XJDF – The evolution of JDF

Background

JDF was initially designed at the turn of the millennium. Any technology will show strengths and weaknesses in such a long timeframe and therefore CIP4 is evaluating a format, currently referred to as XJDF which is a redesign of JDF that takes the evolution of interface technology into account while retaining the valuable lessons learned during the last 15 years. The major goal is an improved alignment of XJDF with common XML technologies that will allow faster and more efficient implementation of JDF workflows by allowing the use of industry standard XML tools.

Author	Dr. Rainer Prosi CIP4 Technical Officer (Heidelberger Druckmaschinen AG)
Published	*June 12 2015*

Table of Contents

Historical JDF Design requirements

Current JDF is designed as a digital representation of the entire job ticket. The assumption at the time was that JDF would be passed entirely from one device to another. Thus all aspects and interrelations of processes in one job in the graphic arts need to be encoded into a single JDF file. This design has one major drawback: The file contains data that may be irrelevant to the recipient. For instance a RIP might receive JDF instructions that contain all necessary data for RIPping but additionally a set of printing instructions dedicated to a printing press and further postpress instructions for finishing devices. Advancements in workflow processors and Management Information Systems (MIS) have made the requirement for a complete description of the entire job in one file less relevant. Generally the entire job is stored internally in some object model (which may but need not be JDF) and only dedicated chunks of JDF are targeted to the recipient device. Nonetheless, Due to requirements caused by defining complete workflows in one file, reading a JDF file is more complex and error-prone than reading a dedicated XJDF file.

Coexistence of JDF and XJDF

The crown jewels of JDF are the dictionaries of relevant parameters for processes and product descriptions in the graphic arts that span prepress through press all the way to finishing and delivery. These dictionaries must be retained. Luckily the descriptions are independent of the details their syntactic encoding. It is possible to move JDF forward to a simpler encoding such as the one proposed by XJDF that retains the ideas and details but removes some of the undesired complexities. This idea has been successfully executed in a similar manner when PDF was introduced. PDF retains the overall graphics model of Postscript while changing the encoding and removing undesired features from the language. CIP4 will even go one step further and release twin versions of JDF and XJDF in parallel. CIP4 will thus ensure that all details that are introduced in XJDF are introduced in the respective twin version of JDF by editing the details in a common master specification. Thus applications that prefer not to migrate to XJDF will still have a maintained JDF specification that takes new developments in the graphic arts into account.
Automatic translation of JDF to and from XJDF is a major design goal of XJDF. It should be easy to make existing JDF applications XJDF compliant by simply converting XJDF to JDF and then using the application's existing JDF import or export capabilities. CIP4 will also make translation tools available that can convert XJDF to JDF and vice versa.

Note that the XJDF specification will continue to be encoded in XML although not even this is a hard requirement. For instance, CIP4 is also working on defining a PDF encoding of JDF Product intent for use within PDF/VT files that applies the same idea – retain the dictionaries, regardless of the encoding – to allow PDF creators to define aspects of the desired printed product.

Data Model – Implementation or Interface

JDF represents a complete job ticket including all process dependencies. As mentioned in the introduction, these interdependencies are modeled in the object model of the implementation of a JDF enabled MIS or production control system. Therefore these interdependencies only need to be transferred to device controllers that control multiple lower level devices and also accept scheduling and routing information from a higher level controller. Simple devices – the vast majority of devices such as printing presses, platesetters or finishing devices, that receive instructions, execute them and return the results – have no use for this additional information.
JDF also defines methods for extracting partial information for distributed processing, a technology referred to as "Spawning and Merging". These implementation specifics are being removed from the XJDF specification and implementations are assumed to use whatever proprietary means that they deem necessary to ensure consistency of their internal data model.

JDF Node hierarchy

JDF allows for very flexible grouping of the process nodes in a hierarchical structure. This is designed to group device processes that belong together in common process groups. It unfortunately also makes reading JDF more difficult for devices because they need to traverse a tree of JDF nodes to find the node or nodes that are targeted at them. It also makes the use of XPath to identify data locations very complex because the parent JDF node cannot be referred to by a simple XPath. XJDF remedies this by allowing exactly one XJDF node that is exchanged between a controller and the respective device.

Sets of Resources – Partitioning

Some resources in the graphic arts need to be specified as sets of resources. In order to print a multi-page full color book you will need to describe the C, M, Y and K separation of the front and back surface of every press sheet. Some properties, such as the plate brand will be the same, but others like the image data or the scheduling of the press runs will differ for various parts of the resource set. JDF describes this with hierarchically nested resources of the same type that specify and can even overwrite the respective data. This is concise and avoids redundant specification of identical data but is also unfortunately very difficult to either validate with an XML schema or process with standard XML processors. Standard XML parsers have no notion of inheritance and therefore JDF partitioning can only be processed with dedicated JDF libraries. No reasonable XPaths exist for defining inheritance.
The concept of resource sets obviously needs to be described in a job ticket. The solution that was selected for XJDF is based on a flat list of individual resources. Each resource element has one or more Part elements that contain the context (In our example: separation, side and sheet name). This avoids inheritance at the cost of redundancy. Most JDF implementations actually specified all data redundantly in the resource leaves so that we do not see this as a major drawback. Partitioned resources in XJDF can be readily addressed with XPaths and the structure of the XML schema has been greatly simplified. Whereas JDF requires four schema data types for each resource type, XJDF requires only one.

JDF ResourceLinks

Some resource definitions need to be reused in JDF. For instance sheets that are produced by a press may subsequently be folded in a folder. On the other hand, there is a clear difference between consumption and production of resources and some parameters, like the amounts, may differ for the number of produced sheets and the number of folded sheets due to waste etc. This leads to a situation where the details of one physical object are distributed over 3 locations within the XML: details for the producer are specified in the output ResourceLink of the producing JDF node; details for the consumer are specified in the input ResourceLink of the consuming JDF node; and common data for both consumer and producer are stored in a Resource that resides in a common ancestor node of both consumer and producer. The XPaths for the three data locations are both complex and dependent on one another so that XPath is not a viable technology for traversing JDF. Since XJDF is dedicated to either the consumer or the producer, the dedicated information is all defined in one single location in the only XJDF node., which can be easily identified with simple XPaths.

Redundant Descriptions

JDF has grown over time and has been developed by many experts from various fields in the graphic arts. This has lead to many additions, deprecations and modifications which are not always consistent. One major part of the XJDF project is to reduce the number of variations that can be used to describe similar concepts. For instance, there are 4 variations to describe the imposition of pages on a sheet or product, all of which have their respective raison d'être:

- LayoutIntent describes how pages are distributed on the final consumer product such as a booklet;
- LayoutPreparationParams describes the layout of pages as seen by a simple digital printer;

- StrippingParams describes the Layout requirements from the point of view of an MIS;
- Layout gives detailed specifications so that a RIP can reliably place marks and pages onto a sheet;

XJDF will unite LayoutPreparationParams, StrippingParams and Layout. This may make some tasks that had dedicated solution slightly more difficult, but the experience with simple structures such as LayoutPreparationParams, where each version brought "just one more feature" and finally made it almost equivalent to StrippingParams in functionality but still used different parameters. The result is that we have an unnecessary divide between digital and conventional layout description. XJDF will retain LayoutIntent because it describes the customer view rather than the production view. Imposition description is one of many areas where multiple methods to describe similar properties will be consolidated.
Note that consolidation and cleanup requirements are diametrically opposed to the requirement that translation from JDF to XJDF and back should be straightforward and ideally generic so that there are many discussions and individual decisions to make regarding the ideal solution compared to a backwards compatible solution.

Device Capabilities and Plug & Play

JDF provides a mechanism to describe the range of JDF settings that a device can accept. This essentially allows for instance a RIP to provide an XML capabilities file that states that the RIP is a RIP and not a saddle stitcher. It also allows devices to describe their processing speeds, limitations such as number of colors or media size restrictions and the availability of modules such as duplex units. Unfortunately it has not been widely implemented and therefore plug & play integration remains an illusive goal for JDF enabled devices. Please note that the flexibility and diversity of devices in the graphic arts makes the task of enabling plug & play much more difficult than for instance for memory sticks.
XJDF no longer provides a CIP4 defined syntax for defining device capabilities. It relies instead on XML schemas (XSD) to provide details of the supported subset of XJDF. This removes some features and capabilities but allows use of standard XML schema enabled tools which should allow for more rapid adoption. For instance, there are tools that convert XML to schemas and can create sample XML files for schemas. With these standard tools, combined with the simplified design of XJDF, the goal of plug & play for XJDF becomes achievable.

ICS Documents and Standards

ICS documents – Interoperability Conformance Specifications – will still be required. Since XJDF will cover the same scope of the graphic arts as JDF, it will still be necessary to adhere to a given, predefined subset of the specification. In JDF, ICS documents assume a certain level of knowledge of JDF so that concepts like ResourceLinks and the syntax for partitioning are implied. One goal of XJDF is to enable more self-contained ICS documents. Since XJDF is designed to be easily described using XPaths, it is possible to write ICS documents that clearly define the requirements for a given XPath without assuming any more knowledge than general XML. This allows ICSs or even normative standards bodies like ISO to adopt parts of the XJDF specification without requiring the entire specification.

Change Orders

One complaint about JDF is that it is hard to change details of a job once it has been transferred to the device. Some of the difficulties are independent of JDF because some changes have inherently complex implications. Changing the number of pages or size of a print product typically invalidates the chosen production method and you have to restart production planning from scratch. On the other hand, some tasks like adding varnish should be a fairly simple. This is not the case in JDF, due to the linking of process steps in one large file. In essence, you have to tell the producer of the prior step to change the recipient of its output, add a new process step and then tell the original recipient that it will receive a resource with different properties (in our case it has been varnished) from a different producer.
XJDF allows job changes by making all attributes optional and thus allowing incremental changes to existing processes. XJDF is designed with changes in mind. One possible XJDF implementation is to create a skeleton job and then add the process steps as multiple subsequent change orders.

Product Intent and Gang Jobs

JDF sees product intent and processing as two very similar ideas. The concept is that a product description is very similar to the description of the process to produce that product. Unfortunately, the hierarchy of products and processes allows only one product to be the parent of a process and this assumption is incorrect in the case of gang jobs, where multiple products are produced in one step to optimize production efficiency.
XJDF separates the concepts of product and process completely. Product descriptions of all products produced by a process step can optionally be attached to the XJDF process description. This structure additionally allows for specifying the number of requested final products, which is not always easily inferred from the requested production amount due the combination of requirements for waste compensation and N-Up production.

XJMF and Audits

The Job Messaging Format – JMF – will also be more closely aligned with XML schema requirements. It will be renamed to XJMF and the message families will be made type-safe rather than generic. Since JMF is already a fairly simple XML format, no further structural changes are anticipated and XJMF will be very similar to JMF except for the usual cleanup of deprecated message attributes and some minor consolidation of message types.
XJDF audit elements are very similar to their JDF counterparts except for the following exceptions:

- All audits that were designed to track modifications of JDF over time have been removed. There is no such concept as an evolving JDF.
- Audits designed to track job status and resource usage have been modified to be syntactically equivalent to their XJMF counterparts. An audit is thus simply the consolidated sum of the snapshots that are represented by XJMF messages.

Conclusion

XJDF is a continuation of JDF that should simplify and enable more widespread integration and automation in the graphic arts. It is well aligned with current XML technologies and thus will enable vendors to build automated products and even savvy printers with home-grown systems to automate their workflows more efficiently. This alone will accelerate adoption by providing the following unique features:

- Reducing the number of methods to specify similar concepts will help to avoid incompatible XJDF dialects.
- Reduced Implementation efforts for "out of the box" software and for custom projects.
- Simplified integration and improved availability of off-the-shelf XML debugging tools.
- Integrations will be more robust because the reduced flexibility of the standard will reduce surprises through unanticipated combinations of XJDF features.
- Integration will be more cost effective.
- JDF and XJDF will be maintained in parallel so that both JDF and XJDF will be viable standards that can evolve based on requirements from the industry.
- Transition from JDF to XJDF is available and can be automated, thus allowing mixed JDF/XJDF environments.
- It will be easier to get development teams up to speed with XJDF since developers can easily apply their existing knowledge of XML to XJDF.
- The simplified structure will not necessarily make plug&play in an XJDF environment easy, but it will certainly enable it.
- Bring XJDF into other areas of the graphic arts such as Web To Print, multi-channel or multi-media publishing and digital finishing that have so far shied away from the complexity of JDF.
- Enable use of XJDF for subcontracting and cross-company networks.
- Enable better integration with ERP systems by aligning with standard ERP XML export tools, both for ERP print production integration and ERP print procurement integration.