4

Assisted form filling

Łukasz Bownik, Wojciech Górka and Adam Piasecki
Centre EMAG
Poland

1. Introduction

Semantic technologies that form the basis of the idea of the Semantic Web (Berners-Lee et al. 2001) are currently one of the most popular points of interest in the field of computer science. The main emphasis here is on issues related to the logical layer of semantic applications, that is exploring the methods of machine reasoning within an ever wider range of logic, and the issues of data exploration and integration. This article features a practical application of semantic technologies for solving problems more closely affecting an average computer user. We present a solution to the issue of accelerating the filling of electronic forms based on previous achievements in this field, which combines the capabilities of inference engines with the expressiveness of ontologies stored in OWL (Web Ontology Language) (OWL Guide 2004) and flexibility of data stored in RDF (Resource Description Language) (RDF Primer 2004). The article assumes that the reader has a basic knowledge on the construction and use of the RDF and OWL languages.

1.1. Objective of the article

The objective of this article is to describe ways to facilitate electronic form filling. At the beginning we described the issue and the requirements placed on the solution. Then we described the solution. We characterized the reasons for the selection of specific techniques and architecture of the proposed solution, as well as two alternative approaches to the interaction with the user. Finally, we presented the investigated results of the acceleration of assisted form filling, which was compared with other existing solutions of the stated issue.

1.2. The Issue

The basic issue associated with the concept of public services is the need to fill the forms containing repeated data. Most forms require entering the same data, similar data sets or data dependent on other data contained in another part of the form. This causes psychological discomfort associated with repeated, from the user's perspective, contents. Such an approach is not only unproductive but also increases the risk of errors. Errors in the completed forms, if not detected during the interaction with the system, may have serious and far-reaching consequences, including legal ones. A mechanism capable of assisting in the process of electronic form filling would not only result in a positive acceleration of the process but could also potentially reduce the risk of errors.

1.3. Requirements

The proposed solution of the stated issue is to assist the user in the process of filling in any electronic form through the automatic and pseudo-intelligent filling of the form fields with information available to the system. The basic assumptions that guided the search for solutions were:

- modularity of the developed software that will allow integration with a broad class of information systems;
- independence from the domain of the form which will minimize the number of assumptions about the meaning and format of processed data;
- minimization of interactions with the user that is both a target and measure of the quality of the resulting solution.

The result of research is a portable Java library that implements assisted electronic form filling processes. Later in the article we will present the results of the work and a description of the architecture and operation of the developed software, hereinafter referred to as "the forms module".

1.4. Other solutions

Similar solutions have been proposed, *inter alia* in the projects "RoboForm"[1] and "Mojo Navigator"[2] however they focus primarily on the functionality of a portfolio of passwords integrated with a web browser with an additional history of the field values associated with the forms available on the Internet. They also lack the dynamics associated with the automatic acquisition of data, which is presented by the described solution. On the market there are a lot of other solutions based on a dialogue form of form filling but most of them are firmly based on the dialogue recorded on a particular form (the wizards). Finally, it should be noted that a simple form of field values suggestion is available directly in browsers. However, it is characterized by a very small degree of ergonomics both with regard to the interaction with the user, who must enter the desired values prefixes, and the lack of contextuality, which makes it impossible to fill the whole form at once.

[1] http://www.roboform.com, 2009-01

[2] http://www.mozilla.org/projects/ui/communicator/browser/formfill, 2009-01

2. Design

The following section describes the design of the forms module along with the reasons for selecting particular data representation and architecture.

2.1. Data representation

The main idea of assisted form filling is pseudo-intelligent matching of existing data to form fields. For this purpose, it is necessary to identify the meaning of the fields and sections of the form and find an adequate available data set. The data for assisted filling may come from the user's ontoprofile, whose task is to accumulate the data from the completed forms, or from an external system. In the case when the data are not available (as it will be the case with every new user), assisted filling will not be available either.

Identification of the meaning of data requires their explicit tagging with metadata. For this reason, we need data representation that supports their binding to the metadata. The binary representation and simple text representation, for example in the csv format, do not allow to record the metadata and therefore are not useful. For this purpose, the most suitable are the relational data model (in whose case the metadata consist of names of relations and fields) and languages like XML (eXtensible Markup Language) (XML 2006) and RDF.

The objective of this work was to develop software with a minimum number of assumptions about the meaning and format of the processed data. In the relational data model the meaning and format of data is contained in the database schema that can be subjected to a variety of analyses performed by the software. The relational database schemas have, however, two significant drawbacks. First, the producers of databases quite freely implement the DDL language standard and extend it with many product-specific features. Such diversity results in considerable difficulty in automatic analyzing of database structures. Second, solving the issue of assisted form filling requires maximum flexibility of data representation. The relational database model does not satisfy this condition as it assumes constant or slowly changing structure of the database. Application of the representation of data based on this model would be a very hard assumption regarding the meaning and format of data in a particular installation (e.g. installing a module with respect to a database which describes the forms of medical service appointments would allow to fill only this type of forms). For this reason, the relational data model has been rejected as a representation of the processed data.

Rejection of the relational model narrowed the choice of data representations to those based on XML and RDF. Data stored in these languages are very dynamic in their structure defined by XML schemas and ontologies that can be dynamically loaded and analyzed by an application. XML requires that all data have a tree structure. This structure does not fully correspond with the "natural" structure of data which more often resembles graphs. For this and other less significant reasons, the RDF language was chosen because it allows very natural data recording in the form of graphs and a flexible description of their structure by means of ontologies stored in OWL. An additional advantage is the possibility of using machine-based inference with ontologies and rules which allowed the development of pseudo-intelligent software by means of very expressive logic programs.

2.2. Structure

Data structures on which the module operates are defined by means of ontologies written in OWL. In order to understand the operations of the dialogue module, it is necessary to grasp the data ontology and the form ontology (a kind of data ontology) ideas which lie at the basis of the process.

2.2.1. Data ontologies

Data ontologies define the vocabulary (concepts) used in the course of data exchange between systems. A data ontology is not meant to organize knowledge (which is the case of expert systems) but to determine concepts describing the data which can be understood by many systems. In terms of their concept, data ontologies correspond with XML diagrams and are their functional equivalents.

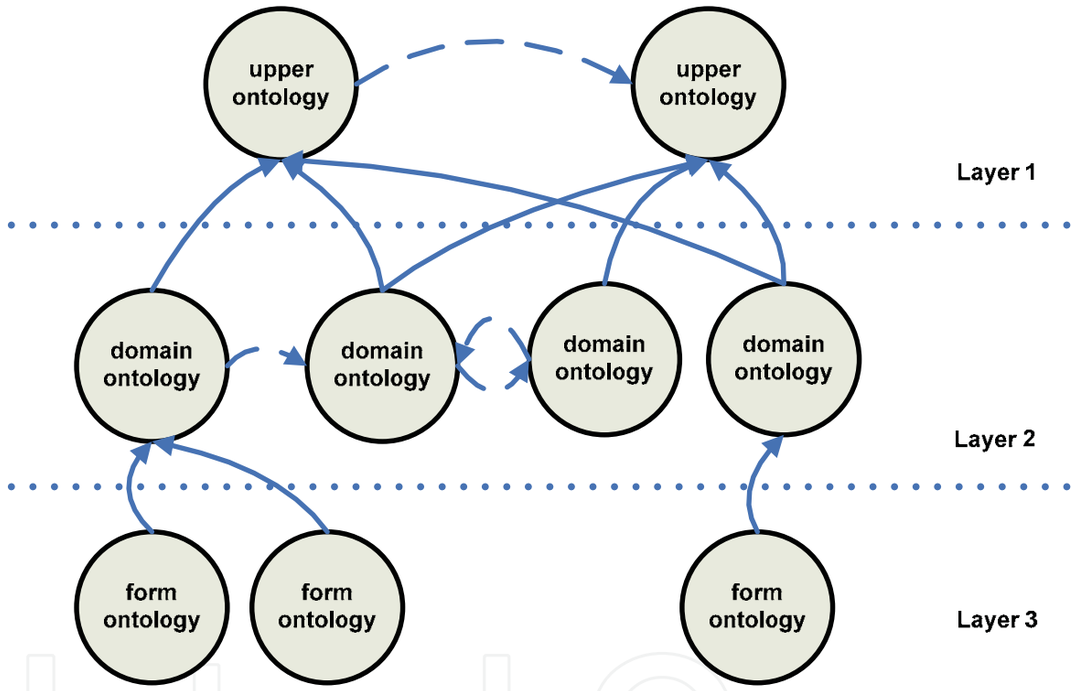

Fig. 1. Layered-tree ontology structure [own].

Data ontologies are naturally arranged in a tree-layer structure shown in Figure 1.
The first and highest level contains definitions of the most general vocabulary (concepts) commonly used by many IT systems (e.g. *person, age, name, family name*). The concepts defined at this level are the basis for the definitions of more specialized concepts placed at lower levels of abstraction. **General ontologies should not (must not, to be precise) impose any restrictions** (e.g. in the form of limits as to the format of field value or number of fields in the form) on the defined concepts because each restriction placed at this level of generalization will diminish the universal force of the defined vocabulary and will raise the probability of conflicts with restrictions defined at lower levels (redundant or contrary restrictions). General ontologies must have total horizontal coherence, i.e. mutual coherence within the entire level of abstraction.

The second, middle level contains definitions of concepts shared by IT systems which operate within a given domain (e.g. medicine). These ontologies should use, to the highest possible extent, the concepts defined at a higher level of abstraction (i.e. they should not define their own concepts indicating, for example, *family name*) as well as add the concepts from this very domain (e.g. *clinic, doctor, vaccine*). At this level of abstraction **only most obvious restrictions** should be defined, i.e. the restrictions which will be forced by all existing and future IT systems operating within a given domain. If it is not possible to explicitly define such restrictions, one should not define them at all because, as it is the case with general ontologies, they may provoke conflicts between ontologies. Domain ontologies must have horizontal coherence within a certain domain, i.e. mutual coherence within each domain at this level of abstraction as well as coherence with general ontologies.
The third and lowest level contains definitions of forms, i.e. specific data packages exchanged between IT systems. A form is a structure of data exchanged between two IT systems. The structure may, but does not have to, be related to the form filled by the system user. Each form there should have a form ontology describing its structure. The ontologies of the form should use, to the highest possible extent, the concepts defined at higher levels of abstraction and **define only such new concepts which are specific to a given form** and cannot be shared with other forms. The form ontology should define all necessary restrictions which will ensure logical coherence and correctness of the form required by the specifications of the IT system. Since for each form there are different rules (restrictions) which determine its coherence and correctness (e.g. one form requires an ID number, the other does not), it is not advisable to define restrictions at higher levels of abstraction. The form ontologies do not have to demonstrate any horizontal coherence (it is practically impossible to provide it). A logical error resulting from horizontal incoherence will never occur because there is separate processing of each ontology in each processing path in the IT system, which was depicted in Figure 2. The form ontologies must be coherent with the applied domain ontologies and general ontologies.

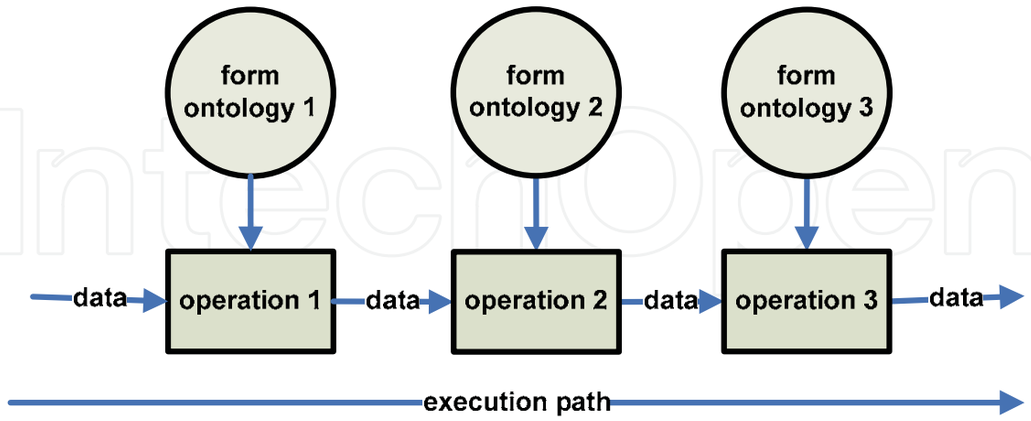

Fig. 2. Physical separation of processing of form ontologies within a single processing path [own].

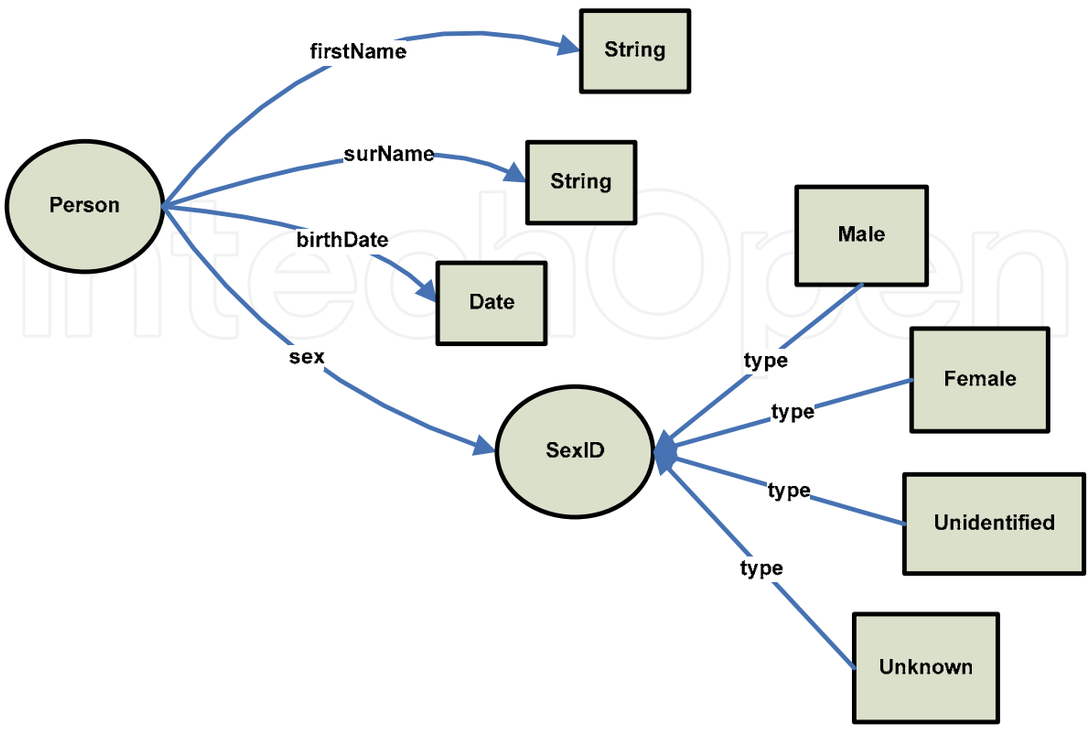

Fig. 3. Example of data ontology [own].

Figure 3 shows a sample ontology describing personal data.

2.2.2. Form ontologies

Form ontologies define the structure as well as coherence and correctness rules of data contained in the described form. A form ontology describes a specific set of data exchanged between IT systems.

The process of constructing a form ontology begins from building the main form class which will lie at the roots of the whole structure. The main class of the form must be marked with the appropriate property `isFormRoot`. Thanks to this assumption, it is possible to automatically detect the root of the form. The definition of the successive sections of the form is carried out by defining separate classes (e.g. *Person* class or *Address* class) linked with object properties to form a tree structure. The definitions of data fields are carried out with the use of data-type properties of classes which make up the sections of the form.

As the form ontology is based on concepts defined in data ontologies which may be much more extensive than the form requires, the `isRequired` and `isOptional` properties were defined. They connect the desired properties with the classes being the sections of the form (see Fig. 4). In order to force the order of processing sections and fields of the form, it is necessary to define the processing order for each pair of properties, representing these sections, by means of the *askBefore* property (see Fig. 4).

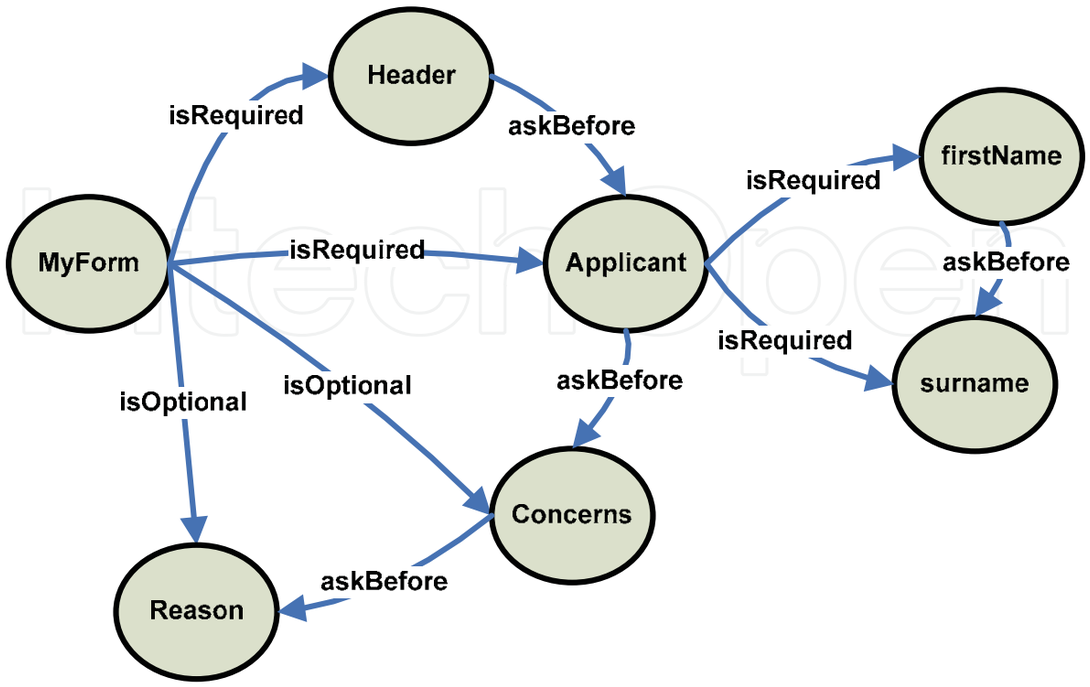

Fig. 4. An example of form ontology[own].

In order to define the desired restrictions, it is necessary to determine new concepts due to the fact that the built-in cardinality concepts have their own defined semantics which does not suit the issue to be solved (OWL Guide 2004).
Form ontologies can contain rules (Horn clauses ((Horn 1951) that extend the functionality of the OWL language. In order to secure logical non-contradiction, the rules must belong to the "DL-safe" set (Grosof et al. 2003). The most frequent application of the rules is the structure of section labels (enabling, for example, creation of a section label for personal data on the basis of the entered name and surname) and automatic assigning of values to certain fields based on the values of other fields.

2.3. The operating principle

The operating principle of the module is based on the fact that a form makes up a tree structure in which successive embedded sections make branches, while fields – leaves of the tree. Thus it is possible to process a form with the use of tree-searching algorithms. To fulfill this task the depth-first search algorithm was selected. This algorithm is the most compliant with the way people fill real forms. The functioning principle of the dialogue module is presented below in a simplified way. While analyzing successive steps one should remember about the iteration-recurrent nature of the algorithm.

1. At the beginning of each dialogue, the forms module detects a proper form class in the form ontology and creates its instance.
2. The newly created instance becomes a currently processed node and, at the same time, the top of the tree.

3. For the currently processed node, all required and optional properties are found and sorted according to the arrangement determined in the ontology. Then the successively found properties are processed.
4. If the currently processed property points at a section of the form, the section is then processed recursively.

In the course of processing the successive properties of the node, the dialogue module tries to detect, in the set of input data, the best adjustments both for the single fields of the form and for the whole sections. The dialogue module asks the user questions only in situations when the desired values do not exist or when it is possible to adjust more than one value to one field or section.

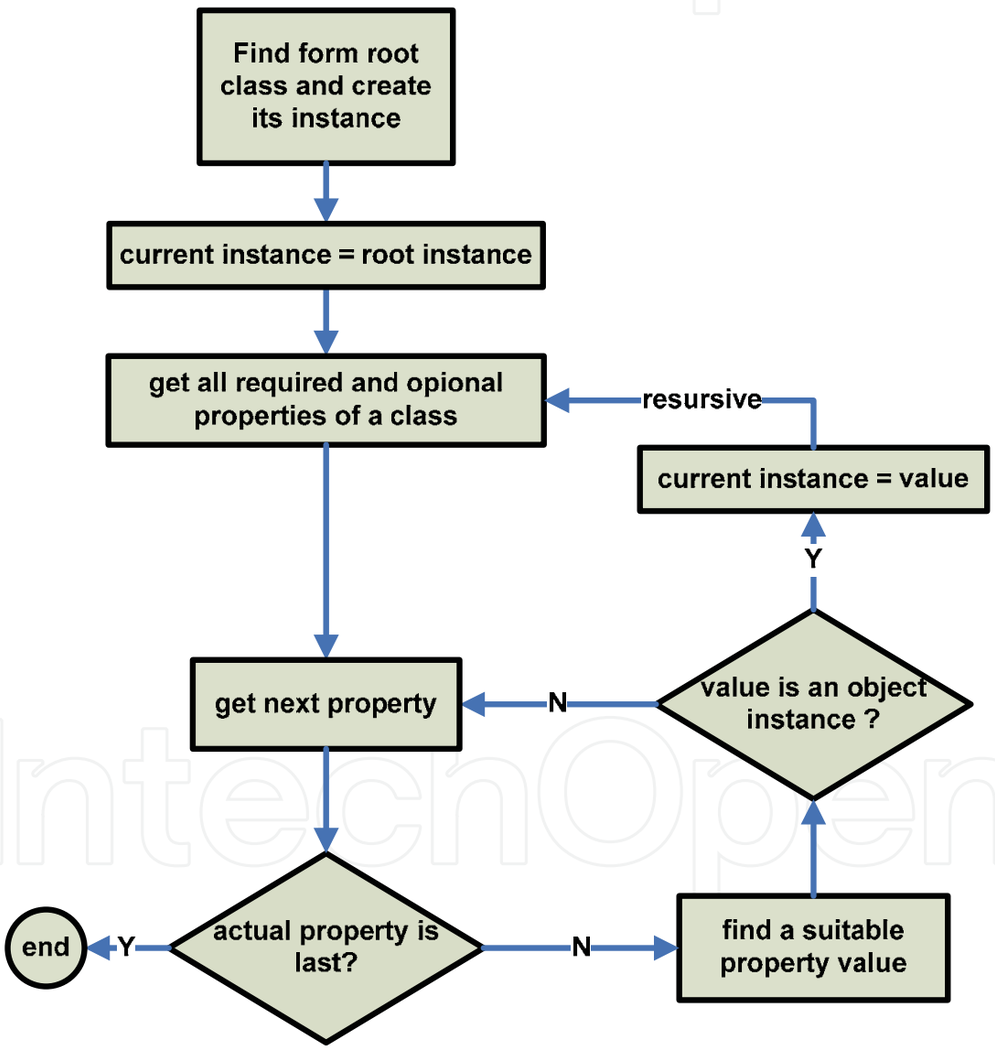

Fig. 5. A simplified version of the algorithm. [own].

During the subsequent processing of the properties of the node the software tries to find in the collection of input data the best match of both the individual form fields as well as the

entire section of the form. The forms module communicates with the user only in situations where the desired values do not exist or if it is possible to fit more than one value into onefield or section. A simplified version of the algorithm is presented in Figure 5.

2.4. The reasoning

The proposed machine reasoning is mainly based on the OWL language but is limited in its expressiveness to a subset called DLP (Description Logic Programs) (Grosof et al. 2003). DLP is a logic system that is a subset of a description logic, on which the OWL language is based, that can be expressed by means of Horn clauses (Horn 1951). DLP provides the key inference capabilities found in OWL necessary for the implementation of assisted form filling, that is the reasoning with class hierarchies and properties. In addition, it allows flexible extension of specific rules for the process itself or even for a specific form.
Applying an inference engine brought a significant simplification of the code responsible for retrieving the data with required meaning through moving a significant computing burden into a logic program which contains the rules of logical inference. This way, the matching algorithm is limited to the search of resources belonging to the respective classes or values of the respective properties.

2.5. Architecture

The architecture of the proposed solution is briefly described below. The basic elements of the forms module are responsible for performing assisted filling process logic, and storing the data gathered during filling forms. The user interface and data access layers have been omitted.

2.5.1. The form processor

The forms module is composed of four basic parts:

- RDF storage providing access to RDF data and ontologies[3] ;
- a forward inference engine based on the RETE algorithm (Forgy 1979);
- a module that controls the process of form filling developed as a result of the work (implemented in Java), which, together with two previous elements, makes up a forms processor;
- an ontoprofile allowing the storage of the data collected during form filling processes

Figure 6 presents a layered structure of the forms module.

[3] Jena – A Semantic Web Framework for Java. http://jena.sourceforge.net, 2009-01

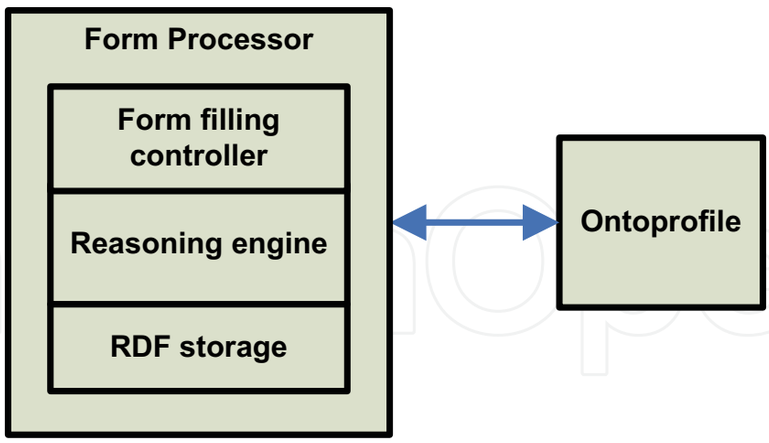

Fig. 6. The forms module structure [own].

Effectiveness of the module depends on the input data stored in the user profile.

2.5.2. The ontoprofile

The ontoprofile is used to store data collected during the assisted form filling processes which are used to support the filling of successive forms. Its main task is to integrate data coming from completed forms with the data collected so far and to make these data accessible to the successive forms. Proper integration is driven by the integration policy which determines what should be integrated and how. The data contained in the ontoprofile are stored in the form of RDF graphs. A physical way in which they are stored is not important but for the prototype installation a relational database has been used.

3. Prototypes

The following describes the two prototypes built during the experimental work. We described the way of interaction with each of the prototypes and the reason for the rejection of one of them.

3.1. The dialogue version

The first prototype of the forms module assumes a dialogue based interaction between the user and the computer in the form of a sequence of questions and answers, which recalls the manner of interaction with the wizard. The questions are asked by the software, and the answers are provided by the user. The forms module operates as a filter of information. With a set of input data (may be empty) and ontologies, the forms module "tries to" fill in the form with input data and data supplied by the user. Filling the form takes on a dialogue with the user while maintaining the principle of minimal interaction which assumes that the user is queried only in such cases where the data are unavailable or inconclusive. The usefulness of the forms module, as measured by the number of interactions with the user (the fewer the better), increases with the amount of input data available in the system.

In order to illustrate the functioning of the dialogue module let us check how the module works with a simple form. We assume that it is necessary to fill the form as follows:

Form:
Personal data of the user:
Name;
Surname;
Address:
City;
Street;
House No;
Purpose of submitting the form;

When the process of filling the form is invoked for the first time, the dialogue with the user will look as follows:

System: Enter "Name".
User: John
S: Enter "Surname".
U: Brown
S: Enter "City".
U: London
S: Enter "Street".
U: Portland Place
S: Enter "House No".
U: 47
S: Enter "Purpose of submitting the form".
U: Application for the permission to use a company car

After filling and accepting the form, the data from the form are stored in the user's ontoprofile. At the next interaction between the user and the module, the dialogue will look as follows:

S: I have found personal data for "John Brown, Portland Place 47, London". Shall I apply them (Y/N)?
U: Y
S: Enter "Purpose of submitting the form".
U: Application to access financial data.

As one can see in the above successive filling of the form, the number of necessary interactions with the user is 6 times smaller. The reduction was possible due to the fact that the forms module was able to automatically fill the fields with available personal data. Asking the question about the field "Purpose of submitting the form" results from the updating policy of the user's profile – this issue, however, is beyond the scope of the paper. Please note that the functioning of the module is based on the real meaning of data defined

in an ontology which allows to move data between different forms using common concepts defined in the shared data ontologies.

3.2. The form view based version

The dialogue based version of the forms module described in the previous chapter is an initial prototype that enabled to verify the validity of the approach and demonstrated the ability to achieve the stated goals. From the point of view of ergonomics, however, the use of the prototype proved to be inadequate. The assisted form filling based on a dialogue which involves asking the user about the values of consecutive fields turned out to be uncomfortable. The user was constantly losing context in which the currently filled field existed. He/she was also unable to determine at what stage of the process he/she currently is and when the process will be completed. In addition, the inability to go back in the process of form filling made the solution unacceptable.

These problems resulted in a departure from the dialogue based approach to a more natural way, based on the form view, in which the user can freely choose the order in which he/she completes the form, and freely modify the contents of already completed parts. In this prototype the software tries to pre-fill the largest possible part of the form, at the same time giving the user the right to accept or change the suggested values. This approach has significantly increased the comfort of working with the software, which no longer imposes the schema of work with the form and brings its action to the role of an assistant. It should be noted that the change in the interaction has not changed the idea of the forms module and its architecture.

The operation rules of the assisted filling based on the form view will be described by examples. The following is a series of screenshots of a working forms module.

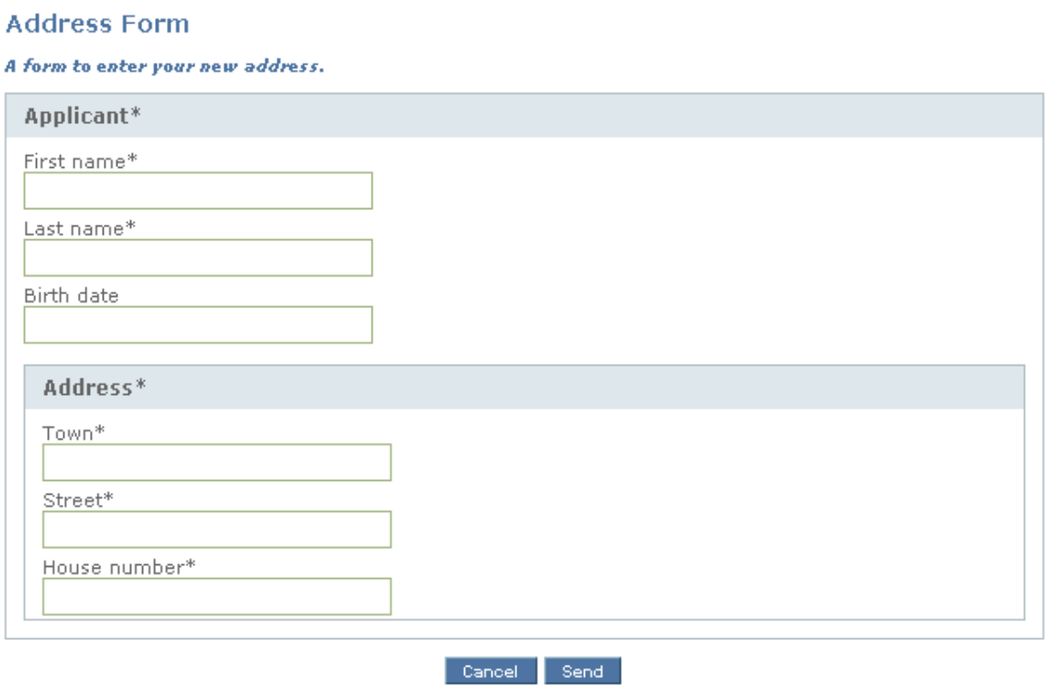

Fig. 7. An empty form.

Figure 7 presents a blank form at the first run. The form is used to update the address of a system user. A user profile does not contain any data yet because no form has been completed so far, so all fields are empty.
Figure 8 shows the same form after the introduction of sample data. After submitting the form to the system, the data will be saved in the user's ontoprofile. Their meaning will be stored as metadata that will come from the form fields from which they originate.
Figure 9 shows a re-run of the forms module. In this case, the software was able to retrieve data from ontoprofile and attribute it to the form according to their meaning. As you can see, it is so accurate that the form no longer requires any interaction with the user. In addition, the software was able to create objects representing labels of the identity and address data and place them in dropdown boxes located below the sections headers. This enables the user to select alternative proposals. The lists also allow to select an empty position which deletes the contents of each field in the section.
Figure 10 represents a situation in which the user has deleted the contents of the form fields and introduced the new values of personal data. In an attempt to minimize the number of necessary interactions, the software suggests him/her an existing address value stored from a previous interaction. The user can choose the proposed address or enter new values.

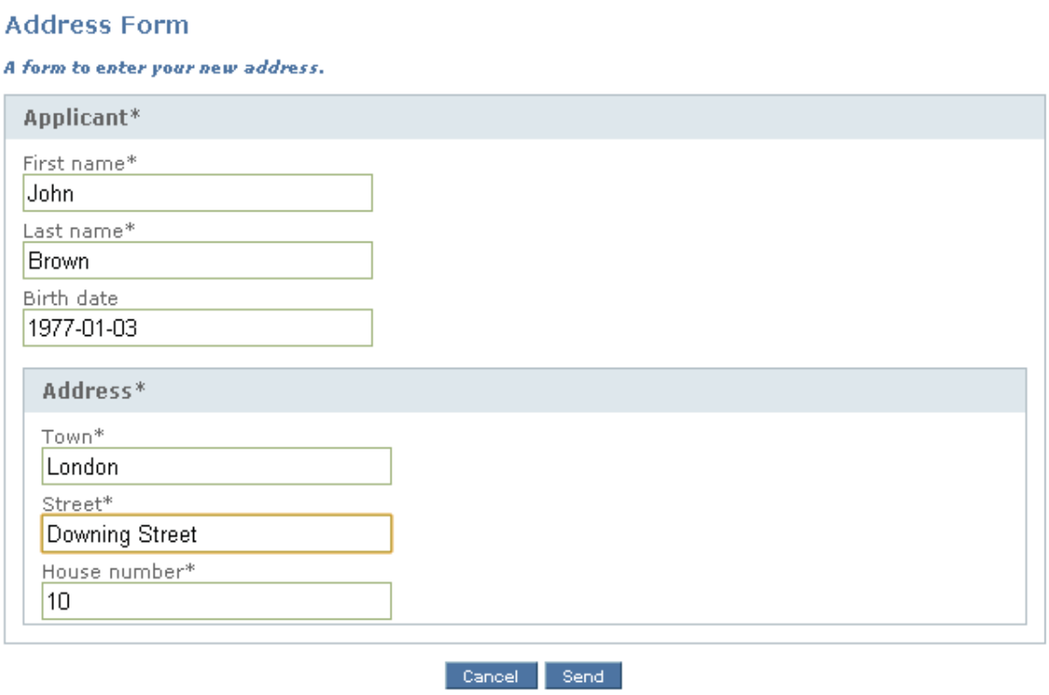

Fig. 8.A fully filled form.

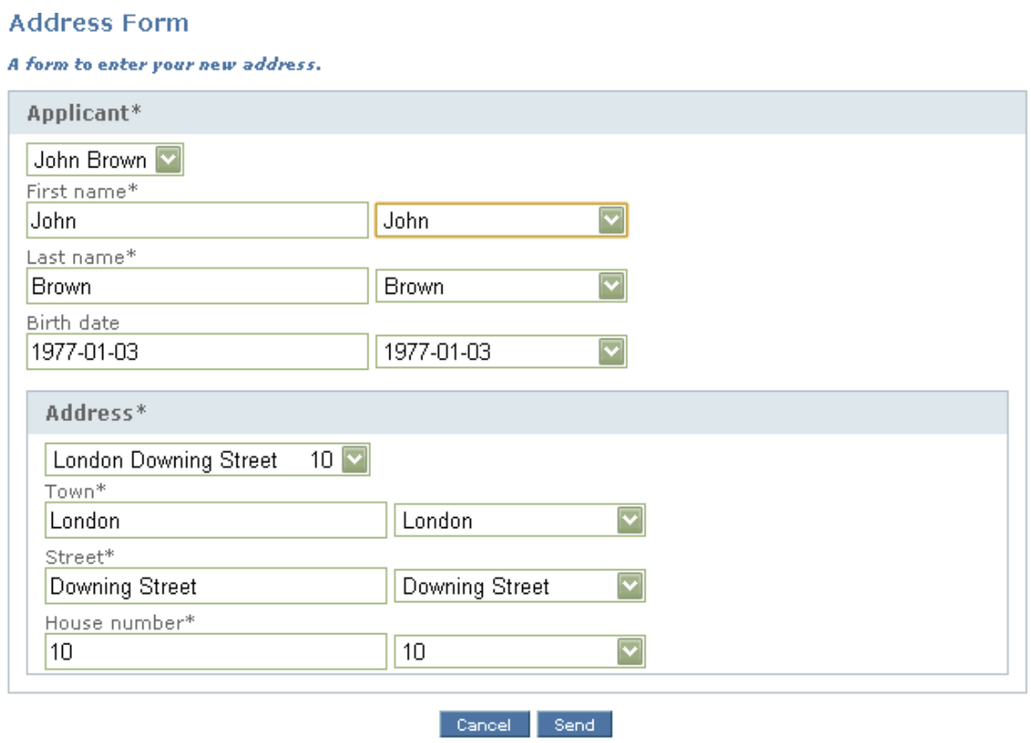

Fig. 9. Second run of the forms module.

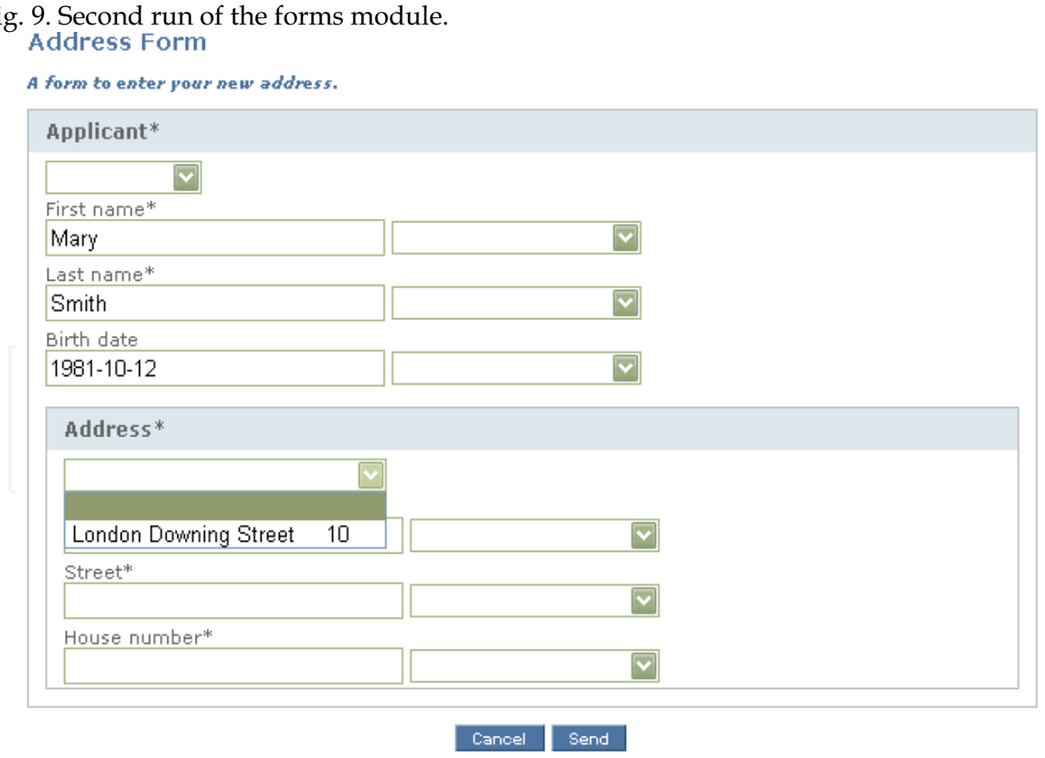

Fig. 10. Changing the data during a second run of the forms module.

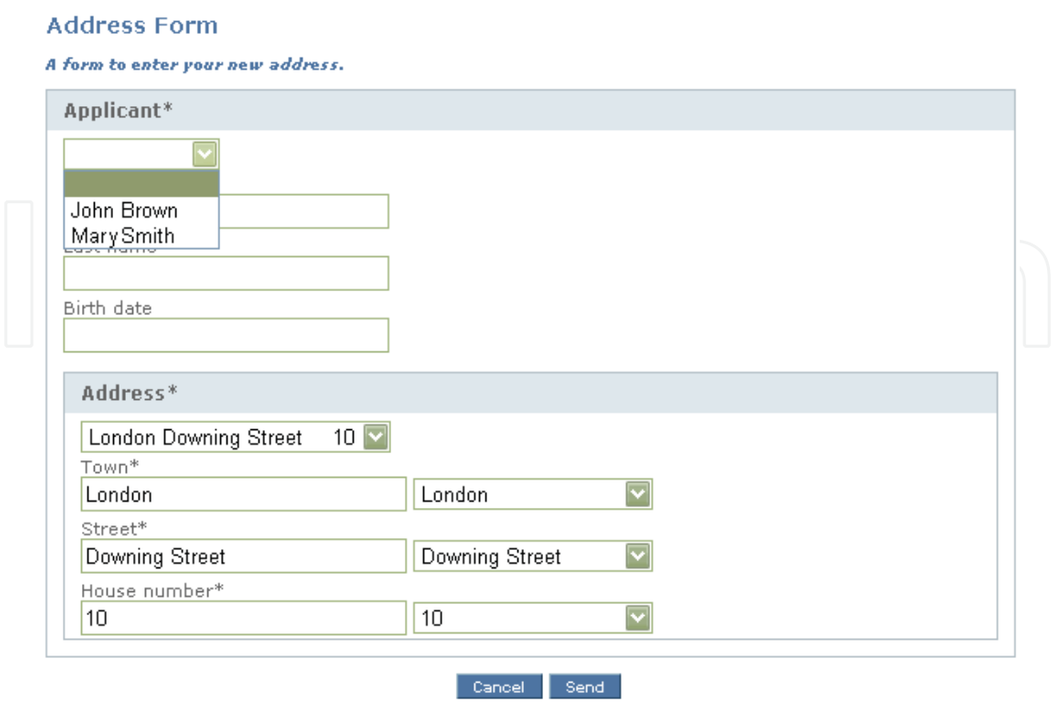

Fig. 11. The third launch of the form module.

Figure 11 demonstrates the third launch of the form module. In this case, the software found two objects with the meaning matching to the "Applicant", and therefore allowed the user to select from two proposals.

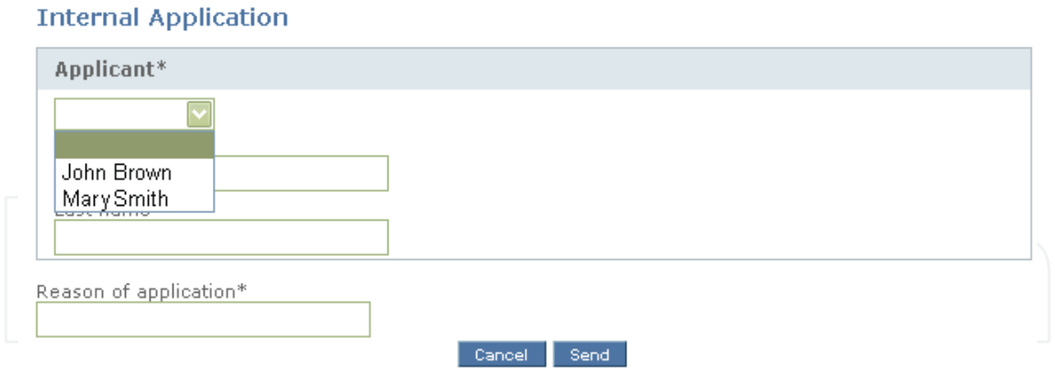

Fig. 12. Matching data to a different form.

In Figure 12 there is a different form launched for the first time. The user's ontoprofile already includes data gathered during the processing of earlier forms, which can be used in assisted filling. It should be noted that although the form shown in Figure 12 is different from the one you saw in the previous figures, the semantic description of the data and forms made it possible to match the correct data to the corresponding fields and sections of the form. Such a solution makes it possible to transfer data between forms if only they are assigned the same meaning.

4. The acceleration measurements of assisted form filling processes

The measurements of acceleration of assisted form filling processes were made on the two forms shown earlier and on a more complex form shown in Figure 13. The measurements allowed to estimate the degree of acceleration of the process of assisted form filling in relation to the time required to fill out the forms without support.

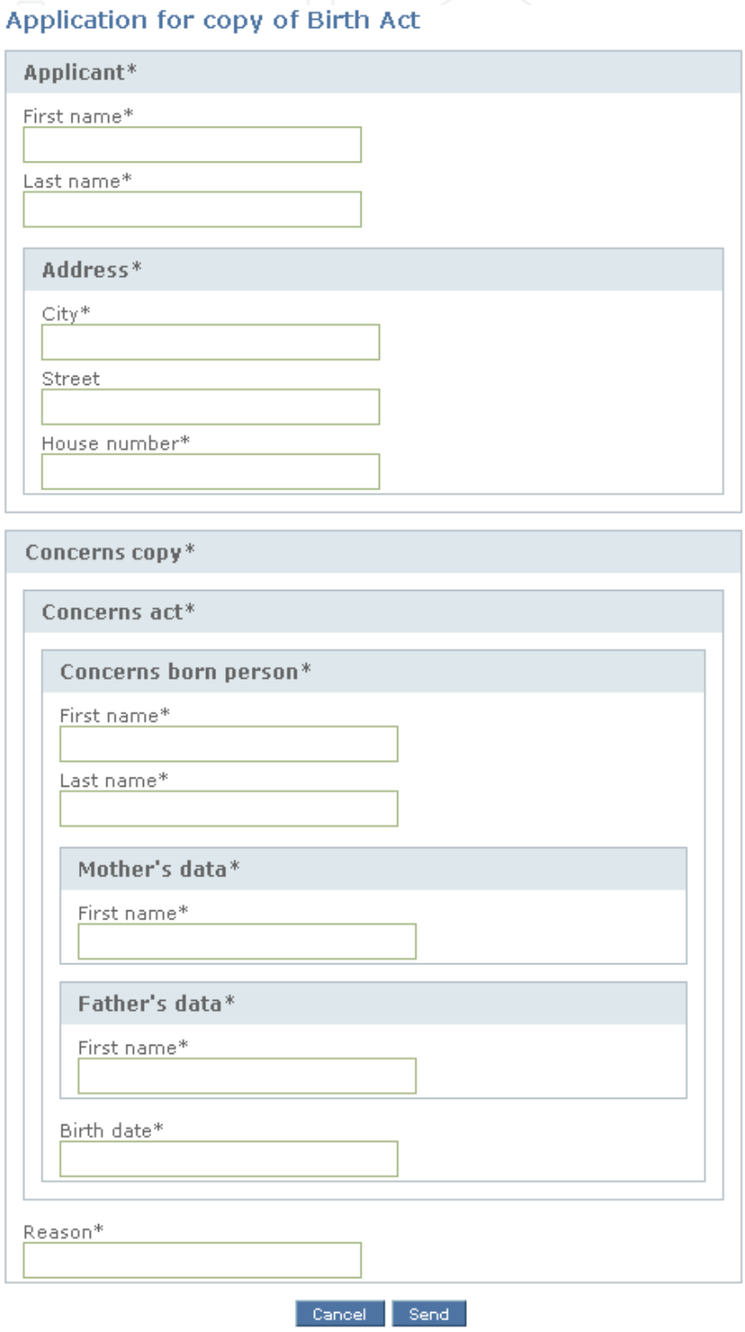

Fig. 13. A complex form [own].

4.1. The estimation of maximal acceleration

To estimate the maximum acceleration it was required to construct test scenarios involving the best (fastest) and worst (the most interaction intensive) processes of filling in the forms. For this purpose, we constructed the following test cases:

- T1 – the user manually completes all fields (the ontoprofile is empty) - requires a large amount of interactions (key presses) because, due to the lack of data, the assisted filling functionality is not available;
- T2 - the user accepts all proposed values of the fields making up only those which are missing - this case is optimal;
- T3 - the user deletes all proposed fields and brings his/her own values - the worst case because, apart from entering the data, it is necessary to delete already suggested values;
- T4 - for each section the user selects a proposed value from the list - it is an almost optimal case but most often it appears while using a module.

Before we began to measure real acceleration of assisted form filling processes we had estimated the maximum possible accelerations. The estimates were made by counting the number of interactions (clicks and key presses) required to fill each form. This quantity was obtained by counting all actions (filling in the fields, selecting a value from the list, etc.) necessary to complete the case at the optimal test conditions, and then by multiplying them by the appropriate weight of each action. We defined the following actions:

- approval of the form = 1 interaction (a mouse-click on the button);
- selecting a proposed field or section value from the list = 2 interactions (a click on a dropdown list unroll button and a click on the selected item);
- filling in a field = 7 interactions (6 key presses corresponding to the average length of words in the Polish language and a tab key press to make a transition to the next field).

Test case	Number of interactions		
	Internal Application	Address Form	Application for copy of Birth Act
T1	22	43	78
T2	8	1	1
T3	24	45	82
T4	10	3	12
Average acceleration	2,59	29,33	43,33

Table 1. Estimated maximum acceleration of the process of filling in the forms using the forms module [own].

Table 1 presents the estimated peak acceleration of the form filling processes. As you can see, these estimates are very high and the most complex form called *Application for copy of Birth Act* can be traced up to even 40-times here. In addition, it can be seen that the

acceleration increases with the complexity of the form (Figure 14). This is due to the fact that with an increase in the nesting of sections of a form the efficiency of the process increases too, because it is possible to suggest values of the whole form tree branches.

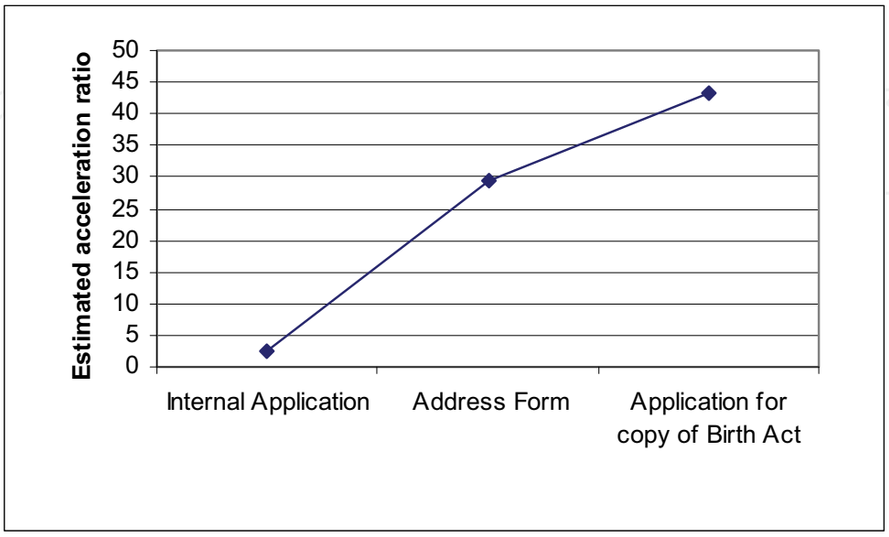

Fig. 14. Estimated acceleration of the process of filling in the form in relation to the degree of complexity, with the use of the module forms [own].

To highlight the difference of the solution, we estimated the acceleration of the processes of filling the same forms with the use of the functionality available in the Firefox 3 browser. In this case, the following events were recognized as actions:

- approval of the form = 1 interaction (a mouse-click on the button);
- selecting the proposed value for a field = 3 interactions (typing the first letter which activates the auto complete option, a click of the mouse selecting a proposed value, transition to the next field with a tab key press);
- filling in a field = 7 interactions (6 presses of the keys corresponding to the average length of words in the Polish language and a tab key press to make the transition to the next field).

Table 2 presents the results of the calculation of the estimated acceleration for specific test cases. The acceleration growth as a function of the complexity of a form is indicated in figure 15.

Test case	Number of interactions		
	Internal Application	Address Form	Application for copy of Birth Act
T1	22	43	78
T2	10	19	34
T3	22	45	82
T4	10	19	34
Average acceleration	2,20	2,32	2,35

Table 2. Estimated maximum acceleration of the process of filling in the forms using the mechanism available in the Firefox 3 browser [own].

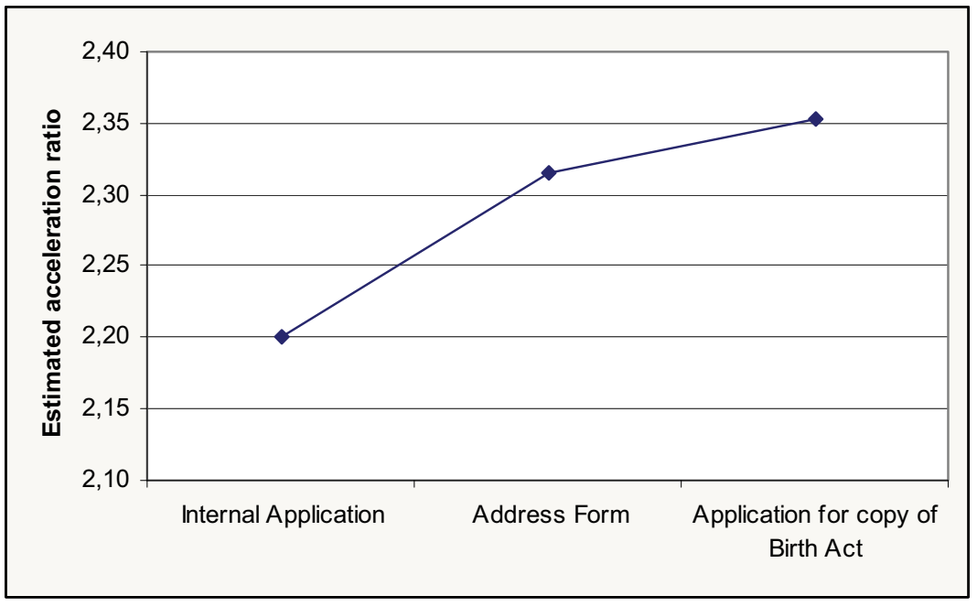

Fig. 15. Estimated acceleration of the process of filling in the form in relation to the degree of complexity, using the mechanism available in the Firefox 3 browser [own].

As you can see, the difference between the estimated acceleration offered by the forms module as compared to the one offered by Firefox is dramatic as the former exceeds the latter 20-times. In addition, it should be noted that the solution offered in the browser does not show a significant growth of acceleration with the increasing degree of complexity of the form (only about 5%).
In the following section we present the results of tests performed on the acceleration of assisted form filling with the users' participation. The results will verify the above estimates.

4.2. Acceleration measurements with the users' participation

After having estimated the maximum acceleration of the forms completion, actual measurements were made with test users. The group consisted of four people. The first

three had no previous contact with the tested software, while the fourth was one of its developers, thus an advanced user. The scenario was to perform four test cases described earlier for each of the three sample forms. To reduce the size of the table we changed the names of the forms to:

- F1 - Internal Application
- F2 - Address Form
- F3 - Application for Copy of Birth Act.

We measured the time the users took to fill out the forms in the specific test cases. The acceleration of the form filling processes has been defined as the average ratio of the best to the worst cases. The acceleration was calculated using the formula below, where a is the acceleration of the process of filling a specific form, and t_n is the time of completion of the n-th test case.

$$a = \left(\frac{t_{T1}}{t_{T2}} + \frac{t_{T1}}{t_{T4}} + \frac{t_{T3}}{t_{T2}} + \frac{t_{T3}}{t_{T4}}\right) / 4$$

Then we calculated the average acceleration for each user (Table 3). At this point, you may notice a significant difference between the acceleration achieved by the new users and the value of acceleration achieved by the advanced user (User 4) which was two times higher. For this reason, the fourth user's samples were considered abnormal and excluded from further calculations.

Finally, the arithmetic means of accelerations were calculated for each form (on the basis of the results of users 1, 2 and 3) as well as the arithmetic mean of acceleration for all forms which is 3.36 times.

Then we measured the accelerations of form filling processes using mechanisms available in Firefox 3. Because of a different mechanism, the test cases T3 and T4 have become synonymous with cases T1 and T2 (the same scenario of interaction) and therefore have been omitted. The results of the measurements can be found in Table 4.

Then we calculated the arithmetic means of the acceleration for each form, and the arithmetic mean for all forms, which was 1.13 times.

Scenario	Time needed to fill in a form(in seconds)										
	User 1			User 2			User 3			Us	
	Time	Acceleration		Time	Acceleration		Time	Acceleration		Time	A
			Average			Average			Average		
F1-T1	12			46			25			10	
F1-T2	10			24			14			6	
F1-T3	17			23			55			14	
F1-T4	7	1,76		23	1,47		14	2,86		8	1,7
F2-T1	22			51			35			18	
F2-T2	4			16			11			1	
F2-T3	31			46			62			23	
F2-T4	5	5,96		10	3,94		15	3,82		15	10,9
F3-T1	54			110			68			49	
F3-T2	16			38			24			5	
F3-T3	65			131			78			52	
F3-T4	17	3,61	3,78	28	3,74	3,05	23	3,11	3,26	11	7,3

Table 3. The results of the measurements of acceleration of the assisted form filling using the forms n

Scenario	Time needed to fill in a form(in seconds)										
	User 1			User 2			User 3			Us	
	Time	Acc.	Avg.	Time	Acc.	Avg..	Time	Acc.	Avg.	Time	A
F1-T1	12			46			25			10	
F1-T2	10	1,20		51	0,90		22	1,14		7	1
F2-T1	22			51			35			18	
F2-T2	18	1,22		46	1,11		33	1,06		17	1
F3-T1	54			110			68			49	
F3-T2	49	1,10	1,17	99	1,11	1,04	61	1,11	1,10	43	1

Table 4. The measurements of acceleration using a form-filling mechanism for Firefox 3[own].

4.3. Analysis of results

The analysis of estimates let us assume that the acceleration processes of assisted form filling will grow more or less linearly with the increase of the forms complexity. This fact has not been confirmed by the results of tests presented in Figure 16.

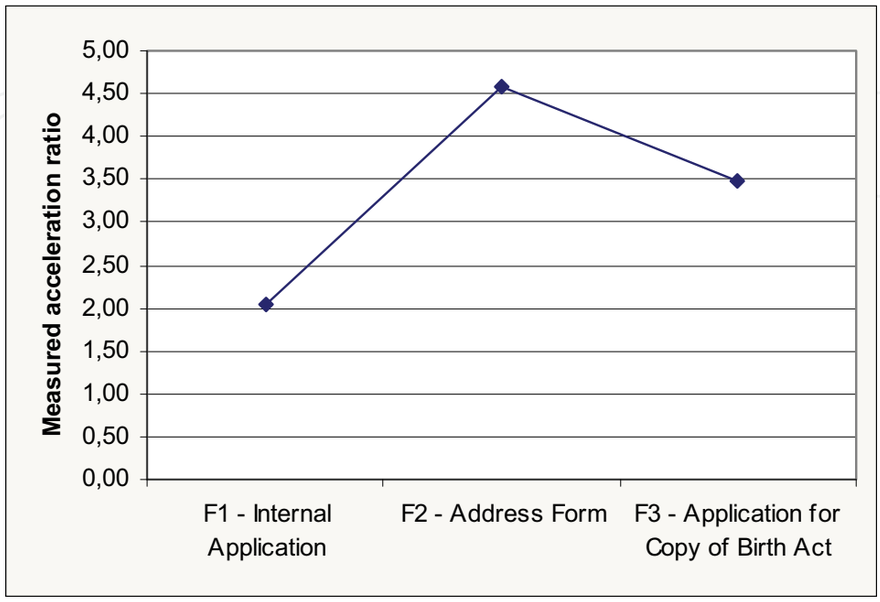

Fig. 16. The measured average acceleration of the process of assisted form filling in relation to the degree of complexity of forms [own].

The dramatic difference between the estimated and the measured acceleration comes from the time the users spent on analyzing the structure and meaning of forms being filled, which was not included in the estimates. The time needed to analyze the meaning and structure of forms is also responsible for the noticeable decline of acceleration of processes with the increase in the form complexity, which was depicted in the figure 16. It is reasonable to assume that the two trends are going to cancel each other out resulting in roughly constant acceleration. However, it is necessary to perform tests with more users and more diverse forms to determine which trend has a dominating influence on the final value of the acceleration.

Despite the need for further research we can already predict some consequences. In case of users with well-developed motor skills the dominant factor will be the time they spend on the analyzing the meaning of the form. In such cases, the benefit associated with accelerating the process of form filling will be small, and the usefulness of this solution will be considered only in terms of easy use.

Another extremely different group will consist of people with low motor skills, for whom the main concern will be to type in the field values using the keyboard. They can be both disabled people and novice computer users who have not yet dealt with rapid typing. For such people the dominant factor will be the time taken to fill in individual fields. In such cases, the mechanism of assisted form filling will provide a significant time benefit and a significant increase in the convenience of the software use.

However, the largest group of users will probably be represented by those with the average motor skills for whom the acceleration of the form filling processes in the order of three to four times is sufficiently good.
In addition, it should be noted that, despite the difference between the estimated and the measured acceleration of assisted form filling processes, the solution presented by the forms module obtains significantly better results (by order of magnitude) than the mechanism available in web browsers, which at the efficiency of 10-20% puts the solution into question.

5. Conclusions

The application of semantic technologies allowed to implement working software that assists the user in filling any electronic forms. As both research and implementation are still in progress, the usefulness of this solution was based only on approximate estimates and on several studies involving the users, which does not allow a thorough evaluation. The outcome of the work, however, allows us to believe that the chosen direction has good chances of success.

6. References

Berners-Lee T., Hendler J., Lassila O., The semantic web. Scientific American Magazine, May, 2001.
Forgy C., On the efficient implementation of production systems. Carneggie-Mellon University, 1979.
Grosof B., Horrocks I., Volz R., Decker S., *Description Logic Programs: Combining Logic Programs with Description Logic .2003,* http://www.cs.man.ac.uk/~horrocks/Publications/download/2003/p117-grosof.pdf, 2009-01.
Horn A., On sentences which are true of direct unions of algebras. Journal of Symbolic Logic, 16, 14-21, 1951.
OWL Web Ontology Language Guide. W3C, 2004, http://www.w3.org/TR/owl-guide, 2009-01.
OWL Web Ontology Language Overview. W3C, 2004, http://www.w3.org/TR/owl-features, 2009-01.
OWL Web Ontology Language Reference. W3C, 2004, http://www.w3.org/TR/owl-ref, 2009-01.
OWL-S: Semantic Markup for Web Services, W3C, 2004, http://www.w3.org/Submission/OWL-S, 2009-01.
OWL Web Ontology Language Semantics and Abstract Syntax. W3C, 2004, http://www.w3.org/TR/owl-semantics, 2009-01.
Resource Description Framework (RDF); Concepts and Abstract Syntax. W3C, 2004, http://www.w3.org/TR/rdf-concepts, 2009-01.
RDF Primer. W3C, 2004, http://www.w3.org/TR/rdf-primer, 2009-01.
RDF Vocabulary Description Language 1.0: RDF Schema. W3C, 2004, http://www.w3.org/TR/rdf-schema, 2009-10.
RDF Semantics. W3C, 2004, http://www.w3.org/TR/rdf-mt, 2009-01.
Extensible Markup Language (XML) 1.0 (Fourth Edition). W3C, 2006, http://www.w3.org/XML, 2009-01.

Engineering the Computer Science and IT

Edited by Safeeullah Soomro

ISBN 978-953-307-012-4

Hard cover, 506 pages

Publisher InTech

Published online 01, October, 2009

Published in print edition October, 2009

It has been many decades, since Computer Science has been able to achieve tremendous recognition and has been applied in various fields, mainly computer programming and software engineering. Many efforts have been taken to improve knowledge of researchers, educationists and others in the field of computer science and engineering. This book provides a further insight in this direction. It provides innovative ideas in the field of computer science and engineering with a view to face new challenges of the current and future centuries. This book comprises of 25 chapters focusing on the basic and applied research in the field of computer science and information technology. It increases knowledge in the topics such as web programming, logic programming, software debugging, real-time systems, statistical modeling, networking, program analysis, mathematical models and natural language processing.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen