Exploiting Form Semantics and Validation Checks to Improve e-Form Layout

Abstract. On their route to the electronic era, organisations, release on the web more and more complex form-based services, which require users to enter numerous data items interrelated by business rules. In such a context, it is crucial to provide optimal form layouts, in order to present the service users with interfaces that facilitate their work. This paper presents an integrated environment, which exploits data item interrelations manifested by the business rules (or validation checks) to optimise the layout of the web forms comprising a complex service. The approach is validated through its application on a tax return form e-service.

Keywords: Usability, form layout, form semantics, validation checks, integrated environment, optimisation

1 Introduction

Most of the interactions of citizens with public authorities are performed through forms, which are filled in by the citizens and processed by the authorities. Forms used in such transactions may range from simple documents with less than ten fields, such as a statement for address change, to highly complex document sets, such as tax return forms or social benefit claims.

Form layout and field placement plays a significant role, regarding the ease of use of the forms, both by the citizens and the public agencies. Even in simple forms, using consistent layout across forms allows users to familiarise more quickly with the forms and exploit the knowledge amassed from using one service in the context of other services [1]. In complex multi-form services, form layout and field placement are of even higher importance, since:

- Imposing a document structure aids the users in locating the fields that they need to use. In an unstructured document, this procedure is tedious and frustrating.
- Placing conceptually related fields closely together assists the user in the process of gathering the appropriate data from relevant documents (e.g. balance sheets when filling in tax return forms) and crosschecking the values.

As public administration authorities and organisations are adopting e-government practices, electronic procedures for filling in and submitting forms are provided to the citizens. The approach usually followed for designing the layout for electronic forms is to mimic the appearance of their paper counterparts, possibly automating this process with appropriate software (e.g. [2], [3], [4]). Document structure of the paper form is retained, mapping original field groupings to distinct web pages, interconnected via appropriate navigation controls, as depicted in Figures 1 and 2. Producing a single web pages containing all fields is –although possible- undesirable since it contravenes with web page and user interface design recommendations and

places heavy requirements to the client programs (browsers). This practice, however, does not always yield optimal results due to the following reasons:

1. the design process of paper-based forms takes into account the issue of imposing a document structure for allowing users to locate fields within the form, but usually disregards issues related to the number of different areas that a user needs to fill in. This due to the fact that, in a paper-based environment looking up a value in a different page (or form) entails no significant cost; in an on-line environment, however, such lookups imply additional requests to the service delivery platform (web server, WAP server etc), increasing thus both the total time needed for users to complete the task at hand and the server load.

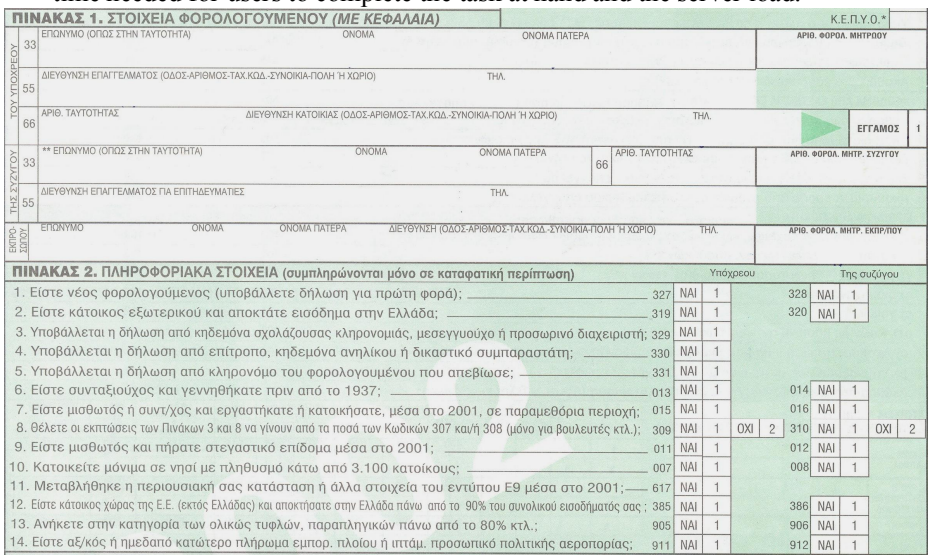

Fig. 1. The two first areas –personal data and general information– of the Greek paper-based tax return form

2. When paper-based forms *are* optimised regarding to the issue of the number different areas, this optimisation is frequently targeted for the back-office procedures they are involved in, rather than for end-user convenience, since a form is only filled once by the end-user but usually processed in multiple stages in the back-office.
3. Electronic form submittal is typically complemented with a number of validation checks, which enforce business rules upon the values filled in the various form fields (e.g. the VAT amount paid may not be higher than the 18% of the overall value of purchases). When some validation checks fail, users are requested to correct the values they have entered; this process is eased if all fields involved in the validation check appear in the same page. Placing such fields in the same page, besides relieving the user from the need of navigating through pages, allows for performing some checks in the front-end, facilitating early error detection and correction.

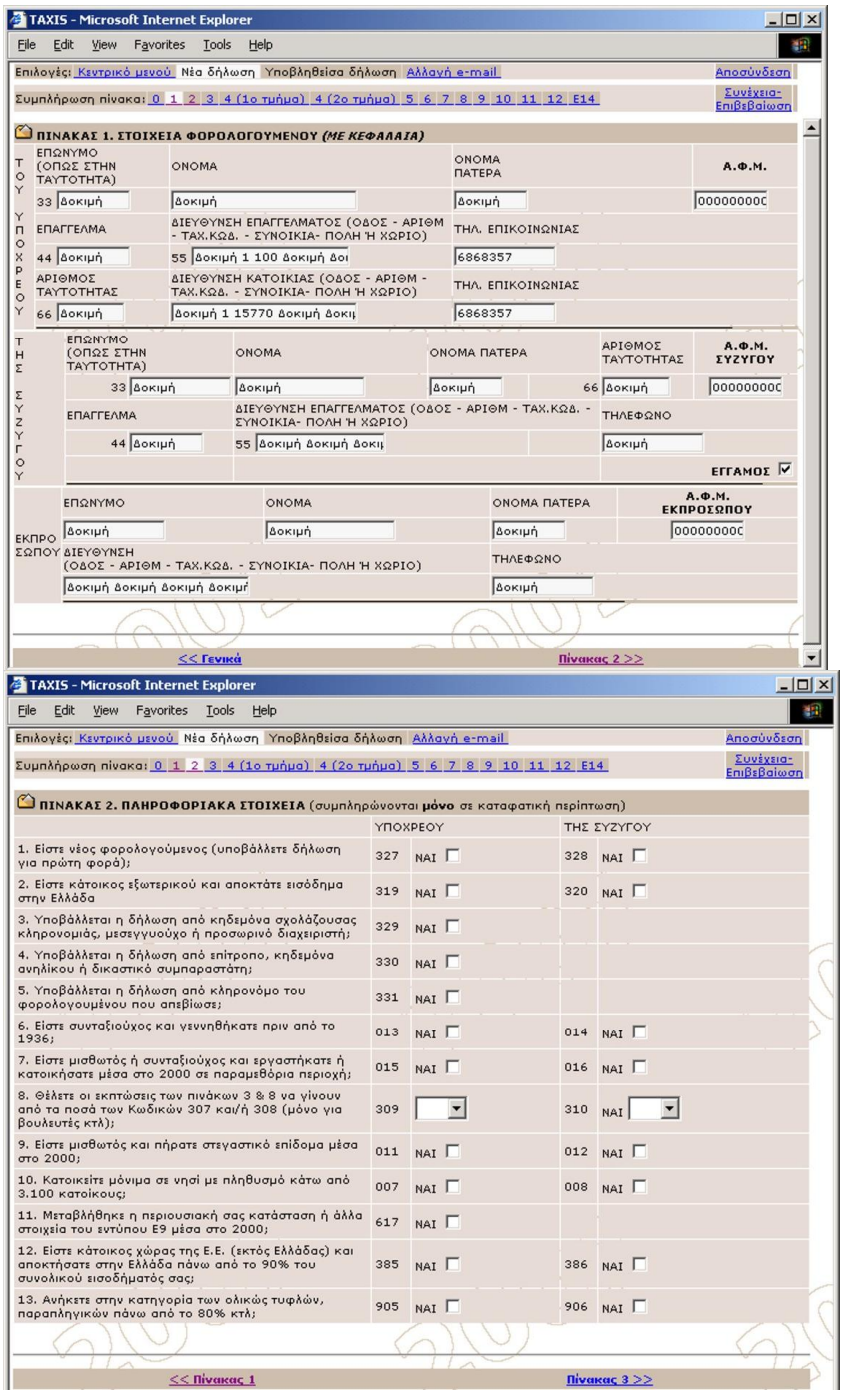

Fig. 2. The electronic version of the Greek tax return form: areas have been mapped to different pages with appropriate navigation controls (page top and bottom)

Notably, in paper-based services this stage is performed by the public servants that receive the forms with minimal involvement from the citizens, so design of paper forms usually does not take into account this aspect.

Alternatively, organisations may opt to design electronic forms from scratch, using specialised software (e.g. [5], [6]); however no support for form layout optimisation has been insofar included in this software.

In this paper we present an approach that integrates all phases of electronic service development and deployment, including definition of form fields and their semantics, form layout and validation checks. By integrating these phases, it is possible to store all relevant information in a single repository, which may then be exploited in an optimisation phase for providing hints to the service designers regarding the improvement of the electronic service layout. The platform may also encompass generic user interface design best practices and policies (e.g. maximum number of elements in a single form, field and label alignment etc), providing thus a holistic solution to the issue of designing the user interface of electronic services.

The rest of the paper is organised as follows: Section 2 outlines the electronic service development methodology and describes the features of the development environment that enable the application of form optimisation procedures. Section 3 presents the methods employed to optimise the form layout and section 4 presents a case study for the application of the proposed methods. Finally, section 5 concludes and outlines future work.

2. e-Service Development Methodology and the Development Environment

Electronic services are complex software artefacts, comprising of interrelated components, with each component addressing a particular aspect of service delivery. In more detail, the design and implementation of an electronic service includes the following stages:

1. *identification of the data that the electronic service should collect.* This requirement is directly related to the purpose of the electronic service, as dictated by the organisation's business rules. For example, a tax return form electronic service should collect data about the citizen's income, expenditures and tax deductions, while a real estate transfer declaration e-service should collect data about the seller, the buyer, the real estate being transferred, the notary involved, the price and the payment details. For each piece of data that must be collected, an appropriate input area (text box, radio button, drop down list, checkbox etc) should be presented to the service end-user, in order to enable data input. This step is generally performed by domain experts, who have a deep knowledge of the rules, regulations and directives that apply to each electronic service.
2. *Validation checks*. Data entered by the electronic service end-users must be checked to determine whether it is compliant to the organisation's business rules governing the domain area that the e-service falls in. Validation checks may involve a single data item (e.g. *it is mandatory to fill in the social security*

number or *income is a numeric element accepting positive values*) or multiple data items (e.g. *if a child's birth year is entered, the child's name must be provided as well; pre-paid taxes from salaries, if declared, must be less than the 35% of the declared income from salaries*). Validation checks are generally catalogued by domain experts and the IT staff implementing the services provides the necessary code to enforce these checks.

3. *Input forms*. Input forms constitute the interface through which end-users enter the requested data. A service may include one or more forms, and each form contains a number of input areas corresponding to the data items identified in step (1). Item placement on the forms is decided by domain experts, who should ensure that the placement is meaningful, grouping together input areas that are semantically related. If a paper form for the same service exists, this may serve as an initial layout for the electronic service forms. Input forms are typically created by HTML/interface experts, with the assistance of IT staff for inclusion of validation checks.
4. *Back-end code.* When the service end-user fills in and submits the electronic forms of an e-service, the provided data values are sent to the server that hosts the e-service. At this stage data should be validated for correctness and, typically, stored in a database for further processing within the organisation's workflow. Back-end code may also cater for retrieving values from registries to pre-fill certain areas (e.g. the areas corresponding to the citizen's name and social security number may be retrieved from a registry and appear as pre-filled in; this can only be performed in services where the user *logs in*, thus it is possible to know the user identity). Back-end code is provided by IT staff.

The development environment discussed in this paper supports the first three phases of the e-service development methodology and partially the fourth. More specifically, the development environment allows for defining the data items that pertain to the electronic service, and the definition of validation checks. Regarding the input forms, the development environment provides facilities for defining the *thematic categories* (termed *semantic axes* in this paper) to which the service data items may be assigned, to aid the system in computing feasible partitionings of the data items to forms that are meaningful for the end-user. The development environment allows also for defining *layout constraints*, e.g. which is the maximum number of input areas that may be placed on a single page. Combining information from validation checks and semantic axes, the system generates HTML form drafts, which can be then customised (aesthetic touches and/or modification of the automatically computed layout) to be used in the e-service. Finally, in relation to the back-end code, the development environment provides facilities for generating code for validation checks, which must then be integrated with the code for database storage or any other service-specific code crafted by the IT staff. A screenshot of the development environment front-end is presented in Fig. 3.

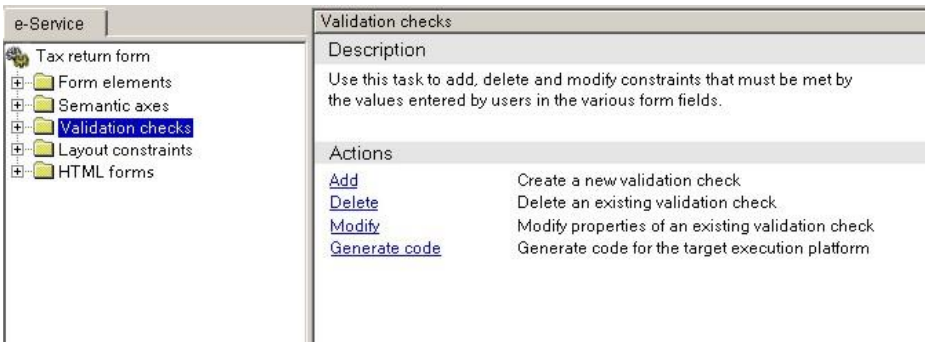

Fig. 3. The development environment front-end

Providing an integrated environment for supporting all development phases, rather than a stand-alone tool, was considered necessary, since the latter choice would require the same information to be entered multiple times (e.g. validation checks would have to be entered *both* in the programming language used in the service delivery platform *and* in the tool supporting the optimisation phase). This duplication would further complicate the development process and might lead the tool to fall into desuetude.

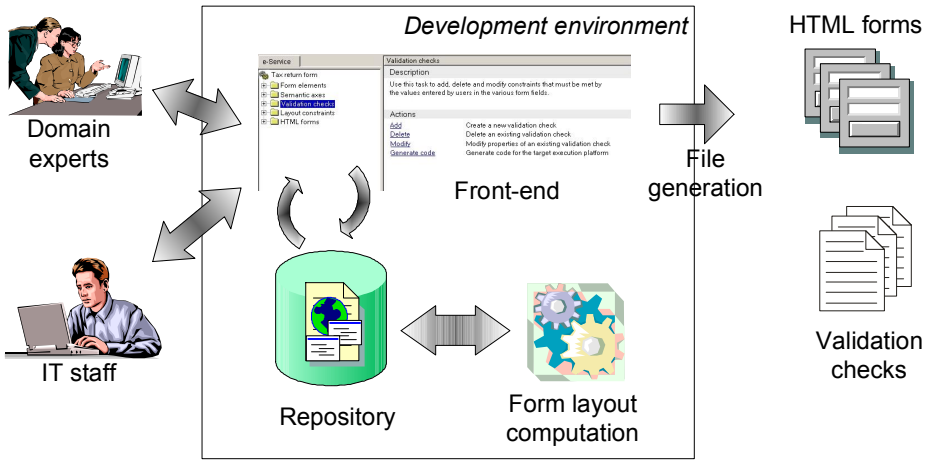

Fig. 4. Architecture of the development environment

The architecture of the development environment is depicted in Fig 4. The development environment users enter the data regarding the form elements, the semantic axes, the validation checks and the layout constrains through the front end. All these information are stored in a single repository, allowing thus for enforcement of integrity constraints (e.g. a validation check should only include existing form elements). When all relevant information have been entered, the form layout computation procedure may commence, which extracts information from the repository and creates a partitioning of the data items into HTML forms. The development environment users may then review the computed partitioning, perform

any adjustments consider necessary and finally invoke the file generation procedure, which creates HTML files, corresponding to the e-service HTML forms and files containing executable representations of the validation checks. HTML forms may be then edited and customised through any appropriate tool, while code files can be integrated with the rest of the service back-end code written by the IT staff, to complete the service development cycle.

The functionality offered by the development environment is detailed in the following paragraphs.

2.1 Definition of the form elements comprising the electronic service

This facility allows the designers to define all form elements that may be needed in the context of the electronic service, such as form fields, associated text labels, form headers/footers and navigation controls. For input fields, in particular, a number of parameters may be specified, including the type of the field (text field, check box, drop down list etc), the data type that is accepted (number, string, etc), the range of allowable values, whether it is editable or display only and so on. A number of the details entered for each field (e.g. the type of the field and the text labels) are not directly related to the issue of optimising the layout of the forms involved in the transaction service, but are required for other aspects of the electronic service development, which are supported by the development environment.

2.2 Designation of semantic axes

Semantic axes are *thematic categories* under which form fields can be classified. For example, for modeling a tax return form, candidate semantic axes include, amongst others, *income, expenditure, pre-paid taxes, salaries, real estate* and *informational*. Each field within the service may be assigned to any number of semantic axes – for instance, the field in which income from salaries is filled in can be assigned to the semantic axes *income* and *salaries* while the field representing the pre-paid taxes from stipendiary occupation falls under the axes of *salaries* and *pre-paid taxes*. Semantic axes are thus *candidate form areas* (for paper forms) or, equivalently, distinct web pages within the transaction service (for the electronic environment). It is important to note here that even though a single field may be assigned to multiple semantic axes, it will appear only once in the final electronic service layout: multiple appearances of the same field have proved to confuse users, rather than help them. Determination of the most suitable semantic axis to use for each field in the final layout is discussed in section 3.

2.3 Definition of validation checks

Validation checks are part of the business rules governing the electronic service, specifying conditions that values entered by the users must fulfill. Validation checks are important for determining optimal field placement, since –as stated in section 1– it

would be beneficial if fields involved in the same validation check appeared close together within the final form layout. Within the development environment, service developers (domain experts and IT staff) enter validation checks through an editor, supporting the following types of checks:

1. *A Requires B*. If a value is entered in field *A* then a value must be entered in field *B*. For example, if the *Married* indication is checked, the *Wife's Name* field must be also filled in.
2. *A Precludes B*. If a value is entered in field *A* then field *B* should be left blank. For instance, if the user fills in the field *profits from trade business*, the field *losses from trade business* should not be filled in, since an enterprise may not have simultaneously profits and losses from the same activity.
3. *A cmp B * c*, where *A* and *B* are form fields, *cmp* is a relational operator (=, ≠, >, ≥, <, ≤) and *c* is a constant value. This validation check category allows for modeling of arithmetic constraints on form fields such as *the expenses declared for transports must be less than or equal to the total expenses* (in this case, c = 1), or *pre-paid taxes may not be more than 45% of the total income* (c = 0.45).
4. *Custom check.* This category of validation checks is used to model any constraint that does not fall in groups (1) – (3), mainly complex ones. For these checks, domain experts may only specify the fields involved in the check, while IT staff supplies the actual code that will be used in the service delivery platform to implement the actual checking.

Using this categorisation scheme allows domain experts to easily enter all validation checks falling into the first three groups in a graphical, intuitive environment. Generating the code that will implement these checks within the service delivery platform is handled via an appropriate module that is plugged in the development environment (different plug-ins may cater for different service delivery platforms). From statistical analyses regarding the validations involved in the Greek Taxation electronic services (a rather complex environment), it was found that 82% of the total number of validation checks could be modelled using these constructs, and only 18% of the checks actually required IT staff involvement for its final implementation.

Finally, a *weight* is associated to each validation check, specifying how important is to keep the fields involved within this check closely together. The value of the weight (in the range 1-100) is determined by the domain experts based on their experience, regarding the number of documents usually failing this validation check, the number of citizens using any of the form fields involved in the check etc.

2.4 Specification of layout constraints

The final step in the specification of the electronic service is the definition of constraints and options that hold for the service as a whole, and for each individual page. Such constraints include:

1. the maximum number of fields that may be placed in a single web page.
2. the maximum number of pages that should be used for placing the various fields.

3. whether two or more distinct thematic categories (semantic axes) may be placed in a single web page (provided that the maximum number of fields per page is not exceeded) to reduce the overall number of web pages within the service.

It should be noted that some of these goals are often contradictory; for instance, in order to minimise the overall number of pages within a service, the number of fields per page must be increased. Service designers should determine a "golden mean" between contradictory goals, to produce a suitable layout.

2.5 HTML form customisation

Once all service specification steps have been completed and the form layout has been computed (form layout computation is discussed in section 3), the development platform users may review and customise the proposed solution. More specifically, the development platform users may check how the system has partitioned elements into distinct pages and may move selected data items from one page to another. Moreover, users may decide to merge two pages, in order to reduce the overall number of distinct pages within the service. Users may also save the computed layout and modify data entered in steps (2.2) to (2.4), in order to compute alternative layouts. (For step 2.3, in particular, only the *weight assignment* of validation checks is expected to change). When the users are satisfied with the computed layout, they file generation procedure for the service may be invoked.

3 Computing an Optimal Form Layout

When the appropriate information has been entered into the system, the process of computing an optimal form layout may begin. The objective of this procedure is to split the fields required for the electronic service into a number of web pages that will have the following properties:

- Web pages should contain conceptually related fields. According the modelling used in the development environment, fields are considered conceptually related if they have been assigned to the same semantic axis.
- Fields interrelated with validation checks should be positioned on the same page whenever possible, to avoid extraneous navigation.
- The overall number of fields within a single page should not exceed the limits specified in the form layout constraints.
- The total number of pages should be minimised.

In order to compute a solution with the above characteristics, the system constructs an undirected graph G = (V, E), whose vertices v are the fields that appear within the electronic service. Two vertices v_1 and v_2 representing fields f_1 and f_2 are connected with an edge e if there exists a validation check VC that relates the values of f_1 and f_2. For each edge, a weight W is assigned, which is equal to the weight assigned by the domain expert to the validation check. If there exist multiple validation checks VC_1, VC_2, ..., VC_n involving fields f_1 and f_2 and having weights W_1, W_2, ..., W_n, respectively, then fields f_1 and f_2 are connected with a single edge e whose weight w is set to

$\sum_{i=1}^{n} W_i$. The objective of the optimisation algorithm is to partition the vertex set V into mutually disjoint subsets V_1, V_2, ..., V_m, such that the cost of the weights of all edges interconnecting vertices belonging in different subsets is minimised; the costs of edges connecting vertices within the same vertex subset is disregarded. An additional constraint for vertex subsets is that for each such subset, all vertices (fields) included in this subset should be assigned to the same semantic axis S_i; however, fields assigned to the same semantic axis are not placed necessarily on the same vertex subset, i.e. a semantic axis may be split in multiple vertex subsets.

The vertex subsets V_i will actually be the different web pages comprising the electronic service. Intuitively, the cost of the edges connecting vertices (fields) in different subsets (pages) is a measure of the extraneous navigation actions that users will perform for the purpose of looking up values of fields that have been placed on different pages. The constraint of formulating vertex subsets V_i with fields belonging to the same semantic axis guarantees the semantic affinity of web pages.

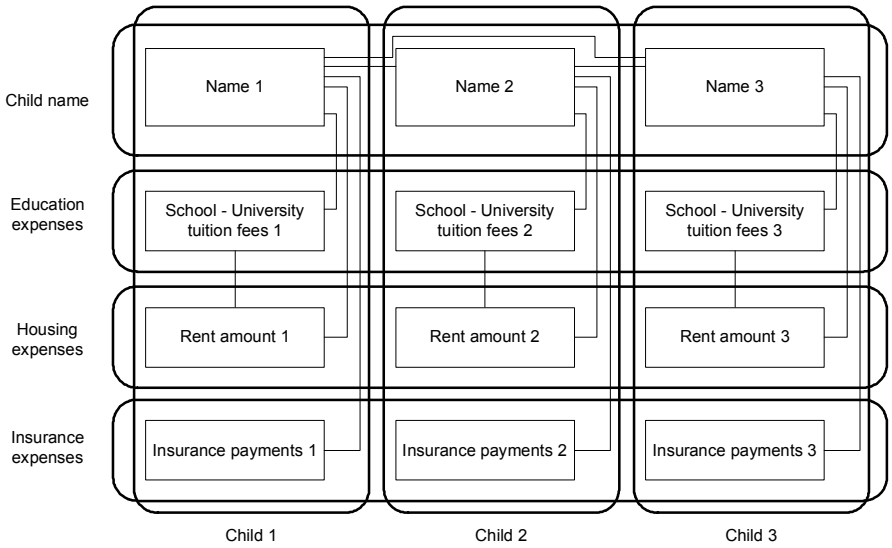

Fig. 5. Graphical representation of the layout problem

Fig. 5 presents a graphical representation of a problem instance where expenses related to a family's children are declared. In this figure, rectangles are used to denote form fields; two rectangles are connected with a line if they are involved in the same validation check. Round-ended rectangles represent the semantic axes and enclose the fields associated with them. In this example, there exist seven semantic axes, defining two promising field partitioning schemes: the first one splits input fields per child, while the second splits input fields based on expense category. Using the first partitioning scheme only three validation checks involve fields in different vertex subsets; using the second partitioning scheme, the number of such validation checks increases to 12, thus the first partitioning should be favoured. (In the figure, weights have been omitted for clarity purposes; the type of all dependencies is *requires*).

According to the description presented above, the task of optimising the layout of a transaction service is isomorphic to the graph partitioning problem ([7], [8]). This problem, and many variants of it, has been extensively studied (e.g. [9], [10], [11], [12]) and software libraries for solving it have been made available ([13], [14], [15]). The graph partitioning paradigm is also used in VLSI design (e.g. [16]) and parallel computing (e.g. [17]). The software used for the prototype environment is the hMETIS package ([18, 19, 20]) which directly supports n-way graph partitioning, formulates high-quality partitions and is very efficient, even in low-end workstations.

In order to compute the optimal layout, the development environment traverses the graph G and creates a *hypergraph description file*, in the format required by HMetis ([21]). Then, the *shmetis* command of the HMetis package is invoked, which reads the hypergraph description file and produces a partitioning. The number of partitions that will be computed is passed as a command-line parameter to the *shmetis* command. Note here that the HMetis package provides only facilities for partitioning a graph to a given number of partitions, while the constraints given in the development environments state that a *maximum number N of partitions* (web pages) should be used. To overcome this limitation, the *shmetis* command is invoked N – 1 times, with the first invocation computing a two-way partitioning, the second invocation a three-way partitioning and so on. The results of each invocation are first checked against the additional layout constraints (maximum number of fields that may be placed in a single web page); if these constraints are met, then the computed partitioning is tagged as a *candidate solution* and retained. Each solution is associated with the *Sum of External Degrees* metric of the partitioning quality provided by HMetis, which is directly equivalent to the optimisation target of the development environment (i.e. it represents the cost of the weights of all edges interconnecting vertices belonging in different subsets). When all the invocations have been completed, the Sum of External Degrees metrics of the retained solutions are compared, and the one with the smallest value is selected as the final solution.

Finally, the development environment makes the computed solution accessible through the *HTML forms* hyperlink of the development environment front-end, enabling users to perform modification to the proposed layout (see section 2.5) or directly request the generation of the respective HTML pages. The generated HTML pages may be finally processed by HTML experts and/or by specialised software to provide for the final aesthetic touches.

4. A Case Study: the Greek Tax Return Form

In order to validate the proposed approach, an experiment was set up. In this experiment the case of the Greek tax return form was studied; this case was considered to be a good example of a complex service, since it includes approximately 800 fields. In the paper form, these fields are broken down in 12 thematic areas. The electronic version of the service follows closely the layout of the paper form using 13 pages (one thematic area has been mapped to two web pages, due to excessive number of fields). The electronic version is complemented by 195 validation checks (not including validation checks regarding the data type and value

range of individual fields), in which the values of 503 declaration fields are correlated.

In the current electronic service layout, 49 validation checks (25.1% of the overall number) involve fields that have been placed on different web pages. The service was made available to tax payers starting from the fiscal year 2001 through the address http://www.gsis.gov.gr/e-srv/e1-2001. In the first year 30.000 citizens (approximately) used the service to submit their tax return forms, while in the two following years the number of service users rose up to 120.000 (fiscal year 2002) and 150.000 (fiscal year 2003)[1]. The software accepting the citizen's declaration was crafted to collect (anonymous) logs regarding the validation checks that failed during the submission of the tax return forms and the service users' navigation across the pages.

When the submission period for each year was over, log files were collected and statistically analysed. Statistic analysis, showed that that these 49 validation checks accounted for the 54% of the errors detected in electronic submissions; in other words, one out of two users that has made an error in the declaration normally needed to issue two (or more) requests for web pages before he/she is able to correct the error and resubmit the tax return form. Besides the inconvenience in the error correction process, users had to request more pages (and thus need more time for completing the task) in the process of the initial form filling as well, since the fields accepting the information they needed to type in were scattered among different pages.

This behaviour has been attributed to the fact that the semantic axes used in the paper version were not as appropriate for the electronic version. More specifically, the paper version uses the semantic axes *personal details, information, deductions due to accessibility issues, income, additional information, expenditure details, expenditure deductions, other deductions, pre-paid taxes, family details, tax payer profile information* and *bank details for tax returns*. Most of the "important" validation checks that co-related fields from different forms (i.e. the 49 validation checks producing the 54% of the errors) included certain fields from the *income* and *pre-paid taxes* semantic axes, which were also included in the candidate semantic axes of *stipendiary activities details, agricultural activities details, trading activities details, freelance worker details, real-estate income details* and *overseas activities income details*. Each such candidate semantic axis includes the income, the pre-paid taxes and any other information pertaining to the specific activity class.

The information regarding the fields of the tax return form, the semantic axes, the validation checks and the layout constraints were input in the development environment and the computation of an optimal layout was requested. It has to be noted that during the input of validation checks, domain experts were asked to fill in the weight of each validation check solely based on their experience and intuition, without disclosing to them the results of the statistic analysis. For confirmation purposes, the weights assigned by domain experts were matched against the error frequency of each validation check, as shown by the log files, to reveal that all assigned weights were very close to the error frequency, with only two exceptions; this finding provides an initial indication that it is possible to rely solely on the

[1] From the fiscal year 2002 and onwards, the service is available through the address http://www.taxisnet.gr

experience of the domain experts for weight assignment, though further experimental confirmation is required.

Once all relevant information was entered in the development environment, the optimisation procedure was invoked to generate a layout. The layout proposed by the system adopted the candidate semantic axes as the basic logical dimension for partitioning the service fields. Thus the semantic axes *income* and *pre-paid taxes* were entirely abolished from the proposed layout, having been replaced by the semantic axes of *stipendiary activities details, agricultural activities details, trading activities details, freelance worker details, real-estate income details* and *overseas activities income details*. Although the number of abolished semantic axes is lower than the number of the newly introduced ones, the overall number of pages in the proposed service layout was equal to the original page number (i.e. 13), due to the following reasons:

1. the *income* semantic axis contained 86 fields, thus it was initially divided into two pages of 42 and 44 fields.
2. the newly introduced semantic axes contained a smaller number of fields (16, 20, 23, 12, 19 and 14; note that these figures include the fields from the *pre-paid taxes* semantic axis as well), and was consequently possible to place two of these semantic axes in a single web page.

Furthermore, 18 fields were moved to different web pages, since they were assigned to the pertinent semantic axis *and* were more tightly coupled (through validation checks) with the fields of the page they were moved to. These rearrangements resulted to a decrease of the number of validation checks involving fields from different web pages, which was reduced to 34 (from its initial number of 49), accounting statistically for the 24.3% of the total errors detected in electronic submissions (from the initial value of 54%). The value of the partitioning quality metric *Sum of External Degrees* dropped from an initial value of 167.3 to 86.32, accounting for an improvement of 48.4% (a smaller value for the *Sum of External Degrees* metric indicates a better partitioning; the optimisation target is to minimize this value)

In order to verify the effectiveness of the new layout, an experiment was set up. In this experiment 216 citizens participated on a voluntary basis. The citizens' profiles varied, regarding their expertise on taxation issues: 14 of them were professional accountants, 131 stated that they were "quite familiar" with taxation issues and completed their tax return forms on their own, while the remaining 71 requested for help (either professional or from their family environment) to do their tax return forms. Familiarity with computers also varied in the sample, with all participants having used a computer at least once, but only 143 of them "feeling comfortable with computer usage" and 37 of them "having used an electronic service in the past". The sample was divided into two subgroups of 108 citizens each, taking care that the distribution of citizens with expertise (domain or computer) among the subgroups was fair. The first subgroup was requested to use the original form of the service, while the second subgroup was asked to use the service version with the optimised layout. Both services were hosted on the same machine, a Pentium 4/2 GHz with 512 Mbytes of RAM. Each service user was also given a printed copy of the tax return form (s)he would use in the experiment, for the purposes of familiarising with the layout and noting down figures, as needed. The software handling the submissions again

maintained logs of the validation errors that were encountered during the experiment and the users' navigation activities; this time, however, logs were correlated with the individual submitting the form, in order to facilitate coupling of behaviours with user profile information.

After the users had used the service, the logs were collected and analysed, and the resulting figures from the two subgroups were compared. The subgroup that used the optimised version was proved to need significantly less time for correcting errors (an average of 2:17 minutes, as opposed to 3:56 minutes needed by the other group) and also issued fewer web page requests after the first error was flagged (an average of 1.3 requests, while the second group had an average of 2.8 requests). Besides the improvements in the statistical figures in the error correction process, improvements were also identified in the corresponding figures for the *overall* document completion and submission process. More specifically, while the group using the original version completed the submission process in 13:32 minutes with an average of 16.8 page requests, the group using the version with the optimised layout needed 10:02 minutes and 14.3 requests, in average, to complete the same tasks. These improvements are ascribed to a more effective field placement, allowing service users to entirely skip certain pages they are not interested in and/or easily transfer figures from other documents (e.g. from income and pre-paid taxes certificates issued by various organisations, company books etc).

The following table summarises the data for the original version of the service and the version with the optimised layout.

	Original version	*Optimised layout version*
Number of fields	811	811
Number of semantic axes	12	16
Number of pages in the web service	13	13
Number of validation checks involving fields from different pages	49	34
Frequency of errors from these validation checks	54%	24.3%
Average user time in the error-correction process	3' 56''	2' 17''
Average web page requests in the error-correction phase	2.8	1.3
Average user time for document submission	13:32	10:02
Average web page requests for document submission	16.8	14.3

Fig. 6. Summarisation of experimental data

These figures suggest that the both the process of initial form filling and error correction are significantly facilitated by the layout optimisation procedure, and extraneous navigation activities are avoided.

5. Conclusions – Future Work

In this paper we have presented a scheme for improving the form layout of complex electronic services, by exploiting semantic information that is attached to form fields

by designers and validation checks that model the business rules governing the service. A prototype for a development environment has been created into which domain experts and IT staff enter information regarding the needed fields, semantic axes, validation checks and overall layout constraints, and the system automatically generates an optimal layout for the service, placing the fields on the appropriate web pages. This optimisation scheme has been verified through a case study on the Greek tax return form.

Future work includes the incorporation of a module for suggesting an optimal placement of fields within the same web page, the formulation of a set of guidelines for service designers covering the overall design phase of electronic services and studying the process of reverse-engineering existing electronic services for the purpose of layout optimization.

References

1. New York State Office for Technology. Forms Design Issues for Electronic Document Management Systems. available at http://www.oft.state.ny.us/cookbook/8_forms.htm
2. Adobe Systems Inc. Adobe e-Paper Products. http://www.adobe.co.uk/education/products/epaper.html
3. PureEdge. PureEdge E-Government Solutions. Available at http://www.pureedge.com/solutions/e-gov/
4. Scansoft. Omnipage 12 Product Information. Available at http://www.scansoft.com/eservices/
5. ELF Solutions. I-32 Forms Design. Available at http://www.elf-uk.com/
6. Fontware Inc., "Create/Form Form Server", http://www.fontware.com/section/products/formsolutions/formserver.asp
7. Ferreira C. E., Martin A., de Souza C. C., Weismantel R., Wolsey L. A. The node capacitated graph partitioning problem: a computational study. Mathematical Programming, 81 (1998) 229-256
8. Kuntz P. A distributed heuristic for finding clusters in vertex sets. International Symposium on Mathematical Programming, Lausanne (1997)
9. Hao, J., Orlin J.B. A Faster Algorithm for Finding the Minimum Cut in a Graph. Proceedings. of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms, Orlando, Florida (1992) 165-174
10. Hendrickson B., Leland R. A Multilevel Algorithm for Partitioning Graphs. Proceedings of the Supercomputing 1995 Conference, San Diego, CA, December 4-8 (1995), available at http://www.supercomp.org/sc95/proceedings/509_BHEN/SC95.HTM
11. Chekuri C., Goldberg A., Karger D., Levine M., Stein C. Experimental Study of Minimum Cut Algorithms. Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (1997) 324-333
12. Hendrickson B., Leland R., can Driessch R. Skewed Graph Partitioning, Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, Philadelphia, PA, (1997)
13. Chaco 2.0 User's Guide, available at ftp://ftp.cs.sandia.gov/pub/papers/bahendr/guide.ps.gz
14. SCOTCH team. Static mapping, graph partitioning, and sparse matrix block ordering package. Available at http://www.labri.fr/Perso/~pelegrin/scotch/
15. METIS team. METIS Family of multilevel partitioning algorithms web site. Available at http://www-users.cs.umn.edu/~karypis/metis/index.html

16. UCSD VLSI CAD laboratory. Hypergraph Partitioning Slot. http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/Partitioning/#V
17. Walshaw C., Cross M., Everett M. Mesh partitioning and load balancing for distributed memory parallel systems. In Proceedings of the Parallel and Distributed Computing for Computational Mechanics, Lochinver, Scotland (1997)
18. hMetis project. Hypergraph Circuit Partitioning. http://www-users.cs.umn.edu/~karypis/metis/hmetis/
19. Karypis G., Aggarwal R., Kumar V., and Shekhar S. Multilevel Hypergraph Partitioning: Applications in VLSI Domain. Proceedings of the ACM/IEEE Design Automation Conference, pp. 526-529 (1997)
20. Karypis G., Kumar V. Multilevel k-way Hypergraph Partitioning. Proceedings of the ACM/IEEE Design Automation Conference, pp. 343-348 (1999)
21. hMetis project. hMETIS manual. Available through http://www-users.cs.umn.edu/~karypis/metis