Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 4 (2011) 668–677

Procedia Computer Science

International Conference on Computational Science, ICCS 2011

IODA - an Interactive Open Document Architecture

J. Siciarek, B. Wiszniewski*

Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology, Narutowicza 11/12, 80-233 Gdansk, Poland

Abstract

Objective of the proposed architecture is to enable representing an electronic document as a multi-layered structure of executable digital objects, which is extensible and without a need to support any particular formats or user interfaces. IODA layers are intended to reflect document content organization levels rather then system abstraction or functional levels, as in software architecture models.

Keywords: document engineering, multi-layered architecture, interactive service composition

1. Motivation and background

Since the beginning of human civilization documents have been widely accepted as natural means for representing information and providing interfaces to human driven processes. Today, when it is strightforward to store and process them as any other digital data, an important issue is not to offer electronic document solutions forcing human users to abandon their habits developed with traditional paper-form documents. Although notations for representing a document in a computer memory, or tools for manipulating its content, may change, simple actions like annotating or marking document pages directly on their image is the most natural and expected functionality of any electronic document system. Surprisingly this is not an easy task given the myriad of formats used to represent a document in a digital form needed for its storage: from binary bitmaps of document scans, through description of document pages in postscript, or editable word processor content, up to logical document structure descriptions with typesetting or markup notations. Many of these formats require different tools to view a document content in a form resembling its hard copy image. Web browsers attempt to bridge that "format" gap by utilizing the mechanism of plugins, but marking a visual content of thus rendered document is not easy due to the problem of robust anchoring, needed for marking and annotating a document at the client side – regardless of a document format and functionality of the server side [1, 2]. A limited solution might be providing functionality at the document server side to enable marking and annotating its content by remote users. It should however be independent of a document format to provide forward compatibility with standards that may appear in the future, as well as escape from the trap of developing a heavy machinery for "programming" papers that may quickly become obsolete before reaching maturity.

A perspective of associating markings of a document content with functions leads to a tempting concept of *executable documents*. This concept will certainly extend a traditional understanding of an electronic document by

*Corresponding author

Email addresses: siciarek@wp.pl (J. Siciarek), bowisz@eti.pg.gda.pl (B. Wiszniewski)

Selection and/or peer-review under responsibility of Prof. Mitsuhisa Sato and Prof. Satoshi Matsuoka

doi:10.1016/j.procs.2011.04.070

leveraging its usability as an interface unit to services doing anything anywhere on the Web. Interactive Open Document Architecture model proposed in this paper provides a solution to the problem of making a document content executable despite of its particular representation format. The proposed solution is light-weight, i.e., does not require documents to be in any particular format, relies as much as possible on the existing technologies and tools and takes advantage of interactive interpretation of a document content by human users rather than automatic parsing with a hard wired code.

2. Multi-layered document architecture

A key point about IODA is that it does not introduce any specific document format nor require implementing document functionality in any particular programming language. It just provides a sort of a document *spine* that binds tools and services that already exist in a Web document ecosystem. A document spine is implemented as an XML file that simply combines three layers shown in Figure 1.

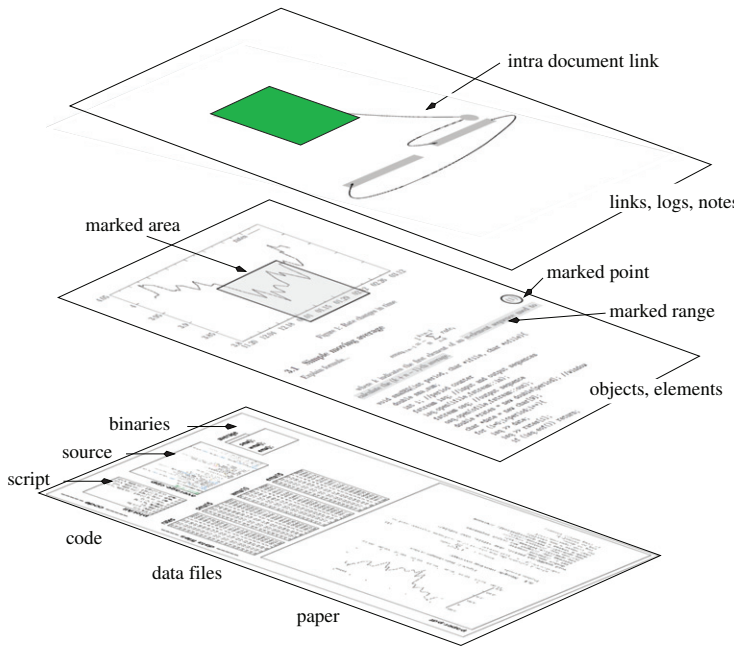

Figure 1: IODA layers

Layers of IODA reflect broad interpretation of a "digital object" from a viewpoint of digital libraries [3]. A *data layer* collects bits and bytes of text and binary files, which are just recorded "facts". Patterns that underlie those data and enable their interpretation constitute an *information layer*. Finally "contexts" of the interpretation patterns constitute a *knowledge layer*.

2.1. Data layer

This is the base layer and contains a *main document* in a form of a file ready to display or print. Document content may be binary, e.g. an image in TIFF or JPEG, or textual, e.g. PDF or RTF. Along with the main document file the data layer may contain or refer to text or binary files with data, scripts, source code, executable binaries, and so on.

Binary files and scripts implement *services* that can make document content executable. IODA distinguishes three kids of such services called respectively: *embedded*, *local* and *external*. Embedded services are executed at the host document server for data stored in the data layer. Local services are performed as browser *plugins* or default *local applications* at the client side – upon downloading data from the data layer. Further modifications of downloaded data do not affect the original content of a document data layer, as they are performed at the client's workstation and

provide a flexible experimentation facility for the paper user. If a service cannot be executed as the embedded or local one, an *external software service* described by the respective entry in a data layer is searched and invoked. Examples include a missing plugin in the user browser, which may be downloaded and installed via an external service and next continued as a local one, processing large volumes of data from some remote site specified by the data layer with a specialized cloud service at yet another remote site, authorization or certification prior to accessing some restricted resources of the data layer, and anything else not provided by embedded or local services.

2.2. Information layer

Interpretation patterns of this layer build upon the data layer components. IODA does not impose any specific format to define these patters. They combine selected semantic objects from a document content (types), with syntactical structures (viewers) of a document layout, and fragments of a document image (markings). Markings take the form of a *point* (a character or a pixel), a text *range* (a string of characters), a specific page *object*, or a *structure*, being a collection of any of the former. Markings associate selected objects with services of the data layer, which may be executed upon receiving events generated by user actions performed on a marked document. In that regard the information layer implements the classic Model-View-Controller design pattern for any user interaction with the executable paper content.

2.3. Knowledge layer

Information layer combines respective data of the data layer to facilitate user interaction with marked fragments of the principal document image. However, user interaction scenarios, e.g. those performed by reviewers, may require *contexts* of interpretation patterns implemented by the information layer. This is a task of the topmost *knowledge layer*. Building up contexts of data interpretation patterns uses *notes* and *links*. Notes constitute a portion of a text or multimedia, like audio, image or video clips attached to one or more selected markings of the information layer. Relationships between markings are defined with links, and provide a context for interpretation patters of document fragments by users. Links may be of intra- and inter-document type. *Intra-document* links relate tags within a single document, while *inter-document* links provide references to markings in other IODA documents. Most operations of this layer are performed by using a standard client browser functionality: viewing a rendered document, exercising functionality of its marked fragments, defining links and navigating between them, as well as editing and attaching notes.

Two additional operations that may be useful at this level of document organization are: *logging* of user actions, and *searching* for documents. The former provides support for resumption, fail-safe recovery of user actions if a connection breaks, or monitoring user actions to notify the interested parties on completing some required actions by the user. The latter may involve various mechanisms for *content-based* search and retrieval of documents to build inter-document links. Such a searching facility may be particularly useful to validate citations and detect plagiarism.

3. Implementation

Implementation of any IODA document requires just a few XML files, of which `data.xsd`, `information.xsd` and `knowledge.xsd` specify directly the respective layers described before:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://ioda.siciarek.pl" ...
    ...>
   <xs:include schemaLocation="./metadata.xsd" />
   <xs:include schemaLocation="./data.xsd" />
   <xs:include schemaLocation="./information.xsd" />
   <xs:include schemaLocation="./knowledge.xsd" />
</xs:schema>
```

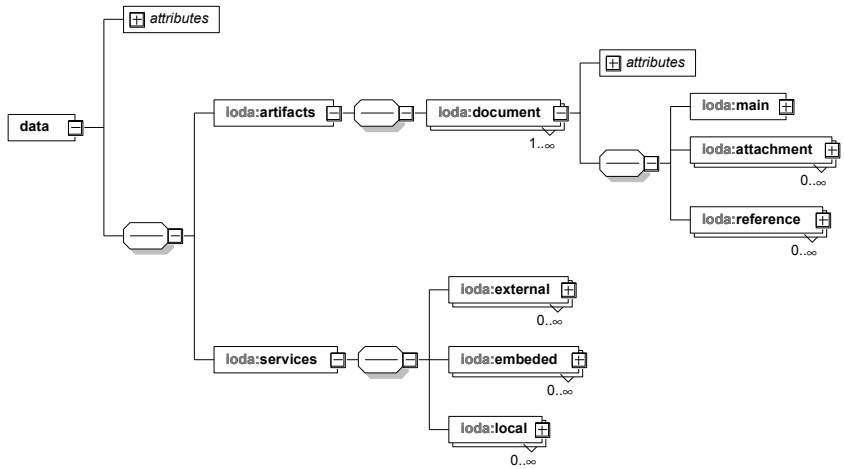

Figure 2: Elements of the data layer

File `metadata.xsd` specifies customary metadada of the main paper, such as `<ioda:author>`, `<ioda:title>`, and `<ioda:reviewer>`, along with its brief `<ioda:description>` and revision `<ioda:history>`.

Logical structure of the data layer specified by the `data.xsd` file is shown in Figure 2. It consists of two essential parts: `<ioda:artifacts>` collecting source data, and `<ioda:services>` supporting the executable paper functionality. The data part includes the `<ioda:main>` paper, its associated `<ioda:attachment>` elements and if needed `<ioda:reference>` elements specifying external data resources. Services specified respectively as `<ioda:embedded>`, `<ioda:local>` and `<ioda:external>` are implemented in a standard `<ioda:url>` form, indicating respectively entries to functions attached to the main paper, document server functions supporting export of data and/or code to the client workstation, and software services of specific external servers

Logical structure of the information layer specified by the `information.xsd` file is shown in Figure 3. Its principal components are semantic `<ioda:objects>`. Depending on the particular format of the `<ioda:main>` document they may directly indicate objects in its source file, e.g. streams in PDF or RTF files, specific markups, e.g. nodes of an XML tree, or fragments of bitmaps – if the main document is just a scan. Objects may be viewed with `<ioda:viewers>` as members of various syntax structures. The most common has been implemented for the first prototype of IODA: `<ioda:table>` elements consisting of `<ioda:field>` elements, and `<ioda:chart>` elements, each one with its specific `<ioda:xrange>` and `<ioda:yrange>` axes and `<ioda:series>` plots of `<ioda:line>` elements. Finally, the `<ioda:exec-spots>` element specify various markings of a rendered document canvas that may intercept user events. Markings may involve a `<ioda:point>` (a pixel or character), a linear `<ioda:range>` of text, a specific `<ioda:object>`, like a formula or a piece of a source code, and `<ioda:structure>` collecting any of the former three types of markings.

Knowledge layer elements shown in Figure 4 include `<ioda:links>`, which associate markings within the same document (intra-document links) or between separate documents (inter-document links), `<ioda:notes>`, which associate markings with user annotations, and `<ioda:logs>`, which register data related to user actions performed in a document. Finally, `<ioda:exec-sets>` specifies the spine component of IODA that combines internal document data, viewers, services and user events to make it executable and provide a front-end interface to the user. The `<ioda:trigger>` element specifies objects to be activated and their associated `<ioda:data-provider>` services called to get out specific data from the document. A more complex behaviors of the activated objects may also be implemented, which ar specialized data processing services specified by the `<ioda:exec>` element producing results presented with associated data viewers specified by the `<ioda:data-viewer>` element.

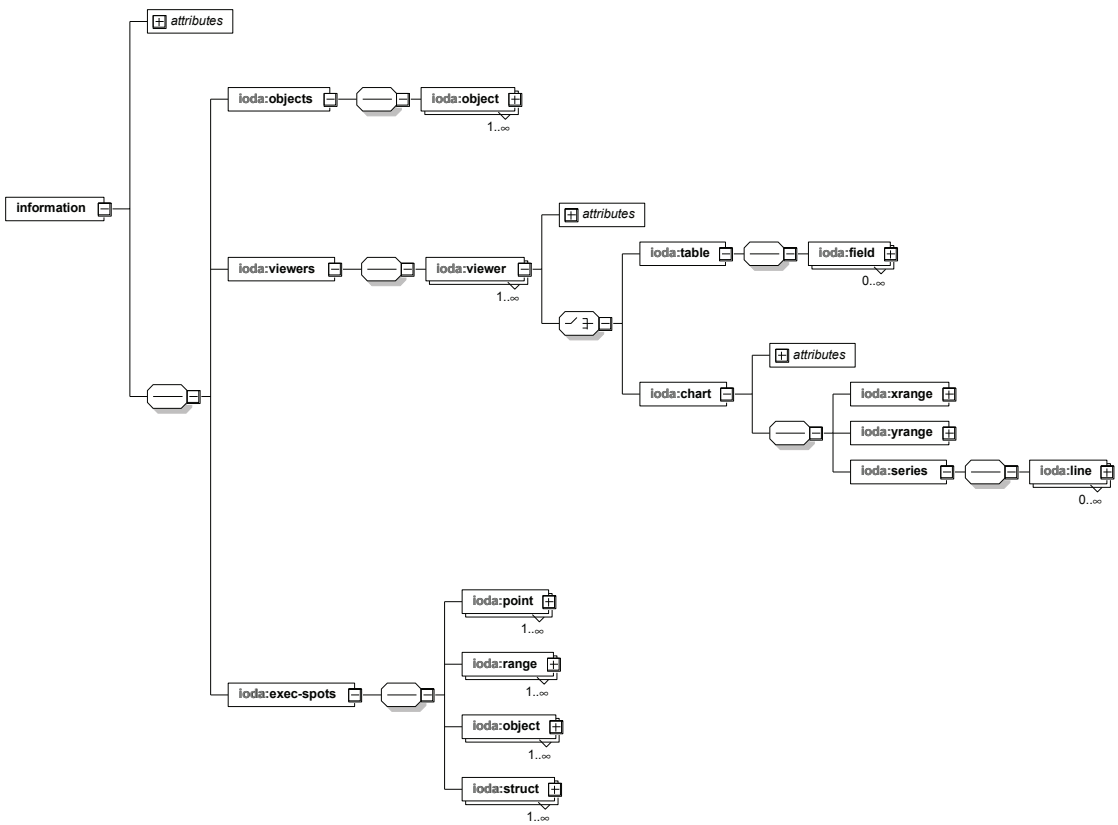

Figure 3: Elements of the information layer

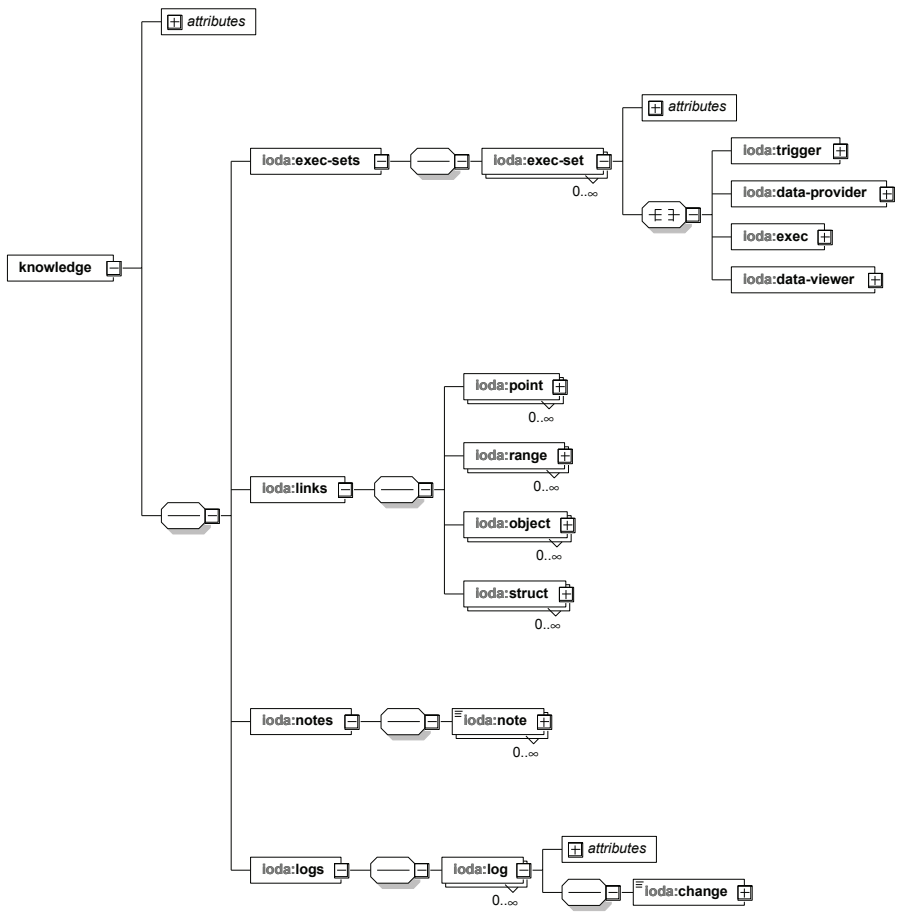

Figure 4: Elements of the knowledge layer

4. IODA document life-cycle

There are in general three phases of a paper life-cycle: *submission*, *reviewing* and *publication*. Submission involves the main document in a printable form, most often PDF. For older papers in data layers scanned images in any binary format may also be considered – for example when building a digital library of executable papers, with their historical predecessors already published.

Upon completing a main document and including it in the data layer, providing attachments and setting-up the information layer, authors make it ready for reviewers. The latter read it carefully and exercise its functionality by interacting with marked document fragments, expand if possible the information layer with their own markings, and also incorporate in the knowledge layer their notes and links. Reviewing is an iterative process, so when the review is complete, authors introduce in the next cycle a revised version of the main document. Depending of the depth of the revision requested by the reviewer, authors edit the content of data and information layers. Changes are tracked by the document server and stored in the knowledge layer to help reviewers in assessing recent author's revisions. If successful, the paper enters the publication phase after several iterations, and its whole or just partial functionality is made available to the public ever after.

4.1. A simple user scenario

In order to demonstrate in practice basic functionality of IODA, a special executable demo paper has been implemented and published on the Web [4] by Authors. Its content explains how to predict trends in money markets with simple, weighted and exponential moving averages. In particular, usability of these metrics is discussed with various charts plotted for some real data, respective formulas for calculating moving averages are given, and their implementation in C++ is presented. The demo paper implements all essential features of IODA for most common objects and structures of a scientific paper, including tables with numerical data, charts, formulas and pieces of code. Detailed presentation of the executable paper demo is beyond the scope of this paper, so just one example user scenario is given below to illustrate the basic concepts introduced before.

The main paper of the demo data layer is a plain PDF file, with attached numerical data text files containing numbers presented in tables, *gnuplot* scripts for generating charts for these data, source code implementing related formulas in C++ and the executable binary code for calculating them.

Consider Figure 5 which illustrates a simple scenario of using the executable demo paper functionality. Numerical objects representing dates appear in three structures specified in the information layer `<ioda:viewers>`: the `<ioda:chart>` element specifying a plot illustrating rate changes in time, and two `<ioda:table>` elements specifying respectively a table with currency exchange rates EUR/PLN in a period from 2010.12.01 to 2011.03.01, and another one listing simple moving average values for window size $n = 5$. Values listed in the latter table have been calculated with a formula marked as a point and linked to that table object. Owing to these definitions, a user marking a chart area shown in the upper right corner of Figure 5 may initiate any predefined chain of events: a range of values labeling axis X of the chart is selected, which in turn implies marking of the shaded area of the table with rates at the upper left corner of Figure 5 and the respective table with simple moving average values at the bottom left corner.

The marked range of dates indicates a sequence of 29 values of EUR/PLN exchange rates that are taken from the respective data file specified by the `<ioda:attachment>` element of the data layer. Owing to the link indicating a formula, which is an object associated via the `<ioda:exec>` element of the knowledge layer with an embedded service provided by a binary code attached to the main paper at the data layer, a new value of a simple moving average for $n = 29$ is finally calculated. Alternatively, instead of associating the linked formula with a binary code calculating simple moving average values, a script may be executed to generate and upload to the user's workstation a complete spreadsheet, ready to use by his/her applications, like Microsoft Excel or Open Office Calc. For particularly long data sequences (for example billions of rates), or formulas requiring higher computational power than offered by a document server (for example $n = 1000000$), some external services specified in the data layer might be called instead.

4.2. Executable paper development framework

The binding spine concept is similar to a *hub document* developed for the MVD (Multivalent Document) architecture over a decade ago. MVD integrates document components of various types into one document by behaviors called *media adapters* [5]. A principal purpose of the MVD architecture was to enable collaborative work on documents composed of distributed layers that may be under control of different authorities, with the excessive use of

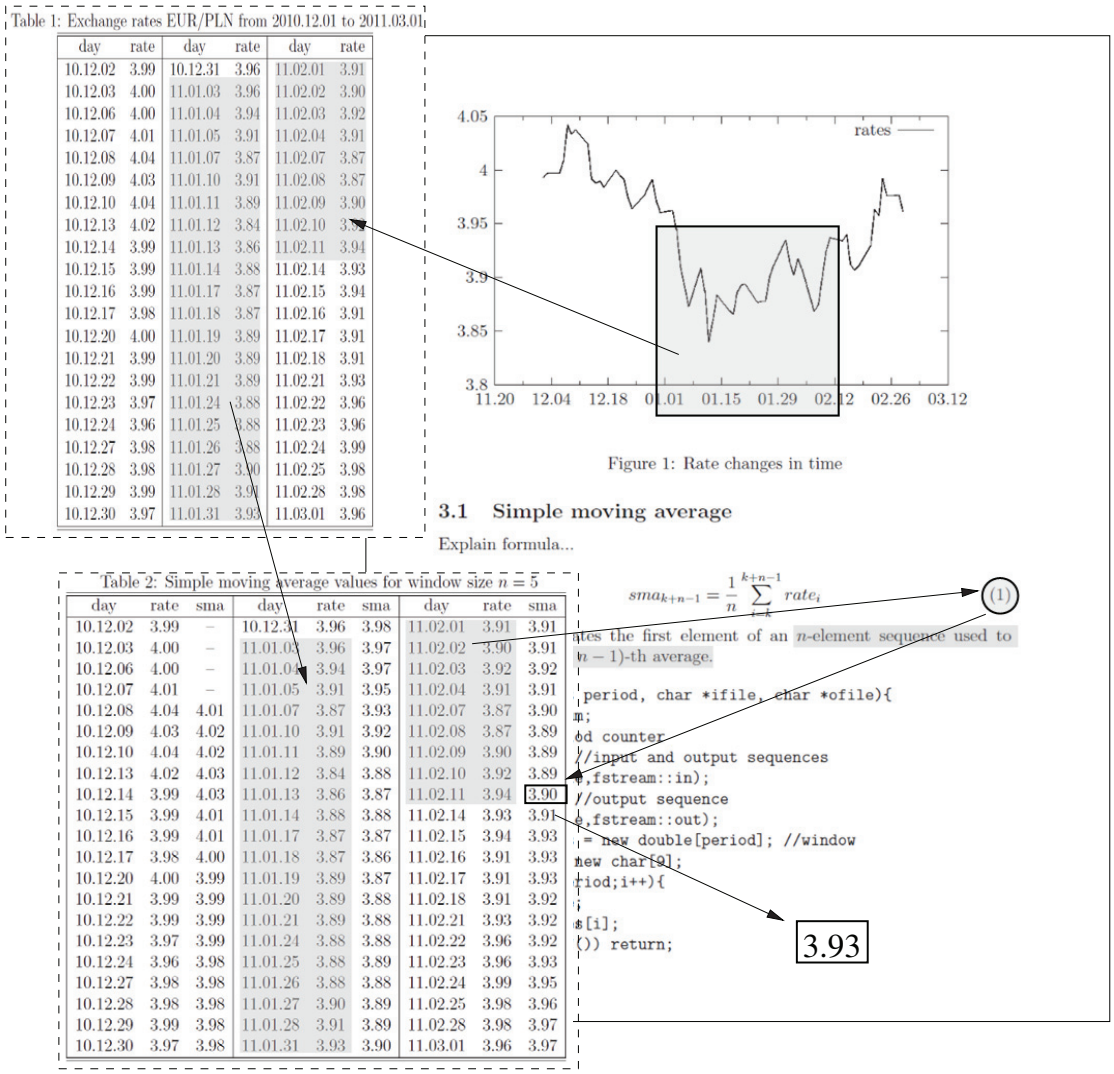

Figure 5: Example use of the IODA executable paper demo

in situ annotations over multiple document formats. One of the challenges there was to develop robust anchoring algorithms, that would maintain annotations of the upper layer reattached correctly despite alterations of the lower layer. The difference between MVD and IODA is that the latter does not require developers to implement functionality of media adapters as embedded behaviors and takes advantage of modern SOA paradigms by expanding document functionality with local and external services executed outside of a document host computer. This reduces the size and complexity of a multi-layered document, minimizes overhead on modifying document functionality upon adding or removing components, and increases flexibility when its execution environment evolves because of releasing new plugins or services that replace older versions specified by its spine component.

Instead of developing hard-wired media adapters proposed by MVD, IODA prefers a light-weight approach involving "orchestration by example" of services registered in the document data layer in a simple three step procedure:

1. Upload the main paper with its attachments and register services in the data layer.
2. Interpret the paper content by naming and selecting objects to be associated with user events via `<ioda:exec-spo` of the information layer.
3. Bind the executable document with `<ioda:exec-sets>` of the knowledge layer

An important point to grasp is that authors creating executable papers conforming to the proposed IODA model may use *exactly* the same tools and resources when preparing results reported in the main paper submitted for publication. For some authors these may be individually developed pieces of code or some standard libraries, like common Linpack [6] or GNU Scientific Library [7], or just a plain spreadsheet, while for others the case would be scripts for some popular symbolic algebra execution tools, like Maple or Mathematica [8], or even unique languages for simulation modeling like Acumen [9].

5. Conclusions

IODA addresses several important issues of the executable paper challenge:

- *Executability* of equations, tables, graphs, pieces of code, etc. is provided by authors interactively selecting and interpreting objects. For simple interpretation patterns, e.g., a set of data associated with a formula in some marked document fragment for which it calculates series of results, execution is straightforward. For more complex patterns involving collections of marked document fragments, e.g. tables of data processed by equations and resulting in a series of graphs and other tables, a dedicated service may be orchestrated using registered services. By logging user actions in the knowledge layer all such experiments can be repeated and manipulated.

- *Short and long-term compatibility;* content executability of the proposed model builds upon executable components of Web browsers, making IODA documents compatible with the users operating systems and architectures, as well as adaptable to future systems to the same extent as Web browsers could be.

- *Validation;* upon submitting an IODA paper, registration of certain specific services of its data layer may be requested by the publisher. In consequence, reviewers may expect a substantial support from a document information layer if set-up correctly by the author when validating results, both in numerical and graphical form.

- *Copyright and licensing;* for particularly sensitive cases, when data or code reported in the paper constitute a proprietary information, access for validation or exercising purposes by reviewers may be implemented as a specialized external service. Data protection mechanisms there may involve a whole spectrum of user access scenarios to Web services involving proprietary data: user authorization to use a particular service, calculation of only general statistics for data hidden behind the service, data anonymization, digital watermarking, etc. Code protection is even simpler, as it may be executed just as a service available only to authorized reviewers. The cost would be additional effort by the author of a submitted paper to implement and make available the respective service.

- *Systems and size;* to convey work done on large-scale computers, or with large data files, it will be strightforward to request IODA external services from some cloud computing platform. The question is of course how large should be the external computer (or data files attached to the submitted paper), compared to the user computer, to necessitate implementing the service of a relevant document fragment as the external rather than a local one. Given the increasing popularity of mobile personal devices, it is most likely that external services proposed by IODA will evolve towards the emerging *Desktop as a Service (DaaS)* model of cloud computing. This perspective would certainly be attractive to publishers and authors, who may profit from executable papers providing specialized services for the wide scientific community and enabling interested viewers to exploit results published in executable papers with their own data within the cloud computing business model.

- *Provenance;* registering and tracking of actions taken on the executable IODA paper is implemented at its topmost knowledge layer with a standard Web server logging facility and is fairly simple.

- *Other issues;* Solutions to protect IODA documents from viruses and code contamination do not go beyond the mechanisms incorporated by today or future browsers and Web services, as they are just XML trees specifying services and tools of a document ecosystem that makes the main paper executable via common browsers. Plagiarism can be handled by incorporating an optional functionality to a document fragment marked by the

reviewer, associating it with a content similarity analysis tool or service (provided or indicated by the publisher). Simple searches may include finding papers of the same author published before to verify originality of the submitted paper under review, while more advanced may involve a range of techniques from substring or keyword similarity matching, up to more recent near similarity search methods [10].

The light-weight approach advocated in the paper assumes the key role of human interaction with digital objects to acquire and discover knowledge in a truly open distributed system, and to enable knowledge based organizations [11].

A prototype system for submitting IODA documents to a publisher is currently under implementation. It will provide authors with the relevant layers building facility, in particular document marking and orchestrating registered services to implement its desired functionality, and reviewers with annotating and exercising paper content.

References

[1] T. Phelps, R. Wilensky, Robust intra-document locations, Computer Networks 33 (1-6) (2000) 105–118.
[2] D. Bargeron, A. Bernheim Brush, A. Gupta, Robust anchoring of annotations to content, United States Patent Application 20060080598, http://www.freepatentsonline.com/y2006/0080598.html (April 2006).
[3] I. Witten, D. Bainbridge, D. Nichols, How to Build a Digital Library, 2nd Edition, Morgan Kaufmann, Burlington, MA, 2010.
[4] J. Siciarek, B. Wiszniewski, How to become a millionaire with some math (an executable paper demo), http://ioda.siciarek.pl (2011).
[5] T. Phelps, R. Wilensky, Multivalent documents, Comm. ACM 43 (6) (2000) 83–90.
[6] J. Dongarra, G. Stewart, LINPACK Users' Guide, Society for Industrial Mathematics, 1987.
[7] B. Gough (Ed.), GNU Scientific Library Reference Manual, 3rd Edition, Network Theory Ltd., 2009.
[8] I. Shingareva, C. Lizarraga-Celaya, Solving Nonlinear Partial Differential Equations with Maple and Mathematica, Springer, 2011.
[9] Y. Zhu, E. Westbrook, J. Inoue, A. Chapoutot, C. Salama, M. Peralta, T. Martin, W. Taha, M. O'Malley, R. Cartwright, A. Ames, R. Bhattacharya, Mathematical equations as executable models of mechanical systems, in: Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems, Stockholm, Sweden, ICCPS '10, ACM, New York, NY, USA, 2010, pp. 1–11.
[10] B. Stein, S. Eissen, Near similarity search and plagiarism analysis, in: M. Spiliopoulou, R. Kruse, C. Borgelt, A. Nurnberger, W. Gaul (Eds.), From Data and Information Analysis to Knowledge Engineering, Studies in Classification, Data Analysis, and Knowledge Organization, Springer Berlin Heidelberg, 2006, pp. 430–437.
[11] M. Godlewska, B. Wiszniewski, Distributed MIND - a new processing model based on mobile interactive documents, in: R. Wyrzykowski, J. Dongarra, K. Karczewski, J. Wasniewski (Eds.), Parallel Processing and Applied Mathematics, 8th Int. Conf., PPAM 2009, Wroclaw, Poland, September 13-16, Vol. 6068 of Lecture Notes in Computer Science, Springer, 2010, pp. 244–249.