

ODF|Plugfest

Bundesministerium des Inneren
14. - 15. Juli, Berlin

Stairway to Interoperability

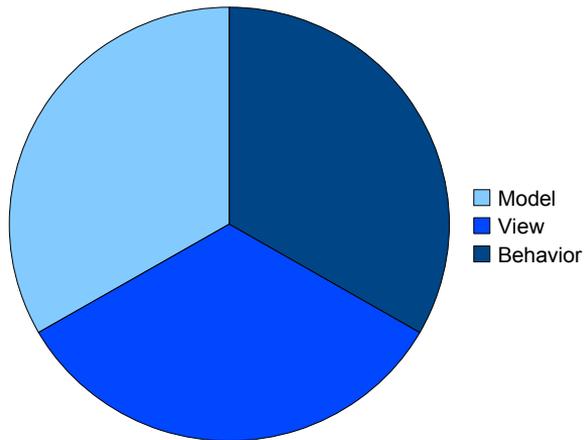
ODF Consultant
Svante.Schubert@gmail.com

Interoperability: Typical User Scenario

- Will my one million ODF documents work as good with ODF application A and ODF application B?
- Can a formal statement about interoperability be given?
- How much is a solution desired?

Interoperability: What the User expects

- Document interoperability is divided in three parts:
 - **Model:** Text, style, metadata, signature/encryption, etc.
 - **View:** Layout as glyph, run, line, block, page, etc.
 - **Behavior:** Script, macro and other executable logic



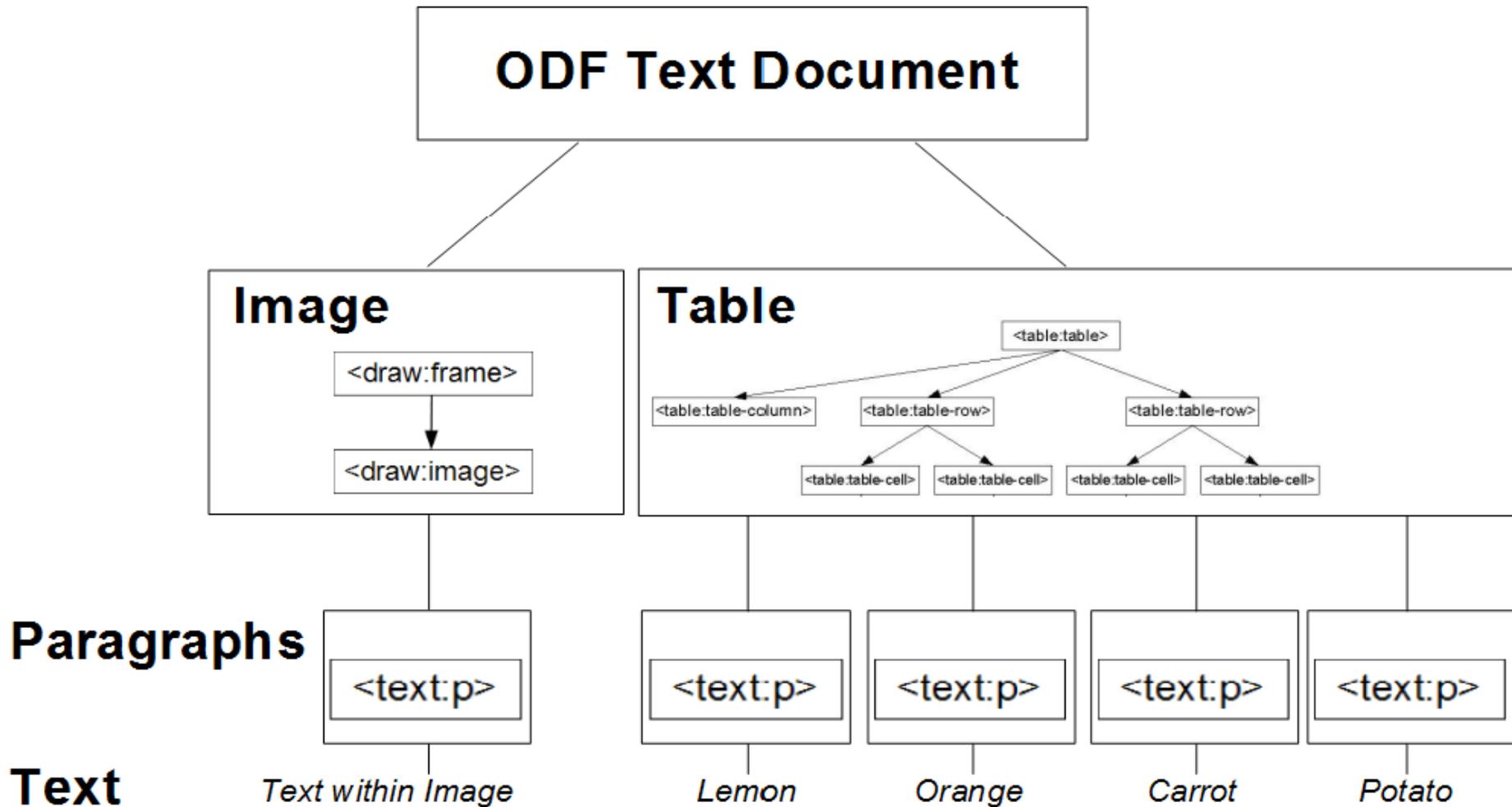
Interoperability: What can we do?

- Current state of interoperability:
 - Currently only ODF model is covered by the ODF specification
- Next steps to archive interoperability:
 - ODF view: Implementations might exchange definitions of their current layout at OASIS
 - ODF behavior can be derived from the ODF model in three steps (focus of the following presentation)

Step 1: Model Simplification by Fragmentation

- Split complexity by splitting ODF documents
 - Using an XML element as fragment?
 - Different togetherness between XML elements
 - Find components that can be added to the document
 - Multiple XML elements group to one component
 - Map XML tree to component tree
 - Components do not influence each other (disjunct)

Document Example with ODF Components



Step 2: Model Simplification by Abstraction of XML

- Easier comparison to other formats (HTML, OOXML)
 - By avoiding unnecessary information / hiding XML implementation details
 - Map XML nodes of a component to properties
- Easier transition to other formats
 - XML nodes of different formats are mapped to similar properties

Step 3: Model Simplification by focus on User Changes

- Specify the implicit known ODF behavior (e.g. insertion of a table row)
- Add methods on components accessing their properties
- Methods represent user changes
- Ensure validity by hiding access to component values
 - Access component properties only by method calls
 - Method call moves document from one valid state to another
 - Lingua Franca for transformations of formats
 - All methods define what can be changed on the document (prerequisite for change-tracking)

Proof of Concept for ODF Behavior: Collaboration

- Scenario: Same ODF document on various devices
 - Opened with browser Office app (HTML DOM)
 - Opened with Office app (proprietary model)
 - Opened with mobile device (proprietary model)
 - Opened on a server (Java DOM)
- Same ODF document concept on all devices, still different run-time model
- How to establish collaboration among different devices
 - Sending ODF or XML diff not feasible
 - Simplify the information being wired, e.g. `insertColumn(..)`
 - Sending change of component like an API call (see [Operational Transformation](#))

Do the following Benefits justify our Efforts? (1/3)

- ODF application conformance tests
 - Automated test: InputDoc + API calls = expectedOutputDoc
 - Centralized test Certification for ODF Apps (e.g. on profiles)
- Easy creation of ODF test documents
 - Problem: Too many test docs, too large & not fully covering
- Standardized ODF behavior for macros/collaboration
 - Compatibility for extensions of ODF applications
- Multiple ODF change calls map to a ODF change set
 - New feature: Apply change-set to another document

Do the following Benefits justify our Efforts? (2/3)

- Easy creation of ODF profiles / categories
 - First, identify the atomic feature of a document before declaring sets of them
- Easy identify / access component for metadata
 - Metadata are related to parts of a document (components finest granularity)
- Easy start for potential layout standardization
 - Layout is another complexity after finding the ODF components (step by step)
 - Agreement on a correct layout is not as easy to find as on implicit ODF behavior

Do the following Benefits justify our Efforts? (3/3)

- Easy comparison / transformation of formats
 - Complexity of mapping XML to components is taken away
- Boost ODF ecosystem
 - Developers no longer need reverse engineering the spec to be able to use ODF as a data vehicle
- Marketing effect
 - By specifying the ODF behavior aside from the existing model, ODF receives differentiator to other formats, which governments might demand

What might possibly block the evolutionary Step?

- Fear of apparent huge amount of work?
 - But better doing it once, instead of partly over and over again
 - But starting with a minor part would work (e.g. table)
- Existing older ODF implementations might dislike the given approach?
 - As it lowers the market barrier for competitors entering the market, implementing ODF
 - As the necessary changes of the source code might be expensive