UDC 004

S. Telenyk, G. Nowakowski,
E. Zharikov, Y. Vovk

INFORMATION TECHNOLOGY FOR WEB-APPLICATIONS DESIGN AND IMPLEMENTATION

Abstract: The problem of the rapid creation of effective web-applications of one class with using formal means are considered. The conceptual approach to its solution is offered on the basis of analysis of the peculiarities of web-applications construction. The approach is based on defining the standard web application architecture and selecting its components using formal methods in accordance with user requirements. A formal logical system is used, which uses the design of web-applications as a process of outputting the formula specified according to the needs of the user, which defines the schemes of execution of the modules of the system. An important feature of the approach is the ability to 3D-visualize the process of designing the system, which creates conditions for effective interaction between developers and machine development tools.

Keywords: web-applications; application construction; business processes; mathematical logic.

Introduction

The creation of information systems today is based on modern methodological concepts that inherited the most important ideas of classical methodologies such as SADT, IDEF0, while enriching them with new ideas. Classical methodologies were characterized, first of all, by a combination of the perfect implementation of the system approach with detailed elaboration of the design stages and taking into account the features of the management object based on the triad "model - algorithm - program".

The most important directions for improving classical methodologies were:

1) structuring software that opened the way for programmer’s teamwork and reuse of software code;

2) describing the created system possibilities using languages of conceptual modeling, which was accompanied by the development of automated design systems;

3) structuring the design process, which enriched the industry with the concept of the system life cycle, repository and unified models of role design systems;

4) prototyping, on the basis of which the concept of parallel development of parts of the system, design and implementation of the system by parts were developed.

By and large, modern methodologies for information systems design, for example Agile-methodology, only successfully use different combinations of these concepts, filling them with real content and complementing the elements of psychology. This conclusion can be made by analyzing the rich experience of using Scrum, Kanban and other technologies.

Such features of modern methodologies as the minimum of documentation, the various schemes of communication of developers and the variety of roles played by them is only a tribute to the time and form of the inclining of the tendency to change.

Formalization is very specific in modern methodologies [1-4]. In classical methodologies it was considered as the basis of automation, CASE-tools. In essence, formalization then seemed to be an inalienable ability of each technology that implemented the methodology, the tool for automatic production of components of the information system, primarily software, databases, based on their formal descriptions. Today, formalization is more related to describing the architecture of the system and descriptions are mainly used to set tasks for project participants, discuss intermediate results, define the next steps [1,2].

But there is a real opportunity to give formalization a more important role in modern methodologies for the creation of information systems. First, this is facilitated by the standardization of architecture and components of information systems. Secondly, within the framework of standardized architectures, many components have been developed that can be used to design new systems. Thirdly, the development of mathematical logic and artificial intelligence in recent years provided as tools for the formal description of systems designed in terms of "what needs to be implemented" and the description of components in terms of "what has already been implemented." Fourthly, the methodology of spatial visualization of designed systems appeared, which can be transferred to the creation of information systems based on the component-based approach.

This paper attempts to use the developed formal models and effective methods for building the technology of automated generation of software systems on the basis of modern methodologies.

As this problem has enormous scales, we have identified one of the most advanced components of modern information systems - the development of systems with web-presentation.

The essence of the problem

The expediency of this tasks class automation is explained by the considerable complexity of design and implementation of subtasks, to which they decompose. Subproblems are duplicated in each project, and changes are subject to data but not their main transformation.

Developers should agree on a universal architecture. Then the application will have the same overall appearance and differ from others only by the presence or absence of one or another component depending on the functional requirements for application.

Adhering to the principles of the three-tier architecture, in the typical application we will allocate levels - representation, business logic and access to data. The level of data access determines the ability to work with data and in general, it is the implementation of the template "Repository". The access to data, if necessary, from the business logic we define in the traditional way: 1) with each entity, we associate

certain generalized behavior, inherent in all elements of the level of access to data; 2) expanding behavior by adding unique methods for the current entity.

Accordingly, a specific repository class inherits its interface, obtaining basic methods common to all and its own methods from its own interface, and also inherits the class that implements the basic behavior, the fact of which contains the implementation of only their methods. The template has a specific architecture and is expanded by adding new classes and interfaces to work with each entity.

Business logic is responsible for processing data obtained from the level of access to data, through the use of behavioral patterns, such as strategy, behavioral classes, in certain cases of commands. Each component implements the end-user functionality that the user expects. According to the incorporated concept, components can be expanded by adding classes with narrowly oriented behavior.

The "Strategy" template aims to implement a set of different behaviors depending on the needs of the component that is level higher and appeals to this strategy. A strategy is described with an appropriate method that takes on input the required parameters, and several implementations of the method.

The "Command" template provides the implementation of a certain behavior on demand, allowing abstract from the logic of the command itself inside the mechanism.

Another mechanism that is used is an interface with behavior descriptions and a implementation class that contains the elements of data processing and the formation of the result required for a component that is in the architecture of the level above.

A web-presentation is a system component that is accessible from the outside and visible to all users. It is intended for receiving and analyzing queries and performing necessary actions by means of calling business logic methods, as well as displaying data on a client side. This component is mainly built on an MVC template, consisting of representations and directly methods invoked when processing client requests.

As the MVC template is built according to its purpose: the model determines which data should be displayed; representation is a graphical component capable of displaying a particular model; the controller determines how to get the model, which presentation to submit it and how to process the request.

This template is designed to differentiate between logic and representation using a model. This allows you to structure components in accordance with naturally defined assignments and expand the capabilities of one component without altering others.

By following the description, the task of creating any web-application can be considered as a task of filling in a particular template. But building even a template of correct architecture, writing monotone and long code require a lot of time. Therefore, the problem of automating the creation of a software system through the automated implementation of the system template based on user requirements. Solving this prob-

lem involves the following steps: designing a database model and generating access levels to it; realization of data processing by built-in strategies on the user's request; generation of presentation. Therefore, the problem of automating the creation of a software system appears by automated implementation of the system template based on user requirements. Solving this problem involves the following steps: designing a database model and generating access levels to it; realization of data processing by built-in strategies on the user's request; generation of presentation.

The structure of the process determines the structure of the program, its general appearance. For the model structure of the program we will use a state diagram.

For many applications in this chart, it's enough to highlight user (U), representation (P), controller (C), business logic (L), data access level (A), data source (D). The natural chain of state changes in the forward direction is caused by the following events: data request (from U to P); create a thread to handle the query and instance of the controller (from P to C); create an instance of the behavioral class and call its method (from C to L); create an instance of the repository and call its method (from L to A); forming a request and sending it to execution (from A to D). In the reverse direction for the construction of the program, it is expedient to select the following transitions: return the result of executing the data request (from D to A); return of access data model (from A to L); processing of selected data (from L to C); transferring results to the user and releasing resources (from C to P); return the result (from P to U).

A wide class of business processes is supported by web applications, authorization and processing of which may be described in a similar way. The difference lies in methods called at the level of business logic, interfaces, classes, and objects required by the user.

Another example is a business processes that require quality, speed, performance, and other characteristics of various objects. Here is a toolkit for performing such calculations.

To create popular today's transactional systems, the components of aggregation of orders data according to the chosen criterion would be acquired, control of transaction execution, system return to the previous state, etc.

A component to support the business process for processing discounts on vouchers could also be an integral part of a broad class of systems. The description of this business process is quite standardized, and the construction of the relevant component would be very similar to the one discussed above.

Consequently, for broad classes of systems of similar purpose, the only one distinguishing the implementation of tools for supporting standardized business processes is behavioral strategy at the level of business logic. It is advisable to implement the system in such cases in such a way that, depending on the conditions that were given when creating a particular component, the program code was generated according to different schemes corresponding to the specified conditions, but the system guaranteed a unique issue of this code for each design request.

Existing Approaches To Creating Web-Based Applications

Attempts to automate the creation of web-applications are made a long time ago.

Of course, it's worth mentioning already made solutions like CMS. The logic of the behavior of these constructors is to use the finished system for the entire range of tasks at the expense of the means of choosing exclusively the theme of the graphic design and adding or disconnecting those or other pages. This entails a number of shortcomings. First, the predominant use of PHP in addition, without differentiating the levels of architecture is not accompanied by the use of templates. The consequence is a huge layer of NOT readable and hard-edited code. Secondly, loading individually each component, which is often superfluous for this site, since it is built on a template designed for the entire functionality, negatively affects performance. Third, there is no system setup at different levels.

Another solution is the composite automation of the capabilities of individual modules through the use of frameworks built on one or another technology. The system is created by composing modules, and modules - by composing projects, and projects - by composing classes, etc. This approach differs by the complexity of the process of templating in the entire solution, in contrast to the template of the pattern and the module, which are quite simple tasks. And in many cases, it creates the need to manually edit the code. On the other hand, templating a solution based on a common template framework group generates new features for expanding functionality quickly through the use of formal methods.

The report considers other approaches to automating the creation of web-applications and concludes that they are partially compliant with the solution to the problem of generating the code of web-applications taking into account the possibility of managing the automation process at each level and the appropriate capability of selecting and integrating existing components based on formal user requirements and descriptions of these components.

For more detailed consideration of the architecture of web-applications, an appropriate alternative to their rapid creation is to template the process of their implementation at each level of the architecture using existing frameworks developed for this or that technology.

But there are several obvious aspects of solving this problem. The level of access to data is subject to templating based on the entities of the database as a separate component. Similarly, you can customize the creation of a REST API based on process descriptions. But the level of business logic requires a detailed description of the effective implementation of template-based capabilities, taking into account the architectural level of the frameworks and component descriptions, in order to ensure their choices, integration, and sufficient flexibility within the settings.

The automation of the rapid creation of web-applications in the case of this approach leaves programmers one of the key issues - combining the resulting fragments into a general application, requiring appropriate changes to the autogenerated product code.

Problem Statement

It is necessary to develop an approach to the creation of software systems, which provides: reducing the cost of writing template code, during the full development cycle from the process of formalizing requirements and ending with the testing process of this code; designing applications based on components of various implementation technologies; Expansion of the functionality by making minor changes to the components.

Conceptual foundations for creating web-applications based on templates

The key idea behind the development is to build a real model based on the basic notion of architecture, to model the application development process. Speaking of the concept of basic architecture we mean the model shown in Fig. 1.

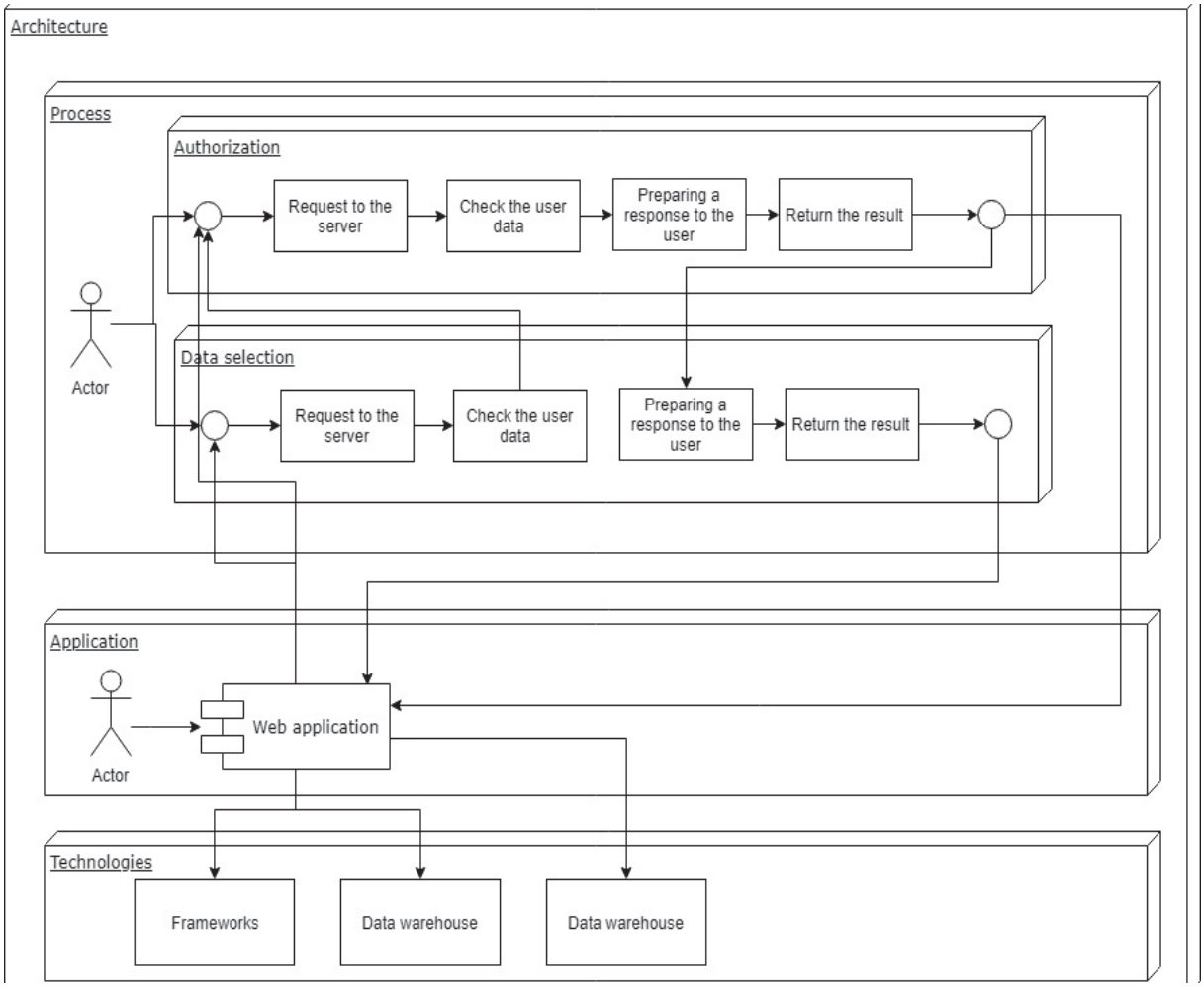

Fig. 1. Basic architecture of the system

A set of template processes can be implemented with one application at the expense of means of using the frameworks and specifications of one or another technology. The application is multilevel and consists of template modules. Each module is constructed of classes and interfaces that have their own structural units

and the corresponding appearance and methods depending on the tasks to be solved in accordance with the requirements of users.

The design of any system begins with the implementation of business processes and the structure of the data warehouse, which will meet the objectives. The user has to put the schema model for the database - to describe the entities and indicate their links.

After this stage, the system builds the basic functionality in the form of a data model of scripts for database generation at the level of data access, using scenarios for work with users and authorizations that the user can choose depending on the needs.

The next stage of the user's work is the generation of the data component directly, namely data selection, data preparation with a number of conditions by combining conditions using the proposed template or creating your own.

After this stage, you need to set the conditions and create a semantic description of the corresponding transformation and processing of data in accordance with the current business process.

The final stage is to generate the presentation with the appropriate selection and editing the most suitable template from the proposed.

The result is a ready-made application, generated on the basis of the composition of the generated modules of the system.

Mathematical model for constructing basic functionality

To select and integrate components in the complete solution, we will use the first-order causal logic, the structural elements of which are described by the authors for the integration of applications in the work [1].

Symbols:

service: (,), [,], {, }, :, <, >, ,;

constant:

1) *individual*, of primary types (int, real, char, bool) - a_1^1, a_2^1, …, a_1^2, a_2^2, ... where each constant a_i^k pertains to type (primary type) k; structural type (construct) - c_1, c_2, ...; procedural type (method) - d_1, d_2, ...; objective type (problem, entity, relation) — e_1, e_2, ...;

2) *functional i-place*, for individuals of type k - h_1^1, h_2^1, …, h_1^2, h_2^2, ...;

3) *predicate i-place*, for individuals of type k - A_1^1, A_2^1, …, A_1^2, A_2^2, ... (this class includes taxonomic, relational and other predicates, as well as traditional relations, at least equality = and order ≥);

variable: for individuals of type k - x_1^1, x_2^1, …, x_1^2, x_2^2, ..., where every variable x_i^k pertains to type k;

logical: $\neg, \wedge, \vee, \leftarrow, \exists, \forall, \Leftrightarrow$.

 ISSN 1560-8956

Individual terms of type *k*:

1) each individual constant a_i^k of type *k* is an individual term of type *k*;

2) each free variable x_i^k for individuals of type *k* is an individual term of type *k*;

3) if h_i^j is a certain functional constant for individuals of type *k* and $\tau_1, \ldots, \tau_j$ are terms for individuals of type *k*, then h_i^j $(\tau_1, \ldots, \tau_j)$ is an individual term of type *k*;

4) there are no other individual terms of type *k*.

Formulas for individuals:

1) if A_i^j is a predicate constant for individuals and $\tau_1, \ldots, \tau_j$ are terms for them, then A_i^j $(\tau_1, \ldots, \tau_j)$ is the atomic formula for individuals;

2) the atomic formula for individuals is the formula for them;

3) there are no other formulas for individuals.

Hereinafter we consider only the systems of Horn clauses (with a single $\rightarrow$ symbol, atomic formulas to its left and right, and an implicit quantifier $\forall$).

Specifiers are constructions of type τ_1, where τ_1 is a term for an individual object. The construct specifiers are:

1) if $e_1^e \ldots e_i^e$ are individual terms of type entity, and $e_1^r \ldots e_i^r$ are individual terms of type relation, and $A_1^j(a_1^1 \ldots a_j^{\ 1}) \ldots\ A_i^j(a_1^k \ldots a_j^{\ k})$ are atomic formulas for the individuals of primary types, and τ is an individual term of type construct, then τ: $(e_1^e \ldots e_i^e, e_1^r \ldots e_i^r, A_1^j(a_1^1 \ldots a_j^{\ 1}) \ldots\ A_i^j(a_1^k \ldots a_j^{\ k}))$ is a construct specifier.

2) there are no other construct specifiers.

Preconditions:

1) if c_1 is a specifier of construct, and Π is a sequence of atomic formulas, then $<\tau_1:(c_1,\Pi)>$ is the elementary precondition;

2) elementary precondition - precondition;

3) if $<\tau_1>$ is a precondition, and τ_2 is an elementary precondition, then $<\tau_1,\tau_2>$ is the precondition;

4) there are no other preconditions.

Post-conditions:

1) if τ_1 is a construct specifier, and Π is a sequence of atomic formulas, then $<\tau_1:(c_1,\Pi)>$ is the elementary post-condition;

2) elementary post-condition - post-condition;

3) if $<\tau_1>$ is a post-condition, and a τ_2 is an elementary post-condition, then $<\tau_1,\tau_2>$ is the post-condition;

4) there are no other post-conditions.

Specifiers of methods:

1) if τ is an individual term of type method, and $<\tau_1>$ is a precondition, and $<\tau_2>$ is a post-condition, then τ: $(<\tau_1>, <\tau_2>)$ is a method specifier;

2) the arity by constructs of method precondition must not be lower than the arity by constructs of method post-condition;

3) there are no other method specifiers.

Specifiers of problems:

1) if $<\tau_1>$ is a precondition, and $<\tau_2>$ is a post-condition, and τ is a term for an individual object of type Problem, then τ: $<<\tau_1>,<\tau_2>>$ is the problem specifier;

2) there are no other problem specifiers.

Clause is an expression of type $\Pi \rightarrow \Lambda$, where Π is a sequence of atomic formulas; Λ is a single atomic formula.

The system’s knowledge base consists of the three main parts. In the first part, ontological axioms describe the methods and other components of the framework available for use. The second part provides an ontology of the system, described using OWL-based languages based on RDF. The third part summarizes the output rules needed to get the desired result for the user. There are rules for the output of two types. The first type output rules are used to integrate many methods and other components into a single application. They take into account the peculiarities of the problem, the preconditions, the stages, the descriptions of methods and other components of the system. Realized with the desired output proof in a certain way defines the architecture of the application that is being created within a defined class of architectures.

Inference rules of first type:

1. If d_1: $(<\tau_1>,<\tau_2>)$ and d_2: $(<\tau_3>,<\tau_1>)$ then d_1: $(<d_2>,<\tau_2>)$

2. If d_1: $(<\tau_1>,<\tau_2>)$ and d_2: $(<\tau_3 \wedge \tau_4>,<\tau_1>)$ then d_1: $(<d_2>,<\tau_2>)$

3. If d_1: $(<\tau_1 \wedge \tau_2>,<\tau_3>)$ and d_2: $(<\tau_4>,<\tau_1>)$ and d_3: $(<\tau_5>,<\tau_2>)$ then d_1: $(<d_2 \wedge d_3>,<\tau_3>)$

4. If d_1: $(<\tau_1>,<\tau_2>)$ and d_2: $(<\tau_3 \vee \tau_4>,<\tau_1>)$ then d_1: $(<d_2>,<\tau_2>)$

5. If d_1: $(<\tau_2 \wedge \tau_2>,<\tau_1>)$ then d_1: $(<\tau_2>,<\tau_1>)$

6. If d_1: $(<\tau_1,\tau_2>,<\tau_3>)$ and d_2: $(<\tau_4>,<\tau_1>)$ and d_3: $(<\tau_5>,<\tau_2>)$ then d_1: $(<d_2,d_3>,<\tau_2>)$

7. If d_1: $(<\tau_1>,<\tau_2,\tau_3>)$ and d_2: $(<\tau_2>,<\tau_4>)$ and d_3: $(<\tau_3>,<\tau_5>)$ then d_2: $(<d_1>,<\tau 4>)$ and d_3: $(<d_1>,<\tau_5>)$

The second type inference rules are used to speed up the inference process. They take into account the semantics of the user’s problem and the semantics of methods and other components of the system, expressed in terms of the system’s ontology. In essence, these rules are used to reduce the search for rules and axioms of the first part of the knowledge base of the system. They describe semantically acceptable at the level of input and output the variants of combining methods and other components of the system in more functionally complete combinations. These combinations can be used to deduce in the space of rules of the first type. The meaningful use of the second type inference rules can be illustrated by the process of 3D visualization of the combination of methods and other components of the system in spatial structures based on common entities on their inputs/outputs.

The semantically controlled inference mechanism

For the inference we will use the approach proposed by the authors in the paper [1]. It's about the procedures for inference and restoring the inference tree. But we will add to this approach the preliminary procedure for searching the spatial

structure of the associated by input/output methods and other components of the system, which at inputs has entities defined by the user as input information, and at its outputs we have the entities defined by the user as the source information. Define the necessary concepts.

Naturally, we focus on the means of knowledge representation and to reduce the search, we use the knowledge base, especially information on effective inference schemes for frequently performed queries and its ability to structure, factorize and abstract. This is a combined inference strategy, at the lower levels of which the birth of a lossy resolvents is blocked. Formally it is a question of combining the approach of R. Kowalski [3] and the analogue method of D. Playsted [4]. In this case, the proof in the abstract space, the result of which is then used to control the inference in the initial search space, is based exclusively on the axioms and inference rules of the knowledge base.

We cut off the hopeless branches of the inference in the initial space in two stages. At the first stage, we formulate the constructions of the associated by input/output methods and other components of the system. At the second stage, we use the inference in the abstract space. For the reflection of the transition to the abstract space, we will use taxonomic connections. Then the inference in the abstract space will be reduced to the inference in the system of types (classes of entities) with a gradual deepening of the system of subtypes up to individuals. As before, to reduce the search in both the abstract (for classes and types) and in the initial (for individuals) spaces, we will use modifications to the Robinson resolution strategy. The properties needed for the mutual adaptation of the method of analogy and modification of the resolution strategy the used reflection do naturally acquire in a responsibly structured knowledge base.

For the analogy method, we use the concept of a multiclause (m-clause) as a multiset of atomic formulas (atomic formula L will be recorded in m-clause as many times as it is repeated). The operations $\cup$ (unification), $\cap$ (intersection), — (difference), $\cdot$ (concatenation) and relation $\subseteq$ (occurrence) for multisets are naturally performed (note that the operations are performed for the left and right parts of the clause separately).

Suppose $A_1 \in C_1$, $A_2 \in C_2$ and α_1, α_2 are substitutions, which allow us to obtain the most common unifier for atomic formulas A_1 and A_2. Then the clause obtained from unification of clauses $C_1\alpha_1$ and $C_2\alpha_2$ by removing L to the left and right of symbol $\rightarrow$ is called m-resolvent of m-clauses C_1 and C_2. If the m-clauses C_1 and C_2 are ordered and m-resolvent is obtained by eliminating the non-underlined atomic formula in both parts, and on the left after it there is no other atomic formula, then we get the ordered linear m-resolvent. If the last atomic formula to the left of the symbol $\rightarrow$ of the ordered m-clause is unified with the underlined atomic formula to the right of the symbol $\rightarrow$ of the same m-clause the ordered linear m-resolvent is obtained by the m-clause reduction.

Definitions of *m*-clauses and *m*-resolvents are used to define the ordered linear *m*-resolutional proof. Let's start with the definition of the *m*-resolutional proof T_m as a pair of $<V,T_h>$, where V is the set of proof vertices, T_h is the set of vertices triples. In the future, the first and second components of T_m will be allocated using the *s*-

$N(T_m)$ and $s\text{-}M(T_m)$ selector functions respectively. Each vertex $n \in s\text{-}N(T_m)$ of the proof T_m is characterized by the mark $s\text{-}L(n)$ and the depth $s\text{-}D(n)$. If $<n_1, n_2, n_3> \in s\text{-}M(T_m)$, then $s\text{-}L(n_3)$ is the *m*-resolvent $s\text{-}L(n_1)$ and $s\text{-}L(n_2)$, and each triple of this type is called *m*-resolution. In the proof T_m, the vertex $n \in s\text{-}N(T_m)$ is called the initial if it is not the third component of any of the triples of $s\text{-}M(T_m)$ (its mark is the initial *m*-clause), or terminal if it is not neither the first nor the second component of any of the triples of $s\text{-}M(T_m)$ (its mark is the terminal *m*-clause).

Now, let's call the *m*-resolutional proof T_m the proof from *S*, if the marks of the initial vertices Tm belong to the set of m-clause *S*. From *S* we deduce *C* if T_m is a proof from *S*, and *C* is a mark of one of the vertices of T_m.

Finally, the ordered linear *m*-resolutional proof from *S* is called *m*-resolutional proof Tm from *S*, all *m*-clauses of which are ordered and for an arbitrary triple $<n_1, n_2, n_3>$ from $s\text{-}M(T_m)$ $s\text{-}L(n_3)$ is an ordered linear *m*-resolvent $s\text{-}L(n_1)$ and $s\text{-}L(n_2)$.

To manage an ordered linear inference, we will use the typification abstraction proposed in [4]. Suppose *f* is such mapping from the set of *m*-clauses into the set of *m*-clauses that: 1) if *m*-clause C_3 is the *m*-resolvent of *m*-clauses C_1 and C_2, while $D_3 \in f(C_3)$, then exist such $D_1 \in f(C_1)$ and $D_2 \in f(C_2)$ that the result of the substitution of some *m*-resolvent D_1 and D_2 belongs to D_3; 2) $f(\varnothing) = \{\varnothing\}$; 3) if the result of some substitution of *m*-clause C_1 belongs to *m*-clause C_2, then for any abstraction D_2 for C_2 exists such abstraction D_1 for C_1 that the result of D_1 substitution belongs to D_2. Such mapping is called *f m*-abstraction mapping, while any *D* from *f*(*C*) is called *m*-abstraction. The typification mapping is understood as a certain mapping ϕ from a set of literals into a set of literals, which reflects each atomic formula into the formula whose terms have the type closest to the basic types in the hierarchy.

Construction of the proof begins with the formulation of the problem by the user. In essence, the possibility of executing the request is checked, taking into account all available resources. Having access to the descriptions of all modules and other components and axioms that determine the possibilities of their application, the inference mechanism combines the modules and other components into a structure (proof), which ensures the receipt of the result from the input data.

Let the ordered linear *m*-resolutional proof be obtained by definition as a pair of sets: vertices $V = \{k_1, k_2, k_3 .. k_n\}$ and vertices triple $Th = \{<k_1, k_2, k_3>, <k_3, k_4, k_5> ... <k_{n-2}, k_{n-1}, k_n>\}$, where k_i, $i = 1,\ldots,n$ are the vertices of proof, k_n is the terminal vertex. The proof is formed, taking on the initial vertices of the problem postcondition and, as a lateral vertex, the appropriate axiom from the knowledge base. The axiom is appropriate if the sequence of atomic formulas corresponding to this vertex of the proof, or any part of it, is an axiom postcondition. The third vertex of this triple is obtained by applying the axiom to the formula postcondition, taking into account the inference rules. The third vertex of the first triple becomes the first vertex of the second triple, and the process repeats until the terminal vertex is reached - the problem precondition. If the atomic formula corresponding to the third vertex of the

triple can be simplified by applying one of the inference rules, it is simplified in the next triangle, which will have only two vertices - a vertex with a formula, which must be simplified, and a vertex with a simplified formula.

The search for the corresponding axioms is done by comparing the construct of the postcondition, which is being processed at the moment, with the axioms constructs from the ontology. Thus, from the ontology a set of axioms is chosen that describe the methods that process the type and format of the objects we need.

If in the knowledge base for the next vertex the corresponding axioms are several, then each of these axioms is used to further build its own "parallel" version of the proof. Thus, during the operation of the inference mechanism, a number of proofs of varying length and complexity can be formed. Upon completion of the inference mechanism work, the proof from the set of proofs is selected with the smallest number of triples of vertices and the smallest number of applied axioms.

To shorten the search, we can pre-select the structures of related methods and other components, the entities of which inputs and outputs are common. And only in the second stage, when the structures are already selected, the preconditions and postconditions are checked and the architecture of the solution is determined.

A set of formulas and a clause that corresponds to the vertices of the proof triples and reflects the application of the axiom of the first part and the inference rules of the first type is the output of the inference mechanism. To restore the structure, we use the proof tree recovery algorithm proposed by the authors in the paper [1].

Then, using the constructor, we are generating by the finished elements the user representation, to which, in the form of a data source, the output points of the business logic of the application are tied.

Implementation of the technology

The technology is realized using modern tools. The user-friendly web-interface allows the end-user to build a database model based on which the base framework of the application is generated (CRUD data operations with the subsequent generation of the interface as a REST API). The next step is to display this general view of the application with the ability to generate by means of a semantic assignment of relationships between the relevant models of one or another application logic behavior and implementation of the business processes of the required application.

Ready templates are integrated into the application with pointing inputs and outputs. Additional functionality is the ability to create and store by user own templates as well as the ability to put them in the public domain. Formally, the template integration can take place at different levels of architecture, ranging from method integration to integration of entire component that consists of classes.

The applications are a composition of modules, which, in turn, are a composition of classes, which, in turn, are a composition of methods, etc. Defining by the user of input and output entities and their characteristics allows you to find the necessary methods and their interconnections and get the template model of the lowest architectural level, presented in the form of classes.

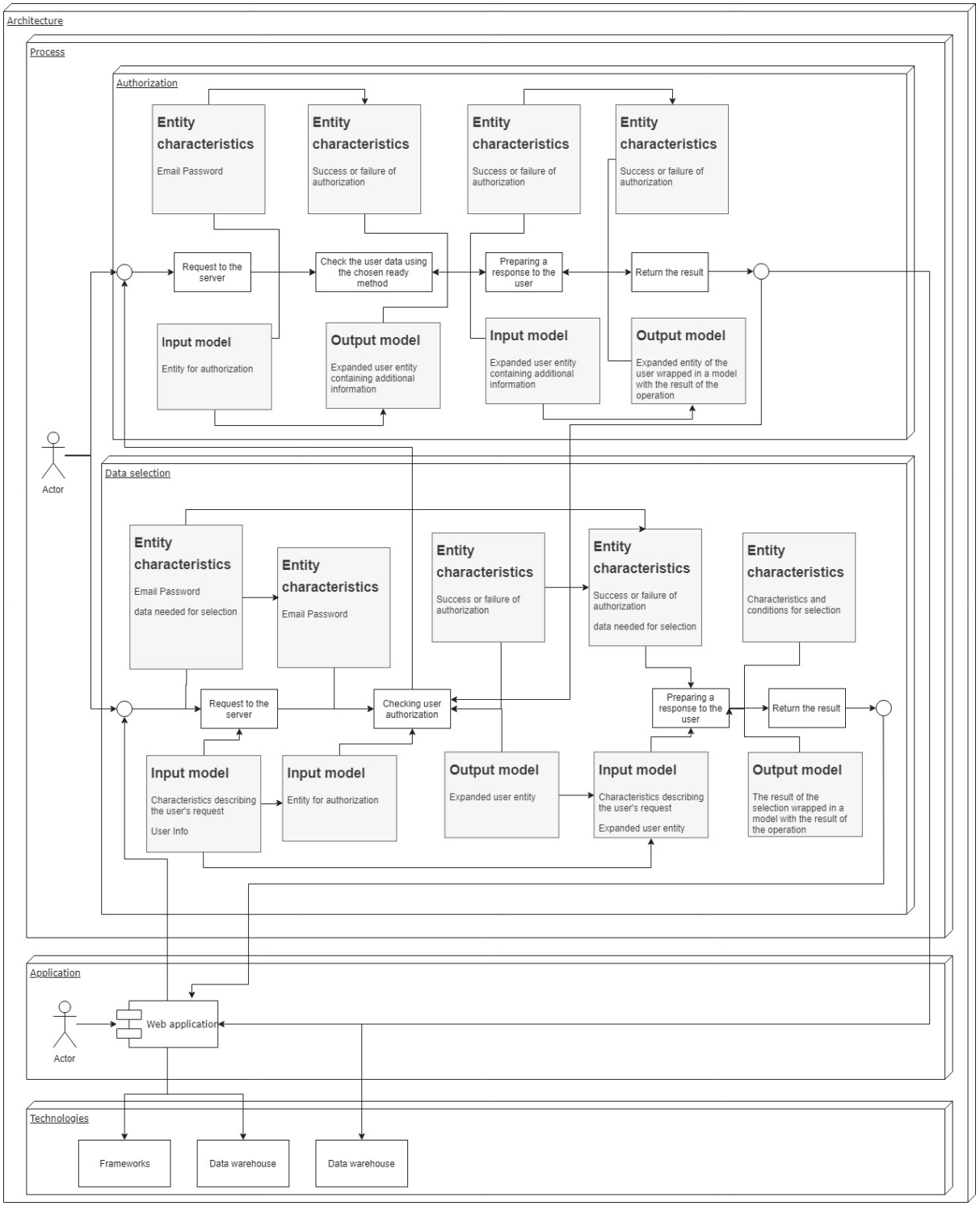

Fig. 2. Composition of business processes elements

At the highest level, the composition model is maintained, but the corresponding classes are already merged through the calls of other methods and the generation of behavior classes, which is the source of the module. Accordingly, the combination of modules follows from the composition of their outgoing points and the established in Fig. 2 connections.

In Fig. 2, the architecture of the system, the semantic model and patterns of the two connected business processes associated with the authorization of users are shown.

Conclusion

In this paper, the problem of the rapid creation of effective web-applications of one class is considered. The performed analysis of the problem and existing decisions confirmed the relevance of the problem and allowed formulate it, identify the key aspects of the study.

The complex approach to the solution of the problem, based on autogenous template solutions for information systems with the possibility of integration into the process of components at different architectural levels, is proposed. The main advantages of the proposed solution are the formal basis for creating applications, which is based on templating the design and implementation processes and the ability to fine-tune the process. In addition, the adopted model for constructing template solutions simplifies the work of developers and helps automate the basic process of developing web-applications for a wide range of business information systems with web-representation.

Analysis of the problem and the existing solutions enabled the choice of appropriate formal means. The mathematical apparatus for the proposed solution includes means of semantic description of business processes and a formal logical system. The first provides a description of business processes and enables them to decompose subprocesses from the highest level of architecture to non-decomposable atomic components, indicating the input and output entity, and the characteristics at which the choice of the corresponding functional unit is made - a template for the processing of the subprocess, or creation own template by choosing the proposed methods. A formal logic system facilitates the selection and integration of components in a complete solution.

Further researches are connected with 3D-visualization of the processes of creating web-applications on the basis of the proposed theoretical model for building a comprehensive solution based on the template of the process of creating an information system with a web-representation.

References

1. S. Telenyk, G. Nowakowski, K. Yefremov and V. Khmeliuk Logics based application integration for interdisciplinary scientific investigations / 2017 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), Bucharest, 2017, pp. 1026-1031.

2. A. Y. Levy Logic-based techniques in data integration / Logic-based artificial intelligence. Springer, Boston, MA, 2000, pp. 575-595.

3. R. Kowalski Computational logic and human thinking: how to be artificially intelligent / Cambridge University Press, 2011.

4. Plaisted, David A. History and prospects for first-order automated deduction / International Conference on Automated Deduction. Springer, Cham, 2015.