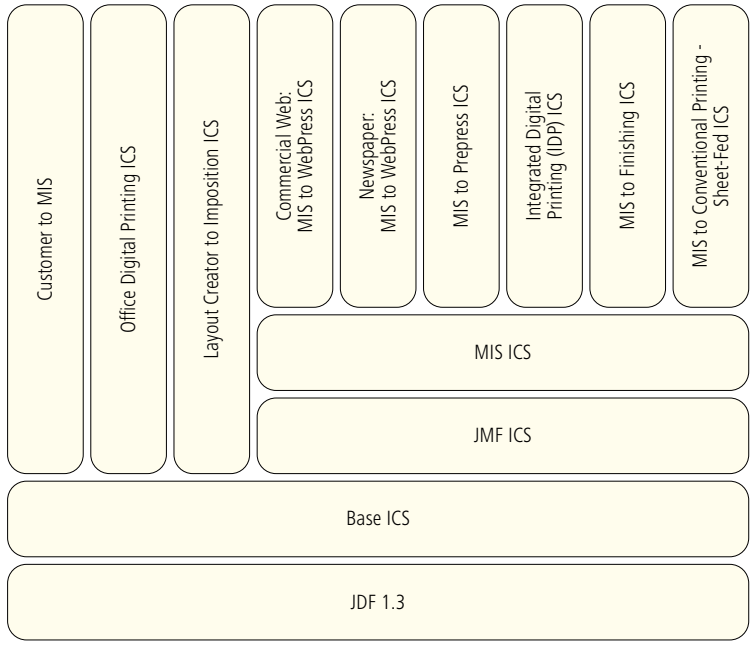

Abbildung 6.31 in ICS-Papieren beschriebene MIS-Schnittstellen

## 6.7 Interoperability Conformance Specifications (ICS)

Schnittstellen sind immer problematisch. Wenn beispielsweise eine Werbeagentur bei einer Druckerei ein Druckprodukt in Auftrag gibt, weiß sie häufig nicht so recht, wie die Content-Daten richtig aufzubereiten sind. Umgekehrt weiß die Druckerei auch nicht, was die Agentur nicht weiß und so entstehen viele Missverständnisse und Frustrationen. Obwohl beide die gleiche Sprache sprechen, klappt die Kommunikation nicht zufriedenstellend. Ein Ausweg aus dem Dilemma bei der Herstellung von Content-Daten ist die Vereinbarung von PDF-Standards wie PDF/X [23] oder PDF/X-Plus [19], welche die große Funktionalität des *Portabe Document Formats* einschränken.

Diese Schnittstellenproblematik hat man in ähnlicher Weise auch im JDF-Workflow: Wenn ein *Agent*, ein *Controller* oder ein *Device* gewisse JDF-Strukturen erzeugt, dann in der Erwartung,

Abbildung 6.32 ICS-Papiere bauen teilweise aufeinander auf.



dass der Empfänger etwas damit anfangen kann. Umgekehrt benötigt der Empfänger gewisse JDF-Daten, die er möglicherweise nicht bekommt. Hier sollen die *Interoperability Conformance Specifications* (ICS) Abhilfe schaffen. Sie schränken für gewisse Software-Komponenten die riesige Funktionalität ein, welches das JDF/JMF bietet.

Bei dieser Art von Problemen gibt es immer zwei Rollen: Einer generiert Informationen und der andere nimmt sie auf – zumindest versucht er es. Insofern ist die Begrifflichkeit von *Agenten*, *Controllern* und *Devices* in diesem Zusammenhang nicht weiterführend und es werden zwei neue Begriffe verwendet:

- Der *Manager* sendet in erster Linie JDF-Dokumente an einen Worker (*Controller* oder *Devices)*. Außerdem kann er auch optional JMF-Nachrichten an diese schicken. Erst in zweiter Priorität liest er auch JDF-Dokumente und JMF-Nachrichten ein, die ihm *Controller* oder *Devices* schicken.
- Der *Worker* empfängt JDF-Dokumente vom Manager (*Agenten* oder *Controller)*, zusätzlich vielleicht auch JMF-Nachrichten. Außerdem kann er (wieder in zweiter Priorität) JDF/JMF-Informationen zurückschicken.

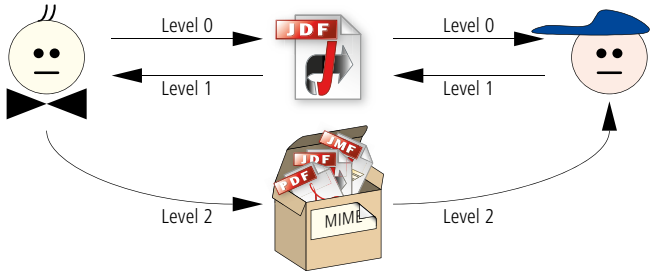



Abbildung 6.33
verschiedene Level
im Base ICS

Über die Wortwahl von *Manager* und *Worker* mag man sich streiten; wir werden aber diesem Sprachgebrauch der ICS-Papiere folgen. Man beachte, dass sowohl *Manager* als auch *Worker* JDF-Daten und JMF-Nachrichten schreiben und lesen können. Ein Manager sendet beispielsweise einen JDF-Prozess an einen Worker, der diesen ausführt und währenddessen und nach dem Prozessende Angaben über verbrauchte und erzeugte Ressourcen zurückschickt. Will man die Rolle des JDF/JMF-Schreibens (oder auch des Modifizierens) betonen, spricht man auch von Produzent (*producer*), bei der Rolle des Lesenden von Konsument (*consumer*). *Manager* und *Worker* sind also manchmal Produzenten, manchmal Konsumenten.

In den ICS-Papieren werden unterschiedliche Schnittstellen beschrieben, die teilweise aufeinander aufbauen. Die verschiedenen MIS-Schnittstellen, die durch ICS-Papiere definiert sind und auf der JDF Version 1.3 beruhen, sind in Abbildung 6.31

```
<ResourcePool>
    <RunList ID="_4719„ Pages="0~-1" Class="Parameter" Status="Available">
        <LayoutElement>
            <FileSpec URL=file://Workflowserver/Pagefiles/Frisch-Werbung.pdf />
        </LayoutElement>
        ...
    </RunList>
    ...
</ResourcePool>
```

Abbildung 6.34
Referenz auf Content-Daten über die URL

zu sehen. Die gesamte Sammlung von ICS-Papieren und deren Hierarchie sind in Abbildung 6.32 dargestellt. Das *MIS to Finishing ICS* beispielsweise beruht auf dem *Base ICS*, dem *JMF ICS* und dem *MIS ICS*, während das *Office Digital Printing ICS* nur auf dem *Base ICS* aufbaut.

Außerdem sind die einzelnen Schnittstellen nochmals in verschiedene Levels klassifiziert, die ebenfalls hierarchisch strukturiert sind.

Einige der ICS-Papiere werden wir noch in den folgenden Kapiteln ausführlicher vorstellen. Jetzt möchten wir nur das *Base ICS* etwas genauer darstellen. Dort sind 3 Levels definiert, nämlich Level 0, Level 1 und Level 2, wobei Level 1 die Funktionalität von Level 0 umfasst und entsprechend Level 2 die Funktionalität von Level 1. In den übrigen ICS-Schnittstellenpapieren werden dann die vorausgesetzten Levels vom *Base ICS* angegeben. Eine Schnittstelle genügt nur der *Base ICS* Level 0, wenn der *Manager* JDF-Dokumente über einen Hotfolder an den *Worker* übermitteln und der *Worker* diese lesen kann. Level 1 ist erreicht, wenn umgekehrt zusätzlich auch der *Worker* JDF-Dokumente an den *Manager* weiterleiten und der *Manager* diese einlesen kann. Für Level 2 sind gleich zwei weitere Forderungen zu erfüllen:

- Allgemein kann es bei einer WMS-Implementierung sinnvoll sein, mehrere Daten gleichzeitig zu übertragen, wie JDF-Dokumente, JMF-Nachrichten, PDF-Druckdaten, Preflight-Profile, ICC-Farbprofile und Voransichtsbilder. Diese können zusammen in ein Paket gepackt werden, wie z.B. in ein MIME-Paket. MIME steht für *Multipurpose Mail Internet Extensions*, denn bei E-Mail-Anhängen können ebenfalls mehrere unterschiedliche Daten zusammen verschickt werden. Bei Level 2 muss nun der *Manager* MIME-Pakete zusammenstellen und der Worker diese interpretieren können (Abbildung 6.33).
- In der Regel werden aber innerhalb eines JDF-Dokuments

zum Beispiel PDF-Dateien referenziert. Das heißt es wird zum Beispiel der Server im lokalen Netzwerk und der Pfad im Dateisystem angegeben, wo sich die Dateien befinden. In Abbildung 6.34 ist eine *RunList* (Seitenfolge) definiert, die nur aus einem PDF-Dokument (*LayoutElement*) besteht, dessen Dateiname und Speicherort über eine Pfadangabe (*FileSpec*) bestimmt ist. Der Wert „0~1“ des Attributs *Pages* besagt, dass alle Seiten dieser Datei verwendet werden sollen. Level 2 verlangt nun, dass der Manager diese Dateien zusätzlich über HTTP erreichbar macht und der Worker sie dort ebenfalls über HTTP abholen kann.

Neben diesen Levels werden in der *Base ICS* viele technische Dinge geklärt, so beispielsweise, aus welcher Anzahl Zeichen eine *ID* bestehen darf. Interessanter (aber auch nur wenig) sind da schon die Auflagen, die an JDF-Knoten und deren Attribute und Unterelemente gestellt werden. So wird beispielsweise festgelegt, dass der *Manager* im JDF-Wurzelelement die *JobID* schreiben und der *Worker* sie wieder lesen muss oder wer welche *Audit*-Elemente in einem *AuditPool* zwingend eintragen muss, welche optional sind und so weiter.

Zusammenfassend lässt sich sagen, dass die ICS-Papiere nicht nur Werkzeuge für Entwickler von JDF-Software sind, sondern auch die wichtigste Grundlage für Integration und Tests zwischen mehreren JDF-Komponenten bilden. Im Fall eines Fehlers in der JDF-Kommunikation sollte man zuerst untersuchen, ob alle Beteiligten sich an die ICS-Regeln halten.

---

**Übungen:**

- Laden Sie von CIP4.org den *JDFEditor* herunter sowie einige JDF-Beispielsdateien. Öffnen Sie diese und untersuchen Sie die Struktur der JDF-Knoten und der Ressourcen.
- Benennen Sie die *.jdf-Dateien in *.xml um und öffnen Sie sie mit einem Browser. Analysieren Sie so den XML-Code auf direkte Weise.
- Trennen Sie mit Hilfe des Editors einen JDF-Unterknoten ab und führen Sie ihn anschließend wieder zusammen. Analysieren Sie die Zwischenzustände mit einem Browser.

# 7 Job Messaging Format (JMF)

Im letzten Kapitel haben wir gesehen, wie mittels JDF Daten zwischen den unterschiedlichen Agenten, Controllern und JDF-Geräten ausgetauscht werden können. Im einfachsten Fall werden JDF-Dateien von einem JDF-Produzenten geschrieben und in den Hotfolder eines JDF-Konsumenten abgelegt, der in bestimmten Zeitintervallen den Ordner nach neuen Daten durchsucht. Diese unidirektionale Schnittstelle über Dateitransfer ist natürlich recht statisch und erlaubt wenig dynamische Interaktion zwischen den Workflow-Beteiligten, die beispielsweise für die Betriebsdatenerfassung, Job-Verfolgung, Job-Änderungen, zentrale Fehlererfassung, Nachweis zum Materialverbrauch, Erfassung des Nutzungsgrads von Maschinen oder zur Systemsteuerung (vor allem beim Hochfahren) wichtig ist. Hierzu wurde zusammen mit JDF das *Job Messaging Format* (JMF) eingeführt. Nicht in jedem JDF-Workflow sind auch JMF-Nachrichtenkanäle implementiert, doch wir werden sehen, welchen zusätzlichen Nutzen sie bringen können. Für ein MIS sind aber beispielsweise erst ab dem *MIS ICS Level 2* gewisse JMF-Nachrichten vorgeschrieben.

Ein allgemeines Verständnis der JMF-Technologie ist wichtig, um zu verstehen, wie ein JDF-/JMF-Workflow funktioniert. Trotzdem muss man sich natürlich im Klaren darüber sein, dass man als Anwender mit JMF weniger zu tun hat. Denn während JDF-Daten zumeist als Dateien vorliegen und damit für eine Analyse mit möglicher Fehlerfindung einfach zugänglich sind, werden JMF-Daten meist über das HTTP-Protokoll verschickt und haben somit keine physische Ablage als Datei. Man kann natürlich HTTP-*Sniffer* einsetzen – also Programme, die HTTP-Pakete abfangen und lesbar machen – doch bei der Vielzahl von Meldungen, die da gewöhnlich über die Netzwerke verschickt werden, ist das eher nicht sehr spaßig. Insofern werden wir auch in diesem Buch die Betonung mehr auf JDF als auf JMF legen und so einige JMF-Feinheiten aussparen.

## 7.1 Kommunikationsmodelle

Doch bevor wir in die Struktur des JMF-Nachrichtenprotokolls einsteigen, möchten wir etwas allgemeiner verschiedene Kommunikationsmodelle diskutieren, die zwischen Software-Modulen in der grafischen Industrie üblich sind:

- Dateitransfer über Hotfolder (oder auch manuell),
- Datentransfer über Druckprotokolle,
- Datenbankschnittstellen,
- Interprozesskommunikation,
- Webservices,
- Nachrichtenprotokolle.

Dateitransfer über Hotfolder hat einen großen Vorteil, nämlich die einfache Administrierung durch Anwender, vornehmlich Systemadministratoren. Dem stehen aber auch einige Nachteile gegenüber:

- Wenn eine Hotfolder-Kette unterbrochen wird (weil beispielsweise ein Rechner nicht läuft), gibt es keine Fehlermeldung.
- Der Datenproduzent bekommt keine Quittung darüber, dass seine Daten erfolgreich gelesen und interpretiert werden konnten.
- Im Fehlerfall bei der Job-Abarbeitung gibt es keine zentrale Überwachungsinstanz, welche die Ursachen erkennen kann.
- Die Kommunikation ist langsam und eignet sich damit weder für BDE noch für bidirektionale Handshaking-Verfahren, bei denen sich zwei oder mehrere Module miteinander abstimmen müssen.

Vielfach wurden und werden immer noch Hotfolder auch bei RIPs eingesetzt, in denen die Druckdaten platziert werden. Üblicherweise sind dann diese Ordner in der Art parametriert, dass gewisse RIP-Einstellungen (Auflösung, Rasterfrequenz,...) assoziiert werden. Bei vielen unterschiedlichen RIP-Einstellungen müssen dann entsprechend viele Hotfolder eingerichtet werden, was unübersichtlich ist. In einer JDF-Umgebung hat man diese Schwierigkeit allerdings nicht, da ja dort die Metadaten – und so auch die Einstelldaten für den RIP – variabel eingetragen werden können und ein Hotfolder nur den Kommunikationskanal an sich darstellt.

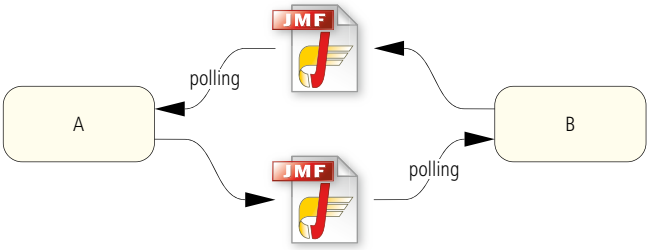
Anstatt über Dateitransfer werden Daten auch über Druckprotokolle an ein RIP-System weitergereicht. So lassen sich über das Druckmenü noch spezielle Einstellungen für einen Job vornehmen. Für JDF ist dieses Verfahren ungeeignet. Auch die herstellerübergreifende Weitergabe von JDF über Datenbanken ist unzweckmäßig, weil hierzu keine standardisierten Zugriffsverfahren existieren. Das Gleiche gilt für die Implementierung von Interprozesskommunikation, die zwar von einzelnen Herstellern

durchgeführt wird, aber nicht offen gelegt ist. Diese Technik findet man beispielsweise bei der PJTF-Kommunikation zwischen den Jobticket-Prozessoren.

Webservices definierten den Nachrichtenaustausch zwischen Applikationen im Web auf der Grundlage von XML [45]. So können beispielsweise Reservierungssysteme von Reisebüros mit Webservices von Hotels kommunizieren, um Anfragen und Buchungen vorzunehmen. Genauso könnten natürlich auch einzelne Module eines Workflow-Managementsystems über Webservices kommunizieren.

Abbildung 7.1
JMF-Nachricht

```
<JMF TimeStamp= "2008-07-25T12:32:48+02:00"...>
 ...
</JMF>
```




JMF über Hotfolder
unidirektional und asynchron
Empfänger muß nicht wärend des Sendens erreichbar sein

JMF über HTTP
bidirektional und synchron


Abbildung 7.2
unterschiedliche Arten der JMF-Kommunikation

Webservices benötigen Nachrichtenprotokolle, wie zum Beispiel SOAP (ursprünglich *Simple Object Access Protocol*) vom W3C. Dieses Protokoll wird in der Tat in Workflowsystemen eingesetzt. Es dient dem Austausch von XML-Nachrichten und kann deshalb auch JMF transportieren. Zum Senden kommt hauptsächlich HTTP zum Einsatz.

JMF-Nachrichten sind kleine XML-Dokumente, die ein Wurzelelement mit der Bezeichnung JMF haben, wie man in Abbildung 7.1 sehen kann. In Unterelementen des JMF-Wurzelelements sind dann die eigentlichen Nachrichten kodiert. Die Nachricht selber wird in der Regel über das HTTP- oder das HTTPS-Protokoll verschickt, also Protokolle, mit denen auch HTML-Webseiten über das Internet transportiert werden. Damit kann eine schnelle bidirektionale Kommunikation aufgebaut werden. Alternativ können manche JMF-Nachrichten aber auch als Dateien in Hotfoldern platziert werden, die der Empfänger wieder auslesen muss. Es gilt damit das gleiche Verfahren, das auch bei JDF weit verbreitet ist (Abbildung 7.2).

## 7.2 JMF-Familien

Was sind nun wichtige Inhalte, die mit JMF transportiert werden können? Hier sind ein paar Beispiele:

- Initialisierung von Geräten,
- Geräte- und Jobstatus,
- Kontrolle von Warteschlangen und deren Einträgen,
- Erledigung von Meilensteinen.

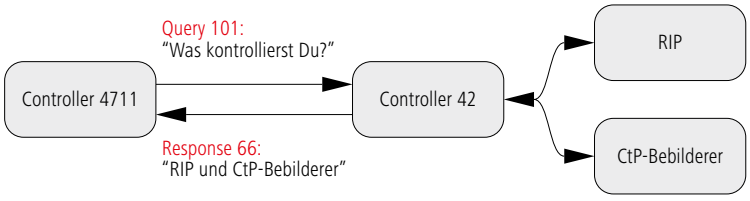
Vor allem das Weitergeben der Zustände von Geräten und Druckaufträgen steht im zentralen Interesse, denn daraus kann für die Anwender vielfältiger Nutzen gezogen werden. Die Job-Verfolgung (*job tracking*) ist damit nämlich genauso möglich wie eine Nachkalkulation. Auch eine zentrale Überwachung der Geräte wird mit Hilfe solcher Meldungen einfacher. In der Praxis ist es häufig nicht besonders schwierig, Meldungen bezüglich Gerätezuständen und Produktionsfortschritten zu erzeugen und zu versenden. Problematischer ist es vielmehr, die Massen von Daten anschließend nach sinnvollen Gesichtspunkten wieder zu filtern. Unter anderem aus diesem Grund findet die Nachkalkulation häufig auch nicht auf Basis von JMF-Nachrichten statt, sondern mit Hilfe der *AuditPools*, die wir in 6.2 vorgestellt haben. Diese *Audit*-Einträge werden typischerweise von Geräten in das eigene JDF eingetragen, das nach Abschluss aller Prozesse an das MIS oder an einen zentralen JDF-Workflow-Server zurückgegeben wird. Das Filtern der JMF-Nachrichten muss also vom Manager, das Schreiben der Audits vom Worker übernommen werden.

JMF-Mitteilungen werden in sechs Kategorien eingeteilt, die so genannten JMF-Familien:

- JMF-Anfrage (*Query*)
- JMF-Kommando (*Command*)
- JMF-Antwort (*Response*)
- JMF-Bestätigung (*Acknowledge*)
- JMF-Signal (*Signal*)
- JMF-Registrierung (*Registration*)

Eine JMF-Anfrage richtet sich an einen JMF-Controller oder an ein JMF-Gerät, um gewisse Information zu bekommen. Es verändert – im Gegensatz zu einem Kommando – nicht den Zustand des Adressaten. Dieser reagiert auf eine Anfrage üblicherweise mit einer

Abbildung 7.3
JMF-Anfrage und -Antwort



```
<JMF TimeStamp="..." SenderID="_4711">
    <Query Type="KnownDevices" ID="_101"/>
</JMF>

<JMF TimeStamp="..." SenderID="_42">
    <Response Type="KnownJDFServices" ID="_66" refID="_101"/>
        <DeviceList>
            <DeviceInfo DeviceStatus="Idle">
                <Device DeviceID="Rip" />
            </DeviceInfo>
            <DeviceInfo DeviceStatus="Running">
                <Device DeviceID="CtP"/>
            </DeviceInfo>
        </DeviceList>
    <Response>
</JMF>
```

Abbildung 7.4
JMF-Antwort
auf eine Anfrage

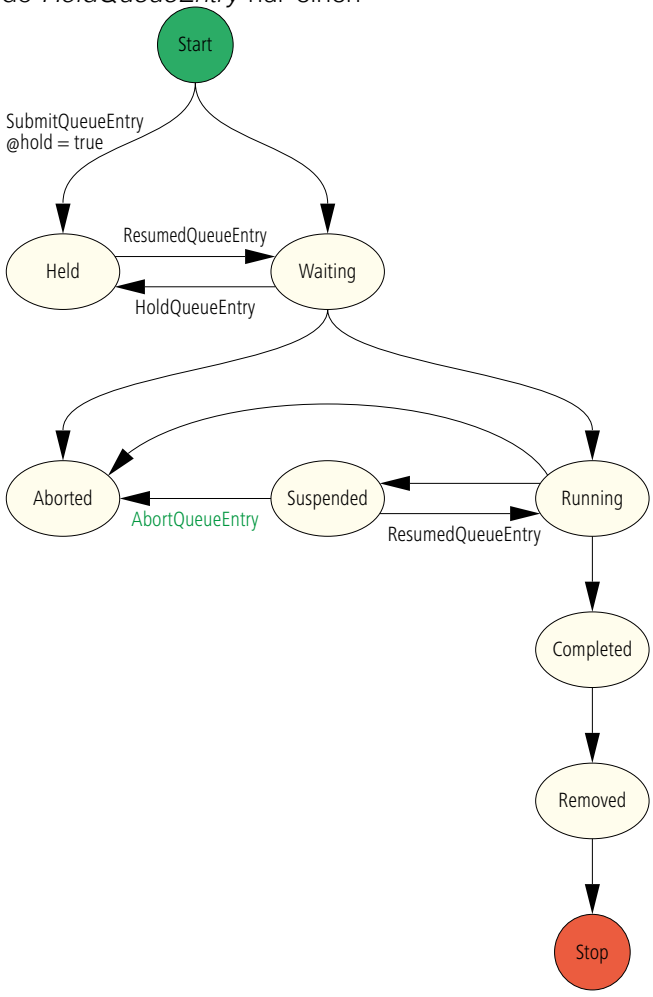
Antwort (*Response*). In 7.3 ist ein Beispiel für eine Anfrage und eine entsprechende Antwort als Grafik zu sehen. Jeder Controller und jedes Gerät hat eine eigene *ID,* und auch die Anfrage sowie die Antwort haben eine Kennung. So kann sich die Antwort eindeutig auf die zuvor erfolgte Anfrage beziehen. In dieser Anfrage möchte der Controller mit der *ID* = 4711 in Erfahrung bringen, welche Geräte der angesprochene Controller mit der *ID* = 42 überwacht. Solch eine Anfrage könnte zum Beispiel in einem Plug&Play-Verfahren für JDF-Module wichtig sein (wobei die Realität noch weit davon entfernt ist). Das Gerät antwortet dann, dass es einen RIP und einen CtP-Bebilderer verwaltet.

In 7.4 ist der entsprechende JMF-Code aufgelistet. Auffällig ist vielleicht, dass die *ID* des Empfängers weder bei der Anfrage, noch bei der Antwort im XML-Code enthalten ist. So ist die *ID* = 42 nicht in der Anfrage und die *ID* = 4711 nicht in der Antwort aufgeführt. Wie kann also die Nachricht überhaupt den richtigen Adressaten erreichen? Die Auflösung dieses Rätsels lautet, dass der Empfänger über das HTTP-Protokoll adressiert wird (oder auch über den Hotfolder). Anschaulich gesprochen, wird die JMF-Nachricht in einen HTTP-Umschlag mit entsprechenden Adressaufklebern gesteckt und dieser dann HTTP-gemäß verschickt. Man erkennt auch im Code, dass die Antwort sich über das Attribut *refID* eindeutig auf die zuvor abgesetzte Anfrage richtet.

Kommandos können Dinge verändern. Das gilt auch für JMF-Kommandos. Es lassen sich beispielsweise Einträge in Warteschlangen durch Kommandos löschen, während eine Anfrage

nur den Status einer Warteschlange erfragen könnte. Doch bevor wir hier weitermachen: Was ist eigentlich eine Warteschlange? Hier ist natürlich nicht die vor der Kasse im Supermarkt gemeint, sondern ganz allgemein eine Datenstruktur, die als Puffer fungiert und in aller Regel nach dem Prinzip *First In - First Out* (wer zuerst kommt, mahlt zuerst) funktioniert. Das können also beispielsweise ein Datenpuffer vor einem RIP sein, in der die zu rippenden Daten zwischengelagert werden, oder aber Zwischenspeicher für JDF-Jobs vor einem Controller oder einem Gerät. Bezüglich Warteschlangen gibt es zwei Sorten von JMF-Kommandos: Entweder betreffen sie individuelle Einträge innerhalb einer Warteschlange oder aber eine komplette Warteschlange selbst. So hält das Kommando *HoldQueue* die gesamte Warteschlange an und kein Job wird mehr weiter bearbeitet, während das Kommando *HoldQueueEntry* nur einen einzelnen Job in der Warteschlange auf Eis legt. Auch wenn Kommandos bezüglich Warteschlangen oder deren Einträge auf den ersten Blick vielleicht unwichtig oder sogar unnütz erscheinen, so sollte man sich klar darüber sein, dass ohne sie Workflows eingeschränkt wären. Nur wenn beispielsweise ein Gerät einem Controller mitteilen kann, dass ein Job nicht mehr warten muss, sondern gerade aktiv ausgeführt wird, kann diese Information wiederum dazu verwendet werden, für den Anwender eine Grafik des Produktionszustandes zu aktualisieren.

Abbildung 7.5
Zustände und Zustandsübergänge bei Warteschlangeneinträgen




Warteschlangeneinträge haben unterschiedliche Zustände, wie:

- wartend (*Waiting*),
- gestoppt (*Held*),
- aktiv (*Running*),
- aufgeschoben (*Suspended*),

```
<JMF SenderID="Controller-1" TimeStamp="2008-08-13T10:05:32+01:00"
Version="1.3"...>
<Command ID="C1" Type="AbortQueueEntry">
       <QueueEntryDef QueueEntryID="job-4711" />
    </Command>
</JMF>

<JMF SenderID="Device-1" TimeStamp="2008-08-13T10:05:33+01:00"
 Version="1.3"...>
    <Response ID="R1" Type="AbortQueueEntry" refID="C1">
        <Queue DeviceID="Device-1" Status="Running">
            <QueueEntry JobID="job-4711" QueueEntryID="job-4711"
             Status="Aborted" />
        </Queue>
    </Response>
</JMF>
```

Abbildung 7.6
Kommando, das einen Job in einer Warteschlange abbricht und die entsprechende Antwort

- abgebrochen (*Aborted*),
- gelöscht (*Removed*).

Diese Zustände können nicht nur Warteschlangen, sondern auch Geräte betreffen, wie in Abbildung 7.4 zu erkennen ist (*DeviceStatus*).

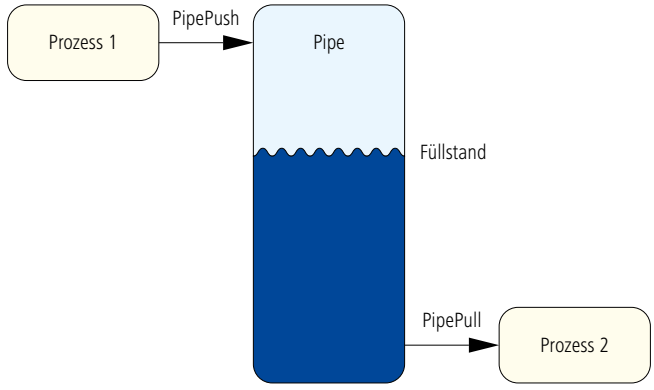
Abbildung 7.5 zeigt die Zustände und einige Zustandsübergänge (um die Zeichnung nicht zu überfrachten, haben wir einige weggelassen). Außerdem sind in einigen Fällen JMF-Kommandos eingezeichnet, welche die Zustandsübergänge

Abbildung 7.7
individueller Warteschlangeneintrag

```
<Command ID="C2" Type="SubmitQueueEntry">
    <QueueSubmissionParams URL="File://HdM/Server1/Jobs/job1.jdf" />
</Command>
```

Abbildung 7.8
Lagerhaltung mit Pipes und JMF-Kommandos zum Füllen und Entleeren

bewirken. Man beachte den Unterschied zwischen *Held* und *Suspended*. In beiden Fällen werden Jobs zur weiteren Bearbeitung ausgesetzt, nur einmal aus dem Zustand *Waiting*



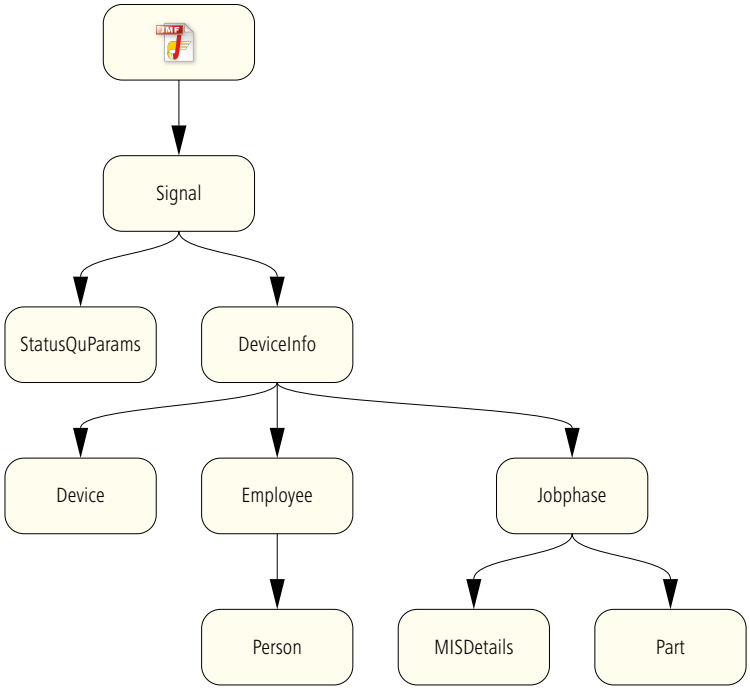
heraus, das andere Mal aus dem Zustand *Running*.

Das Kommando zum Abbruch eines Jobs sowie die Antwort (*Response*) auf dieses Kommando sind in dem Code-Beispiel 7.6 zu sehen. Der Controller-1 schickt ein *AbortQueueEntry*-Kommando (C1) in Bezug auf den Warteschlangeneintrag *job-4711*. Der Empfänger dieses Kommandos ist *Device-1*; seine Antwort bezieht sich auf das Kommando *C1*. Der abgebrochene Job ist zwar weiterhin ein Eintrag in der Warteschlange, allerdings nun mit dem *Status* gleich *Aborted* gekennzeichnet.

JDF-Jobs können, wie wir wissen, als Dateien über Hotfolder transportiert werden. Eine andere Möglichkeit besteht darin, einem Controller oder einem Gerät mitzuteilen, wie und wo diese JDF-Datei abgeholt werden kann, wobei es beim „Wie“ zwei Möglichkeiten gibt: Entweder über normalen Dateitransfer oder über HTTP. Das Code-Beispiel in 7.7 zeigt den Sachverhalt für Dateitransfer.

Weitere Beispiele für Kommandos findet man bei der Datenstruktur *Pipe*, die schon in Abschnitt 2.6 kurz eingeführt wurde. Bei überlappender Produktion legt ein Prozess P1 eine Ressource in eine *Pipe*, aus der zeitversetzt ein zweiter Prozess P2 diese wieder herausholt. Das Besondere an dieser *Pipe*-Struktur ist, dass eine partitionierte Ressource nicht erst komplett in die Pipe eingebracht werden muss, bevor der nächste sie wieder von dort lesen kann. Bei dem in Kapitel 2 erwähnten Beispiel war P1 die Plattenbebilderung und P2 der Druckprozess: Während noch Platten eines Jobs bebildert werden, können bereits die ersten Signaturen gedruckt werden. Die Aktionen, eine Ressource in eine Pipe zu legen (*PipePush*) beziehungsweise zu entnehmen (*PipePull*), sind ebenfalls Kom-

Abbildung 7.9
JMF-Signal



```
<JMF SenderID="_4004"...>
    <Signal ID="S1" Type="Status">
        <StatusQuParams JobID="_0001" JobPartID="_002"... />
        <DeviceInfo DeviceStatus="Idle" TotalProductionCounter="13862.0">
            <Device DeviceID="_4004" Status="Available"... />
            <Employee ID="_111" PersonalID="_013" Roles="Operator"
             Status="Available">
                <Person FamilyName=”Cool” FirstName=“Carl“... />
            </Employee>
            <JobPhase JobID="_001" JobPartID="_002"
             StartTime="2008-07-22T13:18:12+02:00" Status="Completed"
             TotalAmount="1002.0" Waste="71.0">
                <MISDetails CostType="Chargeable" WorkType="Original" />
                <Part SheetName="Blaetter" SignatureName="SIG1" />
            </JobPhase>
        </DeviceInfo>
    </Signal>
</JMF>
```

Abbildung 7.10
Code-Beispiel
für ein JMF-Signal

mandos (Abbildung 7.8).

Schließlich möchten wir noch die *Resource Messages* erwähnen. Das sind Anfragen oder Kommandos bezüglich JDF-Ressourcen. So können beispielsweise Material-, Geräte- oder Jobressourcen abgefragt (*ResourceQuery*) oder auch modifiziert (*ResourceCommand*) werden. Mit Hilfe von Kommandos können entweder JDF-Ressourcen in einer gänzlich neuen Version an ein Gerät geschickt oder vorhandene Ressourcen aktualisiert werden. Wurde zum Beispiel eine Platte belichtet, so kann der entsprechende Controller ein Kommando verschicken, damit der Status der JDF-Ressource oder eines Teils davon auf *available* gesetzt wird.

Liste 7.11
Werte für das Attribut
StatusDetails

BlanketChange
BlanketWash
BreakDown
CleaningInkFountain
ControlDeferred
CoverOpen
CylinderWash
DampeningRollerWash
DoorOpen
Failure
FormChange
Good
InkRollerWash
Maintenance
OutputAreaFull
PlateWash
Repair
ShutDown
SizeChange
SleeveChange
WaitForApproval
WarmingUp
WashUp
Waste

Unter einer Bestätigung (*Acknowledge*) versteht man bei JMF eine zeitlich versetzte (asynchrone) Antwort auf eine Anfrage oder ein Kommando. Denn Anfragen, vor allem Kommandos, können eine Weile benötigen, bis sie ausgeführt werden. So schickt der Empfänger zunächst eine *Response*, dass er die Anfrage oder das Kommando erhalten hat und erst später eine Bestätigung, dass der Auftrag ausgeführt wurde. Im normalen Leben ist die Terminologie allerdings umgekehrt: Ein Auftrag wird sofort bestätigt und später beantwortet.

Signale sind unidirektionale Meldungen, die häufig Statusänderungen von Maschinen oder Jobs betreffen. Controller können auf unterschiedliche Art und Weise solche Signale „abonnieren“, das heißt bekannt geben, dass sie einen gewissen Signaltyp zugeschickt bekommen möchten. In Abbildung 7.9 ist die Struktur eines Signals skizziert, das über den Produktionssta-

tus einer Druckmaschine informiert. Die JMF-Nachricht besteht aus einem Signal, das seinerseits zwei Unterelemente enthält. Das *StatusQuParams*-Element definiert hier den Job, auf den sich das Signal bezieht. Das *DeviceInfo*-Element gibt Auskunft über das Gerät, dessen augenblicklichen Status und auch Details über Produktionszähler und dergleichen. In den Unterelementen *Device*, *Employee* und *JobPhase* stehen weitere Informationen über das Gerät, den Bediener des Gerätes (sofern er/sie sich angemeldet hat) und über den Job. In Abbildung 7.10 ist der Code dieses Signals verkürzt wiedergegeben. In dem Element *JobPhase* erkennt man, dass der Job abgeschlossen wurde (der *Status* ist *completed*), wobei insgesamt 1002 Bogen gedruckt wurden, davon 71 Makulaturbogen. In dem Element *MISDetails* ist außerdem vermerkt, dass der Kunde für den Auftrag zahlen muss (das Attribut *CostType* ist auf *Chargeable* gesetzt) und dass der Originalauftrag ohne Änderung ausgeführt wurde (der *WorkType* ist *Original*). Das Attribut *DeviceStatus* im Element *DeviceInfo* steht auf *Idle*, also „untätig"; andere Zustände sind: *Down*, *Setup*, *Running*, *Cleanup*, *Stopped* oder *Unknown*. Ein weiteres Attribut im selben Element (in 7.10 nicht enthalten) heißt *StatusDetails*. Mit diesem können noch präzisere Angaben zum Zustand von Maschinen und speziell Druckmaschinen gemacht werden. Die Liste 7.11 enthält eine Auswahl der Möglichkeiten. Gerade solch eine Liste verdeutlicht, wie genau und umfassend JDF und JMF spezifiziert wurden.

Das letzte Mitglied der JMF-Familie ist die Registrierung. Hierbei geht es darum, einen anderen zu beauftragen, zu gewissen Anlässen Kommandos an Dritte zu versenden. Beispielsweise könnte ein MIS ein Vorstufen-WMS dazu veranlassen, CIP3-Daten an eine bestimmte Druckmaschine zu senden. Diese Befehlshierarchie benötigt feste Kommunikationskanäle (*persistent channels*), die es zunächst einzurichten gilt. Da diese auch für Signale wichtig sind, möchten wir sie hier genauer darstellen – allerdings erst im nächsten Abschnitt. Auf Registrierungen im Speziellen werden wir ansonsten nicht weiter eingehen.

Zu guter Letzt soll aber noch erwähnt werden, dass auch das JMF-Protokoll mit privaten Inhalten erweitert werden kann. Durch das Hinzufügen von Namensräumen können neue Elemente und Attribute geschaffen werden. Ähnlich wie bei JDF erhöht sich mit solch einer Maßnahme die Flexibilität auf Kosten der Kompatibilität.

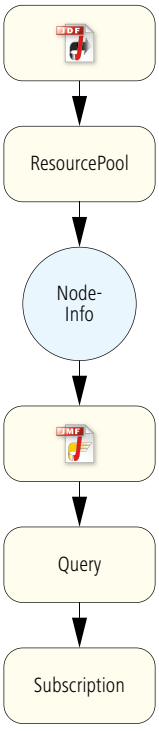



Abbildung 7.12
Mit Hilfe der NodeInfo wird ein Abonnement beantragt

## 7.3 JMF ICS

Die *Interoperability Conformance Specification* für JMF [12] spezifiziert Bedingungen für JDF-/JMF-Module in zwei Ausbaustufen (*levels*).

- Level 1
  a) JDF-Daten werden über Hotfolder weitergegeben.
  b) Manager können feste Kommunikationskanäle nur dadurch aufbauen, dass entsprechende Anfragen oder Registrierungen im JDF-Dokument platziert werden.
- Level 2
  a) Manager können feste Kommunikationskanäle durch JMF-Messages direkt über HTTP aufbauen.
  b) JDF-Daten können über JMF weitergegeben werden.
  c) Es findet ein bidirektionaler JMF-Nachrichtenaustausch zwischen *Manager* und *Worker* statt. Insbesondere können die Verabredungen bezüglich Warteschlangen über JMF realisiert werden.

Das Element *NodeInfo* wurde bereits in Abschnitt 6.5 vorgestellt, über das beispielsweise eine Terminsteuerung für die Durchführung eines Prozesses realisiert werden kann. Tatsächlich können mit Hilfe dieser Ressource auch feste Kommunikationskanäle (*persistent channels*) aufgebaut werden, wie es der Level 1 in Forderung b) verlangt. Diese festen Kommunikationskanäle werden typischerweise dafür benötigt, dass ein Controller Signale von JMF-fähigen Geräten abonnieren kann. So kann also ein MIS festlegen, dass es von einer Druckmaschine regelmäßig über ihren Zustand und über Produktionsdetails informiert werden möchte. Der Begriff „abonnieren“ ist dabei das Zauberwort, denn es wird in der Tat ein *Subscription*-Element (*subscription* = Abonnement) in die *NodeInfo*-Ressource eingebaut. Die Elter-Kind-Beziehungen sind in Abbildung 7.12 zu sehen. Innerhalb einer Anfrage (*Query*) kann also eine feste Kommunikation definiert werden, die so lange Gültigkeit hat, bis sie entweder wieder gekappt wird oder bis der entsprechende JDF-Knoten abgearbeitet wurde.

Natürlich können Anfragen nach diesen festen Kommunikationskanälen auch direkt als JMF-Knoten in HTTP verpackt und versendet werden. Das ist dann die Forderung a), die in Level 2 erfüllt sein muss. Auch die Forderungen b) und c) wurden bereits vorgestellt. Insbesondere hatten wir in Abbildung 7.7 gesehen, wie durch das Kommando von Typ *SubmitQueueEntry* JDF-

Jobs über das HTTP-Protokoll an Geräte gebracht werden können.

Abschließend möchten wir noch einmal unabhängig von den ICS-Levels die verschiedenen Möglichleiten zur Versendung von Nachrichten in einer JDF/JMF-Umgebung zusammenfassen, wobei die Implementierungskomplexität von oben nach unten zunimmt.

- keine JMF-Nachrichten
- JMF-Signale
- JMF-Anfragen / JMF-Antworten
- JMF-Kommandos
- JDF-Übergabe via HTTP

**Übung**

- Laden Sie einen HTTP-Sniffer als Freeware aus dem Internet herunter und installieren sie diese Software. Versuchen Sie HTTP-Pakete, die Sie durch Benutzung eines Browsers generieren, im Sniffer abzufangen und zu lesen.

Einige Hersteller von Auftrags-Managementsystemen:

Alphagraf
Hiflex
Megalith
Optimus
Pagina
Printplus
Prosecco

# 8 Auftrags-Managementsysteme

Auftrags-Managementsysteme (AMS) haben in der grafischen Industrie viele Namen: Sie werden auch Management Information Systeme (MIS), Branchenlösungen, Enterprise Resource Planing (ERP), Produktionsplanung- und Steuerungssysteme (PPS) oder schlicht Kalkulationsprogramme genannt. Natürlich gibt es zwischen den Begriffen Unterschiede – in der Praxis werden sie aber häufig synonym benutzt, und wir werden im Folgenden AMS und MIS gleichermaßen verwenden. Auch wollen wir nicht Begriffsdefinition betreiben, sondern nur die wesentlichen Punkte auflisten, die eine solche Software umfassen kann:

- Vorkalkulation und Nachkalkulation
- Kommerzielle Auftragsabwicklung (von Angeboterstellung bis Fakturierung)
- Kundendatenverwaltung
- Materialwirtschaft
- Schnittstelle zur Produktion
  - Übergabe der Produktionsdetails
  - Produktionsplanung / Disposition (Termin- und Ressourcenplanung)
  - Leistungserfassung / Betriebsdatenerfassung (BDE)
  - Auftragsverfolgung (*Job Tracking*)
- Schnittstelle zu Finanz- und Lohnbuchhaltung
- Schnittstelle zu Kunden (E-Business und Internetportale)
- Schnittstelle zu Lieferanten (Papierlieferant)

Typischerweise sind AMS in Druckereien installiert. Doch nicht alle aufgelisteten Funktionen sind grundlegend für ein Auftrags-Managementsystem und stellen meist nur optionale Module dar, wie zum Beispiel die Produktionsplanung. Auch sind nicht alle ausschlaggebend für den JDF-Workflow, wie etwa die Schnittstelle zur Finanz- und Lohnbuchhaltung. Vorkalkulation, kommerzielle Auftragsabwicklung, Kunden- und Materialverwaltung sind hingegen die wichtigsten Funktionen eines AMS. In den folgenden Abschnitten werden wir deswegen manche dieser Begriffe etwas genauer beleuchten und auch die Abhängigkeiten zwischen ihnen analysieren. Für den JDF-Workflow ist die Schnittstelle vom AMS zur Produktion von ganz besonderer Bedeutung, die wir deswegen im anschließenden Abschnitt untersuchen werden. Die Schnittstelle kann auch in umgekehrter Richtung von der Produktion zum AMS realisiert sein, wobei dann BDE-Daten für die Nachkalkulation sowie Rückmeldungen zum Produktionsstatus und Termine übergeben werden. Die

Kommunikation zwischen Druckproduktеinkäufer und Druckproduktherstellеr mittels PrintTalk ist dann Thema in Abschnitt 8.4.

## 8.1 Grundfunktionalität eines AMS

Es ist nicht Thema dieses Buches, wie eine Kalkulation erstellt wird und wie sie funktioniert. Hier geht es nur darum, welche Inputdaten eine Kalkulation benötigt; die eigentliche Berechnung zur Preisbestimmung eines Druckproduktes bleibt jedoch in diesem Buch eine magische Glaskugel und wird nicht ausgeführt. Die Inputdaten zur Kalkulation sind insofern wichtig, da diese – zumindest teilweise – über JDF an die Produktion weitergereicht werden können.

Zur Kalkulation eines Druckproduktes werden in der Regel Informationen aus den folgenden Bereichen benötigt:

- Geschäftsdaten
- Ausstattung des Produktes
- Produktionsmaschinen
- Materialdaten
- Produktionsprozesse

Die benötigten Produktionsprozesse sind zwar in vielen Fällen auch Schlussfolgerungen aus Produktdefinition und Produktionsmaschinen, hängen in der Druckvorstufe andererseits aber auch von den Eigenschaften der Daten ab, die eine Druckerei vom Auftraggeber erhält.

In den Kalkulationsprogrammen wird recht deutlich zwischen Anwendung und Administration unterschieden. Während die Anwender (Sachbearbeiter, Innnendienstmitarbeiter) die Kalkulation vornehmen, pflegen Systemadministratoren die Stammdaten. Zur Stammdatenpflege gehören insbesondere:

- Einrichten von Kostenstellen
- Eingabe von Kosten und Dauer einzelner Arbeitsvorgänge (Platzkostenberechnung)
- Definition von Produkttypen
- Eingabe von Parametern für die Maschinen der Druckerei
- Einrichten von Kunden- und Materialdatenbanken
- Benutzerverwaltung
- Formularbearbeitung

Der Anwender kann, wenn die Stammdaten korrekt eingerichtet sind, die zur Kalkulation benötigten Daten aus vordefinierten Datensätzen einfach und schnell zusammenstellen. Häufig wird

zunächst der Kunde aus einer Datenbank ausgewählt und damit auch gleich der Ansprechpartner, dessen Kontaktdaten, Liefer- und Rechnungsadresse, und außerdem kann eine Bonitätsprüfung (Kreditwürdigkeit) erfolgen. Bei der Produktdefinition werden die Produktart (Buch, Broschur, Visitenkarte...) aus einer Liste ausgesucht und die Bindungsart, der Seitenumfang, die Farbigkeit, das Endformat, der Beschnitt und die Auflagenhöhe definiert. Bei komplexeren Produkten müssen Produktteile und deren Werte separat angegeben werden. So kann sich zum Beispiel bei einer klebegebundenen Broschur die Farbigkeit des Umschlags von der des Inhalts unterscheiden. In der Rubrik „Produktionsmaschinen“ kann der Anwender eine als Parametersatz hinterlegte Druckmaschine der Druckerei auswählen. Damit liegen dann auch die Plattengröße, gegebenenfalls der Greiferrand, der Papierbeginn, die Widerdruckoptionen und die minimale und maximale Bogengröße bzw. Rollenbreite und Abschnittslänge fest. Des weiteren wird der Bedruckstoff spezifiziert, das Rohbogenformat beim Bogenoffset, die Papierart bzw. Papierklasse, die Grammatur und die Laufrichtung. Anschließend werden, abhängig vom Produkt, ein oder mehrere Ausschießschemata festgelegt, die üblicherweise aus einem Schematakatalog ausgewählt werden können. Bei der Auswahl des Bedruckstoffs können dann auch gleich Preise aus einer entsprechenden Datenbank geladen werden, die aber vom Anwender noch überschrieben werden können. Natürlich können auch Produktionsmaschinen der Druckweiterverarbeitung für einen Druckauftrag entweder vom Operator ausgewählt oder aber vom System automatisch zugewiesen werden.

Schließlich müssen noch einzelne Produktionsschritte definiert werden, die nicht vom Systemadministrator als Standard für die Herstellung des Produkttyps hinterlegt sind. Das können vom Kunden zusätzlich gewünschte Proofs sein, spezielle Aufwände zur Herstellung der eigentlichen Druckdaten oder Ausstattungen des Druckproduktes, die in der Weiterverarbeitung weitere Arbeitsschritte verursachen.

Aus all diesen Informationen berechnet das AMS die Herstellungskosten und schlägt einen Verkaufspreis vor. Danach können Angebotschreiben für den Postversand ausgedruckt, gefaxt oder per E-Mail verschickt werden. Nach Auftragseingang wird dann eine Auftragsbestätigung an den Kunden abgesendet, eine Auftragstasche und die Daten für die Produktion generiert – letztere zumindest bei einem JDF-Workflow.

Die Nachkalkulation dient zur Kontrolle, wie der Auftrag abge-

schlossen hat, und auch, um auftragsbezogene Abweichungen festzustellen. Dadurch können einerseits die Vorkalkulation optimiert, andererseits können auch Verbesserungen zur Produktionseffektivität entdeckt werden. Meist wird der Begriff Nachkalkulation aber etwas eingeschränkt verwendet, nämlich auf die Ermittlung der tatsächlichen Arbeitszeiten der eingesetzten Geräte und den wirklichen Materialverbrauch für den Auftrag. Die Kalkulationsgrundlage, nämlich die Stundensätze von Kostenstellen und damit die Kosten einzelner Arbeitsgänge, wird von der Nachkalkulation weniger erfasst. Eine Nachkalkulation setzt also eine Betriebsdatenerfassung voraus. Diese kann über Stundenzettel erfolgen, welche die Mitarbeiter eines Unternehmens manuell ausfüllen und die anschließend in eine Buchhaltungs-Software übertragen werden müssen. Anstatt die Stundenzettel auf Papier auszufüllen, werden auch häufig die Angaben vom Mitarbeiter in BDE-Terminals eingegeben. Hierzu gibt es schon seit langem herstellerspezifische Systeme, meist Zusatzkomponenten vom Auftrags-Managementsystem. Doch das eigentliche Ziel einer JDF-/JMF-Umgebung ist es, die BDE-Daten weitgehend automatisch zu generieren. Mit JMF gibt es die Chance, dass herstellerübergreifend viele Geräte ihre Daten mit einem standardisierten Protokoll an eine zentrale Auswertungsstelle schicken – also zum Beispiel an das MIS. Denn mit diesen dynamisch erzeugten Daten wird nicht nur die Nachkalkulation weniger aufwändig und trotzdem gleichzeitig exakter, sondern es können auch Gerätefehler zentral gemeldet und der Produktionszustand von Jobs verfolgt werden.

Aber auch in einer JDF-Welt werden nicht alle Daten automatisch generiert. So gibt es häufig Rechner mit entsprechender Software direkt neben nicht anbindbaren Maschinen oder Handarbeitsplätzen, an denen Zeiten erfasst und verbrauchte oder erzeugte Ressourcen eingegeben werden können. Diese Daten werden dann mittels JDF oder JMF an das MIS oder an eine andere Instanz zur Auswertung geschickt.

## 8.2 JDF-Schnittstelle zur Produktion

Die JDF-Schnittstelle zwischen Auftrags-Managementsystem und Produktion ist eine der wichtigsten JDF-Schnittstellen überhaupt. Die Realisierung dieser Schnittstelle ist mittlerweile nicht nur weit verbreitet, sondern auch offen in dem Sinne, dass AMS und Produktionssoftware unterschiedlicher Hersteller über JDF

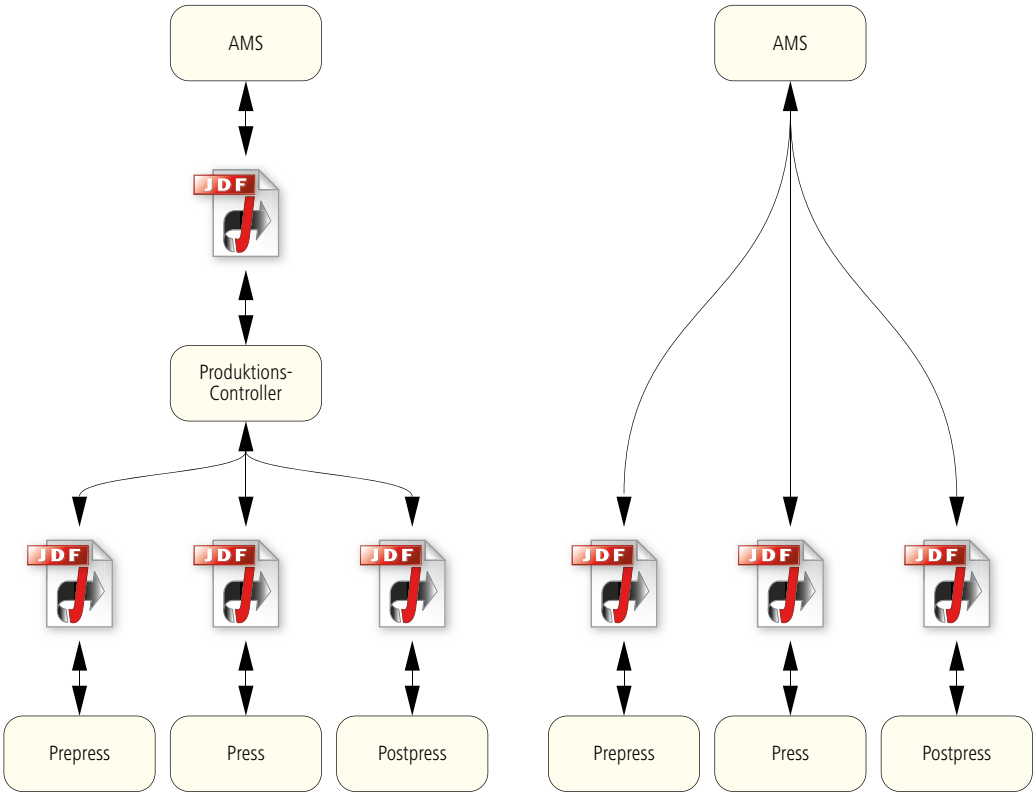



Abbildung 8.1 JDF zwischen MIS und Produktion

und JMF kommunizieren können. Dass dieses nicht für alle JDF-Schnittstellen gilt, werden wir vor allem noch in Kapitel 9 sehen. Für die Druckdienstleister Y und Z in den Abschnitten 2.2 und 2.3 wurden bereits solche JDF-Schnittstellen zwischen MIS und Prepress skizziert.

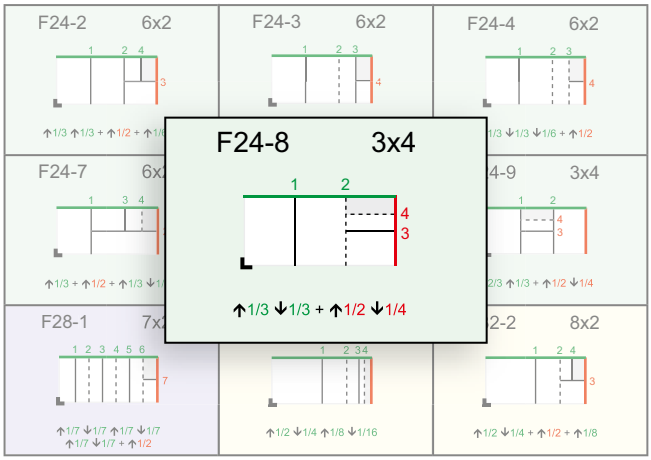



Abbildung 8.2 F24-8 des JDF-Falzartenkataloges

Grundsätzlich gibt es zwei Modelle, wie JDF-Daten zur Produktion weitergegeben werden können. Entweder übergibt das AMS die JDF-Daten einem zentralen JDF-Produktions-Controller, der seinerseits die Informationen an Controller und Geräte in den Abteilungen Prepress, Press und Postpress verteilt, oder es übernimmt selber die Übermittlung an die unterschiedlichen Geräte (siehe Abbil-

dung 8.1). Dieser scheinbar kleine Unterschied hat jedoch große Implikationen. Denn in der Konfiguration links liegt die Workflow-Steuerung in der Produktion, während in der rechten Grafik das MIS diese Aufgabe übernimmt. Führt das MIS die Workflow-Steuerung durch, muss es die technischen Details aus der Produktion wie beispielsweise die Farbzonenvoreinstellungswerte, die Schneidepositionen oder die Falzreihenfolge aus der Produktion entgegennehmen und an die geeigneten Geräte/Controller im Drucksaal oder in der Weiterverarbeitung weiterleiten können. Damit geht die Anforderung an das MIS weit über die klassische, kommerzielle Aufgabe hinaus. Beide in 8.1 gezeigte Konfigurationstypen wurden von Herstellern implementiert.

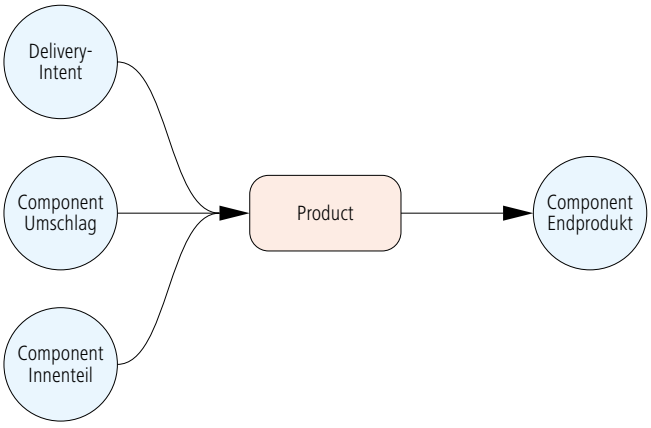



Abbildung 8.3
JDF-Wurzelknoten einer Broschur

In diesem Abschnitt werden wir in unseren Beispielen einen zentralen JDF-Produktions-Controller voraussetzen. Die Übergabe von JDF-Daten für Weiterverarbeitungsmaschinen ist allerdings bisher noch wenig im Einsatz und deswegen wird diese Schnittstelle auch separat in Kapitel 11 behandelt.

Die Schnittstelle wird nicht abstrakt, sondern vielmehr anhand

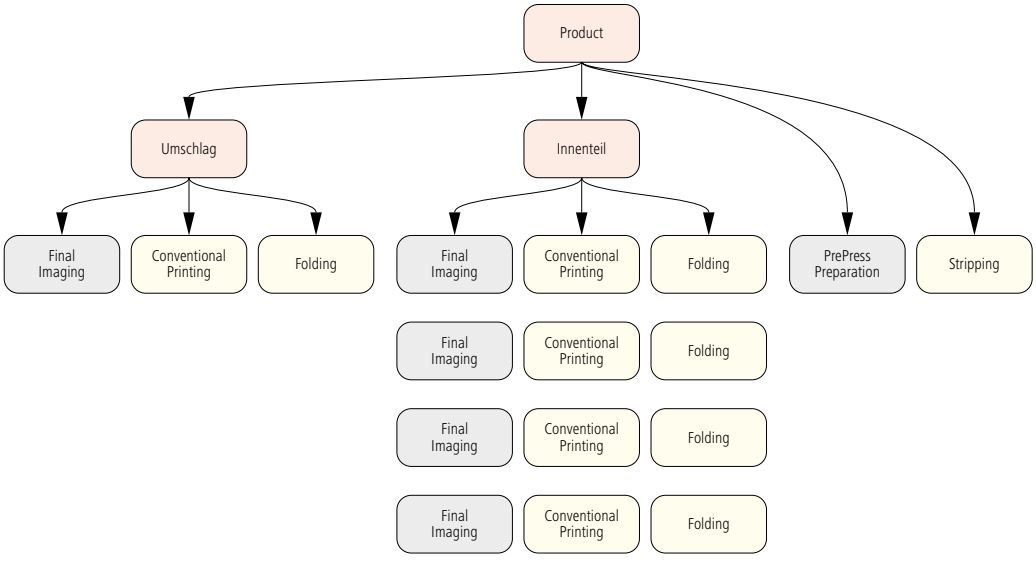



Abbildung 8.4
Baumstruktur der JDF-Knoten

eines Beispiels beschrieben. Bei dem Beispielprodukt geht es um eine 96-seitige, vierfarbige, klebegebundene Broschur mit dem Endformat von 12 x 12 cm. Sowohl der Umschlag als auch der Innenteil wird auf Bogen der Größe 43 x 61 cm gedruckt. Der Falzbogen für den Innenteil ist gemäß F24-8 des JDF-Falzartenkatalogs mit einer 3x4-Ausschießmatrix aufgebaut, so dass es insgesamt 4 Falzbogen gibt (4 x 24 = 96 Seiten). Der Umschlag wird mit 6 Nutzen auf dem Bogen beidseitig gedruckt (Abbildung 8.2).

Die Komponentensicht der JDF-Datei, die typischerweise von einem AMS generiert wird, ist in Abbildung 8.3 zu sehen. Der Produktknoten hat zwei *Component*-Ressourcen als Input, nämlich den Umschlag und den Innenteil. Das Endprodukt ist die Output-Ressource vom Typ *Component*.

Aus Sicht der Produktknoten, *GrayBoxen* und Prozessknoten erhalten wir eine Baumstruktur, die in Abbildung 8.4 wiedergegeben ist. Die roten Rechtecke sind die Produktknoten, die gelben die Prozessknoten und die grauen die *GrayBoxes* – also Prozessgruppenknoten. Da für den Innenteil vier unterschiedliche Falzbogen produziert werden müssen, werden die *GrayBox* der Kategorie *FinalImaging* und die Prozesse *ConventionalPrinting* und *Folding* auch entsprechend vier mal angelegt.

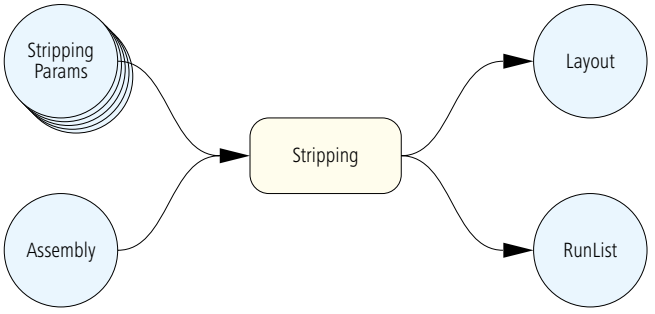
Abbildung 8.5 (unten) Ressourcen der GrayBox PrePressPreparation

Abbildung 8.6 (darunter) RunList-Ressource als Input für PrePressPreparation

Abbildung 8.7 (ganz unten) JDF-Prozess Standbogenerstellung

Einige der hier aufgeführten Prozesse und Prozessgruppen wurden bereits bei der Einführung in JDF erwähnt. Wir möchten sie hier an dieser Stelle nur kurz charakterisieren, bevor wir sie anschließend zumindest teilweise noch genauer analysieren.


```
<RunList Class="Parameter" ID="_001"
NPage="106" Status="Available" />
```




- *PrePressPreparation*: *GrayBox*, die alle Arbeitsschritte von der Aufbereitung der *Content*-Daten bis hin zum Ausschießen umfasst.
- *Stripping*: Bogenmontage
- *FinalImaging*: Diese *GrayBox* umfasst das Ausschießen (*Imposition*), die *GrayBox RIPing*, das Erzeugen

von Voransichtsbilder (*PreviewGeneration*) und die Plattenbelichtung (*ImageSetting*).

- *ConventionalPrinting*: Konventioneller Druck mit physischer Druckform. Hier: Offsetdruck
- *Folding*: Falzen

Der Prozessgruppenknoten *PrePressPreparation* ist sehr einfach strukturiert, wie man in Abbildung 8.5 sieht. Die In-

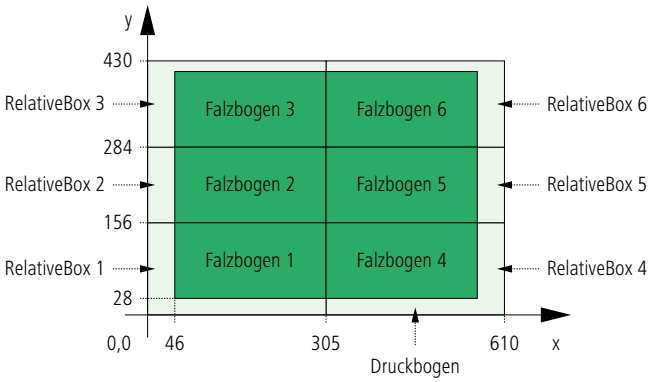



Abbildung 8.8 (oben) StrippinParams enthalten die Falzbogenpositionen

Abbildung 8.9 (unten) StrippingParams enthalten Infos für die Montage

```
<StrippingParams Class="Parameter" ID="_200" PartIDKeys="SignatureName
 SheetName" PartUsage="Explicit" Status="Available">
   <StrippingParams SignatureName="SIG001">
      <StrippingParams SheetName="Umschlag" WorkStyle="WorkAndBack">
         <BinderySignatureRef rRef="_201" />
         <Position MarginBottom="79.370" MarginLeft="130.39"
          MarginRight="0.0" MarginTop="0.0" Orientation="Rotate0"
          RelativeBox="0.0 0.0 0.5 0.362" />
         <Position MarginBottom="0.0" MarginLeft="130.39" MarginRight="0.0"
          MarginTop="0.0" Orientation="Rotate0"
          RelativeBox="0.0 0.362 0.5 0.660" />
         <Position MarginBottom="0.0" MarginLeft="130.393" MarginRight="0.0"
          MarginTop="51.023" Orientation="Rotate0"
          RelativeBox="0.0 0.660 0.5 1.0" />
         <Position MarginBottom="79.37" MarginLeft="0.0"
          MarginRight="130.393" MarginTop="0.0" Orientation="Rotate0"
          RelativeBox="0.5 0.0 1.0 0.3627" />
         <Position MarginBottom="0.0" MarginLeft="0.0"
          MarginRight="130.393" MarginTop="0.0"
          Orientation="Rotate0" RelativeBox="0.5 0.362 1.0 0.660" />
         <Position MarginBottom="0.0" MarginLeft="0.0" MarginRight="130.393"
          MarginTop="51.023" Orientation="Rotate0"
          RelativeBox="0.5 0.660 1.0 1.0" />
         <StripCellParams BleedFace="7.086" BleedFoot="7.086"
          BleedHead="7.086" BleedSpine="0.0" Spine="0.0" TrimFace="11.338"
          TrimFoot="11.338" TrimHead="11.338" TrimSize="340.157 340.157" />
         <MediaRef rRef="_202">
            <Part SheetName="Umschlag" SignatureName="SIG001" />
         </MediaRef>
         <MediaRef rRef="_203">
            <Part SheetName="Umschlag" SignatureName="SIG001" />
         </MediaRef>
         <DeviceRef rRef="_204" />
      </StrippingParams>
   </StrippingParams>
...
</StrippingParams>
```

put-Ressource *RunList* definiert die Druckseiten, die vom Kunden angeliefert werden. Zu diesem Zeitpunktpunkt, also nach Auftragsannahme, ist meist aber nur die Anzahl der Seiten des Druckproduktes bekannt und noch nicht der Name und Speicherort der *Content*-Daten (siehe 8.6). Die Output-Ressource *RunList* hingegen repräsentiert die für die Produktion aufbereiteten Seiten, die insbesondere also normalisiert, farbraumtransformiert und getrappt sind.

Der *Stripping*-Prozess mit seinen beiden Input- und Output-Ressourcen ist in Abbildung 8.7 wiedergegeben. Es gibt für jede Signatur eine Input-Ressource *StrippingParams*, also insgesamt fünf in unserem Beispiel. Diese enthalten im Wesentlichen jeweils die Position der Falzbogen auf dem Druckbogen (siehe Abbildung 8.8) und einen Link auf das Ausschießschema des Falzbogens. In Abbildung 8.9 sind die Positionen der sechs Falzbogen einzeln aufgeführt. Die Angaben sind jedoch nicht ganz einfach zu verstehen. Zunächst wird der Druckbogen in sechs Rechtecke aufgeteilt, deren Ausmaße in dem Attribut *RelativeBox* angegeben sind. Die vier Werte, die dort stehen, haben alle Werte zwischen Null und Eins, denn sie beschreiben die Teile des Druckbogens, wobei die ersten beiden Zahlen die (x,y)-Werte der linken unteren Ecke des Rechtecks und die nächsten beiden die (x,y)-Werte der rechten oberen Ecke definieren. Die *RelativeBox* mit den Werten 0.0 0.0 0.5 0.36279 legt damit das Rechteck unten links fest, das beim Druckbogen unten links – also am Nullpunkt beginnt und rechts in X-Richtung beim halben Bogen und in Y-Richtung bei 36,279 % des Druckbogens endet. Da der Druckbogen 430 mm hoch ist, endet folglich die erste *RelativeBox* in Y-Richtung bei 156 mm (0,36279 x 430 mm). In dieser *RelativeBox* liegt Falzbogen 1 (siehe Abbildung 8.8). Analog lassen sich die anderen Rechtecke berechnen. Innerhalb jeder *RelativeBox* liegt dann in unserem Beispiel jeweils ein Nutzen. Wie dieser Nutzen in der *RelativeBox* liegt, wird durch die Werte *MarginBottom*, *MarginLeft*, *MarginRight* und *MarginTop* in DTP-Punkten angegeben. Die Zeichnung 8.10 zeigt noch einmal diese Sachlage. Die *StrippingParams*-Ressource ist üb-

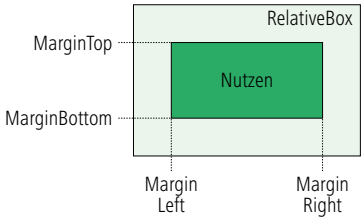



Abbildung 8.10
Position eines Nutzens in der RelativeBox

Abbildung 8.11
Die BinderySignature-Ressource enthält das Ausschießschema.

```
<BinderySignature Class="Parameter" DescriptiveName="F04-01_ui_2x1"
 FoldCatalog="F4-1" ID="_201" NumberUp="2 1" Status="Available" />
```

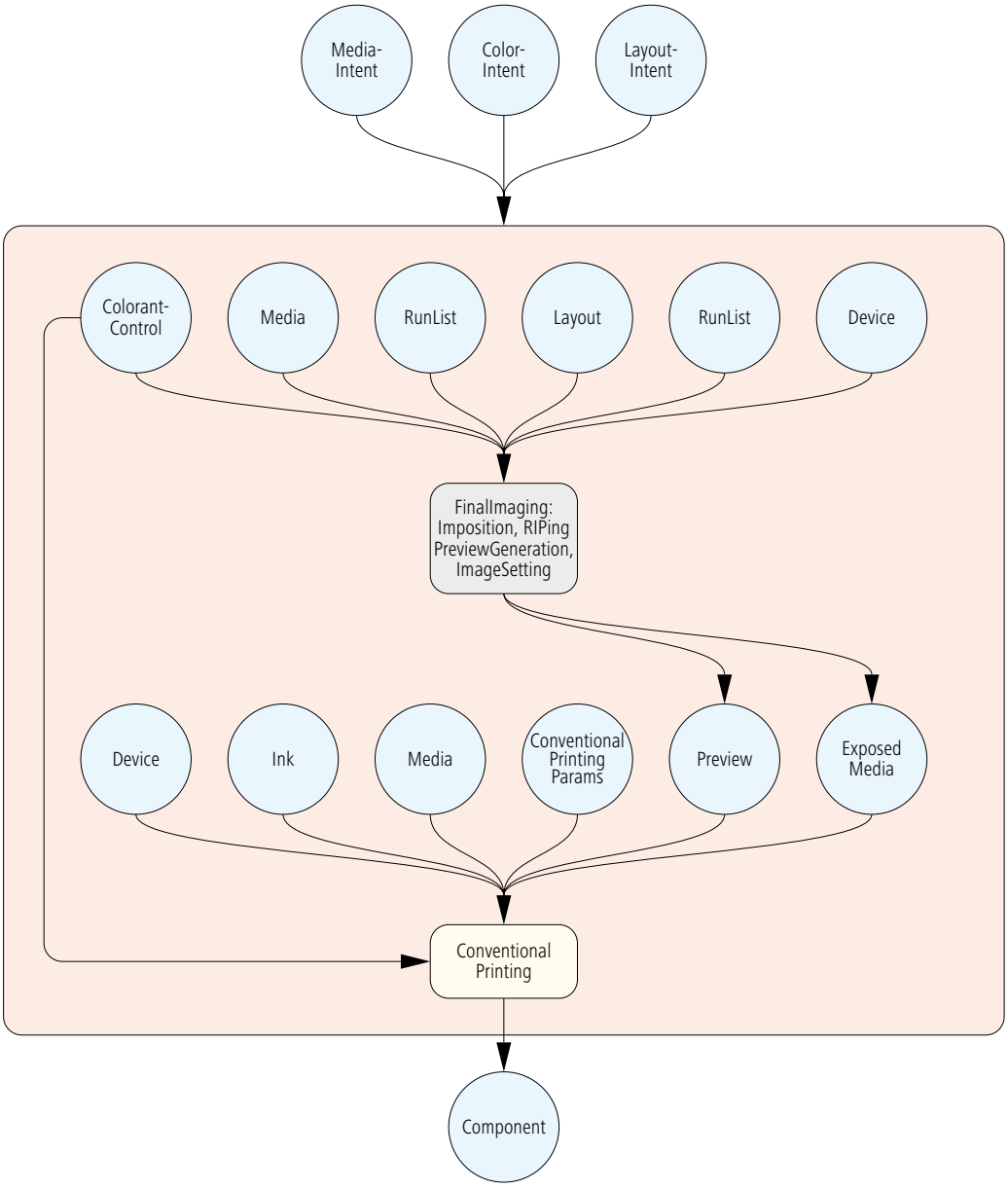



Abbildung 8.12
Produktknoten „Umschlag“ mit seiner Unterstruktur und den Ressourcen

rigens ein gutes Beispiel für eine partitionierte Ressource (zumindest wertvoller als 6.14), die nach dem *PartIDKeys*-Schlüssel *SignatureName SheetName* aufgespalten wird.

Im JDF-Code von Abbildung 8.9 sind außerdem noch ein paar weitere Dinge, die erklärt werden sollten. Im Element *StripCellParams* sind die Werte für die Anschnitte sowie das Endformat des Teilproduktes aufgeführt. Die beiden Referenzen auf

die *Media*-Ressourcen beziehen sich einerseits auf den Bedruckstoff, andererseits auf die zu verwendende Druckplatte. Schließlich sei noch die Ressource *BinderySignature* erwähnt, auf die in der vierten Zeile verwiesen wird und die explizit in Abbildung 8.11 zu sehen ist. Sie enthält offensichtlich das Ausschießschema. In diesem Fall ist einfach ein Eintrag aus dem Falzartenkatalog eingetragen. Alternativ kann aber auch in Unterelementen (*SignatureCell*) von *BinderySignature* das Ausschießschema direkt eingetragen sein. Die *BinderySignature*-Ressource entspricht also einem herkömmlichen Falzmuster, das auf mehrere Falzbogen angewendet werden kann und unabhängig von den Größen der Seiten bzw. Nutzen und der Druckbogen definiert wird.

Die *Assembly*-Ressource hingegen ist in 8.7 nur einmal in dem JDF-Dokument enthalten. In dieser mit der JDF-Version 1.2 eingeführten Ressource können im Allgemeinen Angaben stehen, ob die Bogen zusammengetragen oder gesammelt werden sollen.

In Abbildung 8.12 ist der *ProductIntent*-Knoten „Umschlag“ mit einer *GrayBox* und einem Prozess und deren Verbindungen zu den Ressourcen skizziert. Die vier Ressourcen außerhalb des roten Rechtecks geben die Input- und die Output-Ressourcen des *ProductIntent*-Knotens an. Als Input dienen die Ressourcen oben, welche das geplante Endprodukt beschreiben: In *LayoutIntent* ist die Größe des Endformats, in *ColorIntent* die Farbigkeit und in *MediaIntent* der Bedruckstoff spezifiziert. Die Output-Ressource *Component* unten repräsentiert das Teilprodukt „Umschlag“. Diese Ressource ist dann wiederum Input von dem übergeordneten *ProductIntent*-Knoten, der das Endprodukt definiert.

Die *GrayBox* und der Prozess sind verbunden mit einer Reihe von Ressourcen. Die Ressourcen müssen nicht unbedingt innerhalb des vorliegenden *ProductIntent*-Knotens für den Umschlag liegen, sondern können genauso auch eine Ebene höher im Wurzelelement liegen – also im *ProductIntent*-Knoten für das Gesamtprodukt. Die Pfeile zwischen Ressourcen und den *GrayBoxen* geben also eigentlich nur die *ResourceLinks* wieder.

Auf den ersten Blick mag es so erscheinen, dass die Zeichnung 8.12 fehlerhaft ist, da die Input-Ressource RunList für die *GrayBox* doppelt aufgelistet ist. Das ist aber in der Tat beabsichtigt, da die Inhalte der Ressourcen sich unterscheiden. Im Einzelnen hat die *GrayBox* folgende Input-Ressourcen:

- *Device*: Definition des Plattenbelichters
- *RunList*: Datei, welche die aufbereiteten *Content*-Daten enthält, also den Output von *PrePressPreparation*
- *Layout*: Signaturen des Gesamtprodukts
- *RunList*: Datei(en), die Marken für die Bogen enthalten
- *Media*: Plattendefinition
- *ColorantControl*: Farbgebung der einzelnen Signaturen

Einige der Ressourcen beschreiben nicht nur Dinge des Teilprodukts „Umschlag“, sondern vom gesamten Produkt. Diese Ressourcen (*Layout*, *RunList*, *ColorantControl*) müssen also bei dem JDF-Element des Gesamtproduktes stehen, damit auch das Teilprodukt „Innenteil“ auf diese Ressourcen Bezug nehmen kann. Von den oben genannten Ressourcen sind nur zwei wirklich vorhanden, haben also den Status gleich *Available*: *Device* und *ColorantControl*. Alle anderen sind zwar schon formal angelegt, es fehlen aber noch wichtige Einträge. So ist der

Abbildung 8.13
Elementstruktur des Teilproduktes „Umschlag“

```
<JDF DescriptiveName="Umschlag" Status="Waiting" Type="Product"...>
    <ResourceLinkPool>
     ...
    </ResourceLinkPool>
    <ResourcePool>
     ...
    </ResourcePool>

    <JDF Category="FinalImaging" DescriptiveName="Umschlag(CTP)"
     Status="Waiting" Type="ProcessGroup"
     Types="Imposition RIPing PreviewGeneration ImageSetting"...>
       <ResourceLinkPool>
        ...
       </ResourceLinkPool>
       <ResourcePool>
        ...
       </ResourcePool>
    </JDF>

    <JDF DescriptiveName="Umschlag" Status="Waiting"
     Type="ConventionalPrinting"...>
       <ResourceLinkPool>
        ...
       </ResourceLinkPool>
       <ResourcePool>
        ...
       </ResourcePool>
    </JDF>
</JDF
```

```
<ResourceLinkPool>
    <ComponentLink Usage="Output" rRef="_100" />
    <LayoutIntentLink Usage="Input" rRef="_101" />
    <ColorIntentLink Usage="Input" rRef="_102" />
    <MediaIntentLink Usage="Input" rRef="_103" />
</ResourceLinkPool>
```

Abbildung 8.14 RessourceLinkPool des JDF-Knotens „Umschlag“

Status auf *Unavailable* gesetzt. Das AMS kennt beispielsweise noch nicht den Dateinamen und den Ablageort der Content-Daten und auch die Wahl der Druckplatten wird erst in der Produktion getroffen.

Die *GrayBox „Imposition RIPing PreviewGeneration ImageSetting“* ist im Einzelnen nicht weiter ausgeführt. Nur die Output-Ressourcen dieser *GrayBox* sind festgelegt, nämlich *ExposedMedia* und *Preview*, die wiederum als Input für den folgenden Prozess benötigt werden. Die Stati dieser beiden Ressourcen müssen natürlich auch *Unavailable* sein, da die *GrayBox* noch nicht ausgeführt wurde – oder genauer: da die *GrayBox* noch nicht in Prozesse umgewandelt wurde und diese ihrerseits somit auch nicht ausgeführt werden konnten. Wie wir in Kapitel 6 bemerkt haben, sind *GrayBoxen* ja grundsätzlich nicht ausführbar.

Der folgende Prozess *ConventionalPrinting* hat als Input-Ressourcen:

- *ExposedMedia* bebilderte Platten für diesen Job,
- *Preview* Voransichtsbild jeder Separation (zur Berechnung der Farbzonenvoreinstellung),
- *ConventionalPrintingParams*

Abbildung 8.15 für das Teilprodukt benötigte Farbauszüge

```
<Ink Class="Consumable" ID="_104" PartIDKeys="SignatureName SheetName Side
 Separation" PartUsage="Implicit" Status="Available">
   <Ink SignatureName="SIG001">
      <Ink SheetName="Umschlag">
         <Ink Side="Front">
            <Ink Separation="Cyan" />
            <Ink Separation="Magenta" />
            <Ink Separation="Yellow" />
            <Ink Separation="Black" />
         </Ink>
         <Ink Side="Back">
            <Ink Separation="Black" />
         </Ink>
      </Ink>
   </Ink>

    Analog für die weitere 4 Signaturen des Innenteils

</Ink>
```

| Name | Data Type | Description |
|---|---|---|
| *BillingCode* ? | string | A code to bill charges incurred while executing the Node. |
| *CustomerID* ? | string | Customer identification used by the application that created the Job. This is usually the internal customer number of the MIS system that created the Job. |
| *CustomerJobName* ? | string | The name that the customer uses to refer to the Job. |
| *CustomerOrderID* ? | string | The internal order number in the system of the customer. This number is usually provided when the order is placed and then referenced on the order confirmation or the bill. |
| *CustomerProjectID* ? New in JDF 1.2 | string | The internal project id in the system of the customer. This number might be provided when the order is placed and then referenced on the order confirmation or the bill. |
| *rRefs* ? Deprecated in JDF 1.2 | IDREFS | Array of *ID*s of any Elements that are specified as ResourceRef Elements. In version 1.1 it was the IDREF of a ContactRef. In JDF 1.2 and beyond, it is up to the implementation to maintain references. |
| **Company** ? Deprecated in JDF 1.1 | refelement | Resource Element describing the business or organization of the contact. In JDF 1.1 and beyond, Company affiliation of Contacts is specified in Contact. |
| **Contact** * New in JDF 1.1 | refelement | Resource Element describing contacts associated with the customer. There SHOULD be one Contact [contains (@*ContactTypes*, *"Customer"*)]. Such a **Contact** specifies the primary customer's name, address etc. |
| CustomerMessage * New in JDF 1.2 | | |

Abbildung 8.16 Festlegungen der CustomerInfo-Ressource in der JDF-Spezifikation

Angaben über Druckmaschinentyp (Rolle, Bogen) und zur Wendung,

- *Media* Angaben zum Bedruckstoff,
- *Ink* Angaben zur Druckfarbe,
- *Device* Angaben zur Druckmaschine und
- *ColorantControl* Farbgebung der einzelnen Signaturen.

Auch in diesem Fall haben alle Ressourcen bis auf *Ink*, *Device* und *ColorantControl* den *Status* gleich *Unavailable*.

Die Elementstruktur des JDF-Knotens vom Teilprodukt „Umschlag" ist in Abbildung 8.13 wiedergegeben. Es fehlen nur die Ressourcen und die Links auf die Ressourcen. Schaut man jedoch auf die Skizze 8.12, so ist es klar, dass das *ResourceLinkPool* des Produktknotens „Umschlag" prinzipiell wie in Abbildung 8.14 aussehen muss. Die anderen *ResourceLinkPools* sind analog aufgebaut. Auch die Ressourcen selber sind zum Teil nur wenig spektakulär. So zeigt die *Ink*-Ressource in 8.15 nur, wie für Schön und Wider einer jeden Signatur die benötigten Druckfarben definiert werden. Es handelt sich hierbei um eine partitionierte Ressource, deren Aufschlüsselung durch den Wert des Attributes *PartIDKeys* definiert ist.

Selbstverständlich können hier nicht alle Ressourcen des Knotens „Umschlag" vorgestellt werden und es soll auch nicht wei-

ter auf den JDF-Knoten „Innenteil“ eingegangen werden, der analog zum „Umschlag“ aufgebaut ist. Wir müssen auch zugeben, dass weitere Ressourcen des JDF-Wurzelelementes unter den Tisch fallen gelassen wurden, wie zum Beispiel die *CustomerInfo*-Ressource, welche die Kundendaten enthält. Diese sieht ganz ähnlich aus wie bereits in Abbildung 5.2 und 5.3 gezeigt. Wir werden jetzt nicht den Code noch einmal darstellen und lieber unser Beispiel damit beenden.

Wir möchten stattdessen einen anderen Weg einschlagen, nämlich die abstrakte Definition der *CustomerInfo*-Ressource erklären, wie sie in der JDF-Spezifikation 1.4 wiedergegeben ist. Dieses soll für den Leser den Einstieg in die Spezifikation erleichtern. Die Tabellen sind dort grundsätzlich wie die Tabelle in Abbildung 8.16 aufgebaut: In der linken Spalte stehen die Namen der Strukturelemente (wie ein Attribut, ein Unterelement oder eine Referenz auf eine Ressource), die mittlere Spalte gibt die Datentypen wieder und in der rechten Spalte werden die Strukturelemente erklärt. Auffällig sind in der ersten Spalte die blauen Kommentare wie *New in JDF 1.2* oder *Deprecated in JDF 1.1*. Letzteres bedeutet, dass das entsprechende Strukturelement in Version 1.1 wieder aufgegeben wurde, also kein JDF-Code mit Version größer gleich 1.1 es enthalten darf. Aus Kompatibilitätsgründen muss es natürlich noch in der Spezifikationstabelle stehen – wir wollen jedoch auf solche „veralteten“ Begriffe nicht mehr eingehen. Des Weiteren erkennt man an den Namen Sonderzeichen wie ? oder *. Manchmal (an anderer Stelle der Spezifikation) steht auch ein + oder überhaupt kein Sonderzeichen (siehe zum Beispiel Abbildung 12.8). Die Bedeutung bezieht sich auf die Anzahl der möglichen Instanzen:

? optional
* keinmal oder mehrmals
\+ einmal oder mehrmal

Steht kein Sonderzeichen am Namen, so ist das Strukturelement obligatorisch und muss genau einmal vorkommen. Das Attribut *CustomerID* kann also einmal vorkommen, muss aber nicht, während das Unterelement *Contact* keinmal, einmal oder auch mehrfach vorkommen kann.

Der Datentyp *String* ist einfach eine Zeichenkette, ein *Refelement* ein Element oder aber eine Referenz auf ein Element, während beim *Element* (das kommt in Abbildung 8.16 allerdings nicht vor) das Element immer direkt stehen muss.

Hier die kurze Erklärung der Strukturelemente, die auch in Ver-

sion 1.4 noch aktuell sind:

- *BillingCode*: eine Rechnungsnummer für den Job
- *CustomerID*: Kundennummer des MIS
- *CustomerJobName*: Kundenname
- *CustomerOrderID*: Auftragsnummer des Kunden
- *CustomerProjectID*: Projektnummer des Kunden
- *Contact*: (Referenz auf ein) Element, das einen Kontakt beschreibt
- *CustomerMessage*: Beschreibung einer Nachricht an den Kunden, beispielsweise ein automatischer E-Mail-Versand, wenn der Knoten abgearbeitet wurde

Für die *Contact*- und die *CustomerMessage*-Elemente gibt es wieder eigene Tabellen in der Spezifikation. Da beide Elemente ihrerseits weitere Unterelemente haben (*Address*, *ComChannel*, *Company* und *Person*), findet man diese wiederum in anderen Tabellen spezifiziert. Diese Hyperlinks wurden jedoch nicht eingeführt, um den Leser zu verwirren (obwohl das schon passieren kann), sondern weil Elemente von unterschiedlichen Elter-Elementen verwendet werden und mit dieser Technik aber nur einmal in der Spezifikation stehen müssen. Beispielsweise ist das Element *ComChannel* ein Unterelement sowohl von dem *Contact*- als auch von dem *CustomerMessage*-Element. Tatsächlich muss man als Leser nicht nur die Links nach unten verfolgen (also zu den möglichen Kindelementen), sondern auch gewissermaßen nach oben: Es gibt so genannte „abstrakte Ressourcen“, die Attribute beschreiben, die alle Ressourcen haben können. Diese allgemeinen Attribute (wie beispielsweise *ID*, *Class* oder *Author*) werden dann nicht mehr in den Tabellen der konkreten Ressourcen aufgeführt. Ähnlich wird auch bei JDF-Knoten vorgegangen.

Bisher haben wir fast nur den Datenfluss vom MIS zur Druckvorstufe behandelt und nicht die umgekehrte Richtung, die jedoch für die Nachkalkulation entscheidend ist. Für diesen Weg gibt es zwei Möglichkeiten, die häufig auch parallel genutzt werden:

- das MIS empfängt JMF-Nachrichten vom Produktionssystem,
- das vom Produktionssystem veränderte JDF wird an das MIS zurück transferiert.

Beim Rückempfang des JDFs vom Produktionssystem werden dann vor allem die *AuditPools* ausgelesen, wie bereits in Kapi-

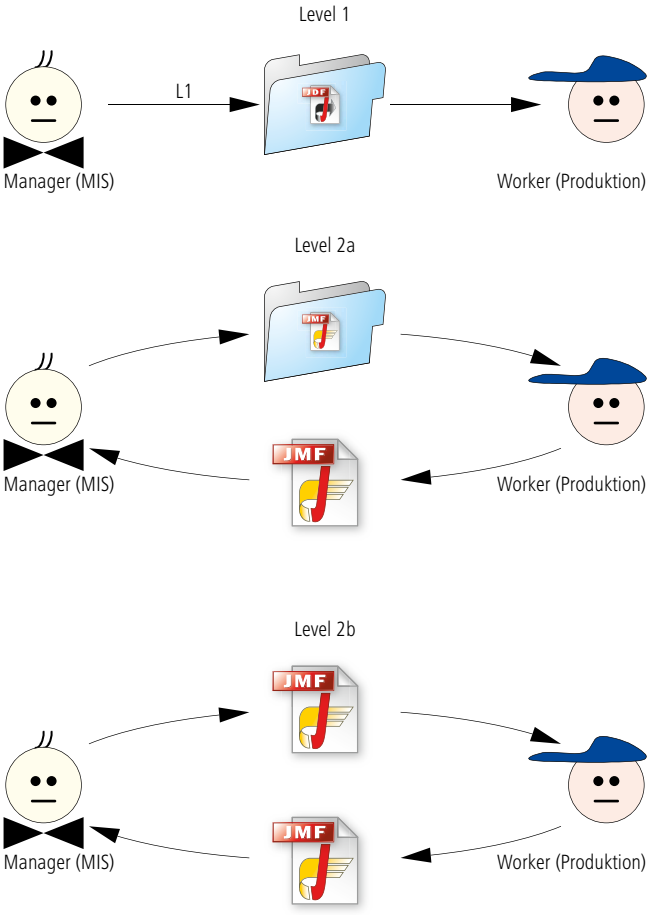



Abbildung 8.17
Level 1 und 2 der MIS ICS

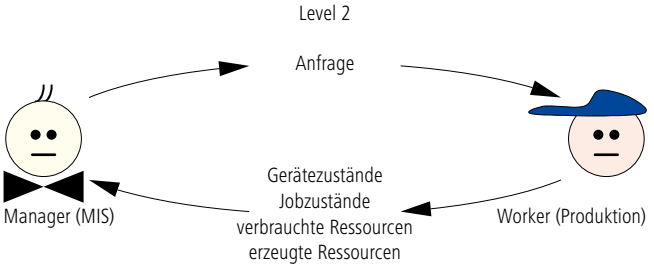



Abbildung 8.18
Der Worker muss auf Anfrage den Manager über Jobs und Ressourcen informieren.

tel 6 ausgeführt.

### 8.3 MIS-ICS-Papiere

Die Bedeutung der Schnittstelle zwischen AMS/MIS und Produktion läßt sich in der Anzahl der entsprechenden ICS-Papiere (Abschnitt 6.7) ablesen. Zurzeit sind die folgenden Unterlagen in der Version 1.3 aktuell; allerdings sind gerade diese relativ kurzlebig und dem Leser wird empfohlen, sich die neueste Version im Internet zu besorgen [12].

- *MIS ICS*
- *MIS to Prepress ICS*
- *MIS to Conventional Printing – Sheet-Fed ICS*
- *Newspaper: MIS to WebPress ICS*
- *MIS to Finishing ICS*

Wir wollen hier aber nur kurz auf das erste dieser ICS-Papiere eingehen, die anderen werden zum Teil in den entsprechenden Kapiteln 9 bis 11 vorgestellt werden. Das *MIS ICS* Papier spezifiziert allgemeine Anforderungen an das MIS, die nicht spezifisch für Vorstufe, Druck oder Weiterverarbeitung sind.

Auch im *MIS ICS* werden – ähnlich dem *JMS ICS* – verschiedene Übereinstimmungsstufen definiert. Hier sind es sogar drei Level, die vor allem abermals JMF-Nachrichten betreffen; wir wollen aber nur die ersten zwei vorstellen (Abbildung 8.17):

- Level 1: Das MIS reicht nur JDF über Hotfolder an den *Worker* weiter (Abbildung 8.17, Level 1),
- Level 2: Das MIS kann zusätzlich JMF-

Anfragen mit Hilfe des *NodeInfo*-Elementes generieren, um einen festen Kommunikationskanal vom *Worker* zum *MIS* aufzubauen (Abbildung 8.17, Level 2a). Hierüber können dann Signale vom *Worker* zum *MIS* versendet werden. Dieses Verfahren entspricht Level 1 im JMF ICS. Der Worker muss in diesem Fall kein HTTP-Server sein, das heißt, er muss keine HTTP-Pakete annehmen und verarbeiten können. Alternativ (Abbildung 8.17, Level 2b) kann in Level 2 aber auch das MIS direkt JMF-Nachrichten verschicken. Bei dieser Option muss der Worker dann sehr wohl auch ein HTTP-Server sein, und man setzt Level 2 vom *JMF ICS* voraus.

Auch andere Forderungen bezüglich JMF werden erhoben. Zum Beispiel muss, wenn sich das MIS mit einer Ressourcen-Anfrage an einen *Worker* richtet, dieser Informationen über Ressourcen, die bisher verbraucht oder erzeugt wurden, zurückschicken (Abbildung 8.18). Eine MIS-Software, die Level 2 des *MIS ICS* erfüllen möchte, muss also entsprechende Anfragen absetzen können (egal, ob über NodeInfo oder über eine direkte JMF-Nachricht). Ein *MIS ICS-Level-2-Worker* muss diese Anfrage empfangen, interpretieren und beantworten können. Ähnliches gilt für Anfragen bezüglich Gerätezuständen und Jobstati. Mit anderen Worten: Jobverfolgung und Gerätezustandsüberwachung setzen MIS ICS Level 2 sowohl für die MIS-Software als auch für die Controller und Geräte in der Produktion voraus. Es gibt in der Praxis durchaus Implementierungen von JDF-Workflows, die diese Funktionen nicht unterstützen.

Doch nicht nur Anforderungen an JMF-Elemente werden in dem *MIS ICS* definiert, sondern auch viele bezüglich JDF. Viele kleine Details müssen festgelegt werden. So wird beispielsweise vorgeschrieben, dass in jedem von einem MIS erzeugten JDF-Wurzelelement in einem speziellen Attribut (*ICSVersions*) der ICS-Level eingetragen wird, nach dem das JDF-Dokument aufgebaut wurde. Weiterhin muss jedes JDF-Wurzelelement auch zwei spezielle Ressourcen als Input haben, nämlich die *CustomerInfo*-Ressource, in der Kundendaten hinterlegt sind, und die *NodeInfo*-Ressource, welche die Terminplanung und optional eine oder mehrere JMF-Nachricht(en) enthält. Jeder von einem MIS erzeugte JDF-Wurzelknoten ist ein Produktknoten, und die Eigenschaften des Produktes sind in *Intent*-Ressourcen hinterlegt. Damit muss jedes JDF-Wurzelelement auch mindestens eine *Intent*-Ressource als Input haben. Gäbe es keine, wäre das Produkt ohne Eigenschaften und damit ziemlich sinnlos.

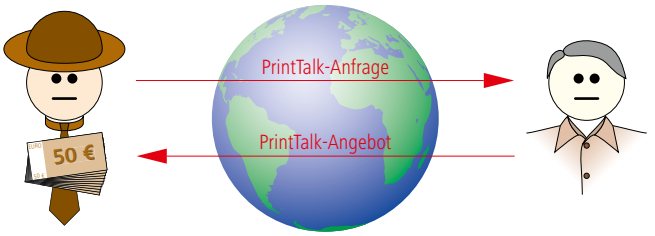



Abbildung 8.19
PrintTalk-Kommunikation zwischen Kunden und Druckerei

Dass jedes JDF-Wurzelement mit einer Output-Ressource *Component*, die das Endprodukt repräsentiert, verknüpft sein muss, haben wir schon früher erwähnt.

Jeder *Worker* muss in den JDF-Prozessknoten nach dem Beenden des Prozesses Informationen über Start, Ende, Dauer und dergleichen in einem *Audit*-Element festhalten. In *MIS ICS* Level 2 muss er auch noch weitere *ResourceAudit*-Elemente in das *AuditPool* eintragen. Diese enthalten Angaben über die vom *Worker* verbrauchten Materialien bei der Ausführung des Prozesses (Platten, Papier, Farbe...).

## 8.4 PrintTalk-/JDF-Schnittstelle zu Kunden

Die Spezifikation von PrintTalk wurde von der gleichnamigen Organisation unter dem Dachmantel der NPES (Association for Suppliers of Printing, Publishing and Converting Technologies) bereits 2000 der Öffentlichkeit vorgestellt. 2004 hat sich PrintTalk in CIP4 integriert und wurde damit als eigenständige Organisation aufgelöst. Die PrintTalk-Spezifikation beruht auf cXML, wie bereits in Abschnitt 5.4 diskutiert.

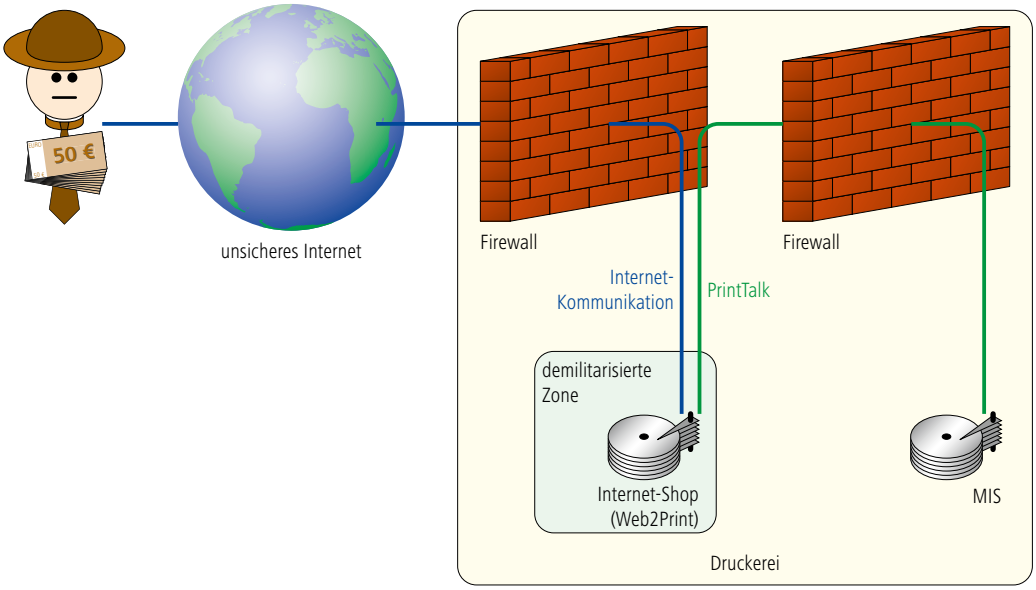



Abbildung 8.20
Web-Server und MIS einer Druckerei. Das PrintTalk-Protokoll kommt nur innerhalb der Druckerei zum Einsatz.

Einige Auftrags-Managementsysteme (AMS) können JDF-Dokumente, die vom Kunden geschickt werden, auch einlesen und daraus ein neues Angebot oder einen neuen Auftrag generieren. So könnte ein Kunde beispielsweise im Programm Acrobat eine JDF-Datei erzeugen, die das Produkt beschreibt, also im Wesentlichen aus Produktknoten besteht. Die Druckerei würde diese dann in das AMS importieren. Damit würde das Auftrags-Managementsystem die nötige Produktbeschreibung von außen erhalten, die geschäftlichen Abläufe wären aber weiterhin nicht abgedeckt. Diese Arbeitsweise hat allerdings einen Haken: Die JobID wird vom Kunden angelegt und passt im Zweifel nicht in den MIS-Nummernkreis der Druckerei. Umgekehrt macht es eher Sinn, wenn die Druckerei im MIS einen Job formal anlegt, die JDF-Datei ohne viel Inhalt dann per E-Mail an den Kunden schickt, der dann die Datei in Acrobat einlesen und mit Auftragdaten vervollständigen kann. Schließlich kann der Kunde dann die so angereicherte JDF-Datei zurück an die Druckerei schicken, die ihrerseits dann die Information dem bereits vorhandenen Auftrag automatisch zuordnen kann.

Nicht nur Endkunden selbst, sondern auch beauftragte Produktionsagenturen /-dienstleister können Auftragsparameter via JDF an die Druckerei kommunizieren. Dabei kann die Agentur bestimmte Parameter (zum Beispiel den Liefertemin und/oder die Auflagenhöhe) automatisch aus dem System des Kunden beziehen, entsprechend anreichern und dann an die Druckerei der Wahl weiterleiten. Bei diesem Szenario sind elektronische Rückmeldungen sehr von Vorteil.

```
<PrintTalk...>
    <Header>
        <From>
          ...
        </From>

        <To>
          ...
        </To>

        <Sender>
          ...
        </Sender>
    </Header>

    <Request>
        <PurchaseOrder...>
            <jdf:JDF ...>
                <jdf...>
                  ...
                </jdf>
                <jdf...>
                  ...
                </jdf>
            </jdf:JDF>
        </PurchaseOrder>
    </Request>
</PrintTalk>
```

Abbildung 8.21
Struktur eines PrintTalk-Dokumentes

Mit JDF kann also eine Kommunikation zwischen Kunden und Druckerei realisiert werden. PrintTalk geht allerdings noch einen Schritt weiter.

Das Prinzip ist mit Abbildung 8.19 schnell erklärt. Geschäftsobjekte, wie Angebotsanfragen oder Angebote,

werden über das Internet zwischen Kunden und Druckdienstleister ausgetauscht. Aber auch Auftragserteilungen, Auftragsbestätigungen, Lieferungsbelege und Rechnungen können auf diese Art kommuniziert werden. Durch die Integration einer PrintTalk-Schnittstelle an ein MIS-System können folglich viele administrative Schritte vereinfacht werden. Im Prinzip geht es also um Portallösungen oder auch um das Schlagwort Web-to-Print.

Grundsätzlich gibt es zwei Konfigurationen: Entweder kommuniziert ein Kunde direkt mit einem Webserver einer Druckerei, die einen solchen Dienst anbietet, oder aber es gibt einen Makler zwischen beiden Parteien. Im zweiten Fall würde sich der Kunde auf dem Webserver des Maklers einloggen, der über PrintTalk Preisanfragen an unterschiedliche Druckereien sendet. Diese erstellen daraufhin Angebote (entweder völlig automatisiert in „Echtzeit“ oder auch zeitversetzt unter der Kontrolle eines Mitarbeiters) und senden diese als PrintTalk-Objekte zurück an den Makler. Der Makler würde seinerseits die Angebote möglicherweise filtern und neu aufbereiten bevor er sie dem Kunden online zur Verfügung stellt. Dieses Verfahren ist in anderen Dienstleistungssparten bereits weit verbreitet, wie zum Beispiel in der Reisebranche. Das automatisierte Abfragen nach Preisen verschärft natürlich den Preiskampf noch weiter.

Verläuft die Kommunikation direkt zwischen Kunden und Druckerei, kann die technische Realisierung abermals unterschiedlich ausfallen. Entweder erzeugt bereits der Kunde mit seinem Browser PrintTalk-Dokumente und sendet sie an den entsprechenden Server der Druckerei, oder aber es gibt eine Standard-Internetkommunikation zwischen Kunde und Druckerei und erst der Web-to-Print-Server in der Druckerei generiert die PrintTalk-Mitteilungen. Im letzteren Fall würde PrintTalk nur ein Druckerei-

Abbildung 8.22
Auftragserteilung über PrintTalk

```
<PurchaseOrder AgentID="CC" AgentDisplayName="CarlCool"
 RequestDate="2008-11-13T11:00Z" BusinessID="A001" Currency="EUR"
 Expires="2008-12-13T11:00Z ">
   <Pricing>
      <Price LineID="_1" DescriptiveName="Hard Cover Books” Amount="6800"
       Price="15000.00" />
      <Price LineID="_2" DescriptiveName="Shipping” Price="980.00"/>
   </Pricing>
   <jdf:JDF...>
      <JDF...>
       ...
      </JDF>
   </jdf:JDF>
</PurchaseOrder>
```

internes Schnittstellenprotokoll zwischen der Web-to-Print-Software und dem MIS darstellen. Diese Situation ist in Abbildung 8.20 wiedergegeben. Natürlich muss ein MIS-Server innerhalb eines geschützten Bereichs im lokalen Netzwerk der Druckerei liegen, also von außen hinter einer Firewall. Der Server, auf dem der Internet-Shop installiert ist, befindet sich üblicherweise in einer demilitarisierten Zone. Der (potentielle) Kunde einer Druckerei kommuniziert also über das Internet mit einem Online-Bestellsystem, das seinerseits PrintTalk-Daten generiert und diese an das MIS weitergibt. Damit können also vordefinierte Produkte von außen mit Hilfe von HTML bestellt werden, und dennoch erhält das MIS in PrintTalk definierte *BusinessObject*-Elemente sowie JDF-Produktbeschreibungen. Diese Konfiguration ermöglicht eine unabhängige Kommunikationsschnittstelle zwischen Web-Portal und MIS, solange nur beide PrintTalk und JDF unterstützen.

Mit PrintTalk kann also eine Business-to-Bussiness-(B2B)-Kommunikation zwischen Makler und Druckerei sowie auch eine Business-to-Consumer-(B2C)-Schnittstelle zwischen dem Kunden und der Druckerei aufgebaut werden. Tatsächlich gibt es bisher nur recht wenige PrintTalk-Anwendungen und man wird erst in der Zukunft sehen, welche Konfigurationen öfter implementiert werden. Möglicherweise ist der Grund der zögerlichen Implementierung auch der, dass MIS-Hersteller häufig ihre eigenen Web-Portal-Lösungen anbieten und deswegen weniger an einer offenen Schnittstelle zum MIS interessiert sind.

In Abbildung 8.21 ist die Struktur eines PrintTalk-Dokumentes abgebildet, die sehr ähnlich zu der allgemeinen cXML-Struktur ist, die wir in Abbildung 5.7 vorgestellt haben. Der Header in dem PrintTalk-Dokument ist völlig identisch aufgebaut wie bei cXML. Auch das Request-Element gibt es in PrintTalk. Dessen Unterelemente sehen aber jetzt etwas anders aus: Es enthält genau ein sogenanntes *BusinessObject*. In dem vorliegenden Beispiel ist es eine Auftragserteilung (*PurchaseOrder*), insgesamt gibt es aber 12 unterschiedliche *BusinessObjects*:

- *RFQ* (Request For Quote): Angebotsanforderung
- *Quotation*: Angebot
- *PurchaseOrder*: Auftragserteilung
- *Confirmation*: Auftragsbestätigung
- *Cancellation*: Stornierung eines BusinessObjects
- *Refusal*: Ablehnung eines BusinessObjects
- *OrderStatusRequest*: Anfrage zum Bestellstatus
- *OrderStatusResponse*: Antwort zum Bestellstatus

- *ProofApprovalRequest*: Anforderung zur Proof-Freigabe
- *ProofApprovalResponse*: Proof-Freigabe bzw. Proof-Ablehnung
- *Invoice*: Rechnung
- *ReturnJob*: Druckerei schickt Job zurück zum Kunden

Jedes *BusinessObject* hat natürlich auch Attribute, welche die geschäftliche Transaktion genauer festlegen. In der Auftragserteilung werden beispielsweise die Währung definiert und auch optional als Unterobjekt die Bezahlungsform und die Höhe des Preises. Im Beispiel 8.22 haben wir sogar zwei Preise, einen für die Herstellung des Druckproduktes und den anderen für die Lieferung. In früheren Versionen von JDF durften auch Preisangaben und Bezahlungsmodalitäten in der *DeliveryIntent*-Ressource stehen. Mittlerweile wurden aber all diese Angaben aus dem JDF herausgenommen und in entsprechende PrintTalk-Elemente verlagert.

Einige dieser *BusinessObjects* enthalten nun ihrerseits einen oder mehrere JDF-Knoten. Das *PurchaseOrder*-Element aus Abbildung 8.22 enthält eigentlich drei JDF-Knoten, wobei nur der oberste wiedergegeben ist. In dem JDF-Element ist dann die Beschreibung des Druckproduktes abgelegt. Teilprodukte des Auftrags werden in weiteren JDF-Knoten spezifiziert. Beispielsweise würden dann wie üblich in dem *ResourcePool* des JDF-Wurzelknotens die Informationen über den Kunden (*CustomerInfo*), über die beabsichtigte Auslieferung des Endproduktes (*DeliveryIntent*) als Input stehen. Wir wollen an dieser Stelle jedoch das Beispiel nicht weiter ausführen, da es genau nach den JDF-Regeln aufgebaut ist, die wir schon in den letzten Abschnitten und in Kapitel 6 beschrieben haben.

---

**Übungen**

- Erstellen Sie mit Acrobat (ab Version 7.0) eine JDF-Datei. Geben Sie hierzu Kunden, Material und das Produkt selber an (z.B. eine Broschur mit 60 CMYK-A4-Seiten Innenteil und Umschlag in CMYK+Sonderfarbe).
- Laden Sie von CIP4.org den JDF-Editor herunter, öffnen Sie die zuvor hergestellte JDF-Datei und analysieren Sie die Struktur.
- Benennen Sie Ihre *.jdf-Datei in *.xml um und öffnen Sie sie mit einem Browser. Untersuchen Sie die JDF-Datei und identifizieren Sie dort Ihre in Acrobat angegebenen Werte.

# 9 Vorstufe

Aus Sicht der Druckvorstufe sieht die JDF/JMF-Welt wie in Abbildung 9.1 aus, wobei jeder Pfeil eine potentielle JDF/JMF-Verbindung darstellt. Typischerweise erhält ein Workflow-Managementsystem der Druckvorstufe seine JDF-Eingangsdaten von einem Auftrags-Managementsystem (Pfeil 1). Je nach Konfiguration kann die Druckvorstufe auch Meldungen an das AMS zurückgeben. Wir haben bereits gesehen, dass die JDF-Daten, die das MIS zur Verfügung stellt, nicht wirklich für die Produktion ausreichend sind (Stichwort: *GrayBox*), das heißt die Druckvorstufe wird in unterschiedlichen Modulen neue JDF-Strukturen und optional auch JMF-Meldungen erzeugen (Pfeile 2). Einen Teil der vom AMS übernommenen und der neu generierten Daten wird die Druckvorstufe an die Druckmaschine(n) (Pfeil 3) und an die Weiterverarbeitung (Pfeil 4) weiterleiten, sei es direkt oder über das MIS. In diesem Kapitel werden nur die Schnittstellen 1 und 2 beleuchtet, während die Schnittstellen 3 und 4 in den Kapiteln 10 und 11 über Druck respektive Weiterverarbeitung analysiert werden. Die Schnittstelle 1 ist geprägt von dem entsprechenden ICS-Papier, dem MIS to Prepress ICS, das in Abschnitt 9.1 diskutieret wird. Die prepress-internen Schnittstellen 2 werden wir anhand von Beispielen in den Abschnitten 9.2 bis 9.5 erklären.

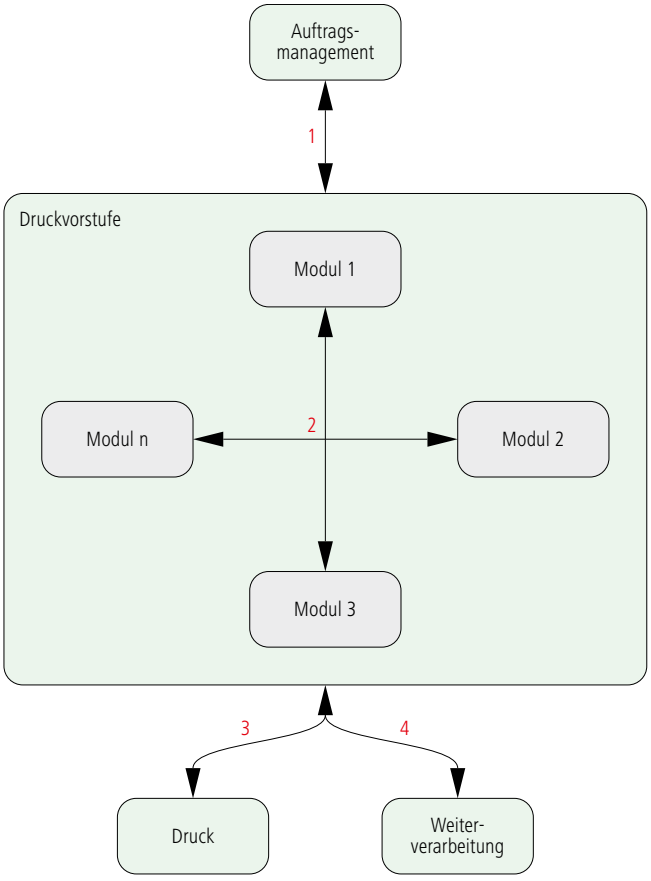



Abbildung 9.1
JDF-/JMF-Verknüpfungen im Bereich der Druckvorstufe

Doch zuvor möchten wir noch einmal an die einzelnen Schritte in der Druckvorstufe erinnern, die wir bereits in Abschnitt 3.2 aufgelistet haben. Dabei hatten wir vorausgesetzt, dass die Druckvorstufe bereits fertige PDF-Daten angeliefert bekommt. Das eigentliche Layout und die PDF-Erstellung sind deswegen nicht mit auf der Liste. Diese Aufgaben sind zwar typisch für den

Akzidenzdruck, es gibt aber natürlich auch Abweichungen. So könnte ein Formproof als Softproof bzw. als Remote-Proof realisiert sein oder sogar ganz entfallen; farbverbindliche Seitenproofs können zusätzlich erstellt werden und so weiter. Workflows für die Verpackungsdruckvorstufe sehen häufig gänzlich anders aus.

Um die Vielzahl von Aufgaben in der Vorstufe zu bewältigen, besteht ein Vorstufen-WMS in aller Regel aus unterschiedlichen Modulen. In Abschnitt 4.3 haben wir gesehen, dass das PJTF die Kommunikation zwischen der Koordinator-Software und den Jobticket-Prozessoren (eben den Modulen) bei der Extreme-Workflow-Architektur realisiert. Diese Funktion übernimmt nun das JDF/JMF (Pfeile 2 in Abbildung 9.1). Ähnlich wie im Fall von PJTF gilt aber auch bei einem JDF-Workflow, dass ein standardisiertes Datenaustauschformat allein nicht ausreicht, um den Anwender eines WMS das Zusammenschließen einzelner Module unterschiedlicher Hersteller zu ermöglichen. Eine Vielzahl von spezifischen Verabredungen ist nötig.

**Beispiele für JDF-kompatible WMS (2009)**

| Hersteller: | Hauptprodukt: |
|---|---|
| AC&C HSH Group | PuzzleFlow |
| Avato System GmbH | PMS Production Management System |
| Agfa | Apogee |
| DALiM | PRiNTEMPO |
| EFI | OneFlow |
| EskoArtwork | BackStage Server |
| Fujifilm | XMF |
| Krause-Biagosch GmbH | KIM |
| LithoTechnics Oty Limited | Metrix |
| Heidelberger Druckmaschinen AG | Prinect |
| manroland | Printnet |
| Kodak | Prinergy |
| OneVision Software AG | Asura Pro |
| RamPage | RamPage Workflow System |
| Screen | TrueFlow |
| Xerox | FreeFlow Print Manager |

## 9.1 Schnittstelle zwischen MIS und Prepress

Bevor wir uns in diesem Abschnitt mit dem *MIS-to-Prepress-ICS* beschäftigen, möchten wir noch ein paar allgemeinere Überlegungen zu dieser Schnittstelle anstellen.

Idealerweise würde man ja neben der Jobdefinition und der Festlegung der Produktionsmittel im AMS bereits auch den Produktionsweg in der Vorstufe festlegen. Und in der Tat, manche Vorstufen-Workflow-Systeme schlagen auf Grund der JDF-Information, die sie vom AMS bekommen, gewisse Arbeitsgänge vor. Da aber das Auftragsmanagement viele produktionsrelevante Details nicht kennt, muss das WMS selber mit vielen Default-Werten die Lücken füllen. Die Produktionswege sind damit aber nicht vollkommen festgeschrieben, sondern können selbstverständlich noch verändert werden.

Insgesamt kann also die Folge von Arbeitsschritten, welche die Vorstufen-Software für einen Druckauftrag ausführt, in mehreren Stufen bestimmt werden:

1. Ein(e) Systemadministrator(in) kann durch Scripting neue Workflows definieren.
2. Der (die) WMS-Beauftragte einer Druckerei kann Defaults für unterschiedliche Arbeitsgänge und Produktionsparameter für Klassen von Druckprodukten festlegen. Diese Einstellungen beruhen aber auf dem grundsätzlichen Workflow, den der Hersteller der WMS-Software vorgibt.
3. Der (die) Anwender(in) kann individuelle Veränderungen des vom WMS vorgeschlagenen Workflows für individuelle Druckaufträge vornehmen.

Für Fall 1 gibt es unterschiedliche Scripting-Möglichkeiten, die entweder von Betriebssystem- oder von WMS-Herstellern gestellt werden. Zur ersten Kategorie gehört *AppleScipt* [8] oder *Windows Script Host* (WSH) [34] mit den Script Engines *VB-Script* oder *JScript*, zur letzteren das *Rules-Based-Automation* (RBA) [30] in *Prinergy* (Kodak). Diese Technologien sind alle sehr mächtig und es würde zu weit führen, sie hier zu behandeln. Bei Interesse für das RBA sei auf den Kasten auf Seite 125 verwiesen.

Im zweiten Fall definiert der/die WMS-Beauftragte Defaults wie zum Beispiel die Default-Rastereinstellungen im RIP oder die Trapping-Werte. Auch kann er/sie Hotfolder für den Input von Content-Daten definieren, eine automatische Seitenzuordnung

auf Standbogen über Regeln für Dateinamen festlegen oder neue Maschinen in den Workflow einbinden.

Im dritten Fall wird in der Regel eine grafische Darstellung des vom WMS angebotenen Workflows auf dem Bildschirm des Anwenders erscheinen. Der Anwender beziehungsweise die Anwenderin hat die Möglichkeit, einzelne Arbeitsschritte für einen Druckjob zu verändern, also beispielsweise das Trapping abzuschalten oder die Default-Rastereintellungen im RIP zu modifizieren.

Die Schnittstelle zwischen MIS und Prepress ist prinzipiell recht kritisch. Denn so wird zwar der Theorie nach ein Auftrag im MIS angelegt und die technischen Angaben an das Prepress-WMS weitergeleitet, welche dann anschließend den Auftrag ausführt; in der Praxis aber finden noch Auftragsänderungen statt, während bereits ein Job in der Vorstufenproduktion bearbeitet wird. Das können für die Vorstufe unproblematische Werte sein, wie

**Rules-Based Automation**

Mit RBA kann man außerhalb der üblichen Prinergy-Bedienoberfläche den Workflow von Prinergy beeinflussen und weiter automatisieren. Dazu kann man mit einem grafischen Werkzeug Regeln erstellen. Bei einer Regel wird zu einem Ereignis, das in Prinergy auftreten kann, eine Aktion assoziiert.

Zum Beispiel könnte man festlegen, dass, sollte ein schwerer Fehler beim Überprüfen der Daten auftreten (im Prinergy-Jargon: beim Refining), eine bestimme Person im Betrieb eine E-Mail-Nachricht hierüber bekommt. So kann man den Import der Content-Daten in das Prinergy-System über einen Hot-Folder organisieren, die Datenprüfung im Hintergrund ablaufen lassen und nur im Fehlerfall nach Erhalt der E-Mail manuell eingreifen.

Ein weiteres Beispiel: Mittels Namenskonventionen der Dateinamen kann man die Seiten automatisch auf den Standbogen platzieren. Eine RBA-Regel fängt das Ereignis „Bogen gefüllt“ ab und das Ergebnis wird automatisch auf einem Formproofer ausgeben.

Über RBA-Regeln lässt sich auch Fremd-Software in die Prinergy-Umgebung integrieren. Außerdem kann man über das grafische RBA-Werkzeug hinaus mit der Computersprache Visual Basic Workflow-Steuerungen definieren.

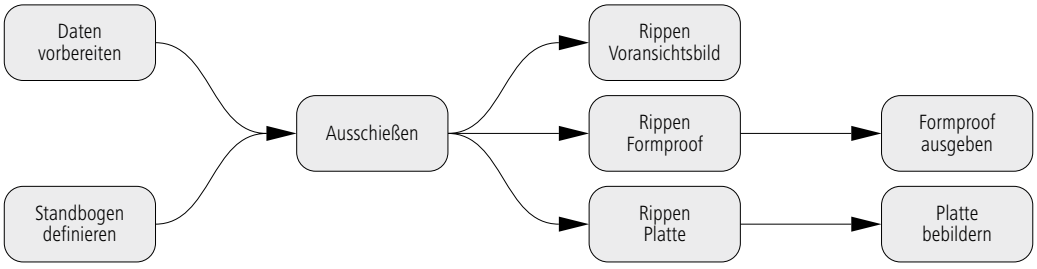
beispielsweise die Auflagenhöhe (die Vorstufe muss allerdings wegen mangelnder Beständigkeit der Druckplatten bei sehr hohen Auflagen ggf. mehrere Plattensätze erzeugen), aber auch Dinge, die sie unmittelbar betreffen, wie die Farbigkeit oder auf welcher Druckmaschine der Job gedruckt werden soll. Zwar gibt es im JDF ein *UpdateJDF*-Kommando, das von einem MIS an das Vorstufensystem geschickt werden könnte, doch die Systeme fangen erst jetzt damit an, diese Möglichkeiten auch zu implementieren. Auch die umgekehrte Richtung findet man natürlich häufig in der Praxis: Ein Operator in der Vorstufe verändert Dinge, über die das MIS informiert werden sollte, damit der Produktionsweg nicht in zwei verschiedenen Versionen als JDF-Beschreibung vorliegt. Das Problem von JDF-Updates ist bisher erst im Ansatz gelöst und wird sicherlich ein wichtiges Thema der nächsten Jahre bleiben.

In dem *MIS to Prepress ICS* werden hauptsächlich *GrayBoxen* definiert, also Informations-Container, die das MIS aufbaut, die aber von der Druckvorstufe noch mit Details aufgefüllt werden müssen. Insgesamt gibt es 11 von diesen grauen Schachteln:

- *PrePressPreparation*
- *ImpositionPreparation*
- *ImpositionProofing*
- *ImpositionRIPing*
- *PlateSetting*
- *PlateMaking*
- *ImpositionSoftProofing*
- *PageProofing*
- *PageSoftProofing*
- *ContentCreation*
- *ProofAndPlateMaking*

Um den Vorstufen-Workflow zu beschreiben, können diese *GrayBoxen* auch wie Bausteine zusammengesetzt werden. Und wie bei Bausteinen üblich, gibt es große und kleine davon. Anhand von dem Aktivitätendiagramm 9.2 möchten wir

Abbildung 9.2
Aktivitätendiagramm Druckvorstufe (Akzidenz)

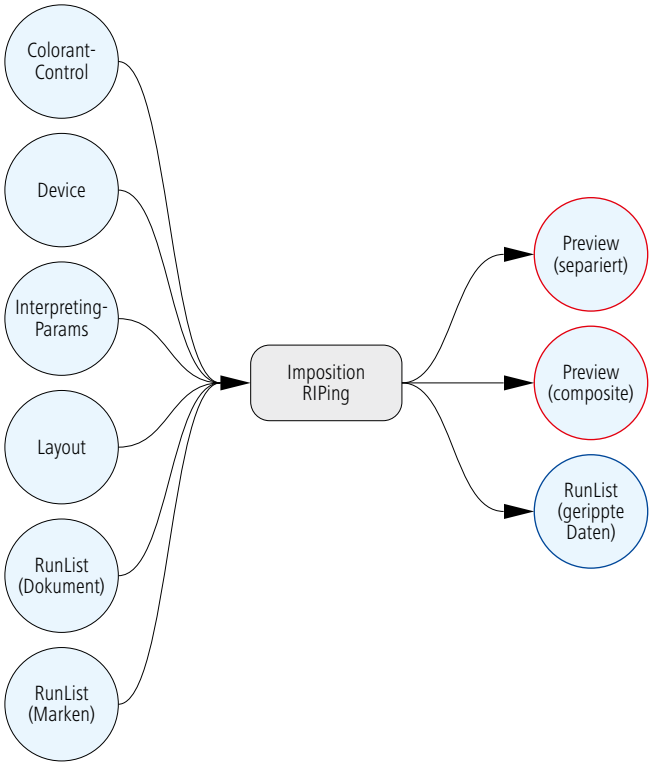


diesen etwas nebulösen Vergleich verdeutlichen. Sehr schematisch ist ein typischer Workflow skizziert, bei dem für jede Ausgabe (Voransichtsbild, Formproof, Platte) jeweils neu gerippt wird. Die RIP-Vorgänge sind allerdings sehr unterschiedlich. Während beim Voransichtsbild eine Auflösung von vielleicht 72 ppi und beim Formproof von 600 ppi erzeugt wird, muss für die Platte die Adressierungsauflösung des Belichters erreicht werden, also meist 2400 oder 2540 dpi. Außerdem werden für die Plattenbebilderung separierte Bilder benötigt, beim Formproof dagegen Composite-Bilder. Ein Voransichtsbild ist mal separiert und mal composite, je nach Verwendungszweck.

Die Aktivitäten in 9.2 spiegeln sich auch in den GrayBoxen wieder:

- *PrepressPreparation*: Daten vorbereiten
- *ImpositionPreparation*: Standbogen definieren
- *ImpositionProofing*: Ausschießen, Formproof rippen, Formproof ausgeben
- *ImpositionRIPing*: Ausschießen, Platte rippen, zusätzlich optional: Voransichtsbild rippen
- *PlateSetting*: Platte bebildern

Abbildung 9.3
GrayBox
ImpositionRIPping




Die *GrayBox PlateMaking* ist gewissermaßen ein Zusammenschluss von *Imposition-Ripping* und *PlateSetting*, also ein „großer Baustein“, der die beiden kleinen überflüssig macht. Ein noch größerer ist die *GrayBox ProofAndPlate-Making*, welche *Imposition-Proofing* und *PlateMaking* umfasst.

Das *MIS-to-Prepress-ICS* legt für jede *GrayBox* einige Attribute fest, vor allem aber die benötigten Input- und Output-Ressourcen sowie deren Attribute. Die drei *GrayBoxen ImpositionRipping*, *PlateSetting* und *PlateMaking* sind in den Abbildungen 9.3 bis 9. 6 mit Ihren Ressourcen dargestellt, wobei *PlateMaking* be-

reits in Abbildung 6.18 illustriert wurde. Normalerweise werden *ImpositionRIPing* und *PlateSetting* hintereinander geschaltet und die blau umrandete *RunList*-Ressource, die typischerweise die gerasterten Separationen im TIFF-Format repräsentiert, bildet die Übergabe zwischen beiden. Abbildung 9.6 zeigt, wie *PlateMaking* die *GrayBoxen ImpositionRIPing* und *PlateSetting* umfasst. Natürlich muss dabei der Output von *ImpositionRIPing*, soweit er nicht nur als Input von PlateSetting dient, auch als Output bei PlateMaking vorkommen. Diese beiden Preview-Ressourcen haben wir zur Verdeutlichung rot umrandet. Außerdem muss *PlateMaking* alle Input-Ressourcen von *ImpositionRIPing* und *PlateSetting* enthalten, bis auf die blau umrandete RunList-Ressource, die ja nur ein Zwischenergebnis darstellt.

Eine MIS-Software kann also die Arbeitsvorgänge in der Druckvorstufe unterschiedlich beschreiben, wie man in Abbildung 9.6 sieht, wobei die Pfeile einfach nur die Folge von *GrayBoxen* symbolisieren. Je nach Freude am Detail kann der *Manager*, das heißt das MIS, unterschiedliche *GrayBox*-Strecken aufbauen. Der *Worker*, also das Workflowsystem in der Vorstufe, muss alle Versionen interpretieren und im Laufe der

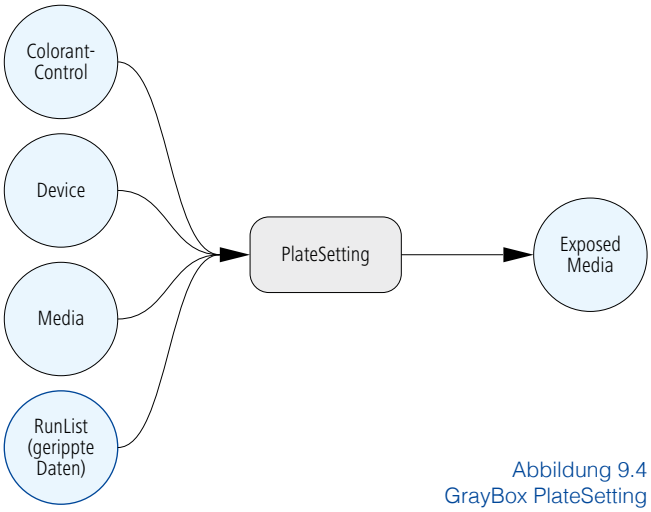



Abbildung 9.4
GrayBox PlateSetting

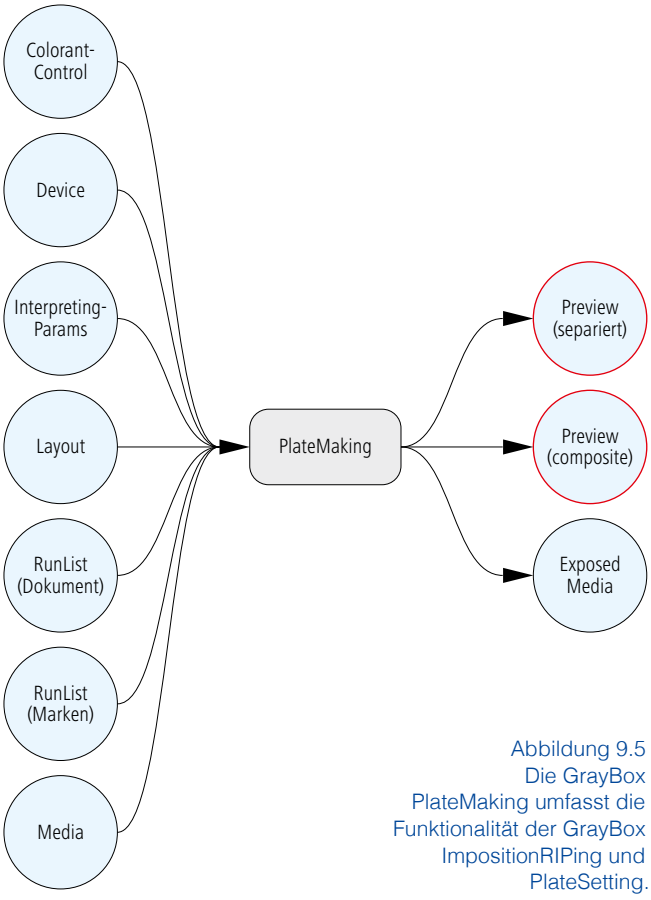



Abbildung 9.5
Die GrayBox PlateMaking umfasst die Funktionalität der GrayBox ImpositionRIPing und PlateSetting.

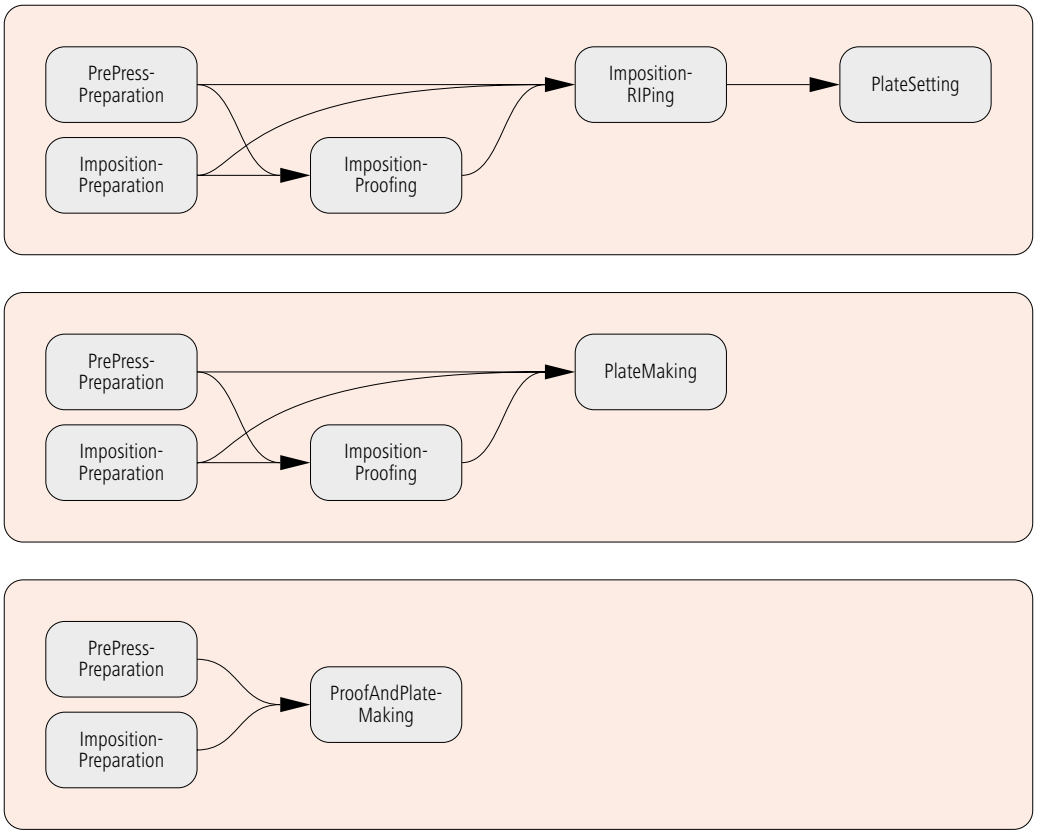



Abbildung 9.6
drei unterschiedliche Möglichkeiten zur Verkettung von GrayBoxen

Produktion zu JDF-Prozessknoten umwandeln können.

Auch im *MIS-to-PrePress-ICS* werden mehrere aufeinander aufbauende Level definiert. Alle vorgestellten *GrayBoxen* sind in Level 1 festgelegt (und damit automatisch auch für Level 2). Level 2 trifft zusätzliche Vereinbarungen zu Versionierungen, wie sie bei mehreren Sprachvarianten eines Produktes vorkommen, und auch zu komplexen Produktionen, bei denen unterschiedliche Produktteile (wie Umschlag und Inhalt) auf einem Bogen zusammen gebracht werden.

Abschließend sei noch vermerkt, dass das *MIS-to-PrePress-ICS*-Papier auch Vorgaben der anderen ICS-Papiere umfasst, auf denen es basiert. So wird *Base-ICS* und *JMF-ICS* in Level 2 vorausgesetzt – und zwar für beide Stufen des *MIS-to-PrePress-ICS*. Die Stufen können also durchaus unterschiedlich sein. Beim *MIS-ICS*, das natürlich ebenfalls eine Voraussetzung für das *MIS-to-PrePress-ICS* darstellt, ist die Situation so, wie man es eigentlich eher auch erwartet: Level 1 vom *MIS-ICS* ist Voraussetzung für Level 1 von *MIS-to-PrePress-ICS* und

Level 2 von *MIS-ICS* ist Voraussetzung für Level 2 von *MIS-to-Prepress-ICS*.

## 9.2 Montage

Die Montage besteht aus den folgenden drei Arbeitsschritten, die hintereinander ausgeführt werden:

- Standbogen erstellen,
- Zuordnen der Seiten/Nutzen auf dem Standbogen,
- Ausschießen, das heißt Berechnung einer Datenstruktur, die alle Druckdaten für die Belichtung einer oder mehrerer Platten enthält. Die einzelnen Seiten werden also mit den Marken zusammengerechnet (typischerweise in eine PDF-Datenstruktur).

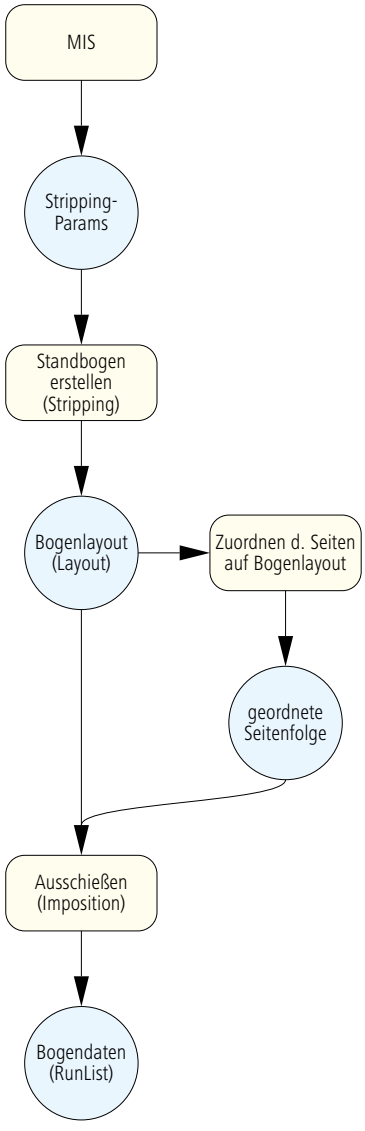
Alle drei Schritte wurden früher in einer Software ausgeführt, der Montage-Software. Mittlerweile ist es aber üblich, nur noch die Standbogenerstellung in dieser Software vorzunehmen, während die Zuordnung im WMS passiert und das Ausschießen im Hintergrund auf einem leistungsstarken Workflow-Server. Durch die immer stärker werdende Verzahnung von Montage- und WMS-Software wird die Trennung allerdings wieder unschärfer.

In diesem Abschnitt geht es aber vornehmlich um die Erstellung des Standbogens, häufig auch Bogen-Layout genannt. Die Aufgaben sind dabei typischerweise:

- Platzierung des Druckbogens auf der Platte,
- Festlegung von Bogen- und Seitengrößen,
- Anwendung eines Ausschießschemas,
- Festlegung der Ränder und Abstände (Greiferrand, Kleberand, Beschnitt...),
- Platzierung von Kontrollelementen (Druckkontrollstreifen, Register-, Falz-, Schneide-, Flatter- und Seitenmarke, Bogensignaturen...) auf Druckbogen und Platte.

Viele dieser Informationen kann ein Auftrags-Managementsystem bereit stellen. Wir haben bereits im vorherigen Kapitel ausführlich die *StrippingParams*-Ressourcen analysiert, die genau diese Informationen

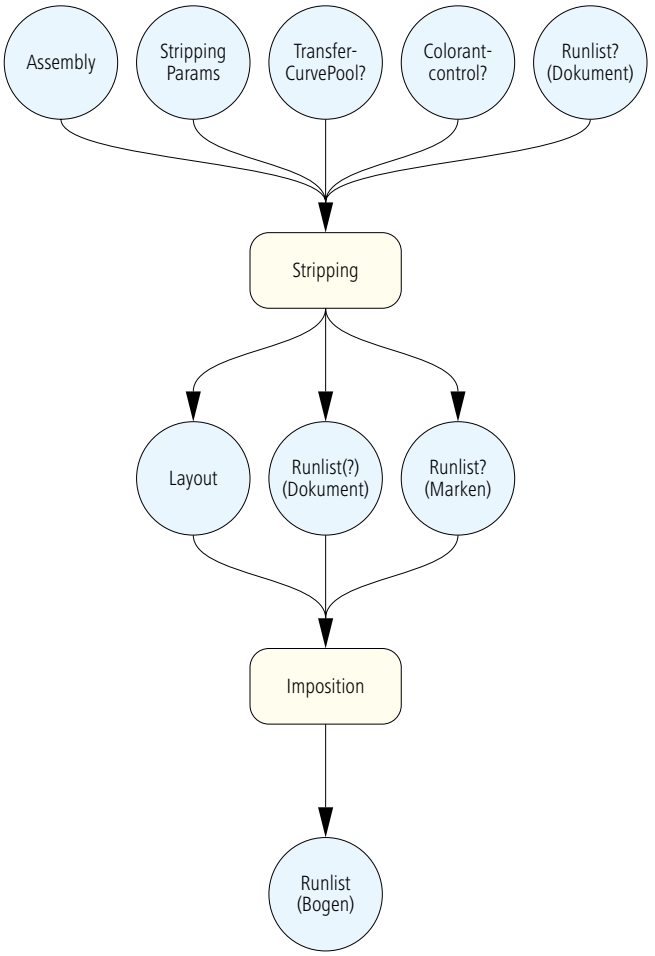
Abbildung 9.7
die Montage als Prozess-Ressourcen-Modell



enthalten. Stark vereinfacht ergibt sich also eine Workflow-Struktur, wie in Abbildung 9.7 gezeigt. Wir möchten betonen, dass nicht alle Prozesse und Ressourcen unbedingt JDF-Strukturen repräsentieren, sondern nur die allgemeinen Aufgaben im Workflow wiedergeben.

In dieser Abfolge steckt ein beträchtliches Automatisierungspotential. Ohne JDF-Anbindung zu einem AMS müssen nämlich beim Anlegen eines neuen Standbogens alle Angaben zum Bogen-Layout manuell in die Montage-Software eingegeben werden. Natürlich können Standbogen gespeichert und für Aufträge mit der gleichen Struktur wiederverwendet oder ggf. leicht modifiziert werden. Bei einer JDF-Integration zwischen AMS und Vorstufe können jedoch alle Werte vom AMS direkt übernommen und die Standbogen automatisch generiert werden. In der Offset-Praxis wird allerdings jeder so erzeugte Standbogen (noch) von einem Operator kontrolliert und in Feinheiten verändert. Voraussetzung für dieses Verfahren ist, dass sich Kontrollmarken relativ zur Bogengröße positionieren und nicht – wie früher üblich – mit festen Koordinatenwerten auf den Bogen platziert werden. Bogen unterschiedlicher Größe benötigen nämlich die Marken an unterschiedlichen Positionen und da diese Positionen nicht vom AMS erhältlich sind, sondern nur Angaben zur Bogengröße, müssen sich die Marken automatisch zurechtrücken. Dieser Vorgang wird zur Zeit eigentlich von der Montagesoftware definiert und durchgeführt. Ab JDF 1.4 können dynamische Marken vom MIS in den StrippingParams charakterisiert werden und der Stripping-Pro-

Abbildung 9.8
die Montage im JDF-Modell



zess muss sie nur noch ausführen. Es ist allerdings erforderlich, dass sowohl das Personal, als auch die Software bei der Auftragsgenerierung die entsprechenden Voraussetzungen dafür erfüllen.

Wir werden nun die JDF-Strukturen zu Abbildung 9.7 genauer analysieren. In Abbildung 8.7 wurde bereits der Prozess zur Erzeugung des Standbogens, nämlich *Stripping*, mit einigen seiner Input- und Output-Ressourcen vorgestellt. Abbildung 9.8 gibt nun die Situation umfassender wieder. Ein „?“ kennzeichnet in dieser Abbildung optionale Ressourcen. Wie man sieht, haben wir nun im Vergleich zu Abbildung 8.7 auch die Input-Ressourcen *TransferCurvePool*, *ColorantControl* und *RunList* (Dokument) für den Prozess *Stripping* mit aufgenommen. Die *TransferCurvePool*-Ressource beinhaltet hauptsächlich die Transferkurven, die auch Tonwertanpassungskurven oder Prozesskalibrierungskurven genannt werden. Sie sind eigentlich nur für den RIP-Vorgang wichtig, und wir werden sie deswegen auch erst im Abschnitt 9.4 genauer beschreiben. Hier wird die Ressource nur dazu gebraucht (wenn überhaupt), um Informationen über Koordinatentransformationen zwischen Standbogen und Platte zu bekommen. In *ColorantControl* wird die Farbigkeit der Separationen festgelegt, so dass auch die entsprechenden Farbmarken auf dem Standbogen generiert werden können. Schließlich wäre es noch möglich, dass eine *RunList*-Ressource, welche das zu druckende Dokument beschreibt, beim *Stripping*-Prozess verwendet wird, um die Seiten einzulesen. Dann könnte man den Standbogen unter Sichtkontrolle der Druckdaten am Bildschirm aufbauen beziehungsweise kontrollieren.

Die *RunList*-Ressource würde in diesem Fall optional weitergereicht an den *Imposition*-Prozess. Das heißt, diese Ressource kann, muss aber nicht Output vom *Stripping*-Prozess sein (deswegen das „?“). In jedem Fall muss aber eine *RunList* (wo immer sie auch herkommt) Input-Ressource vom *Imposition*-Prozess sein, sonst hätte der nichts auszuschießen. Insofern haben wir das Fragezeichen in der Abbildung wieder eingeklammert. Typischerweise wird der *Stripping*-Prozess auch eine Datei (oder auch mehrere) erzeugen, welche alle Kontrollzeichen auf der Platte an der richtigen Position enthält. Diese wird in der Ressource *RunList* (Marken) beschrieben. Über JDF können zwar Funktion und Position von Kontrollmarken übergeben werden, nicht aber deren Aussehen. Insofern muss das separat geschehen; zum Beispiel in PDF.

Zum Abschluss möchten wir in Abbildung 9.9 noch ein Beispiel

```
<Layout Class="Parameter" DescriptiveName="Zustände" ID="_300"
 PartIDKeys="SignatureName SheetName Side" Status="Available">
   <Layout Name="Signature-1" SignatureName=" Signature-1">
      <Layout DescriptiveName="Umschlag" Name="Umschlag" SheetName="Umschlag"
       SourceWorkStyle="WorkAndBack"
       SurfaceContentsBox="0 0 2111.81 1714.96">
         <Media Class="Consumable" Dimension="2111.81 1714.96"
          MediaType="Plate" />
         <Layout DescriptiveName="Schöndruck" Side="Front">
            <ContentObject CTM="1 0 0 1 316.06 188.50"
             ClipBox="307.55 180 1036.06 537.16"
             PositionX="Center" PositionY="Center"
             TrimCTM="1 0 0 1 316.06 188.50" TrimSize="711.49 340.15" />
            <ContentObject CTM="1 0 0 1 1084.25 188.50"
             ClipBox="1075.74 180 1804.25 537.16"
             PositionX="Center" PositionY="Center"
             TrimCTM="1 0 0 1 1084.25 188.50" TrimSize="711.49 340.15" />
            <ContentObject CTM="1 0 0 1 316.06 585.35"
             ClipBox="307.55 576.85 1036.06 934.01"
             PositionX="Center" PositionY="Center"
             TrimCTM="1 0 0 1 316.06 585.35" TrimSize="711.49 340.15" />
            <ContentObject CTM="1 0 0 1 1084.25 585.35"
             ClipBox="1075.74 576.85 1804.25 934.01"
             PositionX="Center" PositionY="Center"
             TrimCTM="1 0 0 1 1084.25 585.35" TrimSize="711.49 340.15" />
            <ContentObject CTM="1 0 0 1 316.06 982.20"
             ClipBox="307.55 973.70 1036.06 1330.86"
             PositionX="Center" PositionY="Center"
             TrimCTM="1 0 0 1 316.06 982.20" TrimSize="711.49 340.15" />
            <ContentObject CTM="1 0 0 1 1084.25 982.20"
             ClipBox="1075.74 973.70 1804.25 1330.86"
             PositionX="Center" PositionY="Center"
             TrimCTM="1 0 0 1 1084.25 982.20" TrimSize="711.49 340.15" />
            <MarkObject CTM="1 0 0 1 0 0" ClipBox="0 0 2111.81 1714.96">
               <RegisterMark Center="1900.86 661.94" Class="Parameter"
                Rotation="0" />
               <RegisterMark Center="210.94 661.94" Class="Parameter"
                Rotation="0" />
               <ColorControlStrip Center="1059.44 1347.70"
                Class="Parameter" Rotation="0" Size="1486.18 28"
                StripType="FOGRA_6_F74_740x10" />
            </MarkObject>
         </Layout>
          ...
      </Layout>
   </Layout>
</Layout>
```

Abbildung 9.9
JDF-Code einer Layout-Ressource

für eine *Layout*-Ressource zeigen, in der Platzhalter für sechs Seiten definiert sind. So ein Platzhalter heißt in der JDF-Sprache *ContentObject*. Wie man sieht, ist die *Layout*-Ressource partitioniert gemäß dem Schlüssel *SignatureName SheetName Side*. Das muss auch so sein, denn die Platzhalter für die Seiten und

vor allem die Positionen der Marken können natürlich von Signatur zu Signatur, von Bogen zu Bogen und auch zwischen Schön- und Widerdruck unterschiedlich sein. Das Layout des Beispielbogens ist in Abbildung 9.10 skizziert. Man beachte, dass in unserem Beispiel die Größe der *SurfaceContentsBox* gleich der Größe (*Dimension*) der Platte (*Media*) ist. Der Nullpunkt liegt bei beiden unten links. Im Allgemeinen könnte die *SurfaceContentsBox* auch kleiner sein, zum Beispiel das kleinste Rechteck, das alle zu druckende Elemente auf dem Bogen umfasst (= *Bounding Box*). Die Attribute vom *ContentObject* sind wie folgt:

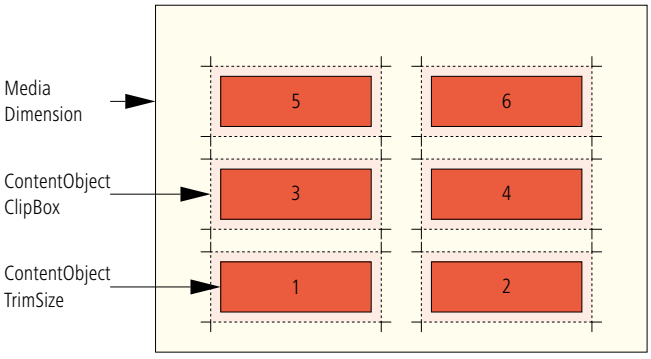



Abbildung 9.10
TrimSize und ClipBox als Attribute von ContentObject

*CTM*:
Die *Current Transformation Matrix* wurde bereits in Kapitel 4.3 als *Dictionary*-Eintrag im PDF vorgestellt. Dieser ursprünglich aus der PostScript-Welt stammende Begriff hat im JDF die gleiche Funktion: Er definiert ebenfalls die Position, Orientierung und Größe eines Platzhalters auf der *SurfaceContentsBox*, also hier auf der Platte. Siehe auch die Übung am Ende des Kapitels.

*ClipBox*:
Hier werden Größe und Position eines Platzhalters in DTP-Punkten definiert. Alle Elemente einer Seite außerhalb der *ClipBox* werden abgeschnitten. Die Position bezieht sich auf die *SurfaceContentsBox*.

*PositionX*, *PositionY*:
Art der Ausrichtung des Inhalts innerhalb eines Platzhalters.

*TrimSize*:
Die Größe einer *Trimbox*, das heißt der Seiten nach dem Beschnitt beziehungsweise das Endformat des Druckprodukts. Die Größe ist in DTP-Punkten angegeben.

*TrimCTM*:
Positionierung einer *Trimbox* auf die *SurfaceContentsBox*.

Als Markenobjekt (*MarkObject*) sind hier nur ein Farbkontrollstreifen „FOGRA_6_F74_740x10“ mit seinen


Abbildung 9.11
Blitzer entstehen durch Passerschwankungen, wenn keine Überfüllungen vorhanden sind.

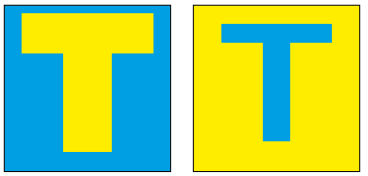


Abbildung 9.12
Die hellere Farbe überfüllt die dunklere.

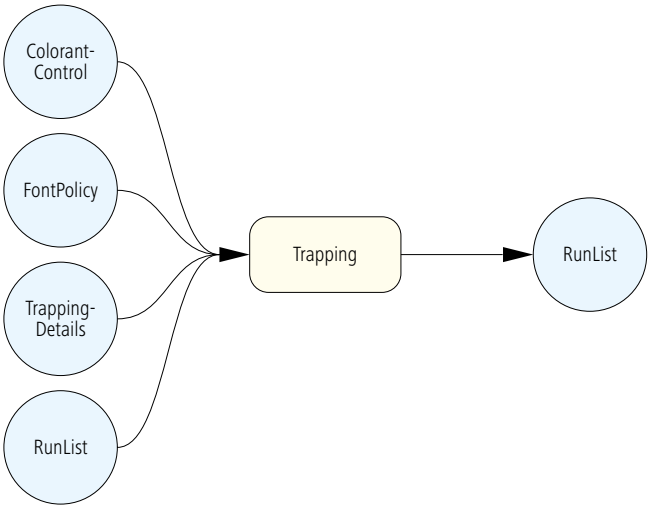
Positionen sowie zwei Registermarken eingetragen.

Ein allgemeiner Hinweis scheint uns noch wichtig: Gerade die *Layout*-Ressource wurde in den unterschiedlichen JDF-Versionen stark verändert. Insofern können JDF-Beschreibungen älterer Versionen auf den ersten Blick völlig anders aussehen als hier vorgestellt.

## 9.3 Trapping

Beim Mehrfarbendruck im Offset kann es passieren, dass der Übereinanderdruck nicht völlig standgenau ist. Diese Passerschwankungen treten vor allem durch Papierdehnung, aber auch wegen Toleranzen in der Druckmaschine auf. Eine Konsequenz dieser Ungenauigkeit ist ein „Blitzer“, wie in Abbildung 9.11 zu erkennen. Deswegen werden beim Trapping unter Umständen angrenzende Flächen vergrößert („überfüllt“). Die allgemeine Regel bei lasierenden Farben ist, dass die hellere Farbe die dunklere überfüllt; siehe Abbildung 9.12. Dann gibt es auch bei kleinen Passerdifferenzen keine Blitzer mehr. Die Kehrseite der Medaille ist, dass natürlich die Trap-Konturen ebenfalls störend auffallen. Insofern wird man sich bemühen, die Trap-Breite möglichst gering zu halten, nämlich gleich der zu erwartenden maximalen Passerdifferenz. Nur wenn in ein schwarzes Objekt überfüllt wird, ist der Trap nicht zu sehen und man kann die Trap-Breite großzügiger wählen. Bei opaken Farben wie Gold oder Silber gilt die Helligkeitsregel nicht mehr. Stattdessen sollte eine opake Farbe aus offensichtlichen Gründen niemals in eine lasierende Farbe überfüllen, sondern höchstens umgekehrt. Bei zwei angrenzenden opaken Farben überfüllt die zuerst druckende Farbe die andere.

Abbildung 9.13
Trapping-Prozess mit Ressourcen



Viele Trapping-Details, wie beispielsweise die Trap-Breite, müssen festgelegt werden. Es gibt hier keinen allgemeingültigen Wert, da er von vielen Faktoren abhängt (Papier, Druckmaschine, Druckverfahren,…). Diese Trapping-Details können in der Ressource *TrappingDetails* definiert werden, die eine Input-Ressource des *Trapping*-Prozesses ist (Abbildung 9.13). In *ColorantControl* findet man die Farbangaben über die lasierenden und opaken Farben, die in dem Druckprodukt vorkommen. In die *FontPolicy*-Ressource wird eingetragen, was geschehen soll, wenn benötigte Schriften nicht zur Verfügung stehen (Prozess abbrechen, Ersatzschriften wählen,…), und die *RunList* schließlich enthält Angaben über die Druckdaten. Das Ergebnis des *Trapping*-Prozesses ist wieder eine *RunList*, nur dass diese nun auf die getrappten Content-Daten zeigt.

Abbildung 9.14 zeigt ein Beispiel für eine *TrappingDetails*-Ressource, wie sie an eine Trapping-Engine geschickt werden könnte. Die Praxis zeigt aber, dass eine Trapping-Engine ähnlich wie auch das RIP nicht unabhängig vom WMS konfiguriert werden kann. Mit anderen Worten: Die Schnittstelle zwischen der Trapping-Engine und dem Vorstufen-Systems ist herstellerspezifisch, auch wenn sie auf JDF basiert.

Wir wollen nun die Elemente und Attribute vom Beispiel 9.14 erklären. In *DefaultTrapping* kann eingestellt werden, ob die Seiten komplett (Wert gleich *true*) oder nur gewisse Zonen (Wert gleich *false*) getrappt werden sollen. In dem Unterelement *Trap-*

Abbildung 9.14
Ausschnitt aus der TrappingDetails-Ressource

```
<TrappingDetails Class="Parameter" DefaultTrapping="true" ID="_333"
 Locked="false" Status="Available">
   <TrappingOrder>
      <SeparationSpec Name="Black" />
      <SeparationSpec Name="Cyan" />
      <SeparationSpec Name="Magenta" />
      <SeparationSpec Name="Yellow" />
      <SeparationSpec Name="Gold-Sonderfarbe" />
   </TrappingOrder>
   <TrappingParams
    BlackColorLimit="0.95"
    BlackDensityLimit="1.6"
    BlackWidth="1.3"
    ImageToImageTrapping="false"
    ImageToObjectTrapping="true"
    ImageTrapPlacement="Normal"
    StepLimit="0.25"
    TrapWidth="0.25"
    ... />
</TrappingDetails>
```

*pingOrder* wird die Reihenfolge festgelegt, wie die Farben übereinander liegen – sie sollte der Druckreihenfolge entsprechen. Wie gerade erklärt wurde, ist diese Reihenfolge bei opaken Farben von großer Wichtigkeit. Im Element *TrappingParams* sind nun die vielen kleinen Einstellungen zum Trapping hinterlegt, von denen wir nur eine kleine Auswahl präsentieren.

Das Attribut *BlackColorLimit* gibt an, ab welchem Rastertonwert eine graue Fläche nach den Regeln der Farbe Schwarz getrappt wird – hier bei 95% Tonwert. In *BlackDensityLimit* wird analog eine Farbe ab einer gewissen Dichte wie Schwarz eingestuft. Der Wert von *BlackWidth* bestimmt, dass die Trap-Breite bei Schwarz das 1,3-fache von der Trapbreite bei den übrigen Farben ist. In *ImageToImageTrapping* und *ImageToObjectTrapping* kann definiert werden, ob benachbarte Bilder beziehungsweise Bilder gegenüber anderen Objekten (Grafiken, Schriften) getrappt werden sollen. Da in dem Beispiel also Grenzen zwischen Bildern und Objekten getrappt werden (Wert *true*), gibt man in *ImageTrapPlacement* noch an, wie das geschehen soll. Hier ist normal ausgewählt, das heißt es gelten die üblichen Regeln, die auf der Helligkeit der lasierenden Farben und der Druckreihenfolge der opaken Farben beruhen. Man könnte auch auswählen, dass beispielsweise das Objekt grundsätzlich in das Bild hinein überfüllt (Wert gleich *Choke*). Ein Farbauszug sollte bei zwei benachbarten Flächen schon deutliche Tonwertunterschiede haben, damit dort überhaupt getrappt wird. Sonst könnte es im Extremfall sogar passieren, dass ein Verlauf getrappt wird. In *StepLimit* wird die Grenze festgelegt. In *TrapWidth* wird schließlich die Trap-Breite vereinbart. Der Wert ist in DTP-Punkt angegeben.

## 9.4 RIPing und Plattenherstellung

Das Produktionssystem hat die Aufgabe, die vom MIS geschriebenen *GrayBoxen* wie *PlateMaking* (siehe Abbildung 6.17) oder *ImpositionRIPing* und *PlateSetting* (siehe Abbildungen 9.4 und 9.5) in einzelne Prozesse, in einen oder mehrere kombinierte Prozesse oder in eine Prozessgruppe, die Prozesse und/oder kombinierte Prozesse enthält, aufzulösen. Auch sind natürlich Kombinationen der drei Möglichkeiten erlaubt. Die Prozesse, die so entstehen, werden dann im Prinzip wie Puzzle-Teile zusammengefügt, indem Output-Ressourcen eines Prozesses wieder Input-Ressourcen des nächsten Prozesses sind. Dabei han-

delt es sich hier meist um *RunList*-Ressourcen, also Angaben über die Daten der Zwischenergebnisse. Es ist jedoch zu beachten, dass bei kombinierten Prozessen diese Zwischenergebnisse auch fehlen dürfen. Manche dieser Puzzle-Teile wurden schon vorgestellt, wie *Interpreting*, *Rendering* und *Screening* (siehe Abbildung 3.10), andere zumindest erwähnt, wie *Imposition* und *ImageSetting* (Abschnitt 6.1). In der Abbildung 9.15 ist eine mögliche Struktur zum RIP-Vorgang und zur Plattenbelichtung noch einmal vollständiger dargestellt. Wir werden nur noch kurz auf die bisher nicht behandelten Prozesse eingehen.

Der Prozess *ContoneCalibration* ist verantwortlich für die Anwendung der Transferkurve im RIP. Er erhält vom Rendering-Prozess eine Bytemap pro Separation, also ein Halbtonbild (alle Tonwerte zwischen 0 und 100%) in jeder Prozess- oder auch Sonderfarbe. Das Resultat des Prozesses ist eine Datenstruktur in gleicher Form, nur dass die Tonwerte unter Umständen verändert wurden. Wie die Tonwerte verändert werden, ist in der Ressource *TransferFunctionControl* spezifiziert. Die Beschreibung der Transferkurven ist analog zu der Beschreibung in PPF (siehe Abbildung 4.11). Tatsächlich können hier auch pro Separation mehrere Kurven verwendet werden. So wird die Plattenausgabe häufig erst mit Hilfe einer Transferfunktion linearisiert, das heißt, es wird eine Tonwertanpassung bestimmt, die bewirkt, dass zwischen den Input-Tonwerten des RIPs und den Tonwerten auf der entwickelten Platte kein oder nur ein minimaler Unterschied ist. Neben der Linearisierungskurve, die nur vom Plattenbelichter, der Platte und unter Umständen auch vom Entwicklungsprozess abhängig ist, gibt es dann noch die eigentliche Transferkurve für den Druck, die abhängig von vielen Parametern ist, wie beispielsweise dem Ausgaberaster, dem Bedruckstoff, der Druckmaschine, den Druckbedingungen, der Farbe. Beide Transferkurven werden dann miteinander verrechnet und ausgeführt.

Der *FormatConversion*-Prozess macht genau das, was sein Name sagt: Er konvertiert ein Datenformat in ein anderes. Hier wird typischerweise eine herstellerspezifische Bitmap in eine TIFF Datei umgewandelt, die Input

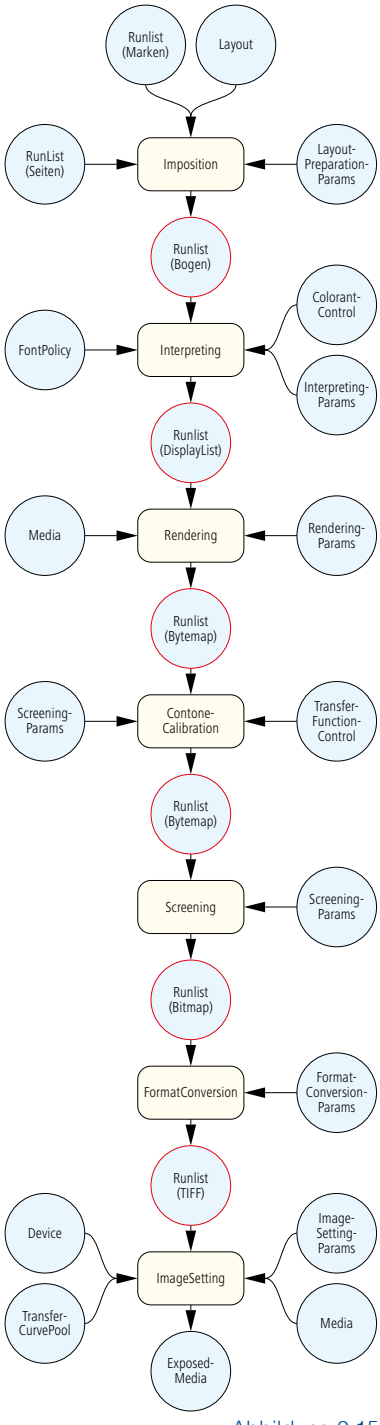



Abbildung 9.15
JDF-Modell der Plattenherstellung

```
<JDF Activation="Active" ID="_001" JobPartID="_1003.I" Status="Part"
 Type="Combined" Types="Imposition Interpreting Rendering ContoneCalibration
 Screening PreviewGeneration FormatConversion ImageSetting" Version="1.3">
...
    <ResourceLinkPool>
        <RunListLink CombinedProcessIndex="0" ProcessUsage="Document"
         Usage="Input" rRef="_77" />
        <RunListLink CombinedProcessIndex="0"  ProcessUsage="Marks"
         Usage="Input" rRef="_21" />
        <LayoutPreparationParamsLink CombinedProcessIndex="0" Usage="Input"
         rRef="_581" />
        <LayoutLink CombinedProcessIndex="0" Usage="Input" rRef="_99" />
        <ColorantControlLink CombinedProcessIndex="1" Usage="Input"
         rRef="_83" />
        <FontPolicyLink CombinedProcessIndex="1" Usage="Input" rRef="_583" />
        <TransferCurvePoolLink CombinedProcessIndex="6" Usage="Input"
         rRef="_71" />
        <InterpretingParamsLink CombinedProcessIndex="1" Usage="Input"
         rRef="_85 " />
        <RenderingParamsLink CombinedProcessIndex="2" Usage="Input"
         rRef="_86" />
        <MediaLink CombinedProcessIndex="2" ProcessUsage="Paper" Usage="Input"
         rRef="_79" />
        <ScreeningParamsLink CombinedProcessIndex="3 4" Usage="Input"
         rRef="_89" />
        <TransferFunctionControlLink CombinedProcessIndex="3" Usage="Input"
         rRef="_34" />
        <ImageSetterParamsLink CombinedProcessIndex="6" Usage="Input"
         rRef="_94" />
        <DeviceLink CombinedProcessIndex="6" Usage="Input" rRef="_286" />
        <MediaLink CombinedProcessIndex="6" ProcessUsage="Plate" Usage="Input"
         rRef="_3085" />
        <FormatConversionParamsLink CombinedProcessIndex="5" Usage="Input"
         rRef="_47" />
        <ExposedMediaLink CombinedProcessIndex="6" ProcessUsage="ExposedMedia"
         Usage="Output" rRef="_81" />
    </ResourceLinkPool>
...
</JDF>
```

Abbildung 9.16 kombinierter Prozess Plattenherstellung

Abbildung 9.17 FormatConversionParams-Ressource

für eine dem CtP-Belichter vorgeschaltete Steuerungs-Software ist. Weil diese Daten Bitmaps enthalten, also binäre Zustände von Pixeln (nur s/w und keine Grautöne), wird das Datenformat auch TIFF-B genannt. Dieses Format ist ein Quasi-Standard zwischen Workflow Management Systemen in der Vorstufe und Be-

```
<FormatConversionParams Class="Parameter" ID="_47" Status="Available">
    <FileSpec Class="Parameter" MimeType="application/octet-stream"
     ResourceUsage="InputFormat" />
    <FileSpec Class="Parameter" MimeType="image/tiff"
     ResourceUsage="OutputFormat" />
</FormatConversionParams>
```

lichtern. Damit ist es möglich, an ein WMS ohne viel Mühe unterschiedliche Belichter anzuschließen. Die Folge ist, dass die Bebilderung der Platten häufig völlig getrennt vom WMS ist. Die CtP-Belichter und ihre vorgeschaltete Software sind in aller Regel dann auch nicht JDF/JMF-fähig. In diesem Fall wird dann der JDF-Workflow bereits die Erzeugung einer TIFF-B Datei mit der Vollendung der Plattenbelichtung gleichsetzen und eine entsprechende Meldung absetzen. Dies ist nach dem *MIS-to-Pre-Press-ICS* auch ausdrücklich erlaubt.

Abbildung 9.16 stellt einen zu Abbildung 9.15 passenden kombinierten Prozess dar. Es wird jedoch nur der *ResourceLinkPool* wiedergegeben, nicht der *ResourcePool*. Man bemerke, dass die rot umrandeten RunList-Ressourcen in 9.15 nicht als Links in dem *ResourceLinkPool* wiederzufinden sind, genauso wenig wie die Ressourcen selber im *ResourcePool*, was für kombinierte Prozesse zulässig ist. Beispielhaft soll hier nur noch die Ressource *FormatConversionParams* in Abbildung 9.17 gezeigt werden. Dort erkennt man, dass der zugehörige Prozess aus einem applikationsspezifischen Byte-Strom ein Bild im Format TIFF herstellen soll.

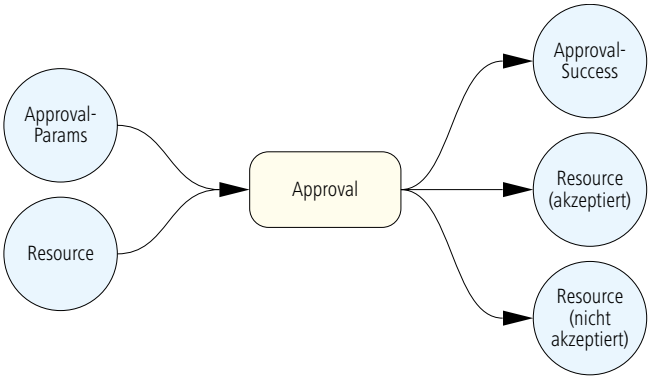
Mit Hilfe eines ähnlichen JDF-Modells kann auch der Ausdruck auf einer Digitaldruckmaschine angestoßen werden. In der Praxis ist es allerdings üblich, dass die Digitaldruckaufträge noch einmal im Front-End der Digitaldruckmaschine gesammelt werden. Dann kann nämlich ein Maschinenoperator noch einmal vor dem Ausdruck eingreifen, um beispielsweise die Aufträge nach zu verwendenden Papierarten zu sortieren.

Auch diesen Abschnitt möchten wir mit einem allgemeinen Hinweis beenden: Wie bereits erwähnt, wird die JDF-Kommunikation zwischen dem umfassenden WMS, dem RIP und einem Plattenbelichter häufig gar nicht mit wirklich offenen Schnittstellen realisiert. Das heißt, ein Hersteller benutzt zwar dafür möglicherweise JDF (und auch andere Kommunikationsoptionen wie Datenbanken), aber nur mit dem (durchaus legitimen) Ziel, seine eigenen Software-Module damit zu steuern. Fremd-Software soll sich hier gar nicht andocken können. Insofern ist das Ergebnis unserer Untersuchungen auch nicht so sehr verwunderlich, bei dem wir festgestellt haben, dass das JDF hier durchaus „kreativ“ eingesetzt wird. Insbesondere wird man an dieser Stelle in der Tat kombinierte Prozesse finden, bei denen ja die Zwischenergebnisse der Prozesse nicht aufgeführt werden müssen. Denn warum sollten auch beispielsweise die Speicherpositionen der Zwischenergebnisse in *RunList*-Ressourcen

eingetragen werden, wenn das WMS eine festgelegte Vorgehensweise hat und die Jobs in eine feste Ordner- oder Datenbankstruktur ablegt oder die Informationen sogar nur im Arbeitsspeicher (RAM) vorliegen.

## 9.5 Proof und Druckfreigabe

Abbildung 9.18
Freigabeprozess im JDF




Ein Proof (oder im Deutschen „Prüfdruck“) soll das spätere Ergebnis des Auflagendrucks möglichst exakt im Voraus darstellen. Es werden unterschiedliche Proof-Verfahren eingesetzt:

- Farbproof, farbverbindlicher Proof oder Kontraktproof (digital)
- Standproof, auch Formproof genannt
- Softproof
- Rasterproof
- Maschinenproof oder Andruck (findet, wenn überhaupt, noch im Tiefdruck oder aber auch im Flexodruck Verwendung)
- Analoger Proof (auf Basis von Filmen)

Hierbei spielen vor allem die ersten drei Technologien ein wichtige Rolle, während die letzten beiden heutzutage nur noch selten Anwendung finden.

Abbildung 9.19
Freigabeprozess für einen Proof

```
<JDF Category="DigitalPrinting" ID="_1" JobID="_2" Status="Ready"
 Type="Combined" Types="LayoutPreparation Imposition Interpreting Rendering
 ImageSetting Approval"...>
...
</JDF>
```

Das Erstellen eines Prüfdrucks ist Teil eines umfassenderen Qualitätsmanagements. Denn natürlich werden auch weitere Dinge überprüft wie Druckplatten, Druckbogen des Auflagendrucks oder auch Teilkomponenten des Druckproduktes. Dabei ist der Zweck, immer eine gewisse Abnahmeprozedur zu durchlaufen, bei der ein Kunde/Auftraggeber oder auch ein interner Mitarbeiter entscheidet, ob das zu prüfende Teil akzeptiert

wird oder abgelehnt werden muss. In der JDF-Sprache werden Ressourcen für die weiteren Produktionsprozesse freigegeben oder nicht freigegeben. Für diesen Abschnitt im Buch sind solche Ressourcen üblicherweise *ExposedMedia*, die Ausdrucke des Farb- oder Standproofs repräsentieren, oder auch *RunLists*, welche die Bilddaten für den Softproof darstellen.

Abbildung 9.18 zeigt diese Situation noch einmal grafisch. Typischerweise wird man eine oder mehrerer Ressourcen haben, die freigegeben werden sollen. Die Ressource *ApprovalParams*

Abbildung 9.20
Informationen zur Farbraumtransformation beim Softproof

```
<ColorSpaceConversionParams ColorManagementSystem="ADBE" ID="R6"
 Class="Parameter" Status="Available">
   <ColorSpaceConversionOp SourceCS="Gray" SourceObjects="All"
    Operation="Untag" />
   <ColorSpaceConversionOp SourceCS="RGB" SourceObjects="All"
    IgnoreEmbeddedICC="false" RGBGray2Black="true" RGBGray2BlackThreshold="1"
    Operation="Tag" RenderingIntent="Perceptual" >
      <FileSpec ResourceUsage="SourceProfile"
       URL="file://Server1/ICCProfiles/sRGBprofile.icm" />
   </ColorSpaceConversionOp>
   <ColorSpaceConversionOp SourceCS="CMYK" SourceObjects="All"
    Operation="Untag" />
   <FileSpec ResourceUsage="FinalTargetDevice"
    URL="file://Server1/ICCProfiles/CoatedFOGRA39.icc" />
</ColorSpaceConversionParams>
```

enthält Detailinformationen über den Freigabevorgang, nämlich im Wesentlichen wer die Freigabe erteilen kann (beschrieben durch das Unterelement *Contact*) und welche Rolle diese Person oder diese Personen dabei einnehmen. Letzteres bedeutet, wer beispielsweise Freigabeerklärungen eines anderen überschreiben darf und dergleichen. Der Output des *Approval*-Prozesses besteht aus den freigegebenen beziehungsweise nicht freigegebenen Ressourcen. Das *Status*-Attribut einer Ressource hat vor dem *Approval*-Prozess den Wert *Draft* (Entwurf), danach entweder einen *Status Available* (angenommen) oder *Rejected* (zurückgewiesen). In *ApprovalSuccess* ist dann noch vermerkt, wer genau den Proof abgezeichnet oder abgelehnt hat, mögliche Kommentare hierzu und gegebenenfalls einen Link auf die Datei, welche die Unterschrift der abzeichnenden Person enthält.

Den *Approval*-Prozess kann man dann als letzten Prozess eines kombinierten Prozesses definieren. Ein Hardcopy-Proof auf einem Digitaldrucker ist demnach ein kombinierter Prozess, der mit den beiden Prozessen „*ImageSetting Approval*“ endet,

wie in Abbildung 9.19 zu sehen. Bei einem Maschinenproof würde analog der kombinierte Prozess die Prozesse *ImageSetting ConventionalPrinting* anstatt *ImageSetting* umfassen. Die Prozesse *ConventionalPrinting* sowie *DigitalPrinting* werden im folgenden Kapitel genauer behandelt.

Für farbverbindliche Proofs, aber natürlich auch für die Plattenbelichtung, werden in der Regel Farbraumtransformationen benötigt. Üblicherweise werden hierzu Farbprofile definiert, welche die Farbeigenschaften von Geräten beschreiben. Für die Eingangsdaten (Digitalkamera, Scanner) stehen zunächst gerätespezifische RGB-Daten zur Verfügung, die mit Hilfe eines Profils in geräteunabhängige Farbwerte gewandelt werden müssen. Für die Druckformherstellung wird dann noch ein zweites Profil benötigt, das den Druckprozess beschreibt. Zur Herstellung eines farbverbindlichen Proofs wird noch ein weiteres Profil benötigt, nämlich das Prooferprofil.

Die Farbraumtransformation erfolgt im JDF durch den Prozess *ColorSpaceConversion*, der typischerweise vor dem *Trapping*-Prozess stattfindet. Wichtigste Input-Ressource ist die *ColorSpaceConversionParams*, in der auch der Speicherort der Farbprofile eingetragen ist. Beispiel 9.20 zeigt eine solche Ressource, deren Elemente und Attribute im Folgenden erklärt werden.

*ColorManagementSystem*:

Hier wird das bevorzugte, herstellerspezifische System zur Farbraumumrechnung definiert („Adobe“).

*FileSpec*:

In dem Attribut ist das ICC-Profil zur Definition der Zielwerte eingetragen.

*ColorSpaceConversionOp*:

Die Operationen zur Farbraumtransformation werden genauer

Abbildung 9.21
Current Transformation Matrix

$$\begin{pmatrix} x & y & 1 \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & 1 \end{pmatrix} = \begin{pmatrix} a_{11}x + a_{21}y + a_{31} \\ a_{12}x + a_{22}y + a_{32} \\ 1 \end{pmatrix}$$

CTM = „$a_{11}$ $a_{12}$ $a_{21}$ $a_{22}$ $a_{31}$ $a_{32}$“

CTM = „1 0 0 1 a b“ entspricht Verschiebung von (x,y) auf (x+a, y+b)

beschrieben, wobei

- in *SourceCS* der Ausgangsfarbraum festgelegt wird,
- in *SourceObject* die Objektarten spezifiziert werden, auf welche die Operation ausgeführt werden sollen, wie *Text*, *LineArt* (Strichgrafik), *ImagePhotografic* (Foto) usw. oder aber *All*, wenn alle Objekte betroffen sind,
- in *Operation* wird definiert, ob Profile an die grafischen Elemente angehängt (*tag*) oder im Gegenteil die angehängten Profile entfern werden sollen (*untag*),
- in *IgnoreEmbeddedICC* angegeben wird, ob eingebettete Profile ignoriert werden sollen (*true*) oder nicht (*false*),
- in *RGBGray2Black* angegeben werden kann, ob RGB-Grau bei der CMYK-Konvertierung wieder auf den Schwarzkanal K abgebildet werden sollte (*true*) oder nicht (*false*), wobei im Fall von *true*
- das Attribut *RGBGray2BlackTreshhold* den Vorgang noch tonwertabhängig einschränken kann; bei einem Wert 0 wird nur das RGB-Schwarz im Vollton, bei dem Wert 0,5 alle RGB-grauen Tonwerte bis 50%, bei dem Wert 1 sämtliche RGB-Grautöne auf den K-Kanal abgebildet,
- in *RenderingIntent* die im Color Management übliche Priorität beim *Gamut Mapping* (Umrechnung von einem Farbraum in einen anderen) festlegt.

In dem Beispiel werden also alle Farbprofile entfernt, die grafische Elemente im Farbraum CMYK oder Grau betreffen, und allen RGB-Elementen ein spezifisches Profil assoziiert, sofern sie bisher noch kein eigenes haben. Die Farbraumtransformation beschreibt einen Vorgang für die Offset-Plattenherstellung.

---

**Übung**

- Zeichnen Sie einen Standbogen mit den Angaben aus Abbildung 9.9. Beachten Sie dabei die Definition der Transformationsmatrizen gemäß Abbildung 9.21. Jeder Punkt (x,y) im 2-dimensionalen Raum wird zunächst formal als 3-dimensionaler Punkt (x,y,1) geschrieben (um auch Verschiebungen zu ermöglichen). Die Transformationsmatrix ist dann folglich eine 3x3-Matrix, wobei die rechte Spalte die festen Werte 0, 0 ,1 hat. Deswegen werden diese drei immer gleichen Werte auch nicht in die *Current Transformation Matrix* (CTM) mit aufgenommen und es bleiben nur 6 Werte übrig. Nach der Matrizenmultiplikation

kann die dritte Dimension wieder entfernt werden und man sieht, dass beispielsweise die CMT="1 0 0 1 a b" eigentlich nur eine Verschiebung einer Koordinate um den Wert (a,b) bewirkt.

# 10 Druck

Auf den ersten Blick ist es verblüffend, dass es in JDF nur drei unterschiedliche Prozesse im Druckbereich gibt: *ConventionalPrinting* (Drucken mit physischer Druckform wie Offset-, Flexo-, Tief- und Siebdruck), *DigitalPrinting* (Digitaldruck) und *Varnishing* (Lackierung). Im Vergleich dazu gibt es deutlich mehr Prozesse in der Vorstufe und in der Weiterverarbeitung, wie man in der deutschen Übersetzung der JDF-Prozessnamen im Anhang dieses Buches sieht. Der Grund für diese Situation ist, dass im Druckbereich anders als in der Vorstufe und in der Weiterarbeitung die Arbeitsvorgänge sehr ineinander greifen und nicht isoliert, unabhängig voneinander und in unterschiedlichen Reihenfolgen durchgeführt werden können (vergl. Abschnitt 3.3).

Wir werden in diesem Kapitel nur auf die JDF-Knoten eingehen, die für den Druck von Belang sind. Wir haben vorher schon – vor allem in Kapitel 9 – die allgemeine Kommunikation zwischen den JDF/JMF-Geräten (hier also der Leitstand einer Druckmaschine) und Agenten und Controllern (entweder MIS oder Workflow Server der Produktion) abgehandelt. Damit aber die neuen JDF-Knoten nicht so völlig aus dem Zusammenhang gerissen erscheinen, möchten wir noch einmal ganz kurz das typische Kommunikationsszenario wiederholen: Der für die Druckmaschine verantwortliche Agent/Controller speichert zunächst eine JDF-Datei in einen Hotfolder, den das JDF-Modul des Druckmaschinenleitstandes regelmäßig abfragt. Alternativ versendet der Agent/Controller ein JMF-Kommando (*SubmitQueueEntry*) an den Leitstand, der seinerseits die darin enthaltene URL dazu verwendet, die JDF-Datei von einem Datei-Server herunterzuladen, auf den beide Parteien Zugriff haben. Die JDF-Datei enthält neben den üblichen Dingen wie Kundennamen (*CustomerInfo*) und Auftragsnummer (*JobID*) auch alle Ressourcenbeschreibungen, die für den Druckprozess notwendig sind und auf die in den folgenden Abschnitten eingegangen werden soll. Für die Betriebsdatenerfassung muss der Agent/Controller entweder mittels JDF (*NodeInfo*) oder mit einer JMF-Message einen festen Kommunikationskanal (*persistent channel*) bei dem Leitstand der Druckmaschine beantragen (*Subscription*). Der wird den Kanal „aufbauen“, dieses mit einer *Response*-JMF bestätigen und regelmäßig die gewünschten JMF-Signale über HTTP an den Agenten/Controller senden. Das geschieht so lange, bis der Agent/Controller ein Kommando schickt, dass er ab jetzt die Signale nicht mehr erhalten möchte (*StopPersistentChannel*). Die Signale werden weder vom Agenten/Controller bestätigt noch vom Leitstand zwischengespeichert, was im Englischen mit dem martialischen, aber treffenden Ausdruck *fire and forget* bezeichnet wird.

Während des Druckvorgangs schreibt der JDF-Leitstand seine Protokolldaten außerdem als *Audit*-Elemente in den *AuditPool* der JDF-Datei. Nachdem der Auftrag abgeschlossen ist, gibt der Druckmaschinenleitstand dann schlussendlich die so veränderte JDF-Datei zurück an den Agenten/Controller, wobei das wieder entweder über speziellen Hotfolder oder mittels einer JMF-Nachricht geschehen kann.

## 10.1 Konventionelles Drucken

Einige konventionelle Druckmaschinen gestatten es, auch Inline Vorstufen- oder Weiterverarbeitungsprozesse durchzuführen wie beispielsweise Digitaloffsetmaschinen, die Platten bebildern oder Rollenoffsetmaschinen, die auch falzen und schneiden können. Der JDF-Prozess *ConventionalPrinting* umfasst solche Prozesse jedoch nicht, und man muss dann kombinierte Prozesse mit Prozessen aus der Vorstufe oder aus der Weiterverarbeitung bilden. Für den Digitaloffsetdruck würde der kombinierte Prozess aus *ImageSetting* und *ConventionalPrinting* bestehen. Beim Rollenoffset würden die beiden Prozesse *ConventionalPrinting* und *WebInlineFinishing* (Inline-Weiterverarbeitung) kombiniert.

Während in der Vorstufe viele Parameter für die Bereiche Druck und Weiterverarbeitung festgelegt werden, hat vergleichsweise der Druck sehr einfache Schnittstellen. Er erhält Vorgaben vom MIS und Vorstufe, produziert aber außer Druckbogen und Sta-

**JDF-Systeme für Offset-Druckmaschinen (2009)**

| **Hersteller:** | **Produkt:** |
|---|---|
| Akiyama | InkZone |
| Goss International | Goss Web Center |
| Heidelberger Druckmaschinen | Prinect Press Manager |
| manroland | printnet PressManager Sheetfed |
| Komori Lithografic Presses | K-Station |
| König & Bauer AG (KBA) | JDFLink für Logotronic |
| Mitsubishi Heavy Industries | IPC Server II |
| Ryobi Druckmaschinen | MIS Connection Software |
| Shinora Machinery Company | Shinora CIP4 Center / Station |
| Sakurai | InkZone Pefect |

tusmeldungen seiner Tätigkeiten keine weiteren Dinge für die anderen. Insofern ist es auch nicht verwunderlich, dass der Prozess *ConverntionalPrinting* viele (optionale) Input-Ressourcen aufweist, aber nur eine Output-Ressource.

Welche Art von Informationen, Festlegungen und Vorarbeiten anderer Abteilungen (vor allem vom MIS oder der Vorstufe) sind für die Druckabteilung von Bedeutung? Zunächst erst einmal unabhängig davon, ob sie diese Daten mittels JDF oder in irgendeiner anderen Form erhält.

1. Druckformen
2. Druckfarbe
3. Angaben über den Bedruckstoff (Größe, Papierklasse...)
4. Festlegung über Schön- und Widerdruck
5. Festlegung der Farbigkeit des Jobs
6. Administrative Daten (Auftragsnummer, Kunde, Abnahmebedingungen, Auflage, Liefertermin...)
7. Farbzonenvoreinstellungen bei Offsetdruck
8. Proof (Formproof, Kontraktproof, Muster, Vorschaubild, Softproof)
9. Positionen von Kontrollelementen auf den Druckbogen
10. Festlegungen von Druckbedingungen (Lab-Werte oder Dichte der Volltonfarben, Tonwertzuwächse, ICC-Zielprofile...)

Nur die ersten sechs Punkte in dieser Liste sind wirklich obligatorisch, die folgenden zwei sind zumindest weit verbreitet, die letzten beiden hingegen führen eine Sonderrolle. So werden die Positionen der Kontrollelemente in vielen Fällen sogar vom Drucker beziehungsweise von der Druckmaschine bestimmt und den Druckformherstellern vorgeschrieben. Denn die Inline- oder Offline-Messsysteme können (wenn sie überhaupt vorhanden sind) die Kontrollelemente nur an gewissen Stellen lesen und interpretieren. Erst wenige moderne Maschinen können auch umgekehrt die Positionen der Kontrollelemente als Metadaten von der Druckformherstellung übernehmen und entsprechend ihre Messinstrumente positionieren. Auch die Festlegung der Druckbedingungen geht in der Regel vom Drucksaal aus, zumindest in der in dieser Beziehung standardisierten Welt des Offsets.

In Punkt 6 haben wir auch die Terminvorgaben zur Auslieferung der fertigen Produkte mit aufgenommen, die in der Regel mit Auftragseingang feststehen. Diese müssen allerdings noch von Disponenten in Schichtenplanung für einzelne Arbeitsgänge in der Produktion umgesetzt werden. Wir möchten die drei folgenden Möglichkeiten ansprechen:

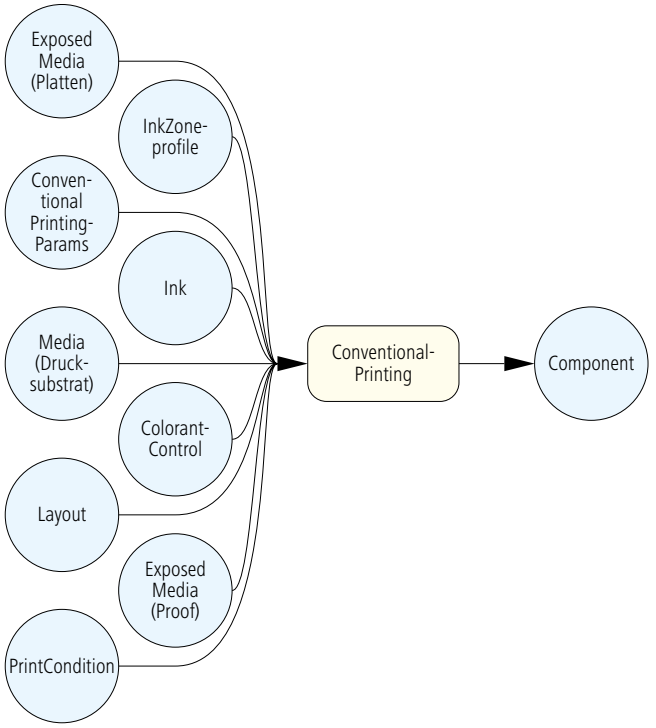
- Die Disposition erfolgt nicht auf der Basis von JDF, sondern

beispielsweise analog mit einer Plantafel oder aber mit einer Software, die über ein herstellerspezifisches Protokoll Terminlisten an die Maschinenleitstände sendet.
- Die Planung der Auftragsreihenfolge wird in einem MIS-Modul vorgenommen und der Auftragsplaner schickt die technischen und administrativen JDF-Auftragsdaten unter MIS-Kontrolle direkt zum Druckmaschinenleitstand.
- Eine der Produktion zugeordnete Dispositionsabteilung erhält die Eckdaten vom MIS, die zur Maschineneinsatzplanung nötig sind.

Der erste Fall ist für unser Thema nicht von Interesse. Im zweiten Fall schickt das MIS die JDF-Auftragsdaten geradewegs in richtiger Reihenfolge an den Leitstand der Druckmaschine, während im dritten Fall das JDF zunächst an die Dispositionssoftware der Produktion geht. Das JDF vom MIS hat also unterschiedliche Adressaten und auch unterschiedliche Inhalte.

In Abbildung 3.12 haben wir bereits die wesentlichen Inputs und den Output beim Offsetdruck vorgestellt. In Abbildung 10.1 zeigen wir noch einmal das gleiche Bild, allerdings nun mit den Namen der JDF-Ressourcen, die wir ja jetzt zum größten Teil in den letzten Kapiteln kennen gelernt haben. Einige mögliche JDF-Input-Ressourcen haben wir jedoch nicht mit aufgeführt, weil sie nur in recht speziellen Situationen benötigt werden. Werden beispielsweise vorgedruckte Papiere verarbeitet, so wird eine *Component*-Ressource an Input benötigt. Oder soll im Flexodruck das doppelseitige Klebeband spezifiziert werden, das für die Montage der Flexodruckplatte auf den Zylinder oder Sleeve benötigt wird, so wird eine weitere *Media*-Ressource vom Typ *MountingTape* benötigt. Andere Inputs, wie die *Preview*-Ressource, sind durchaus weit verbreitet, um die Voransicht des Bogens am Leitstand der Druckmaschine darstellen zu können. Wir werden im Rest

Abbildung 10.1
der Offsetdruck mit seinen Ressourcen



dieses Abschnittes die verschiedenen Ressourcen des Prozesses *ConventionalPrinting* näher beleuchten, auch zum Teil diejenigen, die nicht in Abbildung 10.1 verzeichnet sind.

### Druckparameter und Druckbedingungen

Im JDF-Modell wird zwischen Druckparameter (*ConventionalPrintingParams*) und Druckbedingungen (*PrintCondition*) unterschieden. In der obligatorischen Ressource *ConventionalPrintingParams* sind einige grundsätzliche Parameter für das Einrichten der Druckmaschine hinterlegt. Dabei kann die Art der Trocknung (UV, IR, Heatset) angegeben, das Deaktivieren des Feuchtwerks für den wasserlosen Offset festgelegt oder die maximale Geschwindigkeit in Bogen pro Stunde beim Bogenoffset beziehungsweise Umdrehungen pro Stunde beim Rollenoffset bestimmt werden. Sogar das Druckverfahren (Offset-, Flexo-, Tief- oder Siebdruck) und der Druckmaschinentyp (Bogenmaschine, Rolle mit einer Platte oder mehreren Platten pro Plattenzylinder...) werden spezifiziert. Diese sehr allgemeinen Angaben werden in der Regel nach einem festen Hausstandard eingestellt oder werden implizit dadurch angegeben, dass ein Job einer bestimmten Maschine zugeordnet wird. Insofern werden die Attribute in der Realität keine große Rolle spielen. Der Nutzen ist eher im Zusammenhang mit der Abfrage von Geräten über die Fähigkeiten der zugehörigen Maschine zu sehen. Die sogenannten *DeviceCababilities* werden am Ende von kapitel 11.2 vorgestellt. Es gibt aber auch auftragsbezogene Parameter für eine Druckmaschine, die in dieser Ressource abgelegt sind, wie beispielsweise der *WorkStyle*, in dem die Art des Vorder- und Rückseitendrucks angegeben ist. In Abbildung 10.2 erkennt man, dass *WorkAndTurn* als

Abbildung 10.2
ConventionalPrintingParams-Ressource

```
<ConventionalPrintingParams Class="Parameter" ID="_400"
 PrintingType="SheetFed" Status="Available" WorkStyle="WorkAndTurn" />
```

Wert eingetragen ist, was so viel wie „zum Umschlagen“ heißt (im Widerdruck bleibt die Vorderanlage wie beim Schöndruck, während die Seitenanlage wechselt; kein Plattenwechsel für den Widerdruck). In der Ressource könnte auch geschrieben sein, dass ein Kunde oder auch ein interner Qualitätsbeauftragter die Freigabe zum Druck an der Druckmaschine erteilen möchte. Der Status des Prozesses *ConventionalPrinting* steht dann so lange auf *Stopped* und die *StatusDetails* auf *WaitForApproval* bis das geschehen ist.

```
<PrintCondition Name=" Papiertyp_1_ISO_12647-2" Class="Parameter" ID="_4711"
 PartIDKeys="Side Separation" Status="Available">
   <PrintCondition Side="Front">
      <PrintCondition Separation="Cyan"
       AimCurve="0.0 0.0 0.4 0.56 1.0 1.0" />
      <PrintCondition Separation="Magenta"
       AimCurve="0.0 0.0 0.4 0.56 1.0 1.0" />
      <PrintCondition Separation="Yellow"
       AimCurve="0.0 0.0 0.4 0.56 1.0 1.0" />
      <PrintCondition Separation="Black"
       AimCurve="0.0 0.0 0.4 0.53 1.0 1.0" />
   </PrintCondition>
</PrintCondition>
```

Abbildung 10.3
SOLL-Kurven für die Tonwertzunahmen im Druck

Während die Ressource *ConventionalPrintingParams* also sehr grundlegende Werte für das Grundrüsten einer Druckmaschine bereithält, geht es bei der *PrintCondition* um Sollwerte von Farbe, Dichte und Tonwertzunahmen für den Druck. Auch die spektralfotometrische Farbmessung könnte man hier spezifizieren (Farbtemperatur, Filter, 2° oder 10° Beobachtungswinkel, Messung auf nasser oder trockener Farbe, Messunterlage,...). In der Praxis ist aber auch diese Ressource bisher nur wenig im Einsatz, da zumindest im Offsetdruck in der Regel nach dem Standard ISO 12647 [24] gearbeitet wird und insofern keine auftragsbezogene Festlegung nötig ist. Abbildung 10.3 zeigt aber ein Beispiel, bei dem die Sollkurve für die Tonwertzunahme eingetragen wurde. Die Werte sind analog zu Abbildung 4.11 zu interpretieren.

**Farbzonenvoreinstellung beim Offset**

Das Farbangebot an den Duktor wird bei einer Offsetmaschine klassischerweise dem Drucksujet angepasst (Ausnahme: Anilox-Farbwerke, auch Kurzfarbwerk genannt). Hierzu werden einzelne Farbzonen über die Druckbreite festgelegt. In jeder dieser Zone drückt eine Zonenschraube ein Farbmesser oder einen Farbschieber an den Farbduktor, so dass nur eine definierte Farbmenge fließen kann. Der Farbdosierspalt wird in der Regel elektronisch gesteuert. Die Farbzonenvoreinstellung ist Teil des Einrichtens einer Form.

Die Berechnung der Farbzonenvoreinstellungswerte mittels Vorstufendaten ist schon beim Portable Print Format (PPF) die am weitesten verbreitete Anwendung, wie wir bereits in Abschnitt 4.2 erläutert haben. In der Tat ist sie so üblich, dass vielfach auch weiterhin PPF verwendet wird und nicht JDF. Dabei muss man zugeben, dass die Funktionalität durch JDF nicht höher wird. Insofern bietet der Umstieg von einem Datei-Format zum

anderen zumindest dem Anwender keine Vorteile. Insgesamt ist aber eine JDF-Implementierung natürlich strukturell im Vorteil, weil diese Werte dann mit allen anderen Angaben in eine JDF-Datei eingepackt und weitergeleitet werden können und keine spezielle Behandlung benötigen. Insbesondere entfällt dann die Notwendigkeit, einen PostScript-Interpreter zum Interpretieren der PPF-Datei zu lizenzieren.

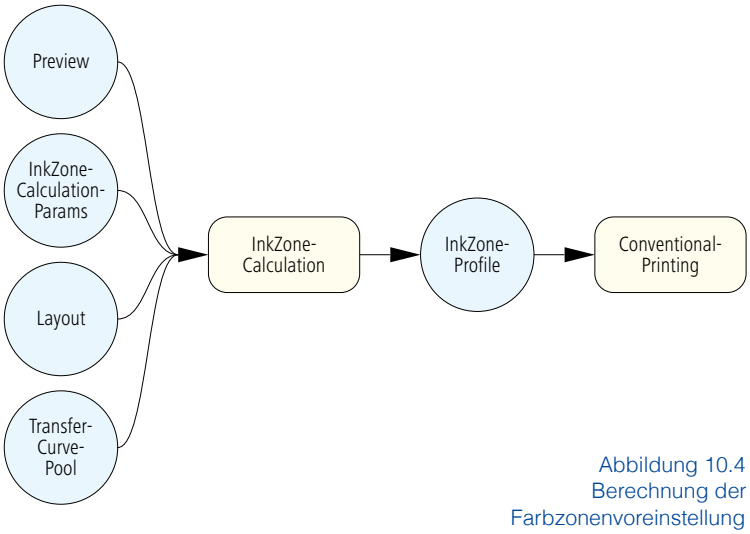



Abbildung 10.4
Berechnung der Farbzonenvoreinstellung beim Offset

Die Berechnung der Farbzonenvoreinstellung ist als JDF-Modell in Abbildung 10.4 zu sehen. Die *Preview*-Ressource ermöglicht den Zugriff auf ein Voransichtsbild (üblicherweise nur um die 50 ppi) des gesamten Bogens. Denn, etwas vereinfacht gesprochen, müssen ja bei der Berechnung der Farbzonenvoreinstellung einfach nur die Pixel pro Separation und Zone aus dem Vorschaubild ausgezählt werden. In *InkZoneCalculationParams* werden die einzelnen Zonen definiert, die bei verschiedenen Druckmaschinentypen unterschiedliche Breiten haben können. Im Beispiel in 10.5 ist die Zonenbreite 3,25 cm (92.125984 / 72 x 2,54 cm) und es gibt insgesamt 23 Zonen parallel zur Zylinderachse. Dass es in Papierlaufrichtung nur eine Zone gibt und deren Höhe gleich dem maximalen Druckformat in Abwicklungsrichtung entspricht, ist formal korrekt, obwohl der Eintrag an sich schon seltsam anmutet. Die Zonenpositionen sind druckmaschinespezifisch, das Vorschaubild zeigt jedoch nur

Abbildung 10.5
Farbzonenparameter

```
<InkZoneCalculationParams Class="Parameter" ID="_25" Status="Available"
 ZoneHeight="1451.338583" ZoneWidth="92.125984" Zones="23" ZonesY="1" />
```

den Bogen. Um nun die Farbzoneneinstellung berechnen zu können, muss natürlich die Position des Bogens bezüglich der Zonen angegeben werden. Dies geschieht am einfachsten dadurch, dass die Position des Bogens auf der Platte während des Druckes definiert wird. Diesen Eintrag findet man in dem Attri-

but *SurfaceContentsBox* der in diesem Buch bereits mehrmals aufgetauchten *Layout*-Ressource. Und schließlich werden auch noch mögliche Transferkurven benötigt, um die Tonwertveränderungen beim RIPing der Belichtungsdaten auch beim Voransichtsbild vornehmen zu können (vergleiche Abschnitt 4.2).

Im JDF werden die Farbzonenvoreinstellwerte in der Ressource *InkZoneProfile* abgelegt. Natürlich werden diese Einstellungen nicht als Öffnung des Farbdosierpalts in Mikrometern angegeben, weil das viel zu maschinenspezifisch wäre. Stattdessen ist jeder der Voreinstellungswerten eine Zahl zwischen 0 und 1, wobei diese die durchschnittliche Flächendeckung in einer Zone darstellt. Diesen Faktor muß man, um einen Prozentwert zu erhalten, allerdings noch mit Hundert multiplizieren. Hieraus die Öffnung des Farbdosierspalts zu errechnen geschieht mit Hilfe einer Anpassungskurve, die an der Druckmaschine einmal für bestimmte Druckbedingungen (Farbe, Papier) ermittelt wird und nicht mehr Teil der JDF-Beschreibung ist. Bei unverändertem Druckbild, also gleichen Farbvoreinstellwerten, muss nämlich das Farbangebot an ein Naturpapier höher sein als an ein Bilderdruckpapier. Diese Unterschiede werden an der Druckmaschine einmal ermittelt, in einer Datei abgespeichert und für den jeweiligen Druckauftrag herangezogen. So

Abbildung 10.6
Farbzonenvoreinstellwerte

```
<InkZoneProfile Class="Parameter" ID="_3" PartIDKeys="SignatureName SheetName
 Side Separation"...>
   <InkZoneProfile SignatureName="S1">
      <InkZoneProfile SheetName="Umschlag">
         <InkZoneProfile Side="Front">
            <InkZoneProfile Separation="Black" Status="Available"
             ZoneWidth="92.125984" ZoneSettingsX="0.004947916666669755
             0.008625565610862806 0.040955175339369494
             0.05258389894419598 0.036772011689294073
             0.04679616798642827 0.03305889423077225
             0.04669306184012362 0.023023897058826546
             0.03196920955882654 0.026367305335598803
             0.022763303638766233 0.01726845305430168
             0.027979296285825045 0.03681620003771036
             0.0802906414969863 0.07557833710407524
             0.07194204845399976 0.05891538225867557
             0.08456872171945984 0.08174302413273286
             0.029526241987182487 0.009106099170440482"
             ZoneHeight="1451.338583"
             ZoneSettingsY="0.04123003711309234" />
            <InkZoneProfile ... />
          ...
         </InkZoneProfile>
      </InkZoneProfile>
   </InkZoneProfile>
</InkZoneProfile>
```

kann das Farbwerk, ohne dass auch nur ein Bogen gedruckt worden wäre, bereits im Voraus auf den Bedruckstoff eingestellt werden. Damit die Steuerungskomponente immer die richtigen Daten für die entsprechenden Druckbedingungen verwendet, sollte sie eben auch auf diese Information aus der JDF-Auftragsbeschreibung zugreifen. Hat sie falsche Werte (weil sie nicht auftragsspezifisch angepasst wurde) oder gar keine, wird dies dazu führen, dass mehrere Regelzyklen (Abzug machen, Bogen ziehen, messen/beurteilen, nachjustieren, neuen Abzug machen etc.) durchgeführt werden müssen, was wiederum mehr Zeit und Einrichtebogen erfordert.

In Abbildung 10.6 ist ein Beispiel-Code für ein *InkZoneProfile* zu sehen. Die Ressource ist bis hin zur Separation partitioniert, denn die Farbzonenvoreinstellwerte sind in der Regel für jede Farbe verschieden. In dem Beispiel sind nur die Werte für die Farbe Schwarz angegeben. Die 23 Werte in dem Attribut *ZoneSettingX* geben die Flächendeckungswerte der einzelnen Zonen an, also von links beginnend und auf zwei Stellen hinter dem Komma beschränkt: 0,49%, 0,86%, 4,09%, 5,25% und so weiter. Das *ZoneSettingY* enthält den Durchschnittwert der Flächendeckungen aller Zonen, hier 4,12%.

### Kontrollmarken

Typischerweise werden mehrere Kontrollelemente auf Druckformen belichtet. Sie lassen sich der Funktion nach untergliedern in:

- Kontrollelemente für die Druckformherstellung,
- Kontroll- und Steuerelemente für den Druck und
- Kontroll- und Steuerelemente für die Weiterverarbeitung.

In diesem Kapitel sind nur die Kontrollelemente für den Druck relevant. Die wichtigsten sind

- Passer- bzw. Registermarken,
- Druckkontrollstreifen,
- Seiten- oder Anlagemarke, auch Schiebe- oder Ziehmarke genannt und
- Druckbogensignaturen.

Natürlich sollten Marken in bestimmten Bereichen platziert werden. Im Bogendruck wird die Seitenmarke genau auf den Bogenrand gesetzt, der Druckkontrollstreifen parallel zu den Zylindern entweder in der Mitte oder auch am Ende des Bogens und so weiter. Solange jedoch die Marken mitgedruckt werden, um anschließend von einem Mitarbeiter optisch überprüft oder auch mit einem Handmessgerät ausgemessen zu werden, ist die genaue Position für den Überprüfungsvorgang

```
<Layout Class="Parameter" DescriptiveName="Lay" ID="_111" Name="Lay"
 PartIDKeys="SignatureName SheetName Side" Status="Available">
   <Layout Name="SIG1" SignatureName="SIG1">
      <Layout DescriptiveName="Blätter" Name="Blätter" SheetName="Blätter"
       SourceWorkStyle="WorkAndTumble" Status="Available"
       SurfaceContentsBox="0 0 2111.81102362 1714.96062992">
         <Media Class="Consumable" Dimension="2111.81102362 1714.96062992"
          ID="_3062" MediaType="Plate" Status="Available" />
         <Layout DescriptiveName="Schöndruck" Side="Front"
          Status="Available"
          SurfaceContentsBox="0 0 2111.81102362 1714.96062992">
         ...
            <MarkObject CTM="1 0 0 1 0 0"
             ClipBox="0 0 2111.81102362 1714.96062992" Ord="0">
               <RegisterMark Center="2028.42519684 659.11417322"
                Class="Parameter" MarkType="8AR_R_106x164" Rotation="0" />
               <RegisterMark Center="83.38582677 659.11417322"
                Class="Parameter" MarkType="8AR_L_106x164" Rotation="0" />
               <ColorControlStrip Center="1059.44881889 809.11811023"
                Class="Parameter" Rotation="0" Size="1798 28"
                StripType="FOGRA_5_F74_740x10" />
         </MarkObject>
      </Layout>
   </Layout>
</Layout>
```

Abbildung 10.7
Marken-Objekt in einer Layout-Ressource

unwichtig. Erst wenn automatische Inline- oder auch automatisch geregelte Offline-Messsysteme eingesetzt werden, wird das Wissen über die exakte Platzierung der Kontrollelemente wichtig, damit die Sensoren sich auf diese überhaupt einstellen können. Und solche Messsysteme gibt es immer mehr: Inline-Messsysteme für den Farbauftrag, gemessen in Dichte, gibt es für den Offset-, den Flexo- und auch für den Tiefdruck. Zumindest im Offset gibt es auch Inline-Kontrollsysteme auf spektralfotometrischer Basis. Auch mitgedruckte Passermarken werden vielfach inline von CCD-Kameras gelesen und ausgewertet mit dem Zweck, die Umfangs-, Seiten- und Diagonalregister automatisch nachzuregeln.

Ein Vorstufenmitarbeiter legt die Position der Kontrollelemente auf dem Standbogen im Montageprogramm fest. Bei „dynamischen“ Marken, die wir in Abschnitt 9.2 behandelt haben, wird die exakte Platzierung aber erst durch Software berechnet. Die Markenpositionen stellen also ein (weiteres) typisches Beispiel dar, bei dem Informationen abteilungsübergreifend weitergereicht werden müssen. Diese Aufgabe kann natürlich JDF übernehmen, und wir sehen in Abbildung 10.7 mehrere Res-

```
<ExposedMedia Class="Handling" ID="_123" PartIDKeys="SignatureName SheetName
 Side Separation"...>
   <ExposedMedia SignatureName="Signatur_1">
      <ExposedMedia SheetName="Umschlag" Status="Available">
         <ExposedMedia Side="Front" Status="Available">
            <ExposedMedia Separation="Cyan" Status="Available" />
            <ExposedMedia Separation="Magenta" Status="Available" />
            <ExposedMedia Separation="Yellow" Status="Available" />
            <ExposedMedia Separation="Black" Status="Available" />
      </ExposedMedia>
         <ExposedMedia Side="Back" Status="Available">
            <ExposedMedia Separation="Cyan" Status="Available" />
             ...
         </ExposedMedia>
      <MediaRef rRef="_1234">
         <Part SheetName="Umschlag" SignatureName="Signatur_1" />
      </MediaRef>
   </ExposedMedia>
</ExposedMedia>

<Media Brand="745x605_Azura" Class="Consumable" Dimension="2111.811
 1714.961" ID="_1234" MediaType="Plate" PartIDKeys="SignatureName SheetName"
 Status="Available">
   <Media Class="Consumable" Dimension="2111.81102362 1714.96062992"
    SignatureName="Signatur_1" Status="Available">
      <Media Brand="745x605_Azura" Class="Consumable"
       Dimension="2111.811 1714.961" MediaType="Plate" SheetName="Umschlag"
       Status="Available" />
   </Media>
</Media>
```

Abbildung 10.8
Platteninformation für den Druckprozess

sourcen vom Typ *MarkObject*, die in einer *Layout*-Ressource eingebettet ist.

### Druckformen und Bedruckstoff

Druckformen (Platten, Tiefdruckzylinder, Sleeves oder Siebe) gibt es in zwei Zuständen: Druckformrohlinge und fertige Druckformen, welche die Druckbildinformationen enthalten. Im Offset kann man etwas verkürzt auch von unbelichteten und belichteten Platten sprechen, wobei der Begriff Bebildern richtiger wäre, da längst nicht alle Platten im sichtbaren Spektrum empfindlich sind. Die mögliche Plattenentwicklung wird hier einfach dem Belichtungsprozess zugeschlagen. Unbelichtete Platten werden in der JDF-Ressource *Media* beschrieben, wobei das Attribut *MediaType* gleich *Plate* gesetzt werden muss. Bedruckstoffe werden mit der gleichen Ressource definiert. Unter *MediaType* muss dann die Art des Bedruckstoffs angegeben werden, wie *Paper* (Papier), *Foil* (Folie) oder *CorrugatedBoard* (Wellpappe).

Für den Druckprozess werden jedoch belichtete Platten benötigt (wenn man einmal von unbeschichteten Blindplatten im Zeitungsdruck absieht). Diese werden in der *ExposedMedia*-Ressource aufgeführt. In der Praxis verwenden die Druckereien nur einen Plattentyp und auch die Größe der Platten ist fest für jede Druckmaschine vorgeschrieben. Insofern interessiert die Drucker eigentlich nur, ob die belichteten Platten für einen Auftrag verfügbar (*available*) sind oder nicht. Es gibt auch Automatis-

Abbildung 10.9
Bedruckstoffinformationen

```
<Media Class="Consumable" ID="_111" MediaType="Paper"
 PartIDKeys="SignatureName SheetName" Status="Available"...>
   <Media Class="Consumable" SignatureName="Signature_1" Status="Available">
      <Media SheetName="Umschlag" Class="Consumable" ProductID="_999"
       Status="Available" Dimension="1729.1338582677165 1218.8976377952756"
       Grade="1" GrainDirection="LongEdge" Thickness="215.0" Weight="215.0"
       FrontCoatings="Glossy" BackCoatings="Matte" Brand="Chromolux" />
   </Media>
</Media>
```

men, die den Auftrag automatisch an die Druckmaschine weiterleiten, sobald alle Platten eines Bogens beziehungsweise einer Form verfügbar sind. In Abbildung 10.8 ist genau diese Information zu sehen. Man erkennt auch, dass es eine Referenz auf die Ressource *Media* gibt, in die der Plattentyp und die Plattengröße ausgezeichnet sind.

Im JDF-Code in Abbildung 10.9 erfahren wir in der partitionierten Ressource *Media* einiges über den Bedruckstoff für den Umschlag:

- Bogengröße ist 61 x 43 cm (*Dimension*)
- Papierklasse 1 gemäß ISO 12647-2:2004 (*Grade*)
- Laufrichtung parallel zur langen Papierkante (*GrainDirection*)
- Papierdicke von 215 Mikrometer (*Thickness*)
- Papiergewicht von 215 g/m² (*Weight*)
- Vorderseite glänzend gestrichen (*FrontCoatings*)
- Rückseite matt gestrichen (*BackCoatings*)
- Marke Cromolux (*Brand*)

Aber auch weitere Attribute könnten in diese Ressource geschrieben werden, wie der Lab-Wert des Papiers oder der Grad der Opazität. Leider werden diese Werte aber (nur selten oder) nie vom Papierlieferanten geliefert (beispielsweise in der Papierpreisliste).

```
<ColorPool Class="Parameter" ID="_500" Status="Available">
    <Color CMYK="1.0 0.0 0.0 0.0" ColorType="Normal" Name="Cyan"
     NeutralDensity="0.61" />
    <Color CMYK="0.0 0.0 0.0 1.0" ColorType="Normal" Name="Black"
     NeutralDensity="1.7" />
    <Color CMYK="0.0 1.0 0.0 0.0" ColorType="Normal" Name="Magenta"
     NeutralDensity="0.76" />
    <Color CMYK="0.0 0.0 1.0 0.0" ColorType="Normal" Name="Yellow"
     NeutralDensity="0.16" />
    <Color CMYK="1.0 0.0 1.0 0.0" ColorType="DieLine" Name="ProofColor" />
    <Color CMYK="0.2 0.3 0.4 0.5" ColorType="Normal" Name="PANTONEDeepBlue
     Lab="20. 30. 40.">
</ColorPool>
```

Abbildung 10.10
ColorPool-Ressource

### Farben und Farbkontrolle

Unter dem Terminus „Farbe“ können unterschiedliche Angaben gemacht werden: Die Festlegung der Farbigkeit pro Bogenseite, die Farbreihenfolge in der Druckmaschine, die farbmetrischen Eigenschaften von Druckfarben, die Farbnamen in den Content-Daten, das Farbmodell des Ausgabegeräts und so weiter. Und auch in JDF werden diese Informationen in unterschiedlichen Ressourcen abgelegt.

In der *Ink*-Resource sind die Druckfarben aufgelistet. Im einfachsten Fall sind das nur die Prozessfarben, wie in Abbildung 8.15 zu sehen. Dort ist der Schöndruck CMYK und der Widerdruck nur einfarbig Schwarz. Selbstverständlich können auch Sonderfarben oder Lacke angegeben werden. Die Separationsnamen sollten aber übereinstimmen mit den Farbnamen in der *ColorPool*-Ressource. Dort sind alle Farben aufgelistet, die in dem Auftrag verwendet werden, also unter Umständen auch solche, die nicht auf der Druckmaschine gedruckt werden. Hier werden die Farben auch genauer spezifiziert, wie zum Beispiel nach dem Farbentyp (*ColorType*), der angibt, ob es sich um deckende, transparente oder lasierende (*Normal*) Farben handelt

Abbildung 10.11
ColorantControl-Ressource

```
<ColorantControl Class="Parameter" ID="_2201" PartIDKeys="SignatureName
SheetName Side" ProcessColorModel="DeviceCMYK" Status="Available">
    <ColorPoolRef rRef="_500" />
    <DeviceColorantOrder>
        <SeparationSpec Name="Black" />
        <SeparationSpec Name="Cyan" />
        <SeparationSpec Name="Magenta" />
        <SeparationSpec Name="Yellow" />
        <SeparationSpec Name="PANTONEDeepBlue" />
    </DeviceColorantOrder>
     ...
</ColorantControl>
```

oder auch die neutrale Dichte der Farben (*NeutralDensity*). Wir hatten bereits im Kapitel über die Vorstufe gesehen, dass solche Farbeigenschaften wichtig für den Trapping-Prozess sind. Auch Lab-Werte der Farben oder die CMYK-Repräsentierung der Farben – auch der Sonderfarben – können hier eingetragen werden. Abbildung 10.10 ist ein einfaches Beispiel für eine *ColorPool*-Resssource. Der CMYK-Wert einer Farbe entspricht den (Näherungs-) Tonwerten von CMYK in Prozent, allerdings normiert auf 1 anstatt auf 100, wie sonst üblich. Der *ColorType* ist in dem Beispiel bis auf eine Ausnahme immer *Normal*, also lasierend. Die Ausnahmefarbe wird nur zum Auslinieren des Bogenproofs verwandt. Insbesondere darf also diese Farbe nicht zu den anderen Farben getrappt werden, keine entsprechende Platte belichtet und natürlich nicht auf der Druckmaschine gedruckt werden.

Als letzte Ressource, die mit Farbdefinitionen zu tun hat, wollen wir hier noch einmal die *ColorantControl*-Ressource aufführen (wie auch schon in 6.4 mit Abbildung 6.18, 8.2 mit Abbildung 8.12 und 9.1 mit den Abbildungen 9.8 und 9.13). Diese enthält alle Informationen, die benötigt werden, die Farben der Content-Daten auf die Farben des Ausgabegeräts abzubilden. Diese Information ist natürlich für eine Druckmaschine völlig unerheblich. Dennoch wird auch diese Ressource im Druckprozess benötigt, weil ein Unterelement, das *DeviceColorantOrder* heißt, die Farbreihenfolge in der Druckmaschine angibt, also typischerweise Schwarz, Cyan, Magenta und dann Gelb im Bogenoffset (Abbildung 10.11).

### Proof und Preview

In aller Regel erhält der Drucker einen Proof, meist einen Formproof, der nicht farbverbindlich auf einem Tintenstrahldrucker ausgegeben wurde. Zusätzlich wird in einer vernetzten Konfiguration auch ein kleines Vorschaubild des Bogens am Leitstand der Druckmaschine angezeigt, sobald ein Auftrag ausgewählt wird. In manchen Fällen (wie bei der Druckerei Y in Kapitel 2.2) werden auch farbverbindliche Softproofs an den Leitstand gereicht, der natürlich mit entsprechendem Normlicht und kalibrierbaren Monitoren ausgestattet sein muss. Auch farbverbindliche Seitenproofs, farbverbindliche Bogenproofs und bereits früher gedruckte Muster werden für einen Abstimmungsprozess herangezogen. Sowohl Proofs als auch Voransichtsbilder können natürlich mittels Ressourcen im JDF beschrieben werden. Ein Hardcopy-Proof wird in der Regel durch eine *Component*, ein Voransichtsbild oder ein Softproof durch eine *RunList* repräsentiert, wie bereits in Abschnitt 9.5 ausgeführt wurde. In manchen Fällen (vor allem im Verpackungsbereich) werden Hardcopy-Proofs auch mittels

```
<Component Class="Quantity" ComponentType="Sheet" ID="_123"
 PartIDKeys="SignatureName SheetName Condition" Status="Unavailable">
    <Component SignatureName="Signature_1">
        <Component SheetName="Umschlag" />
    </Component>
</Component>

<ComponentLink Amount="1" Usage="Output" rRef="_012">
    <AmountPool>
        <PartAmount Amount="2040.0">
            <Part Condition="Good" SheetName=" Umschlag "
             SignatureName=" Signature_1" />
        </PartAmount>
        <PartAmount Amount="364.0">
            <Part Condition="Waste" SheetName=" Umschlag "
             SignatureName=" Signature_1" />
        </PartAmount>
    </AmountPool>
    <Part SheetName=" Umschlag " SignatureName=" Signature_1" />
</ComponentLink>
```

Tabell 10.12 Bestandteile der Druckbogen

Plattenbelichtern oder speziellen Proofgeräten hergestellt, die gerasterte Bitmap-Daten verarbeiten können (Rasterproofs). In solchen Fällen werden die Proofs durch spezielle Varianten der *ExposedMedia*-Ressource repräsentiert.

### Component

Das Ergebnis des Druckprozesses sind Druckbogen, ein Zwischenprodukt des Gesamtprozesses. Übersetzt in die JDF-Sprache heißt das, der Output des *ConventionalPrinting*-Prozesses ist eine *Component* mit *ComponentType* gleich *Sheet*. Doch diese Komponente wird nicht nur bis zum Druckbogen partitioniert, sondern darüber hinaus wird sie auch noch nach Zuständen (*Condition*) unterteilt, wie in Abbildung 10.12 zu sehen ist. In der Referenz auf die *Component*, also im *ComponentLink*, wird diese Unterscheidung erst deutlich: Die Druckbogen werden in Makulatur (*Waste*) und Gutbogen (*Good*) unterschieden. Und für jeden der beiden ist auch eine Sollbogenanzahl angegeben. Bei 2040 Gutbogen sind 364 Makulaturbogen eingeplant.

Allgemein gilt, dass ein *AmountPool* gewissermaßen einen Zähler von Ressourcen darstellt, die von einem Prozess entweder verbraucht oder erzeugt werden. Genau genommen ist aber ein *AmountPool* nur eine Art von Notizzettel, auf den ein Ressourcenzähler seine Werte schreiben kann.

JDF-Implementierungen findet man in Offsetdruckereien, und dort meist im Bogenoffset. Im Rollenoffset, Tief-, Flexo- oder Siebdruck ist der Verbreitungsgrad gering und beschränkt sich auf MIS und Vorstufe (siehe Kapitel 12). Eine JDF-Integration von Tief-, Flexo- oder Siebdruckmaschinen ist uns derzeit nicht bekannt. Zwar umfasst der *ConventionalPrinting*-Prozess auch diese Druckverfahren, aber die entsprechenden Druckmaschinenhersteller scheinen bisher recht wenig Interesse an dem Thema zu haben. Viele von ihnen sind nicht einmal Mitglieder der CIP4-Organisation. Das könnte mehrere Gründe haben:

- Die ursprünglichen Initiatoren von JDF sind Hersteller von Offsetmaschinen und die Anforderungen der anderen Drucktechnologien wurden weniger und auch erst später berücksichtigt;
- im Tief- und Flexodruck sind Vorstufe und Druckereien häufig getrennte Unternehmen, was die Integration erschwert;
- im Offset hat die Weitergabe der Farbzonenvoreinstellungswerte einen besonders hohen Kosten-/Nutzungsgrad, den es so bei den anderen Drucktechnologien nicht gibt;
- Druckstandards gibt es im Wesentlichen nur für den Offset, während bei den anderen Technologien meist nach Hausstandards oder sogar nach speziellen Kundenstandards gedruckt wird;
- die Variabilität von Bedruckstoffen ist höher als beim Offset, was Automatismen erschwert;
- bei den Auflagenhöhen im Offset, die meist niedriger sind

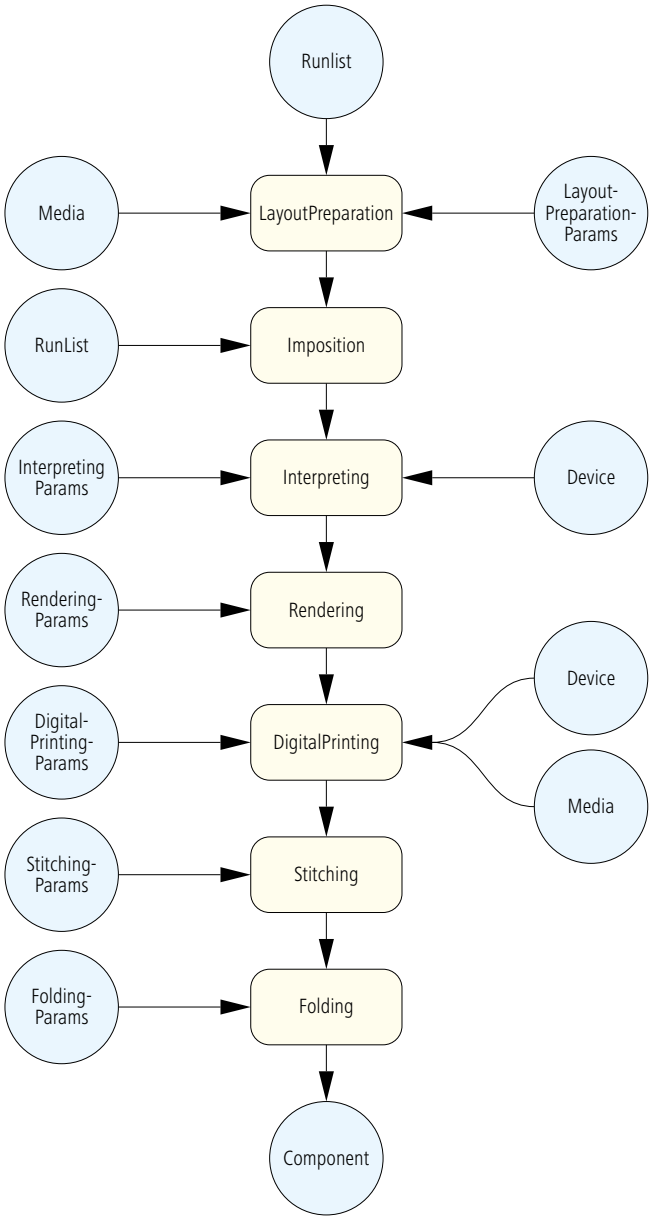
Tabell 10.13

| Nr. | Integrierte Drucksystene | Digitaldrucker in Büro / Copy-Shop |
|---|---|---|
| 1 | LayoutPreparation | LayoutPreparation |
| 2 | Imposition? | Imposition |
| 3 | ColorSpaceConversion? | |
| 4 | Interpreting | Interpreting |
| 5 | ColorSpaceConversion? | |
| 6 | Rendering | Rendering |
| 7 | ColorSpaceConversion? | |
| 8 | Screening? | |
| 9 | Imposition? | |
| 10 | DigitalPrinting | Digitalprinting? |
| 11 | Folding?, Stitching?, Trimming?, HoleMaking?, CoverApplication?, SpineTaping? | Folding? |
| 12 | | Stitching? |

als im Tief- oder Flexodruck, fallen die Einrichtezeiten mehr ins Gewicht;
- besonders im Tiefdruckbereich gibt es nur sehr wenige große Druckereien, die ihre individuellen Auftragssteuerungsysteme haben.

Trotzdem ist vorstellbar, dass sich diese Situation in näherer Zukunft ändern könnte. Denn zumindest die JMF-Nachrichten über Materialverbrauch und über die aktuellen Zustände der Druckmaschinen und Druckaufträge sind natürlich in jedem Fall für eine geregelte Produktion von Interesse.

Abbildung 10.14
Praxisbeispiel Digitaldruck




## 10.2 Digitaldruck

Der Digitaldruck hat bekanntermaßen seine Stärken vor allem bei niedrigen Auflagen, bei kurzen Produktionszeiten und beim personalisierten Drucken. Damit ist eine möglichst durchgängige Automatisierung eine wichtige Voraussetzung für die Wirtschaftlichkeit dieser Drucktechnologie und JDF folglich ein potentiell zweckdienliches Datenformat.

Man kann den Digitaldruck in zwei Anwendungsbereiche unterteilen, deren Abgrenzung aber nicht eindeutig, sondern eher fließend ist:

- Büroumgebung und Copy-Shop,
- professioneller Druck mit Digitaldrucksystemen.

Digitaldrucksysteme unterscheiden sich von Copy-Shop Druckern vor allem durch mehr

Ausschießoptionen, höhere Farbtreue (Farbmanagement) und erweiterte Inline-Weiterverarbeitungsoptionen. Diese Kategorisierung spiegelt sich auch in den entsprechenden zwei ICS-Papieren wieder, die *Office Digital Printing ICS* und *Integrated Digital Printing (IDF) ICS* heißen. JDF-vernetzte, integrierte Digitaldrucksysteme sind auch häufiger Teil eines „Hybrid-Workflows", bei dem die Daten unter einem WMS sowohl für den Offset als auch für den Digitaldruck aufbereitet werden. Aber nicht nur die Offset-Formherstellung, auch die Zylindergravur im Tiefdruck kann JDF-unterstützt stärker automatisiert werden. Denn auch in diesem Bereich gibt es Implementierungen, bei denen JDF-Daten, die ein MIS exportiert, von dem Gravur-System übernommen und Produktionsmeldungen von der Formherstellung an das MIS zurückgemeldet werden.

Im Vergleich zum Bogenoffset unterscheidet sich der Digitaldruck in prozesstechnischer Hinsicht vor allem in zwei Punkten:

- Im Digitaldruck wird keine physische Druckform hergestellt – der JDF-

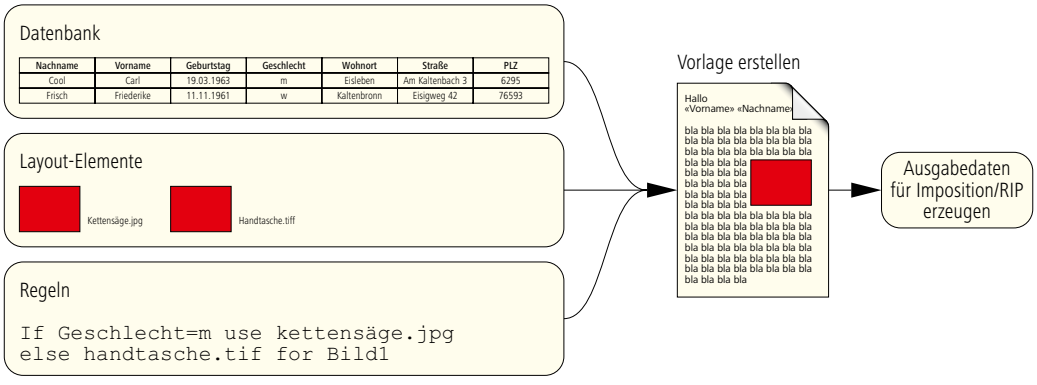
Abbildung 10.15
Praxisbeispiel Digitaldruck

```
<?xml version="1.0" encoding="UTF-8"?>
<JDF Status="Ready" Type="Combined" Types="LayoutPreparation Imposition
 Interpreting Rendering DigitalPrinting Stitching Folding" Version="1.3" …>
    …
   <ResourceLinkPool>
      <ComponentLink Amount="65" CombinedProcessIndex="5" Usage="Output"
       rRef="_2503">
         <Part SignatureName="Sig1" />
         <Part SignatureName="Sig2" />
         <Part SignatureName="Sig3" />
      </ComponentLink>
      <LayoutPreparationParamsLink CombinedProcessIndex="0" Usage="Input"
       rRef="_2504" />
      <MediaLink CombinedProcessIndex="0 4" Usage="Input" rRef="_0918" />
      <DigitalPrintingParamsLink CombinedProcessIndex="4" Usage="Input"
       rRef="_2507" />
      <DeviceLink CombinedProcessIndex="2 4" Usage="Input" rRef="_2509" />
      <RenderingParamsLink CombinedProcessIndex="3" Usage="Input"
       rRef="_2511" />
      <InterpretingParamsLink CombinedProcessIndex="2" Usage="Input"
       rRef="_2510" />
      <StitchingParamsLink CombinedProcessIndex="5" Usage="Input"
       rRef="_2513" />
      <StitchingParamsLink CombinedProcessIndex="6" Usage="Input"
       rRef="_2514" />
      <RunListLink CombinedProcessIndex="0 1" ProcessUsage="Document"
       Usage="Input" rRef="_9437" />
   </ResourceLinkPool>
</JDF>
```

Prozess *ImageSetting* ist unnötig.
- Eine Digitaldruckmaschine hat häufig Inline-Weiterverarbeitungskomponenten integriert.

Der zweite Punkt führt dazu, dass der Digitaldruck im JDF durch einen kombinierten Prozess repräsentiert wird, bei dem die Zwischenergebnisse der Einzelprozesse nicht beschrieben werden müssen. Die möglichen Prozesse sind der Reihenfolge gemäß den beiden ICS-Papieren in der Tabelle 10.13 wiedergegeben. Zunächst fällt bei den integrierten Drucksystemen auf, dass die beiden Prozesse *Imposition* und *ColorSpaceConversion* mehrfach aufgelistet und mit einem Fragezeichen als optional gekennzeichnet sind. Damit ist gemeint, dass beide Prozesse an unterschiedlichen Positionen der Abarbeitungssequenz stehen können, aber auch, dass sie immer an genau einer Position stehen müssen. Die Weiterverarbeitungsprozesse sind alle optional und können in mehr oder minder beliebigen Reihenfolgen durchgeführt werden. Die Digitaldrucker im Büro haben erwar-

Abbildung 10.16
Teilaufgaben beim personalisierten Drucken




tungsgemäß keine dedizierte Farbmanagementkontrolle und weniger Möglichkeiten in der Weiterverarbeitung.

Durch die Vielzahl der beteiligten Prozesse kann ein solcher kombinierter Prozess sehr viele Input-Ressourcen haben. Als Output des kombinierten Prozesses erhält man immer eine *Component*-Ressource. Auch Zwischenergebnisse der Prozesse sind Ressourcen vom Typ *Component* oder *RunList*, sofern sie überhaupt aufgeführt sind. In Abbildung 10.14 ist ein einfaches Beispiel zu sehen, das zwar nicht alle Möglichkeiten

wiedergibt, dafür aber aus der Praxis stammt. In Abbildung 10.15 ist der dazugehörige Code aufgelistet.

Der Prozess *LayoutPreparation* ist ähnlich wie der in Abschnitt 9.2 behandelte Prozess *Stripping*. Beide erzeugen nämlich als Output-Ressource ein *Layout*, also einen Standbogen, der zum Ausschießen (*Imposition*) benötigt wird. Beide haben auch Input-Ressourcen, durch die sie mit den Informationen versorgt werden, die sie für die Layout-Generierung benötigen. Für den *Stripping*-Prozess heißt die Input-Ressource *StrippingParams*, für den *LayoutGeneration*-Prozess wird sie mit *LayoutGenerationParams* bezeichnet. Der Unterschied der beiden Prozesse liegt in der Anwendung. Denn *LayoutPreparation* wird im Digitaldruck verwendet, während *Stripping* in der Offset-Druckvorstufe im Zusammenhang mit einem MIS. Möglicherweise wird es aber hier in einer nächsten JDF-Versionen Veränderungen geben, denn in der Version 1.4 ist bereits bei der Beschreibung der *LayoutPreparationParams*-Ressource vermerkt, dass diese in zukünftigen Versionen wieder aufgegeben werden könnte.

**Personalisierung**

Personalisiertes Drucken, auch *Variable-Data Printing* (VDP) genannt, bedeutet, dass einzelne Elemente (Text, Grafik, Bild) für jede Seite/Nutzen pro Druckbogen unterschiedlich sein können. Üblicherweise gibt es beim VDP einen statischen Teil, der auf jedem Bogen identisch ist, und einen variablen Teil, der von eine Datenbank oder einer Kontrolldatei gesteuert wird. Ein solcher Auftrag besteht meist aus den folgenden Teilaufgaben (Abbildung 10.16):

- Bereitstellung eines Datenbestandes (Datenbanken oder Tabellen) mit Informationen über die Adressaten,
- Design einer Vorlage (Template) für Seite/Nutzen, worauf die statischen und variablen Elemente positioniert sind, häufig mit Plug-Ins von Layout-Programmen realisiert,
- Definition der Regeln zur Verknüpfung der variablen Teile der Vorlage mit dem Datenbestand,
- Zusammenfügen aller Informationen zur Ausgabe auf dem RIP eines Digitaldrucksystems.

Für die Ausgabe der personalisierten Daten auf einem RIP könnte man auch normales PDF verwenden. Dann müsste aber jede Seite/jeder Bogen immer neu gerippt werden, selbst die statischen Elemente. Auch der variable Anteil besteht typischerweise nur aus einer Selektion einer Sammlung von wenigen festen Elementen. Insofern wäre diese Vorgehensweise sehr ineffektiv. Stattdessen sollten besser spezielle Sprachen

und natürlich auch entsprechende RIPs verwendet werden, die es ermöglichen, grafische Elemente nur einmal zu übertragen und zu rippen, aber mehrmals und variabel zu verwenden.

Hier ist neben den vielen herstellerspezifischen Formaten, auf die hier nicht eingegangen wird, in erster Linie die auf XML-basierende *Personalized Print Markup Language* (*PPML*) [11] [36] [37] [38] mit der Unterspezifikationen *PPML/GA* (*Grafic Art*) zu nennen. Das Datenformat *PPML/VDX* (*Variable Data Exchange*), das auf PPML und PDF beruht, und das von Adobe relativ neu spezifizierte Format PDF/VT (*Variable Transactional*) sind weitere Optionen. Von einigen Spezialisten aus der Branche wird prophezeit, dass das PDF/VT-Format [25] das PPML in Zukunft verdrängen wird, vor allem, da es auch in der *Print Engine 2* (der OEM-RIP-Technologie von Adobe Systems [1]) integriert ist.

PPML wurde genau wie JDF im Jahr 2000 vorgestellt. Diese herstellerunabhängige Sprache wird von der 1996 gegründeten Organisation PODi (Digital Printing Initiative) entwickelt. Ist nun PPML ein im Digitaldruck zu JDF konkurrierendes Format oder ergänzen sich die beiden? Um das zu beantworten wollen wir zunächst die wichtigsten Eigenschaften vom PPML betrachten.

PPML erlaubt es, digitale Objekte wie Bilder, Texte, Seiten, die in unterschiedlichen Datenformaten vorliegen (EPS, PDF, JPEG, TIFF,...), in eine PPML-Datei zu integrieren, wobei diese Objekte nicht unbedingt in der PPML-Datei eingebaut sein müssen, sondern auch referenziert werden können. In PPML können diese Objekte dann positioniert, transformiert und beschnitten werden. In PPML kann man auch Objekte als wiederverwendbar definieren, so dass sie dann nur einmal gerippt und anschließend für erneute Verwendungen in einem Cache gespeichert werden. Grafische Objekte werden also nicht in PPML gespeichert, sondern in den auch sonst üblichen Formaten. Stattdes-

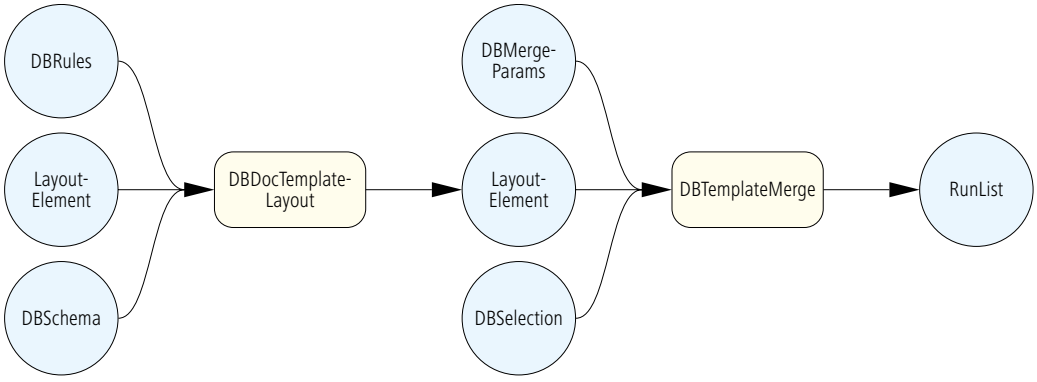



Abbildung 10.17
JDF-Modell für das personalisierte Drucken

sen beschreibt PPML nur, wie diese Objekte zusammengestellt werden müssen, um einzelne Seiten zu erzeugen.

Stark vereinfachend wird manchmal behauptet, dass PPML einfach nur die VDP-Seiten beschreibt wie PDF, nur stärker strukturiert, und der JDF-Workflow erst bei den fertigen Seiten beginne – folglich PPML und JDF nur ergänzende Formate seien. Dies ist aber aus mehreren Gründen nicht ganz richtig:

- JDF beschreibt auch die Seitenerstellung, dargestellt durch den Prozess *LayoutElementProduction*,
- PPML umfasst optional auch Workflow-Metadaten, wie beispielsweise die Beschreibung des Standbogens [37].

Obwohl sich folglich beide Formate in einigen Dingen überschneiden, ergänzen sie sich wiederum in vielen anderen. So können beispielsweise die Eigenschaften des Bedruckstoffs in PPML nicht genau spezifiziert werden, in JDF sehr wohl (in der Ressource *Media*). Zwar konnte eine PPML-Datei sogar JDF-Daten oder aber eine Referenz auf eine JDF-Datei enthalten, doch dieses wurde mit der Version 2.2 wieder zurückgenommen, da PPML in Zukunft nur als Content-Beschreibung dienen soll. Umgekehrt kann eine *RunList*-Ressource sich auf PPML-Daten beziehen. Diese Ressource ist dann Input für einen PPML-Konsument. In [39] sind detailliert die Übereinkommen zwischen den beiden Formaten geregelt. Beispielsweise gibt das Attribut *IgnorePDLImposition* in der JDF Ressource *LayoutElement* an, ob die Bogenlayout-Information aus den PPML-Daten oder aus dem JDF-Daten genommen werden soll.

Die Abbildung 10.17 gibt das JDF-Modell für das VDP wieder. Die *RunList*-Ressource auf der rechten Seite repräsentiert eine PPML oder eine andere VDP-Sprache. Sie ist ihrerseits typischerweise Input von einem *Imposition*-Prozess. Auf der linken Seite haben wir den Prozess *DBDocTemplateLayout*, der eine Vorlage für die statischen und variablen Elemente erzeugt, wobei die Verknüpfung zur Datenbank bereits definiert ist. Für diese Aufgabe werden Regeln benötigt, welche Text- oder Grafikelemente in die Vorlage eingebracht werden sollen. Da die Implementierungen der Regeln aber von Applikationen sehr unterschiedlich gehandhabt werden, sind sie in den *DBRules* nur als lesbare Kommentare hinterlegt. In der Ressource *DBSchema* wird der Datenbanktyp angegeben (SQL, XML oder durch Komma getrennte Texteinträge) und auch abermals ein lesbarer Kommentar zum Schema. In *LayoutElement* sind die einzelnen Content-Elemente, die für die Vorlage und als variable Elemente benötigt werden, beschrieben und deren URL angegeben. Der *DBDocTemplateLayout*-Prozess erzeugt eine neue *LayoutElement*-Ressource, welche die Vorlage repräsentiert. Um sie wirklich als Vorlage zu kennzeichnen, muss im Attribut *Template* der Wert *True* eingetragen sein. Schließlich hat der Prozess *DBTemplateMerge* die Aufgabe, aus der Vorlage eine VDP-Sprache wie PPML zu erzeugen. Hierzu wird als Information benötigt, wo im Dateisystem die Datei(en) abgelegt werden soll(en), was in *DBMergeParams* zu finden ist. In der Ressource *DBSelection* sind die URL der Datenbank, die Indizes der Datenbanksätze und die Datenbankzugriffe in der datenbankspezifischen Sprache (wie SQL) enthalten.

# 11 Weiterverarbeitung

Vernetzte Weiterverarbeitungsmaschinen gab es vereinzelt bereits, bevor JDF publiziert wurde. So erhielten beispielsweise Schnellschneider ihre Daten über PPF, wie es in Kapitel 4.2 beschrieben wurde. Oder es wurden proprietäre Lösungen entwickelt, bei denen die wichtigsten Daten für das Falzen (Bogengröße und Falzart) für einen Auftrag in einem EAN-Code kodiert und auf eine Auftragstasche gedruckt und später mit Hilfe eines Barcode-Lesers an der Falzmaschine wieder eingelesen wurden. Und es gab und gibt noch eine andere Lösung: Um einen Auftrag zu starten, wird beispielsweise für einen Planschneider oder für eine Falzmaschine zunächst ein Schneid- beziehungsweise Falzprogramm definiert. Um jedoch die (teuren) Spezialmaschinen nicht mit dieser Arbeit zu blockieren, werden auch Arbeitsvorbereitungsrechner aufgestellt, in denen die Programme festgelegt und anschließend über ein Netzwerk an die Weiterverarbeitungsmaschine(n) übertragen werden können. Das führt zur Verkürzung der Rüstzeiten an den Spezialmaschinen. Außerdem sammeln diese Rechner auftrags- und maschinenbezogene Daten der Produktionsmaschinen (vor allem bei Falzmaschine und Sammelhefter) wie Einrichtezeiten, Maschinenstatus, Produktionsgeschwindigkeit und so weiter. Die Maschinenhersteller haben hierzu proprietäre Lösungen entwickelt, und auch die Protokolle zwischen der Arbeitsvorbereitungs-Software und den Maschinen sind herstellerspezifisch (ähnlich wie zwischen den Leitständen der Druckmaschinen und den Druckmaschinen selber). Doch eine große Verbreitung haben solche vernetzten Konfigurationen nicht, und auch jetzt ist die JDF-Anbindung in der Weiterverarbeitung noch recht zögerlich.

Damit haben wir zwei Kandidaten für JDF/JMF-Schnittstellen, die auch beide von der Industrie angeboten werden:

- die Bedienkonsole der Weiterverarbeitungsmaschine und
- die Arbeitsvorbereitungsstation für eine oder mehrere Weiterverarbeitungsmaschinen.

Einmal ist also – um wieder die CIP4-Begrifflichkeit zu verwenden – das JDF-Gerät direkt an der Produktionsmaschine, das andere Mal auf einem eigenen Rechner, der über das lokale Netzwerk mit der oder den Maschine(n) kommuniziert. In beiden Fällen können die importierten JDF-Werte dann zur Grundeinstellung herangezogen werden, welche die Anwender in der Software noch modifizieren können. Im zweiten Fall werden die fertigen Produktionsprogramme dann an die Maschinen im herstellerspezifischen Format über das Netzwerk weitergeleitet. Und umgekehrt müssen dann auch die Produktionsdaten von dem proprietären Protokoll in JMF-Nachrichten und JDF-Strukturen über-

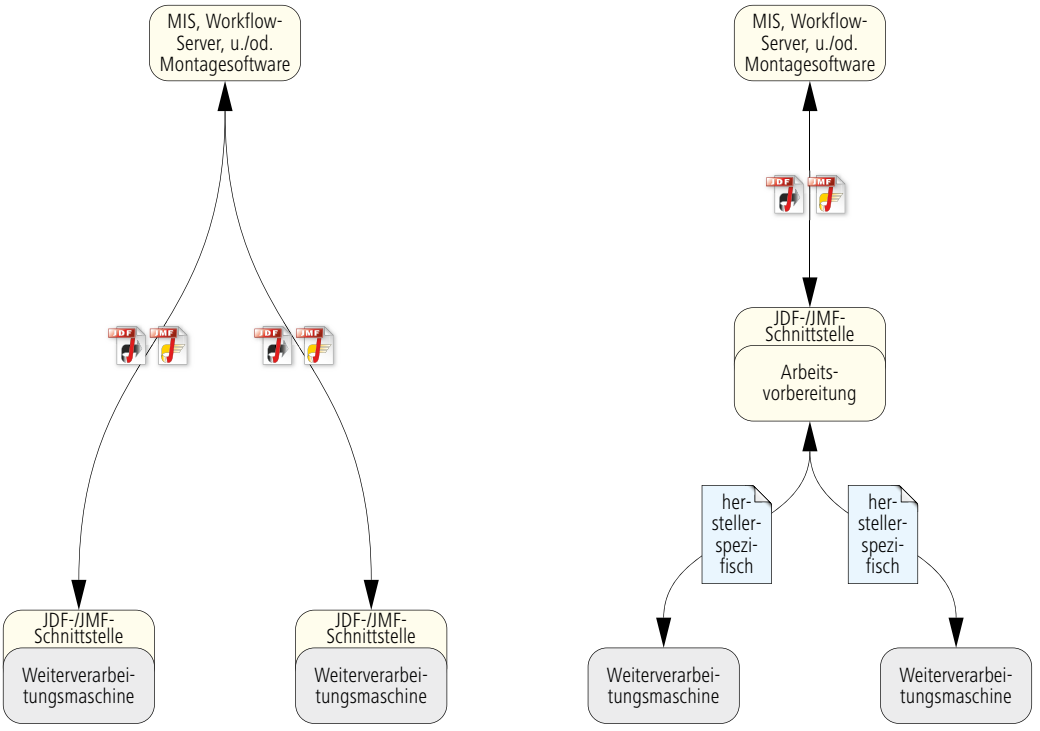



Abbildung 11.1
zwei unterschiedliche Konfigurationsprinzipien von JDF-/JMF-Schnittstellen in der Weiterverarbeitung

setzt werden. Abbildung 11.1 zeigt die beiden Konfigurationen als Skizze. Der Unterschied zwischen den beiden Lösungen besteht vor allem darin, dass eine separate Arbeitsvorbereitungsstation Aufträge in der Reihenfolge noch sortieren und ggf. auf verschiedene Maschinen verteilen kann. Die Bediener der Maschine(n) sind damit entlastet. Bei einer JDF/JMF-Schnittstelle, die in einer Maschine integriert ist, muss entweder das MIS oder der Bediener direkt an der Maschine die Produktionsreihenfolge festlegen und dafür sorgen, dass die Umrüstzeiten

**Beispiele für JDF-kompatible Postpress-Anbindung (2009)**

| **Hersteller:** | **Hauptprodukt:** |
|---|---|
| Ferag AG | iQ |
| Heidelberger Druckmaschinen | Prinect Postpress Manager |
| Hohner Maschinenbau GmbH | ixFrame (der ixact GmbH) |
| Horizon International Inc. | i2i |
| MBO | Datamanager |
| Müller Martini | Connex |

möglichst gering sind. Auch müssen unter Umständen mehrere JDF/JMF-Schnittstellen gepflegt werden, wenn sie bei den Produktionsmaschinen angesiedelt sind, und Software-Updates können damit schwieriger werden. Andererseits macht die Lösung mit einer Arbeitsvorbereitungsstation eigentlich nur dann Sinn, wenn mehrere Produktionsmaschinen eines Herstellers damit gesteuert werden können.

Die JDF-Geräte der Weiterverarbeitung erhalten JDF/JMF-Informationen entweder vom MIS, von einem Workflow-Server der Produktion oder unmittelbar von einem Montageprogramm. Letztere sind diejenigen, die eigentlich die Positionen der Schneid- und Falzmarken festlegen, insofern auch die Informationen weiterleiten können. Doch sind natürlich auch Konfigurationen möglich, bei denen die Montageprogramme die Information zunächst an das MIS oder an einen Workflow-Server zurückgeben, oder dass der dann seinerseits die Weiterverarbeitung bedient. Eine andere Möglichkeit ist, dass ein Montageprogramm, völlig vom MIS gesteuert, den Standbogen erzeugt und keine Möglichkeiten zur Positionsveränderung von Seiten hat. Dann kann das MIS direkt die Schneid- und Falzpositionen auch für die Weiterverarbeitung bestimmen.

## 11.1 Planschneider

Rohbogen sowie auch Zwischenprodukte (wie beispielsweise

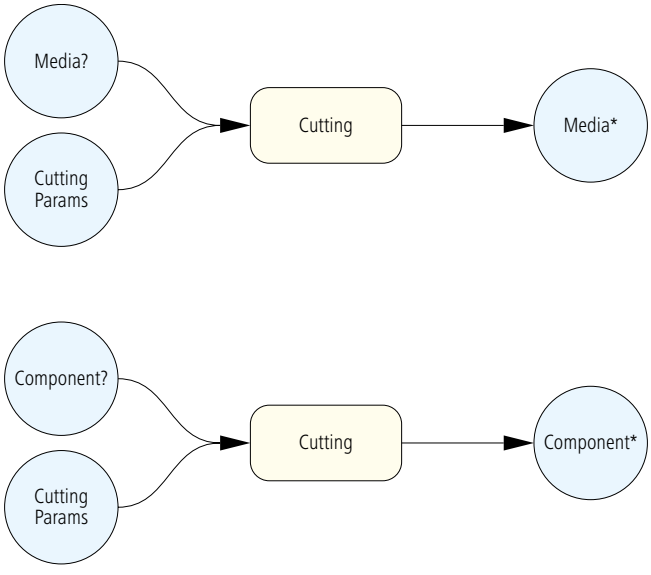



Abbildung 11.2
JDF-Modell des Schneideprozesses für Rohbogen (oben) beziehungsweise Zwischen- oder Endprodukte (unten)

Druckbogen) werden üblicherweise mit einem Schnellschneider - auch Planschneider genannt - geschnitten. Das Schneiden von Rohbogen vor dem Druckvorgang dient einerseits dazu, das erforderliche Druckformat, andererseits aber auch um rechte Winkel bzw. Parallelität an der oder den Druckanlagen zu erhalten. Druckbogen hingegen werden geschnitten, um mehrere Falzbogen aus einem Druckbogen zu bekommen oder auch um die Größe eines Falzbogens zu reduzieren, da beim Dreiseitenbeschnitt nach dem Falzen die Ränder nicht zu groß sein dürfen.

In der JDF-Notation sind Rohbogen Ressourcen vom Type *Media*, Zwischen- oder auch Endprodukte vom Typ *Component*. Der Schneideprozess (*Cutting*) erhält also als Input entweder *Media*- oder *Component*-Ressourcen und erzeugt auch den gleichen Ressourcentyp als Output. In Abbildung 11.2 ist das Modell zu sehen. Die Fragezeichen, die bei den Input-Ressourcen stehen und Optionalität bedeuten, müssen hier so interpretiert werden, dass genau eine der beiden Optionen zwingend vorgeschrieben ist. Die Output-Ressourcen sind mit einem Stern gekennzeichnet, der für keine oder mehrere Anzahlen von Ressourcen steht. Auch hier muss dazugesagt werden, dass genau eine Ressource aufgeführt werden darf, die vom Typ identisch zur Input-Ressource ist, und diese mindestens einmal vorkommen muss.

Die für den Schneidevorgang wichtigste Information steht in den *CuttingParams*. Ähnlich wie beim PPF können auch beim JDF Schneidblöcke (*CutBlock*) definiert werden (siehe Abschnitt 4.2 und insbesondere Abbildung 4.12). Dabei wird nur spezifiziert, wo geschnitten werden muss, nicht aber die Schnittfolge festgelegt. Diese kann in der *CuttingParams*-Ressource alternativ zu den Schneidblöcken auch festgelegt werden. Jede Schnittbeschreibung ist ein Element vom Typ *Cut*. In älteren JDF-Versionen hatte man auch

Abbildung 11.3
Definition von Schneidblöcken

```
<CuttingParams Class="Parameter" ID="_77" SheetName="Umschlag"
 Status="Available">
   <CutBlock BlockName="Umschlag_1" BlockSize="768.18 396.84"
    BlockTrf="1.0 0.0 0.0 1.0 96.37 14.17"/>
   <CutBlock BlockName="Umschlag_2" BlockSize="768.18 396.84"
    BlockTrf="1.0 0.0 0.0 1.0 864.56 14.17" />
   <CutBlock BlockName="Umschlag_3" BlockSize="768.18 396.84"
    BlockTrf="1.0 0.0 0.0 1.0 96.37 411.02" />
   <CutBlock BlockName="Umschlag_4" BlockSize="768.18 396.84"
    BlockTrf="1.0 0.0 0.0 1.0 864.56 411.02" />
   <CutBlock BlockName="Umschlag_5" BlockSize="768.18 396.84"
    BlockTrf="1.0 0.0 0.0 1.0 96.37 807.87" />
   <CutBlock BlockName="Umschlag_6" BlockSize="768.18 396.84"
    BlockTrf="1.0 0.0 0.0 1.0 864.56 807.87" />
</CuttingParams>
```

die Positionen von Schneidmarken in *CuttingParams*-Ressourcen schreiben können, seit Version 1.3 werden diese Informationen aber in der *Layout*-Ressource abgelegt, ähnlich wie wir es in Abbildung 10.7 für ein Druckkontrollelement gesehen haben.

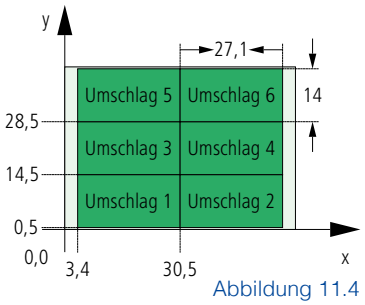
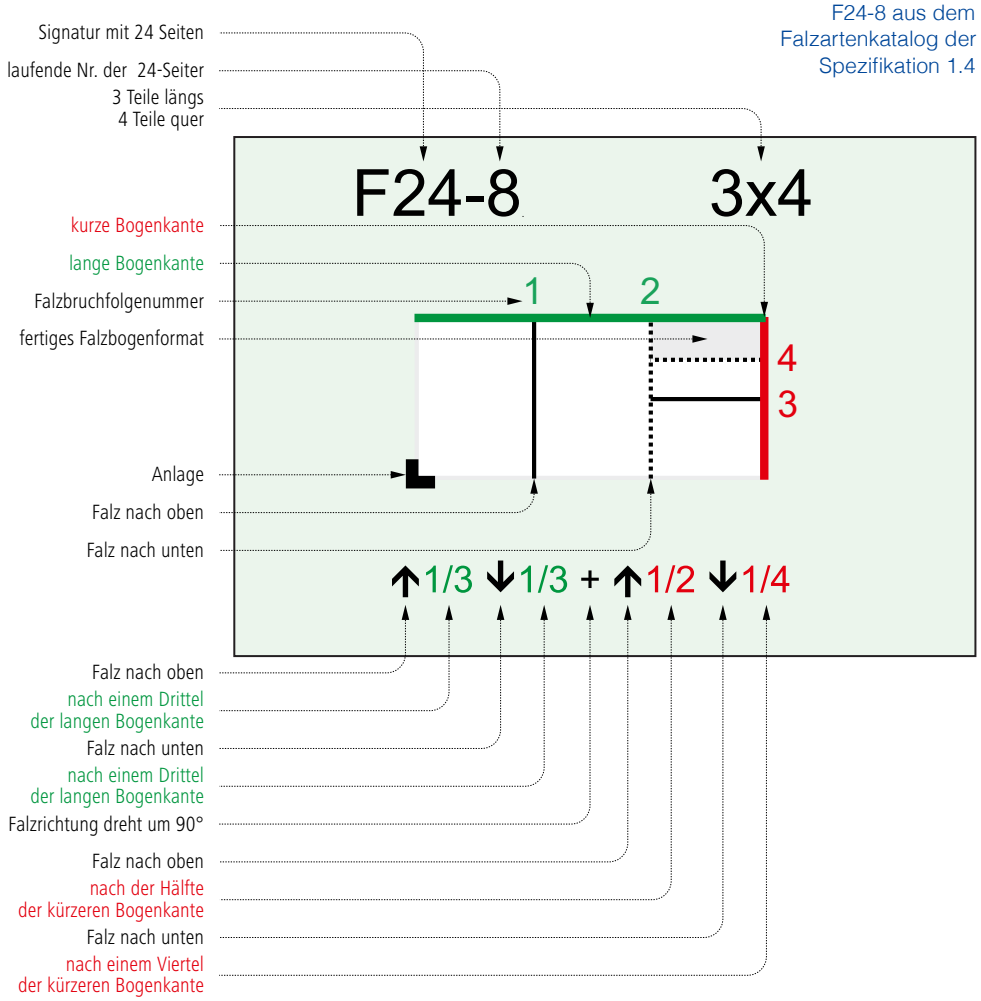



Abbildung 11.4
Position der Schneidblöcke auf dem Druckbogen

Abbildung 11.3 zeigt den JDF-Code einer *CuttingParams*-Ressource, die sechs Schneidblöcke enthält, nämlich für jeden Umschlag einen Block. Zur Übersichtlichkeit haben wir in der Abbildung übrigens ein paar Attribute entfernt und alle Werte auf zwei Stellen hinter dem Komma beschränkt – in der Realität wird man aber die Werte mit einer höheren Genauigkeit eintragen. Die Blockgröße von 768.18 x 396.84 DTP-Punkten entspricht 271 x 140 mm. Die Blockgröße, auf die zunächst geschnitten wird, um-

Abbildung 11.5
F24-8 aus dem Falzartenkatalog der Spezifikation 1.4



fasst Vorderseite, Rückseite und Rücken der Broschur und einen Zentimeter Beschnitt um das Endformat herum. Mit dem Attribut *BlockTrf* wird die Position jedes Blocks festgelegt. Für die Matrix, die im Wert des Attributes steht, gilt die Definition wie in Abbildung 9.21 ausgeführt. Rechnet man die DTP-Punkte in Zentimeter um ergibt sich die Anordnung von Abbildung 11.4. Ein Dreiseitenbeschnitt schneidet die Broschur ganz zum Schluss auf das Endformat. Dieser Arbeitsvorgang wird aber nicht mit einem *Cutting*-Prozess beschrieben, sondern durch den Prozess *Trimming*. Entsprechend gibt es dann auch keine *CuttingParams*, sondern *TrimmingParams*. Dort würde also in unserem Beispiel das Endformat 120 x 120 mm eingetragen sein.

## 11.2 Falzmaschine

Bei modernen Falzmaschinen können viele Dinge, die früher ausschließlich manuell eingestellt wurden, nun in einem Dialogfenster eingegeben und dann automatisch geregelt werden. Der Seitenanschlag beim Anleger, die Walzeneinstellungen gemäß Papierdicke oder der Vorderanschlag beim Taschenfalz wären hier als Beispiel zu nennen. Andere Parameter werden hingegen nicht automatisch gesteuert, wie beispielsweise die Luftzufuhr zur Bogenvereinzelung. Aber es wäre auch hier denkbar, dass Voreinstellungen für bestimmte Papiersorten oder sogar für spezielle Papiere abgespeichert und anhand der Auftragsinformation beziehungsweise des JDFs aufgerufen werden oder dass künftig eine „lernende“ Software für bestimmte Bedingungen entsprechende Einstellungen ermittelt, speichert und auch wieder zur Verfügung stellt.

Typischerweise werden bei solchen Falzmaschinen mit automatischer Grundeinstellung Bogengröße und Falzart in der Menüführung eingegeben, woraufhin das elektronische Einrichtesystem das motorische Einstellen übernimmt. Nach einer Probefalzung erfolgt anschließend die Feinjustierung, das heißt, es werden in mehreren Iterationen die Abweichungen zu den Falzmarken überprüft und nachkorrigiert. Da aber am Anfang nur Bogengröße und Falzart als Parameter zur Verfügung steht, müssen nicht nur der durch den Druck verursachten Papierverzug und Fehler in den vorangehenden Arbeitsgängen, sondern auch Unterschiede der einzelnen Teillängen aufgrund entsprechender Positionierung oder durch einen beim Ausschießen angelegten Vor- oder Nachfalz (vor- oder nachgelagerter Greiffalz) korrigiert werden. Mit anderen Worten: Die Angaben der Falzkatalognummer und Falzbogengröße sind im Grunde genommen nicht ausreichend (siehe Abbildung 11.5), und die Feinjustierung kann deswegen deutlich aufwändiger werden als die Grundeinstellung.

Die gleiche Nachjustierung ist also nötig, wenn im JDF ebenfalls nur das Falzschema und die Bogengröße in den Falz-Ressourcen eingegeben sind. Manche Systeme geben aber auch die genaue Falzposition und die Falzreihenfolge als JDF-Informationen weiter, wobei dann ein eventuell vorhandener Vor- oder Nachfalz bereits eingerechnet ist. Hier ist dann der Nachjustieraufwand und auch die Fehleranfälligkeit geringer.

In Abbildung 11.6 wird eine Falzart für einen 24-Seiter mit der Falzkatalognummer F24-8 in der Ressource *FoldingParams* festgelegt. In vielen Fällen ist diese Information alles, was an Information weitergereicht wird. Im vorliegenden Beispiel werden aber zu-

```
<FoldingParams ID="FOLD_A-1" Class="Quantity" Status="Available"
 FoldCatalog="F24-8" SheetLay="Left">
   <Fold From="Front" To="Up" Travel="943.9370079811025"/>
   <Fold From="Front" To="Down" Travel="1887.8740161811025"/>
   <Fold From="Left"  To="Up" Travel="907.0866141732283"/>
   <Fold From="Left"  To="Down" Travel="1360.6299241732283"/>
</FoldingParams/>
```

Abbildung 11.6
Falzparameter

sätzlich auch die Falzpositionen in den *Fold*-Elementen spezifiziert. Jede *Fold*-Ressource definiert eine Falzoperation, wobei die Reihenfolge der *Fold*-Elemente auch die Falzreihenfolge festlegt. Wenn, wie in unserem Beispiel, sowohl die Falzkatalognummer als auch die einzelnen Falzpositionen von einem Agenten/Controller vermerkt sind, werden die *Fold*-Elemente vom JDF-Gerät vorgezogen. So verlangt es zumindest die JDF-Spezifikation. Das Attribut *From* gibt die Ecke an, von wo aus gefalzt wird, das Attribut *To* die Falzrichtung. Die Kantenbezeichnungen sind in Abbildung 11.7 zu sehen, auch die Anlage vorne-links. Die Travel-Werte sind wie üblich in DTP-Punkten angegeben. Beim ersten Falzvorgang in Beispiel 11.5 wird der Bogen bei x = 33,3 cm (entspricht 2,54·943,9370079811025:72) in y-Richtung gefalzt, wobei der vordere Teil über den hinteren Teil gelegt wird. Der dritte Falzvorgang liegt bei y = 32 cm, wobei der linke auf den rechten Bogenteil zum liegen kommt.

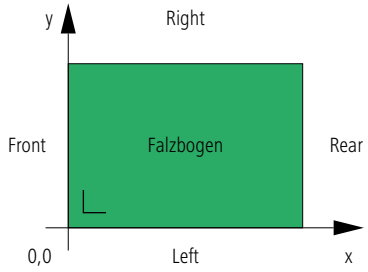



Abbildung 11.7
Bezeichnung der Kanten am Falzbogen

Doch auch wenn die Falzposition in den *FoldingParams* eingetragen sind, handelt es sich trotzdem nur um eine Voreinstellung. Denn nicht nur der bereits erwähnte Papierverzug kann dazu führen, dass die „theoretischen“ Werte korrigiert werden müssen, sondern auch die Grammatur und Produktionsgeschwindigkeit der Falzmaschine verändert die Falzkoordinaten. Und solange es hier noch keine automatisierte Ausgleichsregelung gibt, muss der Operator an der Falzmaschine entsprechend nachregeln.

Falzmaschinen werden häufig auftragsabhängig aus verschiedenen Modulen zusammengefügt. Wird eine Falzart eingegeben beziehungsweise eine *FoldingParams*-Ressource in das JDF-Gerät geladen, welche nicht in der aktuellen Konfiguration gefalzt werden kann, gibt es eine Fehlermeldung erst direkt an der Falzmaschine. Die Mitarbeiter des Auftragsmanagements müssen also die Möglichkeiten der Falzmaschine kennen und auch die Disponenten die aktuelle Konfiguration zur Festlegung der Auftragsreihenfolge. Möchte man an dieser Stelle eine soft-

ware-unterstützte Lösung bereitstellen, ist eine Abfrage der Maschinenkonfiguration beziehungsweise eine Anfrage nach den Fähigkeiten des JDF-Gerätes notwendig. Das ist nicht so trivial, wie es auf dem ersten Blick erscheint, denn ein Produkt hat eine Kombination von Eigenschaften, die für die Entscheidung wichtig ist, ob es auf einer bestimmten Maschine gefalzt werden kann oder nicht. So mag beispielsweise eine Falzmaschine durchaus Papier von 250 g/m² bearbeiten können und auch einen 6-fachen Zickzackfalz durchführen, aber nicht beides zusammen. Diesem Thema widmet sich die JDF-Spezifikation unter dem Begriff *Device Capabilities*. Hierbei kann ein Gerät definieren, welche Knoten, Elemente, Attribute und deren Werte, Ressourcen und JMF-Nachrichten es unterstützt. So kann beispielsweise die maximale Falzbogengröße an das MIS oder einen Produktionscontroller übermittelt werden. Die Möglichkeiten sind aber noch sehr viel weitreichender und es können sogar Leistungsparameter einer Maschine wie maximale Bogenanzahl pro Stunde oder Rüstzeiten beschrieben werden. Auch JDF-spezifische Details, nämlich ob die Geräte-Software Grayboxen, kombinierte Prozesse oder allgemeine Prozessgruppenknoten verarbeiten kann, können ebenfalls kommuniziert werden. Abbildung 7.3 zeigt die prinzipielle Situation.

## 11.3 Sammelhefter

Ein Sammelhefter besteht im Allgemeinen aus drei Module:

- einer Sammelmaschine,
- einem Rückstichheftaggregat und
- einem Trimmer für den Drei-Seiten-Beschnitt.

Beim Sammeln werden unterschiedliche Falzbogen ineinander gesteckt, so dass ein eingesteckter Block entsteht. Hierzu werden mehrere meist in Reihe gebaute Anleger manuell oder automatisch mit Stangen beziehungsweise Rollen mit verschiedenen Falzbogen beschickt, wobei die korrekte Reihenfolge (das heißt von innen nach außen im Endprodukt) einzuhalten ist. Im Betrieb werden dann die Falzbogen vereinzelt, in der Mitte geöffnet und auf einen Sattel gelegt. Indem die Bogen von einem Anleger zum anderen transportiert werden, werden die einzelnen Falzbogen somit übereinander abgelegt. Im letzten Vorgang wird ggf. der Umschlag auf den Block gebracht. Die Menge unterschiedlicher Falzbogen, die gesammelt werden, hängt natürlich vom Umfang des Druckproduktes ab, aber auch von der Anzahl der zur Verfügung stehenden Anleger. Das Einrichten eines Sammelhefters beinhaltet also das Einstellen der unterschiedlichen Anleger auf die Falzbogenformate und den Bedruckstoff.

In einem Rückenstich-Heftaggregat – auch Drahtstichaggregat oder kurz Heftmaschine genannt – werden Drahtklammern durch den Rücken des Blocks gestochen und umgebogen. Die Anzahl, Positionen, Breiten und Formen der Drahtklammern können dabei von Auftrag zu Auftrag variieren. Bei einem Block wird jede Klammer von einem eigenen Heftkopf gesetzt, der von einer Spule kommenden Heftdraht auf die richtige Länge

schneidet, die Klammer vorbiegt, durch den Bundsteg stößt und schließlich umbiegt und damit schließt.

Der Trimmer (Dreiseitenschneider oder auch Dreimesserautomat) hat die Aufgabe, das geklammerte Druckprodukt auf das Endformat zu schneiden und ggf. gleichzeitig damit die Falze aufzuschneiden, damit sich die Seiten aufschlagen lassen. Dazu werden alle Seiten außer die Bundseite beschnitten (Kopf, Fuß, Vorderseite). Einige Trimmer ermöglichen auch einen Mitteltrennschnitt, so dass Doppelnutzenproduktion gefahren werden kann.

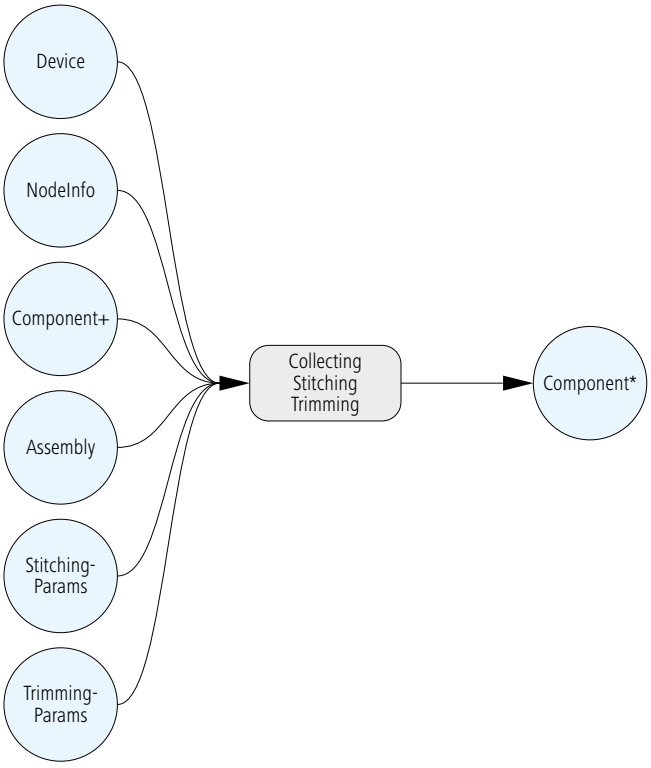



Abbildung 11.8
GrayBox für Sammelheften (Zusammentragen, Klammern und Dreiseitenbeschnitt)

Wie zu erwarten, sind auch im JDF drei Prozesse für Sammelhefter maßgeblich: *Collecting*, *Stitching* und *Trimming*. Da alle drei Module eines Sammelhefters üblicherweise starr gekoppelt sind, ist es folglich zweckmäßig, diese drei Prozesse durch einen kombinierten Prozess oder durch eine Prozessgruppe zu beschreiben. In Beispiel 11.8 ist eine entsprechende *GrayBox* mit Ihren Input- und Output-Ressourcen abgebildet. Die Ressourcen werden im Folgenden kurz erläutert:

- Die *Device*-Ressource enthält die Gerätebezeichnung und die Kostenstelle des Sammelhefters,
- in *NodeInfo* sind die geschätzten Einrichte- und Produktionszeiten eingetragen,
- die *Component* enthält die Beschreibung der Falzbogen,
- die *Assembly* definiert die Reihenfolge beim Sammeln der Falzbogen,
- In *StitchingParams* ist die Anzahl und Form der Klammern vermerkt,
- die *TrimmingParams* enthalten als Information die Endformatgröße des Druckprodukts, auf das schlussendlich geschnitten wird.

Eigentlich erwartet man für den *Collecting*-Prozess auch eine *CollectingParams*-Ressource, und in der Tat ist sie auch in der JDF-Spezifikation aufgeführt – allerdings ohne Attribute und Unterelemente. Sie ist nur ein Container für Erweiterungen und hier nicht weiter ausgeführt. Die Folge der zu sammelnden Falzbogen wird also nicht in dieser Ressource definiert, sondern in der *Assembly*-Ressource. Dort sind *AssemblySection*-Elemente eingetragen, und deren Reihenfolge gibt auch die Abfolge an, wie gesammelt werden muss, wobei das erste Element die äußere Position und das letzte Element die innerste Position im Druckprodukt darstellt. Abbildung 11.9 stellt diese Situation mit drei Falzbogen für den Inhalt und einen für den Umschlag dar. Beim *Selective Binding*, bei dem Falzbogen in variabler Weise gesammelt werden, erhält der *Collecting*-Prozess noch weitere Input-Ressourcen (*DBRules* und *DBSelection*). Damit können unterschiedliche Fassungen für verschiedene Adressaten des Druckproduktes hergestellt werden.

Abbildung 11.9
Sammelreihenfolge der Falzbogen

Zum Abschluss soll noch das Beispiel, aus dem die *Assembly*-Ressource in 11.9 entnommen wurde, im Ganzen erläutert werden (allerdings etwas vereinfacht). Der JDF-Baum ist in Abbildung 11.10 wiedergegeben. In der linken Spalte sind die zentralen Vorstufenprozesse eingezeichnet: Das Überprüfen der Daten auf Produktionstauglichkeit (*Preflight*), das Distillen der PostScript-Daten zu PDF (*PSToPDFConversion*), die Farbraumtransformation (*ColorSpaceConversion*), das Überfüllen/Unterfüllen (*Trapping*) und das Erzeugen eines Standbogens (*Stripping*). Vielfach werden diese Prozesse zunächst auch nur durch die *Grayboxen PrepressPreparation* und *ImpositionPreparation* skizziert. Die mittleren beiden Spalten in der Zeichnung 11.10 zeigen die unterschiedlichen Prozesse beziehungsweise kombinierten Prozesse für die Teilprodukte

```
<Assembly Class="Parameter" ID="_019069" Order="Collecting" "
 Status="Available">
   <AssemblySection AssemblyIDs="Cover_B_1"
    DescriptiveName="F04-01_ui_2x1_1" />
   <AssemblySection AssemblyIDs="Text_1_B_2"
    DescriptiveName="F08-07_li_2x2_2" />
   <AssemblySection AssemblyIDs="Text_2_B_3"
    DescriptiveName="F08-07_li_2x2_3" />
   <AssemblySection AssemblyIDs="Text_3_B_4"
    DescriptiveName="F08-07_li_2x2_4" />
</Assembly>
```

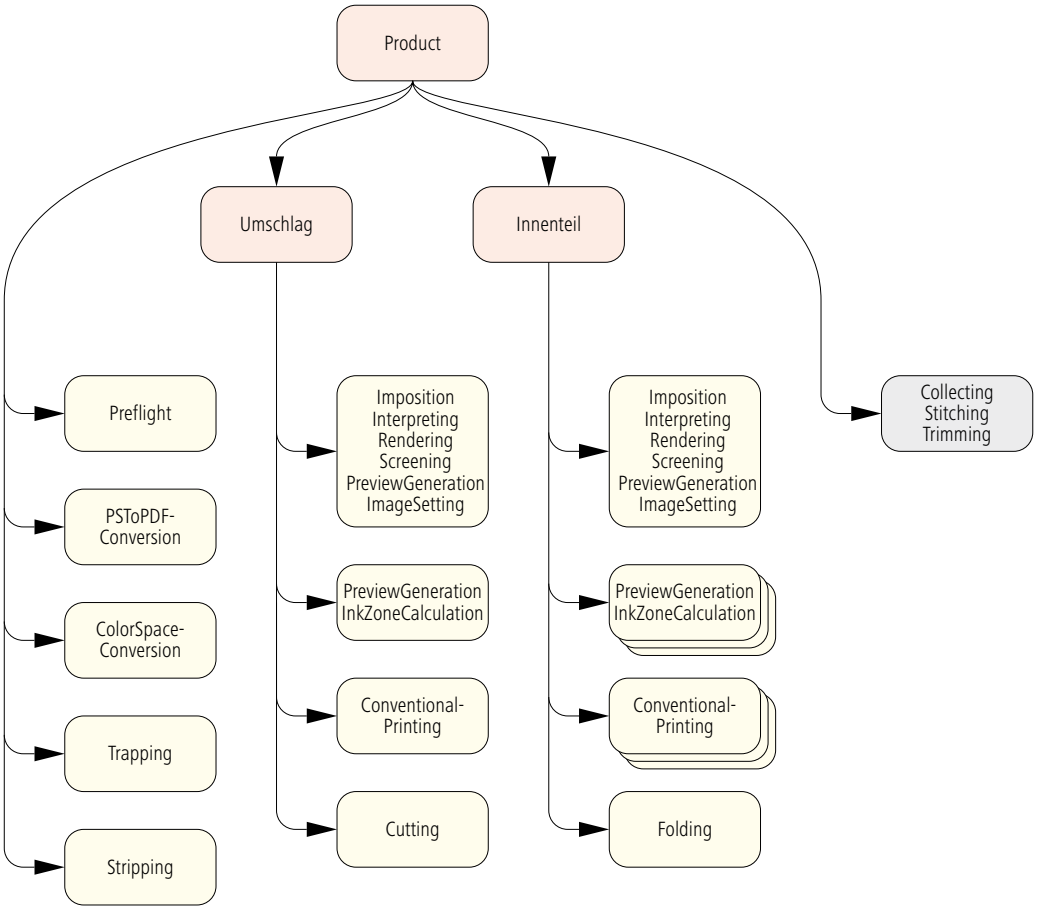



Abbildung 11.10
JDF-Prozess-Baum

„Umschlag“ und „Inhalt“ an. Die oberen beiden kombinierten Prozesse enthalten beide den Prozessnamen *PreviewGeneration*, doch deren Funktionen sind verschieden. Während im kombinierten Prozess „*Imposition ImageSetting*“ ein Vorschaubild definiert wird, das später auf dem Monitor des Druckmaschinenleitstands zu sehen ist, wird das Vorschaubild von dem kombinierten Prozess „*PreviewGeneration InkZoneCalculation*“ nur für die Berechnung der Farbzonenvoreinstellung erzeugt (typischerweise separierte Bilder in 50.8 ppi). Da sich auf dem Druckbogen mehrere Umschlagnutzen befinden, folgt nach dem Druckprozess ein Schneidvorgang. Die Inhaltsbogen hingegen werden gefalzt.

Abbildung 11.11 umreißt noch einmal einen Teil des Produktionsablaufs, wobei nur die wichtigsten „Übergaberessourcen“ aufgeführt werden: die Seiten (*RunList*) für das Ausschießen (*Imposition*), die Platten (*ExposedMedia*) als Schnittstelle zwi-

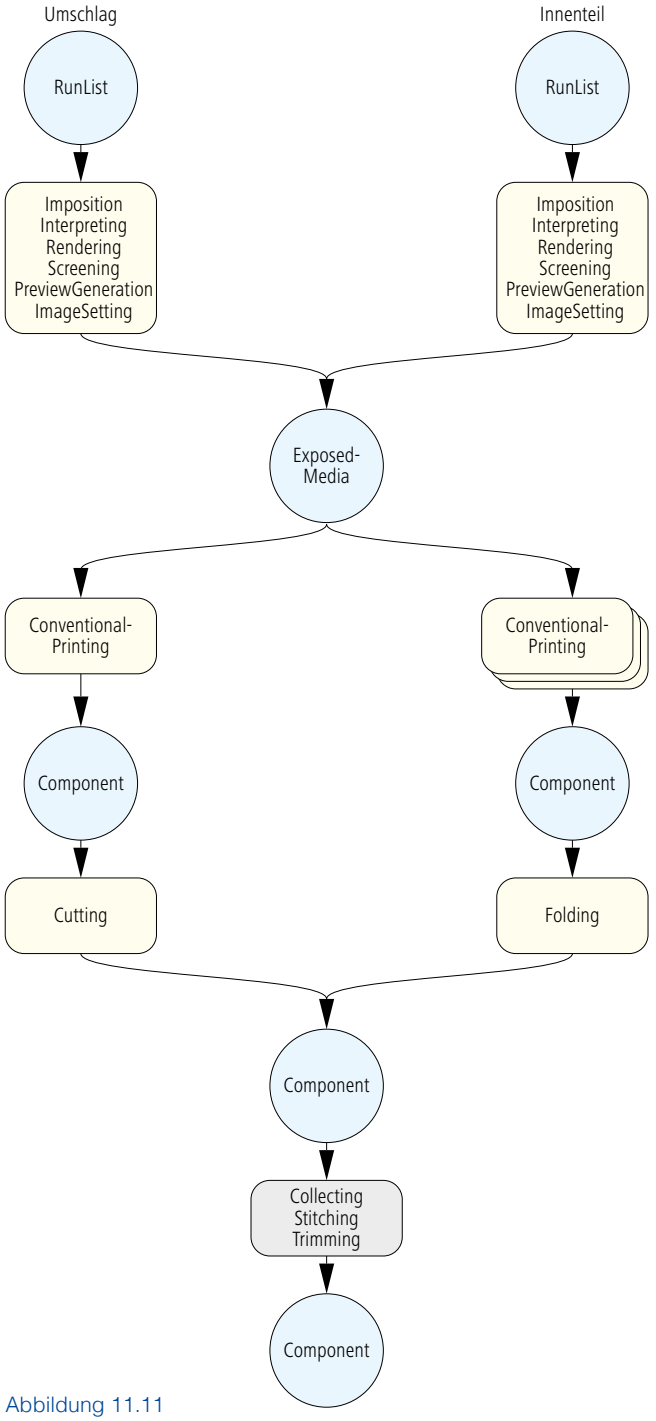



Abbildung 11.11
Produktionsablauf

schen Vorstufe und Druck und schließlich die unterschiedlichen *Component*-Ressourcen Druckbogen, Falzbogen und das fertige Produkt.

In der Übersetzung der Prozessnamen im Anhang sind knapp 50 unterschiedliche Prozesse der Druckweiterverarbeitung aufgeführt; davon wurden in diesem Kapitel jedoch nur fünf ausführlicher behandelt (*Cutting*, *Folding*, *Collecting*, *Stitching*, *Trimming*). Der Prozess der Stanzformherstellung (*DieMaking*) wird noch im nächsten Kapitel vorgestellt. Alle anderen müssen leider unberücksichtigt bleiben.

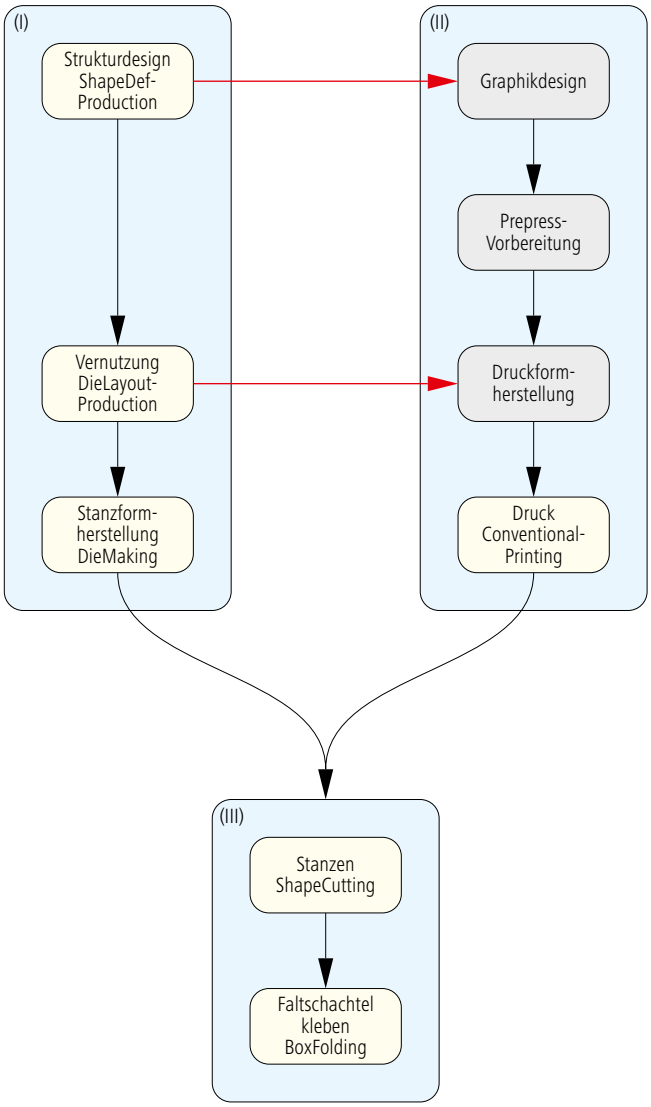
**Übungen:**

- Stellen Sie analog zu 11.11 eine Prozesskette für eine klebegebundene Broschur zusammen, deren Umschlag mit einer Goldfolie geprägt wird. Suchen Sie hierzu die nötigen Prozesse aus der JDF-Spezifikation heraus und führen Sie auch die „Übergaberessourcen“ auf.
- Erstellen Sie zusätzlich noch analog zu 11.10 einen möglichen JDF-Baum für die klebegebundene Broschur.

# 12 Verpackungsdruck

Aus der Sicht des JDF/JMF-Modells gibt es keinen prinzipiellen Unterschied zwischen Verpackungsdruck, Akzidenzdruck und anderen Anwendungsbreichen. Zwar werden im Verpackungsdruck natürlich andere Bedruckstoffe (wie Karton, Wellpappe, Folien oder Verbundstoffe) benutzt und auch andere Druck- und Weiterverarbeitungsmaschinen eingesetzt, dennoch gibt es hierfür aus der JDF-Prozesssicht keine Veränderung. Nur ein paar neue Attribute und Attributwerte für einige Ressourcen müssen definiert werden. Das bereits mehrfach verwendete Attribut *MediaType* nutzt folglich Werte wie *CorrugatedBoard* (Wellpappe) oder *Foil* (Folie). Außerdem wurden Attribute zur Beschreibung dieser Materialien wie *Flute* (Welle) für die Wellpappe eingeführt (zum Beispiel in den Ressourcen *Media* und in *MediaIntend*). Seit JDF 1.4 gibt es auch zunehmende Unterstützung für den Flexodruck. So wurden zum Beispiel für das Attribut *MediaType* auch die Werte *Sleeve* und *MountingTape* (Klebeband) definiert und für die Flexoplatten Werte für die Plattentechnologie (*PlateTechnology*) und die Reliefhöhe (*ReliefThickness*) definiert.

Abbildung 12.1
Aktivitätendiagramm „Faltschachtelherstellung“




Es gibt jedoch Prozesse, die hauptsächlich im Verpackungsbereich eingesetzt werden. In diesem Kapitel sollen nun einige davon vorgestellt und in die drei folgenden Bereiche eingeteilt werden:

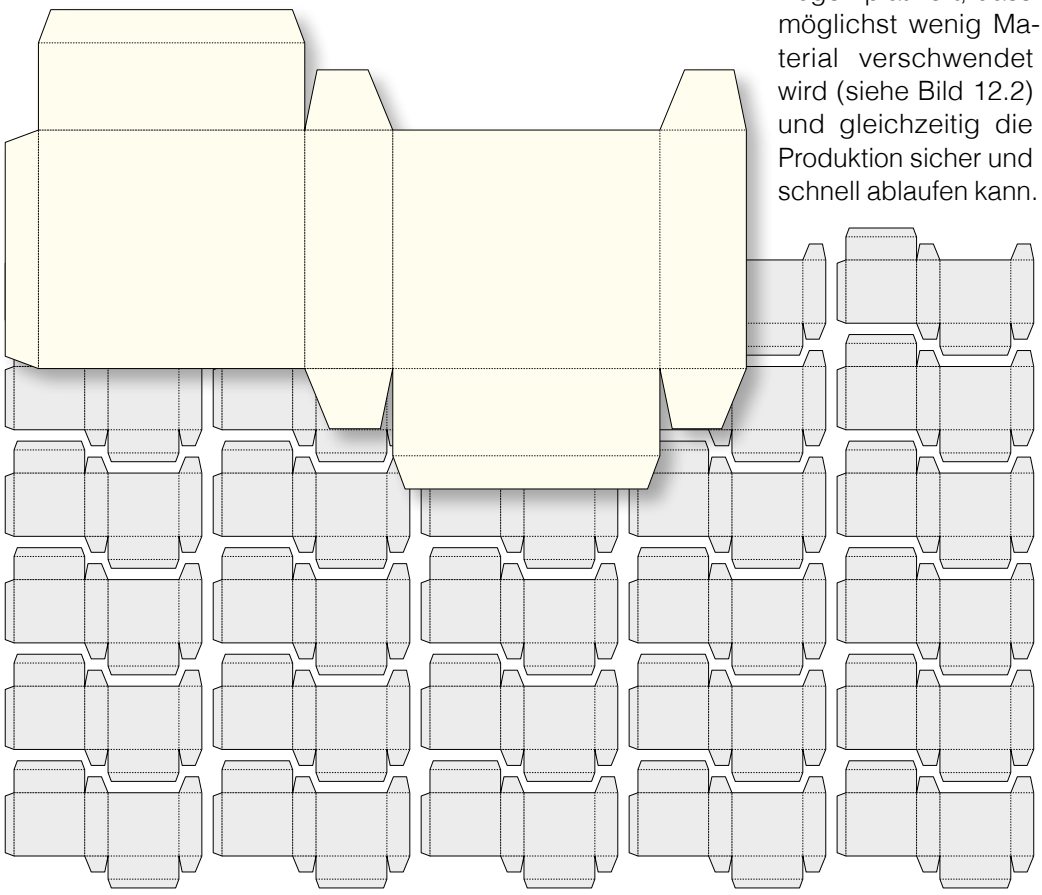
a) Faltschachtelkonstruktion (Strukturdesign), Vernutzung, Stanzformherstellung,

b) Stanzen und Faltschachteln kleben,
c) Barcode-Management inklusive Strichbreitenkompensation, vor allem im Flexodruck.

Die Abbildung 12.1 zeigt ein Aktivitätendiagramm, wobei die in den Beispielen a) und b) aufgeführten Prozesse noch einmal im Produktionskontext der Faltschachtelherstellung zu sehen sind. Die Strichbreitenkompensation in c) ist ein Teilprozess der Aktivität „Prepress-Vorbereitung".

Die mit gelben Rechtecken hinterlegten Aktivitäten entsprechen jeweils einem JDF-Prozess. Beim Konstruktionsdesign (*ShapeDefProduction*) werden typischerweise mit einem CAD-System die Umrisse, Rill- und Perforationslinien für einen Nutzen festgelegt. Bei der Vernutzung (*DieLayoutProduction*) werden anschließend ein oder mehrere Nutzen mehrfach so auf einen Bogen platziert, dass möglichst wenig Material verschwendet wird (siehe Bild 12.2) und gleichzeitig die Produktion sicher und schnell ablaufen kann.

Abbildung 12.2
Einzelnutzen und Vernutzung



Dieser Vorgang wird in der Regel mit dem gleichen CAD-System durchgeführt. Zusätzlich werden noch weitere Dinge festgelegt, die für den Stanzformbau wichtig sind, wie zum Beispiel die Registerlöcher für die Aufnahme des Werkzeugs in der Stanzmaschine. Die exportierten CAD-Daten können dann beim Stanzformbau wieder in die entsprechenden Systeme zur Herstellung von Stanzwerkzeugen importiert werden, wobei ein Stanzwerkzeug aus mehreren Teilen bestehen kann (siehe Abbildung 12.3):

- Stanzform (*CutDie*),
- Gegenzurichtung/Gegenstanzform (*CounterDie*),
- Ausbrechwerkzeuge (*Lower Stripper*, *Upper Stripper*),
- Nutzentrennwerkzeuge (*Blanker*).

Die Gegenzurichtung wird zur Ausprägung der Rillungen und das Ausbrechbrett für das Trennen vom Stanzabfall benötigt. Die Stanzformherstellung (*DieMaking*) erzeugt also ein oder mehrere Werkzeug(e) (*Tool*) für den Stanzvorgang (*ShapeCutting*). In einer Stanzmaschine werden dann häufig auch die Zuschnitte ausgebrochen und die Nutzen getrennt. Beides kann aber auch offline, also außerhalb der Stanzmaschine erfolgen. Die gestanzten Schachtelzuschnitte kommen dann in den Anleger einer Faltschachtelklebemaschine, in der die Schachteln


Abbildung 12.3
Elemente einer Stanzform: gummierte Stanzform im Schliessrahmen (rechts), Stanzplatte mit Gegenzurichtung (links oben), Druckbogen (links unten), gestanzter und ausgebrochener Einzelnutzen (Bildmitte) und fertige Faltschachtel

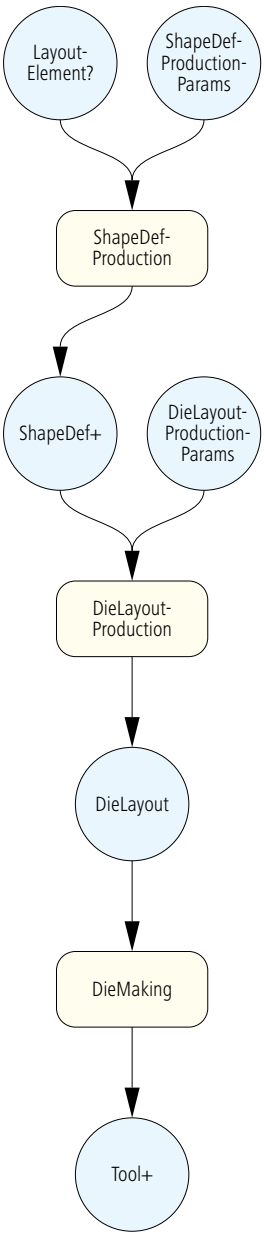



Abbildung 12.4
Stanzformherstellung

gefaltet, geleimt und verschlossen werden. Diese Vorgänge werden unter dem Prozess „Faltschachtel kleben“ (*BoxFolding*) zusammengefasst.

Die roten Linen in der Zeichnung 12.1 bezeichnen spezielle Datenflüsse. Für ein grafisches Design einer Faltschachtel werden nämlich die Informationen über Stanz- bzw. Rillkonturen benötigt, damit das Design zur Falzschachtel passt. Häufig wird hierbei einfach eine entsprechende EPS-Datei oder die PDF-Pfade aus dem CAD-System exportiert und dann in die Grafik-Software wieder eingelesen, so dass die Designer sich daran orientieren können. Außerdem müssen beim Ausschießen (Step&Repeat) in der Druckformherstellung die Nutzen so positioniert werden, dass später Stanzung und Rillung des Druckbogens exakt mit der Nutzenlage übereinstimmen. Da die Positionen der Nutzen aber meist frühzeitig mit dem CAD-System festgelegt werden und nicht in der Druckformherstellung, werden Positionsdaten vom CAD-System weitergereicht – häufig in Form von CFF2-, DXF oder in DDES-Dateien. Diese werden sowohl dem Stanzformbauer als auch dem Druckformhersteller gegeben. Der Grund, warum nicht die Druckformherstellung die Positionen der Nutzen festlegt (wie es ja im Akzidenzbereich mit den Seiten geschieht) liegt daran, dass die Herstellung einer Stanzform längere Zeit beansprucht, zumal sie häufig auch außer Haus gegeben wird. Und die CAD-Zeichnungen stehen in der Regel am Anfang des Herstellungsprozesses, während die Druckformherstellung erst kurz vor dem Drucktermin die Daten für die Plattensätze vorbereitet.

Bei einfachen Etiketten (*Labels*), die von einem Rechteck begrenzt werden (*Bounding Box*), stellt manchmal auch das MIS die Informationen für das Step&Repeat zur Verfügung und übermittelt diese sowohl dem Stanzformbau als auch der Druckformherstellung. Bei Faltschachteln, die in komplexer Weise verschachtelt auf einen Druckbogen montiert werden, muss jedoch in jedem Fall ein CAD-System oder eine andere Produktionssoftware die Vernutzung übernehmen.

## 12.1 Stanzformdesign, Vernutzung und Stanzformherstellung

In Abbildung 12.1 stellt Kasten (I) das strukturelle Design mit CAD und den Stanzformbau dar, während Kasten (II) die Arbeitsschritte der grafischen Produktion bis zum Druck und Block

(III) die Druckweiterverarbeitung repräsentieren. Die Prozesse in (I) wurden erst mit der JDF-Spezifikation 1.4 eingeführt und sind folglich zurzeit noch nicht in einem fertigen JDF-Workflow implementiert. Die hier aufgeführten Beispiele basieren deswegen nicht auf Produktionsdaten. Die Prozesse in Kasten (II) und (III) wurden hingegen bereits vor mehreren Jahren in der Spezifikation festgelegt, und es sind Implementierungen auf dem Markt (wenn auch nur wenige), die diese JDF-Prozesse lesen und ausführen.

In Abbildung 12.4 ist Kasten (I) von Abbildung 12.1 noch einmal als Prozess-Ressourcen-Modell von JDF aufgezeichnet. Für den Prozess *ShapeDefProduction*, also die CAD-Konstruktion einer Faltschachtel (oder Etiketts, Kartonaufsteller, Beutel,…), stehen zwei Input-Ressourcen zur Verfügung: In dem optionalen *LayoutElement* ist typischerweise eine Skizze eingetragen, nachdem die Konstruktion vorgenommen werden soll. Der Eintrag könnte beispielsweise die URL einer EPS-Datei sein, in der die Skizze gespeichert ist. Der Entwurf selber wird meist vom Auftraggeber oder einer (internen oder externen) Werbeagentur vorgegeben. In der Ressource *ShapeDefProductionParams* kann die URL und das Ausmaß eines 3-D Modells der Verpackung verzeichnet sein. Außerdem ist es möglich, den Namen eines Verpackungsstandards einzutragen, beispielsweise aus dem FEFCO- oder dem ECMA-Katalog. Der Output *ShapeDef* des *ShapeDefProduction*-Prozesses beschreibt die Konstruktion der Faltschachtel, wobei entweder eine externe Datei referenziert werden kann, welche die Konturen enthält, oder aber die Konturen selber in die Ressource eingetragen sind. Im letzteren Fall werden Geraden und Bézier-Kurven als Werte dem Attribut *CutPath* zugewiesen. Die Kodierung der Geraden und Kurven erfolgt wie beim PDF (siehe Abschnitt 4.4.1 in [5]). Abbildung 12. 5 verdeutlicht noch einmal die beiden Möglichkeiten. Im ersten Fall ist eine CFF2-Datei als Referenz angegeben, im zweiten hingegen ist ein Pfad spezifiziert. Die Prozessressource *Shape* ist nämlich von Typ Pfad und in dem *CutPath*-Attribut ist

Abbildungen 12.5 zwei verschiedene Möglichkeiten zur Beschreibung von Faltschachtelkonturen

```
<ShapeDef Class="Parameter" ID="_4711" Status="Available">
    <FileSpec URL="file://Fileserver1/CFF2/Faltschachtel.cf2"/>
</ShapeDef>
```

```
<ShapeDef>
    <Shape ShapeType="Path" DDESCutType="101"
     CutPath=" 28 28 m 10 72 l 20 140 144 150 144..."/>
</ShapeDef>
```

der PDF-Pfad definiert. Mit dem Operator *m* wird der Cursor auf eine spezielle Position bewegt, mit dem Operator *l* von dort eine Gerade gezogen und schließlich mit dem Operator *c* am Ende der Geraden eine Kurve angefügt (*m* für *moveto*, *l* für *lineto*, *c* für *curveto*). Die Operanden werden dabei immer vor dem Operator aufgeführt. Das Attribut *DDESCutType* gibt gemäß des ANSI-Standards DDES3 [7] die Art des Schneidens oder Perforierens an.

Eine Verpackung kann natürlich aus mehreren Teilen bestehen, wie es zum Beispiel bei Deckelschachteln der Fall ist. Insofern können auch mehrere *ShapeDef*-Ressourcen Input des *DieLayoutProduction*-Prozesses sein. Außerdem können natürlich auch unterschiedliche Verpackungen auf einem Bogen vernutzt werden. In der Ressource *DieLayoutProductionParams* können einige Step&Repeat-Parameter vorgegeben werden, wie die Anordnung und die Zwischenräume zwischen den Nutzen. Der *DieLayoutProduction*-Prozess bestimmt dann die Vernutzung von einer oder mehreren Konturen und erzeugt die Ressource *DieLayout*, in dem die CAD-Daten beschrieben sind, die für den Stanzformbau und für das Ausschießen benötigt werden. Auch hier wird im Wesentlichen nur der Speicherort der externen Datei registriert. Schlussendlich muss eine physische Stanzform (*Tool*) hergestellt werden, was in dem Prozess *DieMaking* beschrieben wird. Es werden, wie oben erwähnt, Stanzformen, Gegenzurichtungen und Ausbrech- und Nutzentrennwerkzeuge hergestellt. Aber auch Prägewerkzeuge für die Blind- oder Heißfolienprägungen sind Tools, auf die hier jedoch nicht weiter eingegangen wird.

Abbildung 12.6
Vernutzung durch CAD-Daten

Bei den Arbeitsschritten der grafischen Produktion (II) muss ebenfalls ein Layout für das Ausschießen und die Plattenbe-

```
<StrippingParams Class="Parameter" ID="_001" Status="Available"
WorkStyle="Simplex">
    <Position MarginBottom="36.0" MarginLeft="36.0"
     RelativeBox="0.00000 0.00000 1.00000 1.00000" />
    <BinderySignatureRef rRef="_4711" />
    <StripCellParams Mask="DieCut" TrimSize="744.12 752.04" />
    …
</StrippingParams>

<BinderySignature BinderySignatureType="Die" Class="Parameter" ID="_4711"
 Status="Available">
    <DieLayout>
        <FileSpec URL=" file://Fileserver1/DDES3/6-Faltschachteln.dd3" />
            <Station StationAmount="6" StationName="DES1" />
    </DieLayout>
</BinderySignature>
```

lichtung erzeugt werden, gewissermaßen als Gegenstück zu dem *DieLayout* der Stanzformherstellung. In Kapitel 8 und 9 wurden der hierfür nötige *Stripping*-Prozess vorgestellt (siehe Abbildungen 8.7, 8.9 und 9.8). Im Akzidenzbereich werden die Position eines Falzbogens bzw. die Positionen mehrerer Falzbogen auf einem Druckbogen festgelegt und in dem Unterelement zusätzlich noch das Ausschießschema notiert. Bei der Produktion von Verpackungen benutzt man hingegen anstatt eines Ausschießschemas entweder die vernutzen Stanzformkonturen oder ein Step&Repeat-Muster zur Anordnung der Content-Daten. Diese Informationen hierzu können von gewissen MIS-Systemen zur Verfügung gestellt werden, die ihrerseits die Angaben von der CAD-Abteilung erhalten (allerdings bisher noch nicht über JDF).

Der JDF-Code in Abbildung 12.6 zeigt ein Beispiel, bei dem in der *BinderySignature*-Ressource ein Verweis auf die externen CAD-Daten eingetragen sind. In den *StrippingParams* wird zunächst die Position der Stanzkontur auf dem Druckbogen angegeben, ähnlich wie in dem Beispiel in Abbildung 8.8, nur das dort mehrere Falzbogen auf einen Bogen platziert wurden. Und wie vorher auch, enthält das Beispiel das Unterelement *StripCellParams*. Es enthält die Endformatbox sowie eine Angabe über den Clip-Pfad (*Mask*), der identisch zur Schneidkontur der

```
<BinderySignature ID="_4712" Status="Available" BinderySignatureType="Grid"
 NumberUp="2 3"...>
   <SignatureCell FrontPages="0 0 0 0 0 0 " Orientation="Left" />
</BinderySignature>
```

Abbildung 12.7
Step&Repeat von Etiketten

Stanzform ist. Die *BinderySignature*-Ressource kann entweder ein direktes Kind der *StrippingParams*-Ressource sein (wie in Beispiel 8.9) oder aber alternativ außerhalb der *StrippingParams* definiert werden. Im letzteren Fall muss natürlich die *StrippingParams* eine Referenz auf die *BinderyParams* enthalten, wie es in Abbildung 12.6 zu sehen ist. Die *BinderySignature* kennt drei Unterarten (*BinderySignatureType*), nämlich:

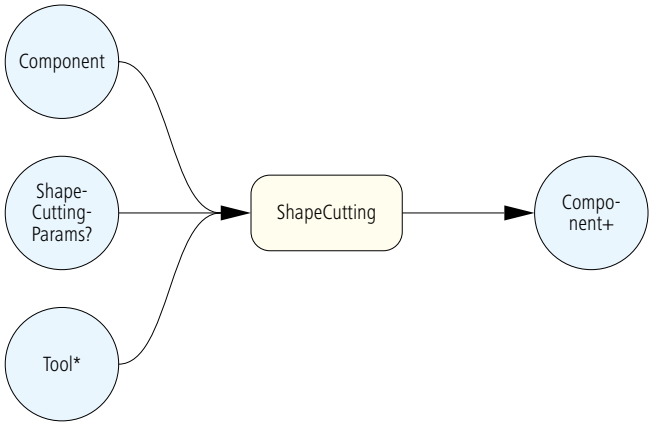
- Falzschema (*Fold*),
- Step&Repeat (*Grid*),
- Stanzkontur (*Die*),

wobei *Fold* als Default-Wert gesetzt ist. Insofern musste in den vorherigen Kapiteln dieses Attribut auch nicht unbedingt aufgeführt werden, jetzt hingegen schon. Wenn die Ressource vom Typ *Die* ist, muss sie (eine Referenz auf) das Element *DieLayout*

beinhalten. Dort ist dann die *URL* einer Datei („6-Faltschachteln.dd3") aufgeführt, die zur Beschreibung der Stanzkonturen dient. Außerdem ist in dem Beispiel noch die Anzahl der *Stations* angegeben, was der Anzahl gleicher Nutzen auf dem Druckbogen entspricht.

Abbildung 12.7 zeigt ein Beispiel für eine *BinderySignature*-Ressource, bei der die Nutzenanordnung über eine Step&Repeat-Matrix definiert wird. Der *BinderySignatureType* ist infolgedessen auch *Grid*, was mit „Gitter" ins Deutsche übersetzt werden kann. Mit dem Attribut *NumberUp* lässt sich die Anzahl der Spalten und Zeilen der Matrix festlegen. Jeder Matrixeintrag entspricht dabei einem Faltschachtelzuschnitt. Das gleiche Attribut lässt sich auch für den Akzidenzdruck anwenden, wenn ein Ausschießmuster zu definieren ist. Dann entspricht jeder Matrixeintrag einer Musterseite. Siehe Abschnitt 8.2, Abbildung 8.11. Mit dem Attribut *FrontPages* repräsentiert jede 0 einen Nutzen gleicher Orientierung, wobei der Bogen nur ein Typ von Etikett enthält (ansonsten müssten mehrere *BinderySignatures* definiert werden).

Abbildung 12.8
JDF-Modell Stanzen




## 12.2 Stanzen und Faltschachteln kleben

Beim Stanzen werden mittels eines Stanzautomaten Formstücke – die so genannten Zuschnitte – aus Druckbogen ausgestanzt. Je nach Konfiguration können dann in einer Ausbrechstation, der zweiten Station des Stanzautomatens, mittels Ausbrechwerkzeug die Stanzabfälle eliminiert werden. Anschließend werden noch die Nutzen getrennt, was inline oder offline geschehen kann.

Die genannten Vorgänge werden in JDF unter *ShapeCutting* zusammengefasst. Als wichtigste Input-Ressourcen dienen einerseits die im letzten Abschnitt diskutierten Werkzeuge (*Tool*) sowie die Druckbogen (*Component*). Abbildung 12.8 zeigt das Modell. Natürlich können (wie sonst auch immer) noch solche Input-Ressourcen hinzukommen, die für jeden Prozess-Knoten zulässig sind, wie beispielsweise *NodeInfo* oder *Device*. Weiter-

hin soll noch darauf hingewiesen werden, dass mit Stanzautomaten auch Blindprägungen vorgenommen und sogar Heißfolienprägemodule in die Maschinen integriert werden können. In diesen Fällen könnte der gesamte Vorgang durch einen kombi-

```
<JDF ID="_343" JobPartID="_1002.0" MaxVersion="1.3" Status="Part" Type="
 ShapeCutting " Version="1.3" >
   <ResourceLinkPool>
       <NodeInfoLink Usage="Input" rRef="_344" />
       <ComponentLink Usage="Input" rRef="172"…>
       <DeviceLink Usage="Input" rRef="_339" />
       <ComponentLink Usage="Output" rRef="_152"…>
       <ShapeCuttingParamsLink Usage="Input" rRef="_772">
          <Part SheetName="Faltschachtel-Druckbogen" SignatureName="SIG1" />
       </ShapeCuttingParamsLink>
   </ResourceLinkPool>
   <ResourcePool>
       <ShapeCuttingParams Class="Parameter" ID="_772"
        PartIDKeys="SignatureName SheetName BlockName" SheetLay="Left"
        Status="Available"…>
          <ShapeCuttingParams SignatureName="SIG1">
             <ShapeCuttingParams SheetName="Faltschachtel-Druckbogen">
                <ShapeCuttingParams BlockName="Block1" />
                <ShapeCuttingParams BlockName="Block2" />
                <ShapeCuttingParams BlockName="Block3" />
                <ShapeCuttingParams BlockName="Block4" />
                <ShapeCuttingParams BlockName="Block5" />
                <ShapeCuttingParams BlockName="Block6" />
             </ShapeCuttingParams>
          </ShapeCuttingParams>
       </ShapeCuttingParams>
   …
   </ResourcePool>
</JDF>
```

nierten Prozess beschrieben werden, der *Embossing* (Prägen) und *ShapeCutting* (Stanzen) umfasst.

Abbildung 12.9
Stanzprozess

Das Beispiel eines ShapeCutting-Prozesses, das ausschnittsweise in 12.9 zu sehen ist, ist noch einfacher aufgebaut als das Modell 12.8. Es enthält nämlich keine Input-Ressource Tool, weil, wie am Anfang von Abschnitt 12.1 bereits gesagt, derzeit noch kein Produktionssystem diese Information bereitstellt. Ein *Blockname* identifiziert im Akzidenzbereich einen *Cutblock* für den *Cutting*-Prozess (siehe Abbildung 11.3), für den Verpackungsbereich identifiziert es einen Nutzen, der ausgestanzt wird. Die Druckprozess *ConventionalPrinting* erzeugt als Output eine *Component*, das heißt einen Stapel Druckbogen mit dem *SheetName* „Faltschachtel-Druckbogen“, aus denen dann die sechs Faltschachteln (Blöcke) ausgestanzt werden.

Die Zuschnitte werden in der Produktion dann in den Einleger einer Faltschachtelklebemaschine gelegt. Dort werden dann die Rilllinien mittels eines Vorbrechers gefaltet, zurückgefaltet und anschließend flach weitergeleitet. Das Vorbrechen dient dazu, dass die Faltschachteln später in den Abpackmaschinen leichter geöffnet werden können. Anschließend werden meist Klebelaschen oder Klebekanten beleimt. Danach werden die Faltschachteln geschlossen. Schließlich werden sie zusammengepresst, verklebt und gesammelt beziehungsweise einem Aggregat zugeführt, welches die Schachteln in Kartons umverpackt.

Faltschachtelklebemaschinen sind häufig für die Anwender individuell maßgeschneidert. So gibt es spezielle Module für Faltschachteln, die im gewissen Sinne besonders sind, wie zum Beispiel solche mit konischen Formen oder auch Pralinenschachteln. Hierbei werden durchaus Komponenten unterschiedlicher Hersteller verbaut. Manchmal werden Maschinen sogar speziell für einen Einsatzzweck konstruiert und auch sonst ist das Umrüsten einer Faltschachtelklebemaschine in der Regel mit vielen mechanischen Einstellungen verbunden, die der Anwender zu leisten hat. Für den möglichen Fall eines Wiederholauftrages werden dann die Einstellwerte der Maschine schlicht notiert. Trotzdem gibt es aber bereits Maschinen, die sich durch Stellmotoren und Software bis zu einem gewissen Grad automatisch voreinstellen lassen, also nicht über Handräder, Hebel und Inbusschlüssel, wie es klassischerweise der Fall ist.

Abbildung 12.10
Voreinstelldaten für eine Faltschachtelklebemaschine

Die Einstellarbeiten auf einer eher typischen Faltschachtelklebemaschine verläuft im Groben wie folgt. Abhängig von Format

```
<BoxFoldingParams BlankDimensionsX="43 388 502 848 960"
 BlankDimensionsY="43 78 156 465 543 578 621" BoxFoldingType="Type01"
Class="Parameter" ID="_771" PartIDKeys="SignatureName SheetName BlockName"
Status="Available">
    <BoxFoldingParams SignatureName="SIG1">
        <BoxFoldingParams SheetName="Faltschachtel-Druckbogen">
            <BoxFoldingParams BlockName="Block1" />
                <BoxFoldAction Action="LongPreFoldLeftToRight"
                 FoldIndex="0 -1" />
                <BoxFoldAction Action="LongPreFoldRightToLeft"
                 FoldIndex="2 -1" />
                <BoxFoldAction Action="LongFoldLeftToRight" FoldIndex="1 -1" />
                <BoxFoldAction Action="LongFoldRightToLeft" FoldIndex="3 -1" />
        </BoxFoldingParams>
    </BoxFoldingParams>
</BoxFoldingParams>
```

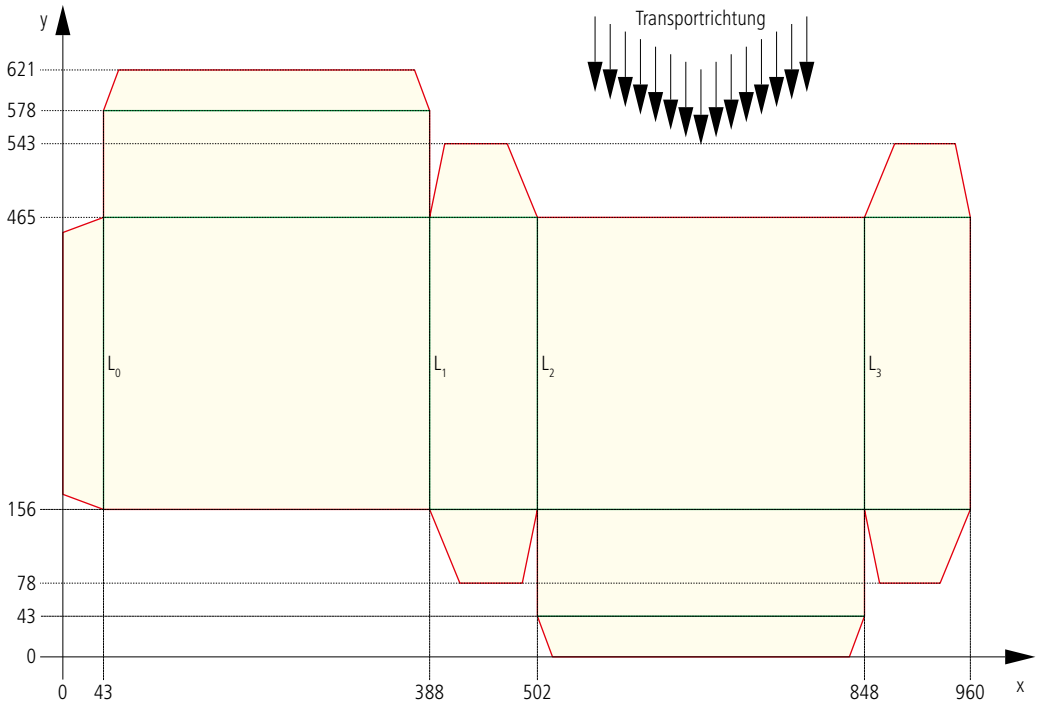



Abbildung 12.11
Standardfaltschachtelform vom Typ 01

und Dicke der zu verarbeitenden Zuschnitte müssen diverse Einstellungen beim Einleger vorgenommen werden. Des Weiteren werden eine oder mehrere Vorbrechstation(en) gemäß Zuschnittstyp und -dicke eingestellt. Dann müssen in dem (den) Leimwerk(en) Breite und Länge der Leimspur und die Klebstoffauftragsmenge abgestimmt werden. Natürlich werden auch die Positionen der Leimwerke festgelegt. Die anschließende Faltung verlangt dann ebenfalls eine spezifische Einstellung, genau wie das anschließenden Pressen und Sammeln der Faltschachteln.

Eine JDF/JMF-Vernetzung einer Faltschachtelklebemaschine kann – wie bei den anderen Maschine auch – mehrfachen Nutzen bringen:

- Die Auftragsdaten werden direkt zum Gerät geleitet.
- Die Fertigungsdaten (BDE) werden unmittelbar wieder an ein Produktionssystem zurückgemeldet.
- Die Voreinstellungsdaten können übernommen werden, so dass die Rüstzeiten verkürzt werden.

Der dritte Punkt wird allerdings (noch) eine relativ geringe Rolle spielen, da nur wenige Maschinen mit automatischer Voreinstellung auf dem Markt sind. Mit Blick auf die Zukunft sollen hier die JDF-Möglichkeiten diesbezüglich dennoch genauer

beleuchtet werden.

Der Prozess *BoxFolding* repräsentiert die auf einer Faltschachtelklebemaschine ausgeführten Arbeiten. Input-Ressourcen von BoxFolding sind im Wesentlichen die *Components* (Zuschnitte), die als Ergebnis des Stanzprozesses entstanden sind, und die *BoxFoldingParams*, die Parameter, welche die Maschine voreinstellen kann. In Abbildung 12.10 ist ein Beispiel zu sehen. Dort werden Falzaktionen (*BoxFoldActions*) beschrieben, die sich im vorliegenden Fall auf eine Standard-Faltschachtel beziehen. Insgesamt sind in der JDF-Spezifikation zehn Standards vordefiniert, darüber hinaus können aber auch eigene Faltschachtelarten festgelegt werden. Der *BoxFoldingType* hat hier den Wert *Type01*, was einer Falzschachtelkonstruktion der Art entspricht, wie sie in Abbildung 12.11 zu sehen ist. Die Werte der Attribute *BlankDimensionsX* und *BlankDimensionsY* sind ebenfalls in Abbildung 12.11 eingezeichnet. Der Nullpunkt ist unten links und die Zahlen sind in DTP-Punkten angegeben. Natürlich können diese Werte variieren und beschreiben hier nur eine ganz konkrete Faltschachtel dieses Typs. Mit dem Schachteltyp sind auch schon die Faltoperationen bestimmt, wobei zu beachten ist, dass die Zuschnitte mit der bedruckten Seite nach unten in den Einleger gelegt werden:

- an Linie $L_0$ muss von links nach rechts vorgebrochen werden,
- an Linie $L_2$ muss von rechts nach links vorgebrochen werden.

Und nach dem Leimauftrag von unten auf der linken Lasche muss

- an Linie $L_1$ von links nach rechts und
- an Linie $L_3$ von rechts nach links gefaltet werden.

Die Leimspur (*GlueLine*) ist in diesem Beispiel nicht definiert, die *BoxFoldAction*-Elemente entsprechen aber genau der Reihenfolge der Faltoperationen. Beim *FoldIndex* ist hier immer nur der erste Wert ausschlaggebend, der Wert „-1“ besagt nur, dass das Falten in Transportrichtung über die gesamte Länge erfolgen soll.

## 12.3 Barcode

Auf dem ersten Blick scheint ein Barcode einfach nur ein wei-

```
<LayoutElementProductionParams Class="Parameter" ID="_300" Status="Available">
    <LayoutElementPart>
        <BarcodeProductionParams>
            <IdentificationField Encoding="BarCode" EncodingDetails="EAN_13"
             Value="0123456789128" />
        </BarcodeProductionParams>
    </LayoutElementPart>
</LayoutElementProductionParams>
```

Abbildung 12.12
Definition eines Barcodes

teres grafisches Element auf der Verpackung zu sein, das beim Design zu beachten und natürlich auch zu realisieren ist. Und in der Tat kann man mit handelsüblichen Layout-Programmen Barcodes erzeugen und auf das Grafikdesign platzieren. Die Generierung von Strichcodes wird zwar meist nicht direkt von den Layout-Programmen unterstützt, aber es gibt Plug-Ins für diesen Zweck. Alternativ kann man Stand-alone Programme einsetzen, in denen die Ziffern für einen Barcode eingegeben werden können und die daraus dann einen entsprechenden Barcode-Font kreieren. Solch ein Font beinhaltet dann nur ein einziges Zeichen, das genau dem vorher definierten Barcode-Muster entspricht. Schließlich lässt sich der Font in ein Betriebssystem installieren und anschließend von jedem beliebigen Layout-Programm verwenden. Da im Verpackungsbereich häufig alle Schriften in allgemeine grafische Pfade umgewandelt werden, muss dann der Barcode-Font auch nicht an den Druckdienstleister weitergegeben werden. Damit könnte also der Barcode für eine Verpackung bereits beim Grafikdesign erstellt werden und die Druckereien hätten damit nichts zu tun.

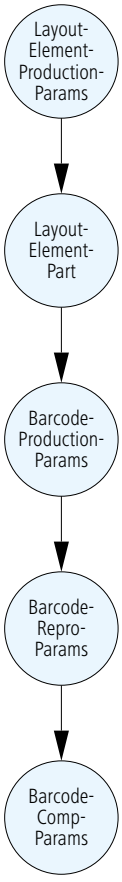



Abbildung 12.13
„Ressourcen-Stammbaum“ für den Prozess LayoutElementProduction

Das Verfahren hat aber einen Nachteil: Vor allem im Flexodruck gibt es eine deutliche Strichverbreiterung, die dazu führen kann, dass der Barcode nicht mehr korrekt elektronisch lesbar ist. Das kann zu einem immensen Schaden führen, weil dadurch nicht nur die Verpackung unbrauchbar wird, sondern sogar das verpackte Produkt, wenn der Fehler erst nach dem Abpacken bemerkt wird. Also muss vorab eine Strichbreitenkompensation für den Barcode eingerechnet werden. Die Höhe der Kompensation hängt allerdings von verschiedenen Faktoren ab (Drucktechnologie, Bedruckstoff, Druckfarbe, Verschnitt…), so dass nur Spezialisten der technischen Verpackungsdruckvorstufe Vorgaben machen können. Deswegen werden im Grafikdesign in der Regel Barcodes nicht gesetzt, sondern erst später beim Platten- oder Druckdienstleister mit speziellen Programmen der Verpackungsdruckvorstufe eingefügt.

Damit stellen Barcode-Details Informationen dar, die vom Auf-

traggeber bis zur Produktion weitergeleitet werden müssen. Diese Kommunikation kann mit Hilfe von JDF erleichtert werden.

Die Druckerei übernimmt den EAN-Code also als Ziffernfolge vom Auftraggeber und trägt diese in ein (geeignetes) MIS-System ein. Von dort wird die Information im JDF an eine Verpackungsdruckvorstufensoftware weitergeleitet, so dass die Mitarbeiter dort die Codes nicht mehr selber eintragen müssen (was immer das Risiko der Fehleingabe birgt). Einen entsprechenden Eintrag findet man in Abbildung 12.12. Die *LayoutElementProductionParams* bilden eine Input-Ressource des Prozesses *LayoutElementProduction*. In diesem Prozess wird entweder die Herstellung von Layout-Komponenten wie Bilder, Grafiken, Text oder auch Barcodes beschrieben oder aber auch die Erzeugung von kompletten Seiten/Nutzen in einem Layout-Programm. Andererseits wird ein MIS solche detaillierten Prozesse nicht definieren, sondern stattdessen eine *GrayBox,* wie die bereits häufiger erwähnte *PrePressPreparation*, oder aber auch die *ContentCreation*. Im Unterelement – in der Generationsfolge ein Enkel – findet man dann die *BarcodeProductionParams*, welche die Beschreibung der Parameter für die Herstellung von Barcodes enthält. Jeder Barcode ist dann in dem Unterelement IdentificationField aufgelistet. In dem vorliegenden Beispiel ist der Typ des Strichcodes (*EncodingDetails*) und der Wert (*Value*) angegeben. Diese Ressource kann fakultativ noch weitere Unterressourcen beinhalten beziehungsweise Referenzen auf weitere Ressourcen anlegen, wie in Abbildung 12.13 zu sehen ist. In die Ressource *BarcodeReproParams* können dann beispielsweise Informationen über Höhe und Breite des Barcodes und in *BarcodeCompParams* die nötigen Strichbreitenkompensationen hineingeschrieben werden, sobald sie bekannt sind.

Ein weiterer Vorteil von Barcode-Metadaten liegt in einer möglichen Qualitätssicherung. Der Wert kann mit einer (Produkt-) Datenbank abgeglichen werden, sodass sichergestellt ist, dass der Barcode wirklich zum Produkt passt. Das wäre bei einem platzierten Grafikelement nicht so einfach möglich.

# 13 JDF/JMF-Projekte

Während wir uns in den letzten zwölf Kapiteln auf einigermaßen gesicherte Fakten berufen konnten, bewegen wir uns nun auf sehr dünnem Eis. JDF-Projekte sind sehr unterschiedlich und die Projektmanagementmethoden sind nicht immer gleich. So ist es schwer, allgemeingültige Regeln oder gar Tipps für eine Neueinführung oder Erweiterung von JDF-Workflows zu geben. Auch sind unsere Anmerkungen sicherlich zumindest teilweise von unseren persönlichen Erfahrungen geprägt. Schließlich möchten wir auch die Implementierung weder zu rosarot noch zu düster zeichnen. Um es uns etwas zu erleichtern, unterteilen wir JDF-Projekte in die drei Typen:

- Workflow-Implementierung mit Modulen eines Anbieters
- Workflow-Implementierung mit Modulen mehrerer Anbieter
- Workflow-Implementierung mit Modulen mehrerer Anbieter und eigenen Software-Anpassungen

Doch bevor wir diese drei Fälle in den folgenden Abschnitten diskutieren, möchten wir noch ein paar generelle Bemerkungen anbringen, auch wenn das als die sprichwörtliche Sonntagspredigt empfunden werden könnte.

Vor allem zu Beginn eines solchen Projektes ist es wichtig, die Erwartungshaltung an den JDF-Workflow zu dämpfen. Was man von einem JDF-Workflow erwarten kann und was nicht, muss nämlich sehr differenziert betrachtet werden und hängt maßgeblich von den Voraussetzungen des jeweiligen Betriebes ab. Je weniger man über das Thema weiß, desto schwieriger ist es natürlich, die komplexe Situation richtig einzuschätzen, zumal, wie so häufig, der Teufel im Detail steckt. Manchmal werden Wunderdinge erwartet, wie zum Beispiel, dass die derzeitig schlechte Betriebsorganisation sich plötzlich wie von selbst in ordentliche Bahnen lenkt. Oder die Mitarbeiter im Drucksaal sind enttäuscht, dass sie vom vieldiskutierten „JDF-Workflow“ gar nichts mitbekommen. Klar, sie erhalten jetzt ein paar Voreinstellungswerte, aber war das schon alles? Und man muss zugeben, dass viele der schönen Verbesserungen, die heute mit JDF/JMF erreicht werden, auch vorher schon mit herstellerspezifischen Lösungen möglich waren. Denn es gab schon früher BDE und auch abteilungsübergreifenden Datenaustausch von Auftragsdaten und Voreinstellungswerten. Trotzdem ist aber unserer Meinung nach das JDF/JMF-Konzept zukunftsweisend und viel breiter und dazu noch herstellerübergreifend angelegt als alles bisher Dagewesene. Das Potential ist sicher noch nicht

ausgeschöpft.

Sich über die Erwartungshaltungen klar zu werden heißt mit anderen Worten, die Argumente für das Engagement in ein JDF/JMF-Projekt zu definieren. Sollen die Kostentransparenz erhöht, die interne und/oder die externe Kommunikation vereinfacht, die Fehlerhäufigkeit reduziert, die Produktionsschritte verbessert, eventuell sogar automatisiert, die Planung optimiert und die Kosten gesenkt werden? Oder alles gleichzeitig? Das Problem ist hier vor allem, dass man es bei diesen Schlagwörtern gern belässt und nicht ins Details geht, wo und wie genau welche Kommunikation verbessert werden soll und so weiter. Sind die Gründe zu abstrakt, werden nebulöse Hoffnungen daran geknüpft, die dann natürlich nicht erfüllt werden können.

Es müssen also klare Ziele definiert werden, sowohl in technischer als auch in betriebswirtschaftlicher Hinsicht. Basis hierzu ist die Analyse des Ist-Zustands. Die Untersuchung des Workflows kann dabei schon wertvoll für sich allein sein, selbst wenn als Ergebnis möglicherweise herauskommt, dass die JDF-Vernetzung für den Betrieb nicht geeignet ist. Denn bei der Analyse werden unter Umständen Mängel entdeckt und zugehörige Lösungen gefunden, die in eine ganz andere Richtung gehen. Ansonsten hilft die Analyse, realistische Ziele für ein JDF-Engagement zu finden. Kurz: Die Druckerei sollte ein Pflichtenheft zu den geplanten JDF-Schnittstellen schreiben und sich die Erfüllung des Geforderten von den potentiellen Zulieferern bestätigen lassen.

Ein JDF/JMF-Projekt muss längerfristige angesehen werden (Stichwort „Skalierbarkeit“). Zunächst macht es einfach schlicht mehr Arbeit und bringt erst später Nutzen für die Firma. Deswegen sind sich auch alle einig, dass man ein solches Vernetzungsvorhaben in mehreren Etappen angeht und Zwischenziele definiert. So könnten die Meilensteine (die natürlich noch viel konkreter festgelegt werden müssten) lauten:

- Vernetzung von MIS zur Vorstufe
- Vernetzung von MIS/Vorstufe zum Drucksaal
- BDE von Produktion zum MIS
- Integration ausgewählter Weiterverarbeitungsmaschinen in das Netzwerk
- Kundenanbindung

In der Regel stellt das MIS den Beginn einer JDF-Vernetzung

dar. Wo genau man dann fortfährt, hängt im Wesentlichen von der vorhandenen Ausstattung ab. Denn man wird sich beispielsweise keine neue Falzmaschine mit Stellmotorik kaufen, nur weil diese (sinnvoll) in eine JDF-Umgebung passt. Das wäre wirtschaftlich sicher nicht richtig. Stattdessen sollten eher vorhandene Maschinen, zu denen JDF-Schnittstellen erhältlich sind, aufgerüstet oder bei fälliger Neuanschaffung auf deren JDF-Kompatibilität geachtet werden.

In allen drei oben vorgestellten Fällen erfordert es die aktive Teilnahme der Anwender, wobei der Grad der nötigen Mitwirkung gemäß der Listenfolge steigt. Wir möchten James Harvey, Geschäftsführer (executive director) der CIP4-Organisation zitieren [21]:

"Implementing process automation, even with JDF, requires active participation by the printer's staff. Different vendors have different philosophies for how they are implementing process automation, and printers may find themselves acting as project managers to work out issues that may arise when multiple systems are integrated."
(Prozessautomatisierung zu implementieren verlangt – sogar im Fall von JDF – eine aktive Teilnahme von Druckereimitarbeitern. Verschiedene Hersteller verfolgen unterschiedliche Konzepte bezüglich der Implementierung von Prozessautomatisierung, und Druckereien befinden sich in einer Projektmanagerrolle, um Schwierigkeiten auszuräumen, die bei der Integration von mehreren Systemen auftreten können.)

Die Beteiligung sollte dabei nicht nur von oben angeordnet werden, sondern das Betriebsklima sollte möglichst von Teamgeist und Innovationsfreude geprägt sein. Denn Workflow-Managementsysteme (mit oder ohne JDF) verändern und verwischen die Grenzen der Verantwortlichkeit und der Abteilungen. So findet in einer JDF-Umgebung typischerweise eine Kompetenzverlagerung von der Vorstufe hin zum Auftragsmanagement statt, und Mitarbeiter müssen dann ggf. ihren Arbeitsplatz wechseln. Die Erhöhung der Transparenz hat natürlich auch die objektivere Beurteilung von Arbeitsleistung zur Folge. Mitarbeiter werden das Projekt nicht unterstützen, wenn ein Klima von Misstrauen und Konkurrenz herrscht. Schon allein die Ist-Analyse kann Widerstände bei den Mitarbeitern hervorrufen. Schließlich muss auch möglichst einvernehmlich entschieden werden, wer auf welche digitalen Daten Zugriff hat und auch, wer nur Lesezugriff hat oder zusätzlich noch schreiben darf.

## 13.1 Workflow-Implementierung mit Modulen eines Anbieters

Natürlich ist der Aufwand zum Einrichten eines JDF-Workflows geringer, wenn nur ein Hersteller beteiligt ist und nicht mehrere. Trotzdem ist er nicht zu vernachlässigen. Denn die Anpassung an die Produktionsweisen und Geräte des Betriebes muss natürlich immer erfolgen. Dieses kann und sollte auch nicht nur in Eigenregie vom Hersteller durchgeführt werden. Denn das hieße, dass man bei jeder kleinen Änderung den Service in Anspruch nehmen müsste.

Ein WMS-Administrator in der Druckerei muss also beispielsweise neue Geräte in den Workflow einbinden, Benutzer verwalten und Defaultwerte für die Produktion setzen können. Da die unterschiedlichen JDF-Module meist auf mehreren Servern verteilt ablaufen, muss ein IT-Verantwortlicher (oder mehrere) für diese Rechner und für das Netzwerk zuständig sein und deren Zugriffsrechte untereinander regeln können. In der Praxis kommt es durchaus vor, dass sich irgendein benötigter Dienst plötzlich verabschiedet oder Rechner wegen eines Betriebssystem-Updates automatisch neu gestartet werden und dergleichen. Nur wenn jeder Mitarbeiter weiß, wie man auf solche Störungen reagieren kann, hat man für längere Zeit Freude an dem JDF-Workflow. Für eine stabile Konfiguration werden also ein IT-Administrator und ein Workflow-Administrator benötigt. Besser ist es allerdings, wenn dieses Wissen in einer Hand liegt. Da diese Position aber ungeheuer wichtig ist, muss mindestens eine zweite Person ebenfalls diese beiden Systemaufgaben beherrschen (Redundanz). Je nach Umfang der JDF-Vernetzung kann das jedoch eine recht schwierige Aufgabe werden. Bei einem weitreichenden Workflowsystem die komplette Übersicht auf Administrator-Niveau zu behalten, ist nämlich keine leichte Aufgabe für den verantwortlichen Mitarbeiter. Und selbst bei den Herstellern wird man eher auf Spezialisten stoßen, als auf technisch versierte Generalisten.

Auch wenn nur ein Hersteller für die Workflow-Installation verantwortlich ist, wird sich eine JDF-Implementierung im Laufe der Zeit ändern und zumeist auch in Abschnitten realisiert werden, wie vorher ausgeführt. Denn die JDF-Workflowsysteme befinden sich noch im Wandel der ständigen Anpassung und Soft-

ware-Patches, Updates und Erweiterungen müssen regelmäßig eingespielt und anschließend getestet werden. Hier kann es hilfreich sein, mit Virtualisierungen oder Platten-Images zu arbeiten, so dass im Fehlerfall leicht wieder auf die ältere Version zurückgegriffen werden kann.

Wenn ein JDF-Workflow in einem Betrieb neu installiert werden soll, dann muss die Konfiguration möglichst vorab beim Hersteller oder einem seiner Referenzkunden überprüft werden, bei der typische Aufträge der Druckerei getestet werden. Denn Automatisierungen mögen für ein Produkt oder eine Produktgruppe sehr gut funktionieren, jedoch für andere gar nicht. Das kann unterschiedliche Gründe haben. Bietet beispielsweise ein MIS, das nur auf einen festen Falzartenkatalog zugreift, ein bestimmtes Ausschießschema nicht an, so kann es natürlich Informationen dazu auch nicht an die Produktion weitergeben. Oder gewisse Einzelheiten, die eine Produktionsart beschreiben, werden schlicht und einfach nicht oder aber fehlerhaft weitergegeben. Irgendwelche Matrizen, die zeigen, welche Systeme miteinander kompatibel sind, sind nur eine erste, grobe Bewertung. Im Detail muss die Kompatibilität dann trotzdem noch geprüft werden. Insofern wird man vielleicht auch Workflow-Automatismen (wie zum Beispiel eine automatische Standbogengenerierung) nur für einen Teil der Aufträge nutzen, andere werden nach wie vor viele manuelle Eingriffe erfordern.

Die Einführung eines JDF-Workflows verlangt, wie schon mehrfach angesprochen, wohldefinierte Produktionsregeln. Diese können aber auch verhindern, dass kurzfristige Veränderungen an Aufträgen einfach und schnell abgewickelt werden. Nachträgliche Änderungen in einen stark geregelten und automatisierten Workflow einzubringen, ist, wenn mal so will, ein Widerspruch in sich, allerdings häufige Notwendigkeit in der alltäglichen Praxis. Hier kann mit dem JDF-Workflow der Änderungsaufwand höher werden als früher mit der Lauftasche, je nachdem, wie weit der Auftrag bereits abgearbeitet wurde.

Bei einem komplexen JDF-Workflow-System müssen viele Rechner, das Netzwerk und diverse Programme gleichzeitig funktionieren. Da gibt es natürlich Ausfälle, und die Fehlereingrenzung ist nicht einfach. Denn ein Hardwaredefekt an einem Netzwerk-Switch zum Beispiel bewirkt einen Fehler in der Anwendung und eine entsprechende (irreführende) Fehlermeldung. Und so kann es durchaus eine Zeitlang dauern, bis ein Fehler wieder behoben werden kann. Um jedoch die Produktion am Laufen zu halten, kann es hilfreich sein, wenn ein Notfall-Workflow definiert ist.

Beispielsweise sollte der Drucker in der Lage sein, einen Plattensatz zu drucken, ohne dass Voreinstelldaten zur Verfügung stehen. Das klingt vielleicht wie selbstverständlich, ist es aber nicht, denn die Implikationen sind manchmal sehr versteckt. Wenn beispielsweise die Positionen der Registermarken nicht mit in dem JDF übergeben werden, funktioniert möglicherweise die Inline-Registersteuerung in der Druckmaschine nicht mehr, und so weiter. Oder eine an einen JDF-Workflow angebundene Digitaldruckmaschine sollte notfalls auch Jobs direkt verarbeiten können, wenn die JDF-Kommunikation Probleme bereitet.

Nicht jeder Automatismus, der denkbar ist und angeboten wird, muss auch immer sinnvoll sein. Das gilt natürlich vor allem hinsichtlich der Wirtschaftlichkeit. Aber auch in technischer Hinsicht gibt es diesbezüglich Einschränkungen. Wird der Start nachfolgender Prozessschritte von Freigaben durch den Kunden, Verantwortliche der Produktion oder von bestimmten Events abhängig gemacht, kann dies den Produktionsablauf unnötig hemmen. So könnte beispielsweise die Bereitstellung eines Auftrages am Leitstand der Druckmaschine von der Fertigstellung der Druckplatten abhängig gemacht werden, das heißt, die JDF-Daten mit den Parametern für den Druckjob würden erst dann geschickt oder in den Hotfolder kopiert, wenn das MIS beziehungsweise das Produktionssystem eine Information über die Fertigstellung der entsprechenden Druckformen vom Plattenbelichter erhalten hat. Diese völlig vernünftige Regelung macht allerdings dann Probleme, wenn die Rückmeldung des Plattenbelichters – aus welchen Gründen auch immer – gestört ist. Dann ist zwar der Plattensatz vorhanden und im Prinzip auch alle Voreinstellungsdaten für die Druckmaschine und trotzdem geht es nicht weiter, weil entweder das System oder zumindest die Druckmaschine nicht die entsprechende Information bekommen hat.

## 13.2 Workflow-Implementierung mit Modulen mehrerer Anbieter

Alles, was im letzten Abschnitt gesagt wurde, gilt natürlich auch insbesondere für den Fall, dass mehrere Anbieter an einem JDF-Workflow beteiligt sind. Ein paar Gesichtspunkte ergeben sich aber zusätzlich.

Die Schnittstellenproblematik ist in der Digitaltechnik allgemein bekannt: Erzeugt ein Software-Modul eines Herstellers ein Da-

tenformat für ein Programm eines anderen, gibt es keine klaren Verantwortlichkeiten, wenn die Kommunikation nicht funktioniert. Das gilt für JDF wie für jedes andere Format auch. Da JDF und JMF aber noch verhältnismäßig neu sind (im Vergleich beispielsweise zu PDF) wird man bei diesen Formaten eher auf Inkompatibilitäten stoßen. Vor allem bei der Neuinstallation und auch bei Versions-Updates ist eine gründliche Untersuchung vorab zu empfehlen. Das ist aber im Wesentlichen nur für den JDF-Datenaustausch möglich und auch nur dann, wenn das JDF über einen Hotfolder weitergereicht wird und nicht in einem MIME-Paket verpackt über HTTP geschickt wird. Man könnte die Pakete zwar abfangen, dies ist jedoch etwas aufwändig. Das gleiche gilt für JMF-Nachrichten. Manchmal kann man bei Workflow-Komponenten einstellen, ob die Kommunikation über Dateien oder HTTP verlaufen soll. Hier sollte man sich für die Dateivariante entscheiden, wenn man Schnittstellen untersuchen möchte.

Hat man die JDF-Geräte (noch) nicht installiert, deren Kommunikation untereinander man prüfen möchte, so muss man sich JDF-Daten von einem Hersteller erzeugen lassen und den nächsten bitten, sie wieder einzulesen. Im anderen Fall kann man natürlich die Schnittstelle direkt untersuchen. In beiden Fällen wird man bei Kompatibilitätsproblemen versuchen, die JDF-Daten zu analysieren. Viele Hersteller haben auch *White-Papers*, die spezifizieren, welche Anforderungen eine JDF-Datei genügen muss, um erfolgreich von der Software eingelesen und verarbeitet zu werden. Wenn nicht, kann man sich auch vom Hersteller der JDF-importierenden Software Beispieldaten geben lassen, die erfolgreich gelesen werden.

Eine Untersuchung der JDF-Datei könnte mit der Überprüfung auf Gültigkeit beginnen (siehe Abschnitt 5.2). Hierzu kann man auf der CIP4-Webseite unter „Technical Resources“ den Menüpunkt „CheckJDFServer“ auswählen und dort seine JDF-Datei mit dem Serverdienst „CheckJDF“ einlesen. Ein Bericht zeigt dann erste mögliche Fehler an, wie zum Beispiel fehlende Attribute, die aber obligatorisch sind, oder nicht auflösbare Links. Optional können auch private Erweiterungen als Fehler markiert werden. An gleichen Stelle kann man auch den Serverdienst „FixJDF“ auswählen und eine JDF-Datei reparieren und auf einen definierten Versionsstand bringen. Die automatische Reparatur hat aber erwartungsgemäß ihre Grenzen. Ist man CIP4-Mitglied kann man sich auch unter „Technical Resources“, „Downloads“, „Internal Source“ das Programm „CheckJDF.exe“ und eine zusätzliche dll-Datei herunterladen. Auch mit dem „JDFEditor“,

den man von der CIP4-Webseite frei herunterladen kann, lassen sich JDF-Dateien validieren (siehe Abbildung 13.1).

Es kann auch hilfreich sein, die Werte des Attributs *ICSVersions* anzuschauen und vielleicht zu überlegen, ob die JDF-Programme auf unterschiedlichen ICS-Versionen basieren. Das

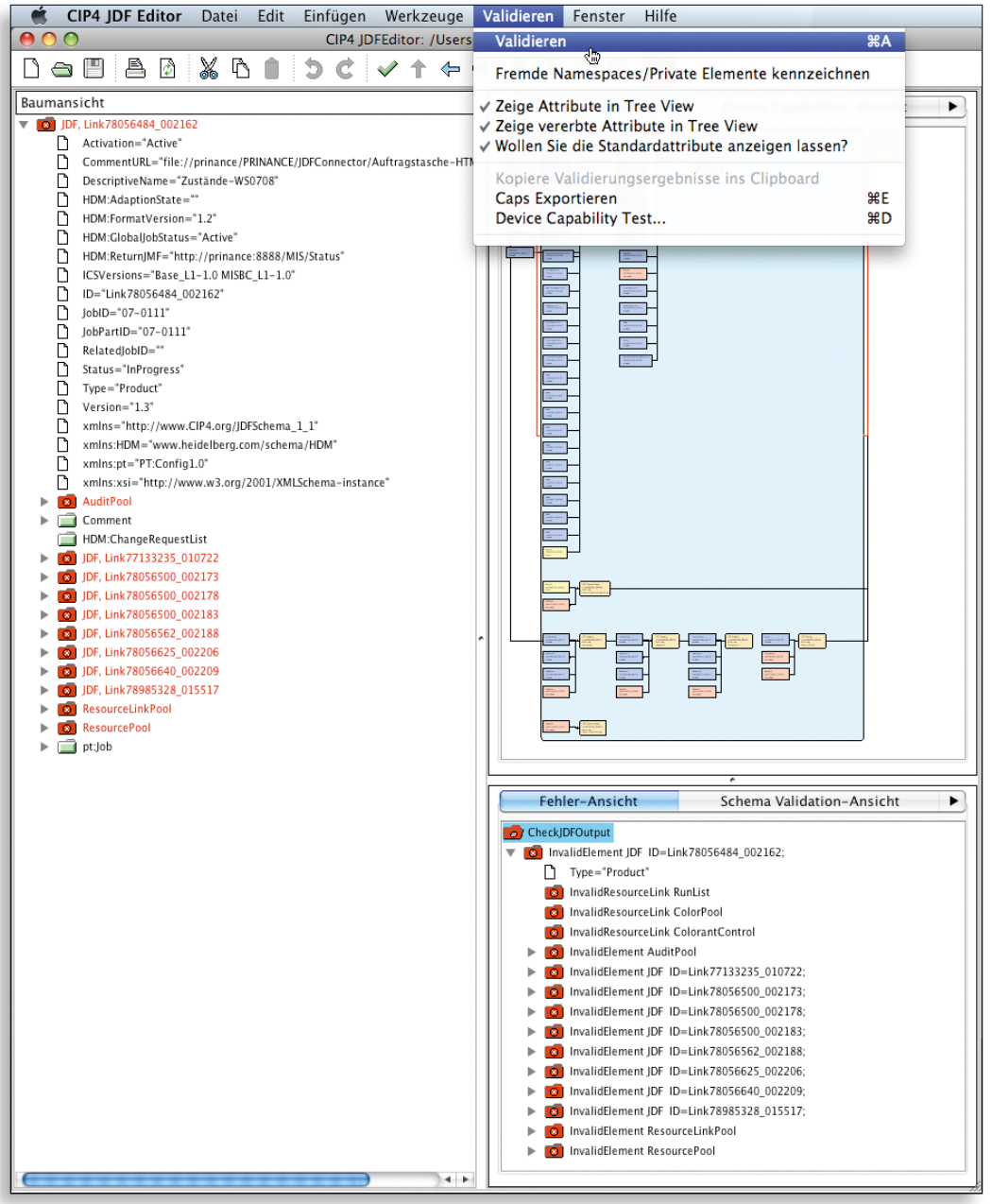



Abbildung 13.1
Validierungsfunktion im CIP4-JDF-Editor

gleiche gilt auch für die JDF-Versionen, die in dem Attribut *Version* verzeichnet sind. Allerdings lehrt die Erfahrung (oder auch das gesunde Misstrauen), dass die angegeben Versionsstände nicht immer mit der Realität übereinstimmen.

Wenn es schon beim Einlesen einer JDF-Datei wegen eines Fehlers zum Abbruch kommt, kann das sehr verschiedene Gründe haben. Vielleicht steht eine Ressource nur nicht direkt an dem JDF-Knoten, der sie benötigt, sondern – völlig legal – bei dem Wurzelelement und wird nicht gefunden (siehe Abbildung 6.5 in Abschnitt 6.1). Solche Probleme sind schwer zu analysieren, ohne den Source-Code der importierenden Software zu debuggen. Hier hilft es höchstens, bei Tests zunächst mit sehr einfachen JDF-Dateien zu beginnen und sie allmählich immer komplexer werden zu lassen. Leichter hingegen ist es, wenn nur einzelne Werte, die eigentlich übertragen werden sollten, nicht von der JDF-lesenden Software angezeigt werden. Dann kann man leicht im JDF-Code nachschauen, ob dort das entsprechende Element/Attribut steht und kann zumindest einem der beiden beteiligten Hersteller den schwarzen Peter zuschieben. Auch wenn das natürlich nicht immer ausreicht, vom Hersteller eine Korrektur dieses Mangels in angemessener Zeit zu erhalten.

JDF-Dateien können übrigens recht umfangreich werden. Die Beispiele in diesem Buch sind immer nur Ausschnitte. In der Realität werden JDF-Dateien unter Umständen viele DIN-A4-Seiten lang und enthalten mehrere Dutzend JDF-Knoten, was natürlich die Analyse erschwert.

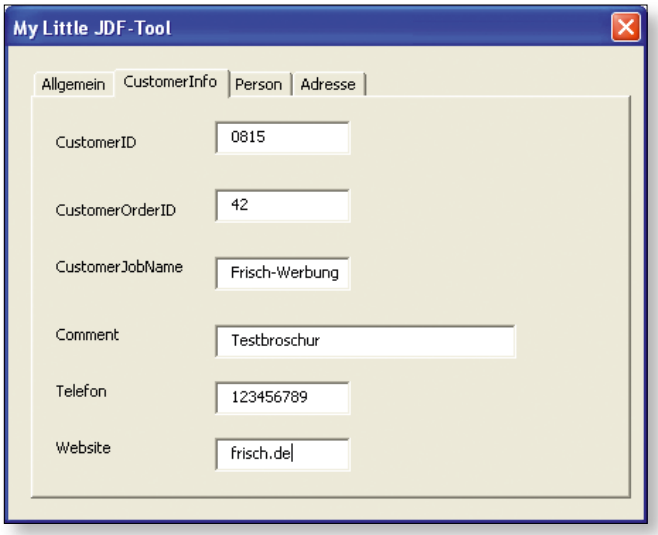



Abbildung 13.2
GUI einer einfachen JDF-Anwendung

Wir möchten es noch einmal in aller Deutlichkeit sagen: Obwohl JDF ein Industriestandard ist, funktionieren die Schnittstellen leider nicht reibungslos. Das kann an privaten Daten liegen, die im JDF transportiert werden (siehe Abschnitt „Standardisierung“ im Kapitel 2), in der Regel sind es aber eher die JDF-Strukturen und die fehlenden oder falschen Informationen in den JDF-Daten. JDF-Schnittstellen zwischen Modulen unterschiedlicher Hersteller funk-

```
Private Sub schreibe_CustomerInfo()
    CustomerInfoID = 11101
    Print #1, " <ResourcePool>"
    Print #1, " <CustomerInfo CustomerID=" & """ & CustomerID & """ & "
     CustomerJobName=" & """ & CustomerJobName & """
    Print #1, " CustomerInfoID=" & """ & CustomerInfoID & """ & "
     CustomerOrderID = " & """ & CustomerOrderID & """ & " > "
    Print #1, " <Comment>" & Comment & "</Comment>"
    Print #1, " <Contact Class=" & """ & "Parameter" & """ & "
     ContactTypes=" & """ & "Customers" & """
    Print #1, " ID=" & """ & ContactRef1 & """ & "
     Status=" & """ & "Available" & """ & ">"
    Print #1, "<Person FirstName=" & """ & FirstName & """ & "
     FamilyName=" & """ & FamilyName & """ & "
     NamePrefix=" & """ & NamePrefix & """ & ">"
    Print #1, " <ComChannel ChannelType=" & """ & "Phone" & """ & "
     Locator=" & """ & Telefon & """ & "
     Status=" & """ & "Available" & """ & "/>"
    Print #1, " <ComChannel ChannelType=" & """ & "WWW" & """ & "
     Locator=" & """ & Website & """ & "
     Status=" & """ & "Available" & """ & "/>"
    Print #1, "</Person>"
    Print #1, "<Address City=" & """; City & """ & "
     Street=" & """ & Street & """ & " Country=" & """ & Country & """ & "
     PostalCode=" & """; PostalCode & """ & "/>"
    Print #1, " </Contact>"
    Print #1, " </CustomerInfo>"
    Print #1, " </ResourcePool>"
End Sub
```

Abbildung 13.3 Umständliches VB-Programm, das JDF erzeugen kann

tionieren also nicht von selbst, sondern müssen angepasst werden. Zwar werden diese zwischen den Herstellern getestet, doch gibt es immer wieder Situationen, vor allem bei komplexen Jobs, dass Fehler auftreten. Und diese Fehler bleiben Anwendern auch meist unerklärlich, da sie weder die Zeit, noch die Tools und vielleicht auch nicht das Know-how haben, um ein Problem zufriedenstellend zu analysieren.

## 13.3 JDF/JMF-Programmierung

Dieser Abschnitt richtet sich an die Personen, die zwar ein Software-Projekt starten wollen und auch ein wenig Erfahrungen mit den Sprachen JAVA oder C++ haben, aber noch nicht mit der Programmierung von JDF-Applikationen. Es richtet sich nicht an die professionellen Software-Entwickler und Experten, die an speziellen Tipps und Tricks interessiert sind.

Natürlich kann man in jeder Computersprache JDF-importie-

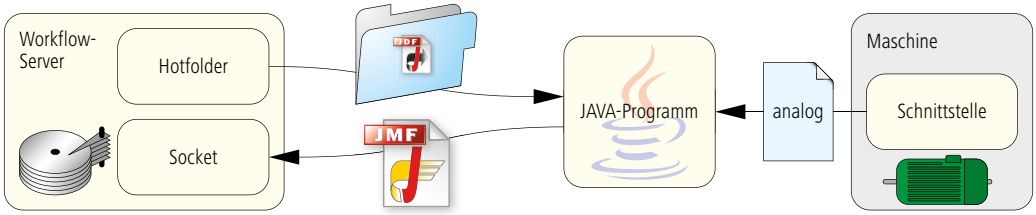



Abbildung 13.4
Projekt zur JDF-/
JMF-Anbindung einer
Maschine

rende oder JDF-exportierende Software erstellen. Man kann zum Beispiel leicht mit *Visual Basic for Applications* eine Bedienoberfläche wie in Abbildung 13.2 erzeugen und auch ein kleines Programm schreiben, das die Eingabewerte des Dialogfensters entnimmt und eine JDF-Datei mit einer *CustomerInfo* als Ressource erzeugt (entsprechend dem Beispiel in Abbildung 6.8). Nur kann man dann auf keine Bibliothek zurückgreifen, und der Code ist mühsam zu erstellen und sehr fehleranfällig. In Abbildung 13.3 ist ein Ausschnitt eines solchen Quick-and-dirty-Programms zu sehen. In der Tat erzeugt dieses Beispielprogramm keinen völlig korrekten JDF-Code, weil z.B. die beiden obligatorischen Attribute *Class* und *Status* fehlen. Es ließe sich aber natürlich entsprechend erweitern. Eine Software zu schreiben, die auf diese Art JDF einliest, ist sogar noch beschwerlicher, wenn keine geeignete Bibliothek zur Verfügung steht. Insgesamt ist von einem solchen Vorgehen abzuraten, wenn man seriös Software entwickeln möchte.

Die CIP4 hat zum Glück genau solche Bibliotheken erstellt, allerdings „nur“ für JAVA und C++. Deswegen sollte man eine JDF-Software in einer der beiden Sprachen entwickeln. Wir werden in dem folgenden Abschnitt nur auf die JAVA-Bibliothek eingehen. Zunächst muss man sich für eine (integrierte) Programmierumgebung entscheiden. Die Liste der Möglichkeiten ist lang – wir haben unser hier aufgeführtes Beispiel mit Hilfe der Entwicklungsumgebung *Eclipse* [17] programmiert, manchmal aber auch mit *NetBeans IDE* gearbeitet.

In Eclipse wird man zunächst eine neues „Projekt“ definieren, dabei können Bibliotheken in Form von jar-Dateien eingebunden werden. Folgende Bibliotheken werden für die Entwicklung von JDF-Applikationen benötigt – sie können im Internet heruntergeladen werden:

- XercesImpl.jar (unsere Version 2.9.0),
- Commons-lang.jar (unsere Version: 2.3),
- JDFLibJ.jar (unsere Version 2.1.3.4 – siehe www.cip4.org).

In unserem kleinen Beispiel haben wir eine Anwendung pro-

```
public Hauptfenster (){ // Konstruktor
    ...
    ListWin = new Thread(this);
    ListWin.start();
    ...
}
public void run(){
    while (true){
        try{
            jobliste.removeAll();
            dateien = ordner.list();
            for (int i =0; i< dateien.length; i++){
                jobliste.add(dateien[i]);
            }

            this.add(jobliste);
            repaint();
            Thread.sleep(10000);
        }
        catch(InterruptedException e){
            e.printStackTrace();
        }
    }
}
```

Abbildung 13.5
Realisierung eines Hotfolders

grammiert, die folgende Funktionen umfasst (siehe Abbildung 13.4):

- Aus einem wählbaren Hotfolder werden JDF-Dateien eingelesen und angezeigt;
- beim Doppelklick auf einen Job werden einige Details zu diesem Job (JobID, Auftragsname, Bogengröße, Lauflänge) aus dem JDF extrahiert und angezeigt;
- in diesem Fenster sind vier Befehlsschaltflächen definiert: Auftragsbeginn, Makulatur, Gutbogen, Auftragsende;
- nach Betätigen der Schaltfläche „Auftragsbeginn“ werden mit einer Lichtschranke alle Bogen gezählt, die durch eine Druckmaschine ohne JDF-Schnittstelle gehen; wählt man „Makulatur“ oder „Gutbogen“, so werden entsprechende Zähler gesetzt;
- wird die Schaltfläche „Auftragsende“ gedrückt, so sendet

Abbildung 13.6
Parsen und Auslesen von Attributwerten

```
JDFDoc JDFDocument = JDFDoc.parseFile(pfad + dateiname);
final JDFNode Root = JDFDocument.getJDFRoot();
...
String JobPartID = Root.getJobID(true);
...
JDFResourcePool RSP = Root.getResourcePool();
final JDFResource CustomerInfo = RSP.getResource("CustomerInfo",0,"");
CustomerJobName = CustomerInfo.getAttribute("CustomerJobName");
```

```
JMFMessage(String JobID,String JobPartID, Long goodCount){
    String DeviceStatus = "Running";
    //Construct a JMF-Message, i.e. the content of the http-package
    JDFJMF JMF = JDFJMF.createJMF(EnumFamily.Signal,EnumType.Status);
    KElement DI = JMF.appendElement("DeviceInfo");
    DI.setAttribute("DeviceStatus", DeviceStatus);
    KElement JP = DI.appendElement("JobPhase");
    JP.setAttribute("TotalAmout", goodCount.toString());
    JP.setAttribute("JobID", JobID);
    JP.setAttribute("JobPartID", JobPartID);
    String content = JMF.toXML();

    //construct a http-header
    long length = content.length();
    String header = "HTTP/1.1 200 OK" + "\nContent-Length: " + length +
     "\nContent-Type: text/html";
...
}
```

Abbildung 13.7
Quellcode für den Versand von JMF-Signalen

das Programm eine JMF-Nachricht an eine einstellbare IP-Adresse und TCP-Port.

Natürlich können wir hier nicht den gesamten Quellcode zeigen, sondern nur ein paar interessante Ausschnitte.

In Abbildung 13.5 ist ein *Thread* mit dem Namen *ListWin* zu sehen, der regelmäßig Dateinamen aus einem Hotfolder liest und sich danach wieder schlafen legt. Er wird vom Hauptfenster aus gestartet. Wenn der Thread aktiviert wird, werden als erstes die Einträge aus der Jobliste entfernt, danach die Namen der Dateien im Hotfolder wieder in die Liste eingetragen und schließlich das Ergebnis im Hauptfenster angezeigt.

Abbildung 13.8
JMF-Signal mit HTTP-Header

Das Parsen und Auslesen von Attributwerten ist in Abbildung 13.6 wiedergegeben. Zunächst wird das JDF-Dokument *JDF-*

```
HTTP/1.1 200 OK
Content-Length: 509
Content-Type: text/html

<?xml version="1.0" encoding="UTF-8"?>
<JMF xmlns="http://www.CIP4.org/JDFSchema_1_1"
 TimeStamp="2008-12-02T18:32:24+01:00" Version="1.3">
 <!--Generated by the CIP4 Java open source JDF Library version : CIP4 JDF
 Writer Java 1.3 BLD 40-->
   <Signal ID="m081202_063224609_000000" Type="Status"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="SignalStatus" />
      <DeviceInfo DeviceStatus="Running">
          <JobPhase JobID="_08-0157" JobPartID="_08-0157" TotalAmout="300" />
      </DeviceInfo>
</JMF>
```

*Doc* geparst und das Wurzelelement mit dem Namen *Root* bezeichnet. Attribute des Wurzelelementes wie die *JobPartID* lassen sich so leicht herausholen. Auch der *RessourcePool*, den wir RSP genannt haben, lässt sich über die *Root* ermitteln und von dort die einzelnen Ressourcen wie die *CustomerInfo*. Man kann hier deutlich erkennen, wie einfach das Ganze ist, wobei wir ein paar Sicherheitsabfragen der Übersichtlichkeit halber weggelassen haben. So gehen wir beispielsweise einfach davon aus, dass die Ressource *CustomerInfo* existiert und diese wiederum das Attribut *CustomerJobName* beinhaltet.

Als Letztes möchten wir in Abbildung 13.7 noch den Quellcode zum Versenden eines JMF-Signals vorstellen. Dort werden die JMF-Message und der HTTP-Header zusammengebaut. Ein *DeviceInfo*- und ein *JobPhase*-Knotenelement werden erzeugt und mit Attributen gefüllt. Zum Schluss wird alles in die Zeichenkette *Content* geschrieben. Hier nicht mehr zu sehen ist, wie mit Hilfe von IP- und Portadresse ein neuer Socket erstellt wird und dort hinein die JMF-Message und HTTP-Header geschrieben werden. Das Ergebnis dieses kleinen Programms ist dann ein JMF-Signal in der Art wie in Abbildung 13.8.

In manchen Fällen möchte man aber keine wirkliche JDF/JMF-Applikation schreiben, sondern vielleicht nur eine herstellerspezifische XML-Schnittstelle einer Software zu einer JDF-Schnittstelle umbilden oder aber JDF-Dateien nur leicht modifizieren. Hierzu muss also ein XML-Dokument in eine andere XML-Form gebracht werden. Diese Aufgabe kann man auch durch die *Extensible Stylesheet Language Transformation (XSLT)* erledigen. Dazu muss man ein XSLT-Dokument schreiben, das dann die Dokumenttransformation steuert. Da es aber viele Bücher (zum Beispiel [9]) über dieses Thema gibt, möchten wir hier an dieser Stelle nicht weiter darauf eingehen.

# Literaturverzeichnis


[1] Adobe System:
Adobe PDF Print Engine 2, White Paper,
http://www.adobe.com/de/products/
pdfprintengine/ (2008)

[2] Adobe System Incorporated:
XMP Specification,
http://www.adobe.com/devnet/xmp/ (2004)

[3] Adobe System Incorporated:
TIFF Revision 6.0,
http://partners.adobe.com/public/developer/
en/tiff/TIFF6.pdf (1992)

[4] Adobe System Incorporated:
PostScript Reference Language, 3. Ausgabe,
Addison Wesley Publishing Company,
ISBN 0-201-37922-8 (1999) oder
http://www.adobe.com/products/postscript/
pdfs/PLRM.pdf

[5] Adobe System Incorporated:
PDF Reference sixth edition, Adobe Portable
Document Format, Version 1.7,
http://www.adobe.com/devnet/pdf/pdf_
reference_archive.html (2006)

[6] Adobe System Incorporated:
Portable Job Ticket Format, Version 1.1,
Technical Notes # 5620 (1999)

[7] ANSI:
IT8.6-2002 (R2007) Grafic technology –
Prepress digital data exchange – Diecutting
data (DDES3)

[8] Apple Inc.:
AppleScript Overview,
http://developer.apple.com/documentation/
AppleScript/Conceptual/AppleScriptX/
AppleScriptX.html (2007)

[9] Behme, Henning und Mintert, Stefan:
XML in der Praxis:professionelles Web-
Publishing mit der Extensible Markup
Language,
Addison Wesley, ISBN 3-8273-1636-7 (2000)

[10] Bohan, Mark et al:
Automation and JDF Workflow, 2006 TAGA
Proceedings

[11] CGATS.20:
Variable printing data exchange using PPML
and PDF (PPML/VDX),
www.npes.org (2002)

[12] CIP4:
Interoperability Conformance Specifications,
Version 1.3 www.cip4.org/ (2007/2008)

[13] CIP4:
JDF-Specification Release 1.4,
www.cip4.org (2008)

[14] CIP4:
The JDF Marketplace;
http://www.cip4.org/marketplace/

[15] Commerce XML (cXML) User's Guide,
Version 1.2.019, http://www.cxml.org (2008)

[16] Dolin, Penny Ann:
Exploring Digital Workflow,
Thompson Delmar Learning,
ISBN 1-4018-9654-5 (2006)

[17] Eclipse Foundation:
Eclipse IDE for Java Developers,
http://www.eclipse.org/downloads

[18] Freund, Jacob; Götzer, Klaus:
Vom Geschäftsprozess zum Workflow,
Carls Hanser Verlag München,
ISBN 978-3-446-41482-2 (2008)

[19] Ghent PDF Workgroup:
PDF/X Plus
www.gwg.org

[20] Hamilton, Eric:
JEPEG File Interchange Format, Version 1.02,
http://www.jpeg.org/public/jfif.pdf (1992)

[21] Harvey, James:
About the JDF User Group
http://www.jdfusergroup.org/
unter: Grafic Arts Information Network
http://www.gain.net [Zugriff April 2009]

[22] Hoffmann-Walbeck, Thomas:
Lehrbuch Digitale Druckformherstellung,
dpunkt Verlag, ISBN 3-89864-182-1 (2004)

[23] International Organization for Standardisation
(ISO) 15930: PDF/X Teil 1 bis Teil 8,
Beuth Verlag (2001 – 2008)

[24] International Organization for Standardisation
(ISO) 12647: Teil 1 bis Teil 7,
Beuth Verlag (2001 – 2007)

[25] International Organization for Standardisation (ISO) 16612-2: Grafic technology – Variable data exchange – Part 2: Using PDF/X-4 and PDF/X-5 (PDF/VT-1 and PDF/VT-2), under development (2008)

[26] International Press Telecommunication Council (IPTC): http://www.iptc.org

[27] Japan Electronics and Information Technology Industries Accociation( JEITA): http://www.jeita.or.jp/english/

[28] JEITA: Exchangeable image file format for digital still cameras: Exif Version 2.2, JEITA CP-3451 (2002)

[29] Kipphan, Helmut: Handbuch der Printmedien, Springer Verlag Berlin, Heidelberg, New York, ISBN 3-540-66941-8 (2000)

[30] Kodak Grafic Communications: "Prinergy 4 Software and Rules-Based Automation", White Paper, http://grafics1.kodak.com/documents/Release%20date.doc (2007)

[31] Koster, Kai: Informations- und Kommunikationstechnologien für Unternehmen, Carl Hanser Verlag, ISBN: 3-446-2118-3 (1999)

[32] Kühn, Wolfgang; Grell, Martin: JDF: Prozessintegration, Technologie, Produktdarstellung, Springer Verlag Berlin Heidelberg, ISBN 3-540-20893-3 (2004)

[33] Kular, Christopher: The Job Definition Format and Print Media, International Journal Of The Book, ISBN: 1447-9516 (2007)

[34] Microsoft: Windows Script Host, http://www.microsoft.com/downloads

[35] Mittelhaus, Michael: Automatisierung in Druckunternehmen, Bundesverband Druck und Medien e.V (bvdm), Art-Nr. 83107 (2005)

[36] PODi - the Digital Printing Initiative: Personalized Print Markup Language – Functional Specification, Version 2.2 ( 2006)

[37] PODi - the Digital Printing Initiative: Personalized Print Markup Language – Imposition Specification, Version 2.2 ( 2006)

[38] PODi - the Digital Printing Initiative: Personalized Print Markup Language – Grafical Art Conformance Specification Specification, Version 2.2 ( 2006)

[39] PODi - the Digital Printing Initiative: Digital Print Ticket – Printing with PPML and JDF, Version 2.0 (2005)

[40] PrintTalk Version 1.3, http://www.cip4.org (2007)

[41] Schwabe, Gerhard et al (Hrsg.): CSCW-Kompendium, Springer Verlag, ISBN 3-540-67552-3 (2001)

[42] St. Laurent, Simon; Fitzgerald, Michael: XML, O'Reilley Verlag Köln, ISBN: 13 978-3-89721-516-0 (2006)

[43] Workflow Management Coalition, www.wfmc.org

[44] W3C (Dave Beckett): Resource Description Framework (RDF), http://www.w3.org/RDF/

[45] W3C: Web Services Description Language (WSDL) Version 2.0, http://www.w3.org/TR/wsdl20-primer/ (2007)

# Glossar

| | |
|---|---|
| Akzidenzdruck | Geschäfts- und Privatdrucksachen wie Visitenkarten, Briefpapiere, Werbeprospekte und Kataloge. Periodika wie Zeitschriften oder Zeitungen, Bücher und Verpackungen zählen im Allgemeinen nicht als Akzidenzdrucksachen |
| AM-/FM-Raster | Auch „periodisches/nicht-periodisches Raster“ genannt. Beim periodischen Raster sind die Abstände zwischen den Rasterpunkten für einen Farbauszug gleich, während sie beim nicht-periodischen Raster variabel sind. AM steht für amplitudenmoduliert, FM für frequenzmoduliert |
| ASCII85 | Kodierung eines Bytestroms mit 85 unterschiedlichen Bitmustern, die im ASCII-Zeichensatz alle druckbare Zeichen repräsentieren |
| Auftrags-Managementsystem | Auch „Branchen-Software“ oder „Management Information System“ genannt. Software zur Kalkulation und Verwaltung von Druckaufträgen |
| Ausschießen | Zusammenführen von Seiten, die in einer Seitenbeschreibungssprache definiert sind, und Marken gemäß einem Standbogen zu einer einheitlichen Datenstruktur, die alle Informationen von einem oder mehreren Druckbogen oder auch nur von einzelnen Seiten der Druckbogen (Schön/Wider) enthält |
| Bitmap | Datenstruktur für Bilder mit 1-Bit Farbtiefe (s/w) |
| Bogen-Layout | siehe Standbogen |
| Bytemap | Datenstruktur für Bilder mit (mindestens) 8-Bit Farbtiefe pro Separation (256 Farbtöne) |
| Einrichten | Vorbereitung einer Maschine |
| Farbmanagement | Auch „Color Management“ genannt. Verfahren und Software-Tools, die Farben auf allen Ausgabegeräten möglichst ähnlich wie die Vorlage wiedergeben. |
| Farbtiefe | Anzahl der Bits, die zur Speicherung von einem Pixel oder Tonwert zur Verfügung stehen |
| Farbzonenvoreinstellung | Werte zur Voreinstellung von Farbzuführungsmengen im Offset-Farbwerk, die zonenweise eingestellt werden können |
| Formproof | Auch „Standproof“ genannt. Meist nicht farbverbindlicher Ausdruck eines Druckbogens auf einem großformatigen Ink-Jet-Drucker zur Überprüfung einer Druckform hinsichtlich Position von Seiten und Kontrollmarken |
| Fortdruck | Druck nach dem Einrichten einer Druckmaschine |
| Greiferrand | Randfläche des Bedruckstoffes, der nicht bedruckt werden kann, da es in der Bogendruckmaschine dort Greifer festhalten und durch die Maschine führen |
| Hotfolder | Unidirektionale Schnittselle zwischen 2 Software-Modulen, die mit einem Ordner realisiert ist. Eine Software schreibt Dateien in den Ordner, während die andere regelmäßig den Ordner auf eingetroffene Dateien überprüft |
| Inline-Finishing | Druckweiterverarbeitungsaggregate, die in einer Druckmaschine enthalten oder mit ihr verkettet sind |
| Interpretation | Transformation einer Seitenbeschreibungssprache im RIP in ein internes Datenformat, das „Display-Liste“ genannt wird. |
| Lauflängenkompression | Verlustfreies Komprimierungsverfahren, bei dem die Wiederholungsanzahl |

| | |
|---|---|
| | gleicher Bitwerte gespeichert wird |
| Laufrichtung | Faserrichtung von Papieren |
| Normalisieren | Umwandeln von Content-Kundendaten in ein einheitliches PDF-Format |
| Montage | Zusammenstellen einzelner Seiten/Nutzen und Kontrollelemente zu einer Vorlage, aus der eine Druckform hergestellt werden kann |
| Nutzen | Positionierung gleicher Produktteile wie Seiten oder Faltschachteln auf einem Druckbogen |
| Papierbeginn | Abstand von Plattenunterkante zu Papierunterkante im Bogendruck |
| Papierklasse | Einteilung von Druckpapieren im Offsetdruck gemäß ProzessStandard Offset (PSO) |
| Preflight | Untersuchung von Druckdaten auf Produktionstauglichkeit mittels eines Programms |
| Rasterfrequenz | Anzahl der Rasterpunkte pro Längeneinheit (Zoll oder Zentimeter). Umgangssprachlich auch Rasterweite genannt. |
| Rastertyp/ -art | Einteilung in AM- oder FM-Raster |
| Rendering | Umwandlung der mathematischen Beschreibungen von Objekten, wie sie in Seitenbeschreibungssprachen (zum Beispiel PDF) vorkommen, in Halbtonbilder |
| RIPing/Rippen | Transformation von Daten, die in einer Seitenbeschreibungssprache definiert sind, in Pixeldaten. Für Plattenbelichter werden Pixeldaten von einem Bit Tiefe hergestellt |
| Screening | Zerlegen von Halbtönen eines Bildes (von Farbtönen mit Farbtiefe echt größer Eins) in druckende und nicht-druckende Elemente (Farbtiefe gleich Eins) |
| Standbogen | Position und Größe von Seiten sowie Hilfsmarken festlegen |
| TIFF-B | Übliches Dateiformat zur Speicherung von Bitmaps |
| Transferkurve | Auch Tonwertkompensationskurve oder Prozesskalibrierungskurve genannt. Kurve oder Tabelle zur Veränderung von Tonwerten im RIP |
| Trapping | Untersuchung angrenzender Farbobjekte und ggf. Vergrößern oder Verkleinern eines oder beider Objekte, um sogenannte Blitzer zu vermeiden (= Unterfüllen/Überfüllen) |
| Trimbox | Auch Endformatrahmen genannt. Größe des beschnitten Formats des Druckproduktes |
| Verdrängung | Auch „Bundversatz“ genannt. Differenz der Positionen von inneren und äußeren Seiten beispielsweise bei einer Rückendrahtheftung, bedingt durch die Falzungen und Sammeln mehrerer Falzbogen |
| Vor-/Nachfalz | Auch Greiffalz genannt. Überstand an einem Falzbogen (vorne oder hinten), der zum automatischen Öffnen des Falzbogens in einem Sammelhefter mit Greifern genutzt wird |
| Web2Print | Layout eines Druckproduktes mit einem Internet-Browser, wobei Standardvorlagen als Basis verwendet werden |

# Deutsche Übersetzung der JDF-Prozessnamen

In die Tabelle haben wir nur die Prozessnamen aufgenommen, die für Version 1.4 aktuell sind. Prozessnamen, die nur für frühere Versionen gültig sind, später aber ausrangiert wurden, sind nicht berücksichtigt.

| **General Processes** | **Allgemeine Prozesse** |
|---|---|
| Approval | Freigabe, Bestätigung, Genehmigung erteilen |
| Buffer | Puffer einrichten (Ressource in Warteposition halten) |
| Combine | Kombinieren (Ressourcen zusammengefügen) |
| Delivery | Ausliefern (von physischen Ressourcen) |
| ManualLabor | Manuelle Arbeit verrichten |
| Ordering | Auftrag / Bestellung erteilen |
| QualityControl | Qualität kontrollieren |
| ResourceDefinition | Ressourcen definieren / vorbereiten |
| Split | Aufteilen (von Ressourcen) |
| Verification | Verifizieren (dass Prozess erfolgreich beendet wurde) |

| **Prepress Processes** | **Prozesse in der Vorstufe** |
|---|---|
| AssetListCreation | Erstellen einer Eingabedatenliste |
| Bending | Platten stanzen/abkanten |
| ColorCorrection | Farbkorrektur |
| ColorSpaceConversion | Farbraumtransformation |
| ContactCopying | Plattenkopie |
| ContoneCalibration | Prozesskalibrierung |
| CylinderLayoutPreparation | Position der Platten auf Plattenzylinder festlegen |
| DBDocTemplateLayout | Dokumentenvorlagen für Personlisierung definieren |
| DBTemplateMerging | Erzeugen von personalisierten Seiten aus einer Datenbank |
| DieDesign | Design der Stanzform |
| DieLayoutProduction | Herstellen der Stanzkonturdaten für Stanzformherstellung |
| DigitalDelivery | Übgerabe digitater Daten definiern |
| FormatConversion | Konvertierung von Bilddatenformaten |
| ImageReplacement | Ersetzung von Vorschaubilder durch hochaufgelöste Bilder (OPI) |
| ImageSetting | Belichtung |
| Imposition | Ausschießen |
| InkZoneCalculation | Berechnung von Farbzonenvoreinstelldaten |
| Interpreting | Interpretieren (Teil des RIP-Vorgangs) |
| LayoutElementProduction | Herstellung von Seiten |
| LayoutPreparation | Standbogendefinition |
| LayoutShifting | Individuelle Verschiebung von Bogendaten pro Separation |
| PageAssigning | Sortierung unsortierter Seiten in Lesereihenfolge |
| PDFToPSConversion | Konvertierung von PDF zu PostScript |
| PDLCreation | Erzeugung von Seiten in einer Seitenbeschreibungssprache |
| Preflight | Datenüberprüfung |
| PreviewGeneration | Voransichtsbild erzeugen |
| PSToPDFConversion | Konvertierung von PostScript zu PDF |
| RasterReading | Einfügen von Bilddatenseiten oder -bogen in die Displayliste |

| | |
|---|---|
| Rendering | Rendern |
| Scanning | Scannen |
| Screening | Rasterung |
| Separation | Separationen erzeugen |
| Stripping | Montieren, Standbogen erzeugen |
| Tiling | Zerteilen von Bogendaten |
| Trapping | Überfüllen / Unterfüllen |

| **Press Processes** | **Druckprozesse** |
|---|---|
| ConventionalPrinting | Konventionelles Drucken (mit physischer Druckform) |
| DigitalPrinting | Digitales Drucken |
| Varnishing | Lackieren |

| **Postpress Processes** | **Prozesse in der Weiterverarbeitung** |
|---|---|
| BlockPreparation | Buchblock vorbereiten |
| BoxFolding | Faltschachtelkleben |
| BoxPacking | Verpacken von Druckprodukten in Schachteln |
| Bundling | Bündeln von Druckprodukten oder Teilprodukten |
| CaseMaking | Herstellen eines Schubers |
| CasingIn | Einschieben eines Buchs in einen Schuber |
| ChannelBinding | Klammern |
| CoilBinding | Spiralbinden |
| Collecting | Sammeln |
| CoverApplication | Buchblock in Softcover einhängen |
| Creasing | Nuten |
| Cutting | Schneiden |
| DieMaking | Stanzform herstellen |
| Embossing | Prägen |
| EndSheetGluing | Anfügen Vor- und Nachsatz |
| Feeding | Papier einziehen |
| Folding | Falzen |
| Gathering | Zusammentragen |
| Gluing | Leimen |
| HeadBandApplication | Zeichenband einlegen |
| HoleMaking | Bohren von Löchern |
| Inserting | Einstecken |
| Jacketing | Buch mit Umschlag versehen |
| Labeling | Etikettieren |
| Laminating | Laminieren |
| Numbering | Nummerieren |
| Palletizing | Palletieren |
| Perforating | Perforieren |
| PlasticCombBinding | Binden mit Plastikbinderücken |
| PrintRolling | Rotationsabnahme (Schuppenstrohm aufwickeln) |
| RingBinding | Einfügen von Blättern in Ringbuch |
| ShapeCutting | Stanzen |
| ShapeDefProduction | Stanzkonturen designen (mit CAD System) |
| Shrinking | Einschweißen mit Schrumpffolie |
| SpinePreparation | Buchrücken für Buchblock vorbereiten |
| SpineTaping | Kleben von Streifen auf Buckrücken |
| Stacking | Stapeln |
| StaticBlocking | Statisches Aufladen von Stapeln für Versand |
| Stitching | Klammern |
| Strapping | Umreifen |
| StripBinding | Bindestrip-Binden |
| ThreadSealing | Fadensiegeln |
| ThreadSewing | Fadenheften |
| Trimming | Dreiseitenbeschnitt |
| WebInlineFinishing | Inline-Weiterverarbeiten beim Rollendruck |

| | |
|---|---|
| WireCombBinding | Drahtkammheften |
| Wrapping | Einwickeln eines Stapels mit Folie oder Papier |

# Abkürzungen

AMS Auftrags-Managementsystem

BDE Betriebsdatenerfassung

B2B Business-to-Business

CFF2 Common File Format, Version 2

CIE Commission Internationale de l'Éclairage

CIM Computer Integrated Manufacturing

CIP3 International Cooperation for Integration of Prepress, Press and Postpress

CIP4 International Cooperation for Integration of Processes in Prepress, Press and Postpress

CSCW Computer Supported Cooperation Work

CTM Current Transformation Matrix

CtP Computer-to-Plate

cXMP Commerce Extensible Markup Language

DDES Digital Data Exchange System

dpi Dots per Inch

DTD Document Type Definition

DXF Drawing Exchange Format

ECMA European Carton Maker Association

EPS Encapsulated PostScript

ERP Enterprise Resource Planing

Exif Exchangeable Image File Format

FEFCO Fédération Européenne des Fabricants de Carton Ondulé

GPS Global Positioning System

HdM Hochschule der Medien

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

ICC International Color Consortium

ICS Interoperability Conformance Specification

IPTC International Press Telecommunication Council

JDF Job Definition Format

JFIF JPEG File Interchange Format

JPEG Joint Photografic Experts Group

JEITA Japan Electronics and Information Technology Industries Association

JMF Job Messaging Format

JTP JobTicket Prozessor

MIME Multipurpose Internet Mail Extensions

MIS Management Information System

PDF Portable Document Format

PDF/VT PDF Variable Transactional

PDL Page Description Language

PJTF Portable Jobticket Format

PPF Print Production Format

ppi Pixel per Inch

PPML Personalized Print Markup Language

PPS Produktionsplanung und -steuerung

PS PostScript

RBA Rules-Based Automation

RDF Resource Description Framework

RFQ Request for Quote

RIP Raster Image Processor

SOAP ursprünglich Simple Object Access Protocol

TIFF Tag(ged) Image File Format

UCS Universal Multiple-Octet Coded Character Set

URI Uniform (Universal) Resource Identifier

URL Uniform Resource Locator

UTF UCS transformation format

VDP Variable Data Printing

W3C World Wide Web Consortium

WfMC Workflow Management Coalition

WMS Workflow Management System

XSD XML Schema Definition

XML Extensible Markup Language

XMP Extensible Metadata Platform

XPS XML Paper Specification

XSLT Extensible Stylesheet Language Transformation

# Index