LATVIJAS UNIVERSITĀTE

Guntis Arnicāns

Informācijas apstrādes rīku izveide neviendabīgai un dinamiskai videi

SAISTĪTĀS PUBLIKĀCIJAS

(Promocijas darba pielikums)

Publikācijas recenzējamos starptautiskos zinātniskos izdevumos

1. V. Arnicane, G. Arnicans, and J. Bicevskis. **Multilanguage interpreter**. In H.-M. Haav and B. Thalheim, editors, *Proceedings of the Second International Baltic Workshop on Databases and Information Systems (DB&IS '96)*, Volume 2: Technology Track, pages 173-174. Tampere University of Technology Press, 1996.
2. G. Arnicans. **Application generation for the simple database browser based on the ER diagram**. In Jānis Bārzdiņš, editor, Databases and Information Systems, *Proceedings of the Third International Baltic Workshop*, Volume 1, pages 198-209. Rīga, 1998.
3. G. Arnicans, J. Bicevskis and G. Karnitis. **The Concept of Setting Up a Communication Server**. In abstracts of papers of *3rd International Conference "Information Technologies and Telecommunications in the Baltic States"*, pages 48-57. Riga, 1999.
4. G. Arnicans, J. Bicevskis and G. Karnitis. **The Unified Megasystem of Latvian Registers: Development of a Communication Server – the First Results and Conclusion**. In abstracts of papers of *4rd International Conference "Information Technologies and Telecommunications in the Baltic States"*, pages 163-168. Riga, 2000.
5. G. Arnicans and G. Karnitis. **Heterogeneous Database Browsing in WWW Based on Meta Model of Data Sources**. In Albertas Caplinskas, editor, Databases & Information Systems, *Proceedings of the 4th IEEE International Baltic Workshop*, Volume 1, pages 174-187. Vilnius "Technika", 2000.
6. G. Arnicans and G. Karnitis. **Heterogeneous Database Browsing in WWW Based on Meta Model of Data Sources**. In Janis Barzdins and Albertas Caplinskas, editors, Databases and Information Systems, *Fourth International Baltic Workshop, BalticDB&IS 2000 Vilnius, Lithuania, May 1-5, 2000, Selected Papers*, pages 167-178. Kluwer Academic Publishers, 2000.
7. G. Arnicans, J. Bicevskis, E. Karnitis, and G. Karnitis. **Smart Integrated Mega-system as a Basis for e-Governance**. *Proceedings D of the 5th International Multi-Conference Information Society IS'2002*, pages 197-201. Ljubljana, Slovenia, 2002.
8. G. Arnicans and G. Karnitis. **Semantics for Managing Systems in Heterogeneous and Distributed Environment**. In Hele-Mai Haav and Ahto Kalja, editors, Databases and Information Systems, *Proceedings of the Fifth International Baltic Conference BalticDB&IS 2002*, Volume 1, pages 51-62. Tallinn, 2002.
9. G. Arnicans and G. Karnitis. **Semantics for Managing Systems in Heterogeneous and Distributed Environment**. In Hele-Mai Haav and Ahto Kalja, editors, Databases and Information Systems II, *Fifth International Baltic Conference, BalticDB&IS'2002 Tallinn, Estonia, June 3-6, 2002, Selected Papers*, pages 149-160. Kluwer Academic Publishers, 2002.

Citas publikācijas, referāti un raksti

1. V. Arnicane, G. Arnicans, and J. Bicevskis. **Multilanguage interpreter**. Submitted paper to *the Second International Baltic Workshop on Databases and Information Systems (DB&IS '96), Tallinn*, pages 14, 1996.
2. G. Arnicans, J. Bicevskis and G. Karnitis. **Development of a Communication Server: First Results and Conclusions**. *Baltic IT Review*, 17(2):29-32, 2000.
3. G. Arnicans, J. Bicevskis, G. Karnitis, and E. Karnitis. **The Mega-system: integration of National information systems. Conceptual and Methodological Baselines**. In Latvian Academic Library Grey Literature database, http://159.148.58.74/greydoc/megasystem_baselines/mega_base.doc, pages 17, 2001.
4. G. Arnicāns. **Domēnspecifisko valodu izmantošanas iespējas**. Referāts, *Latvijas Universitātes Zinātniskā Konference 2002*, slaidi 41, 2002.
5. G. Arnicāns. **Information Processing Tools and Environments**. Referāts, *Latvijas Universitātes Zinātniskā Konference 2003*, lapas 9, 2003.
6. G. Arnicans. **Description of Semantics and Code Generation Possibilities for a Multi-language Interpreter**. Akceptēts iespiešanai *Latvijas Universitātes zinātniskie raksti. Sējums 669. Datorzinātne un informācijas tehnoloģijas.*, lapas 22, 2003.

Hele-Mai Haav, Bernhard Thalheim (Eds.)

Databases and Information Systems

Proceedings of the
Second International Baltic Workshop
Tallinn, June 12-14, 1996

Volume 2: Technology Track

Multilanguage interpreter

Vineta Arnicane, Guntis Arnicans, Janis Bicevskis

University of Latvia
Faculty of Physics and Mathematics
Rainis Blvd. 19, Riga LV-1459, Latvia
e-mail: varnican@lanet.lv, garnican@lanet.lv
and
Riga Institute of Information Technology
Skanstes 13, LV-1013 Riga, Latvia
e-mail: bicevskis@swh.lv

The idea of the multilanguage/multisemantics interpreter (MLI/MSI) has risen from a plenty of problems that require to analyze the given program text and do something with it. Some of such problems are program translation to another programming language or compilation, dynamic program execution or interpretetion, program beautifying and clarifying, determining program complexity, creation of cross-reference tables, static program testing, symbolic testing, automatic testing, program instrumentation with additional text, dynamic testing supporting.

The main characteristics that describe programming language are syntax, semantics and pragmatic. The multilanguage interpreter is a program that receives the source language syntax, the source language semantics and the program written in source language and performs the operations implied by this program and given semantics.

We use attribute grammars as basic concept in MLI/MSI. As metalanguage for expressing grammars we use BNF. The syntactic structure of a given source program as generated by the grammar, can be depicted as a parse tree. Conceptually we parse the input token stream, build the parse tree, and then traverse the tree as needed to evaluate the semantics rules at the parse tree nodes.

Semantics are described by evaluation rules. Rules are not associated with some grammar production rule but are connected to terminals and nonterminals of BNF. We have divided semantics rule for nonterminal symbol into two functions: Pre_function that is executed if we visit node from parent or sibling node and Post_function if we visit node from child. For terminal symbols we write only one function.

The semantics for MLI/MSI is the set of programs written in some metalanguage and some real programming language. These programs are written in such a way that they can be executed on computer. The special support tool is designed for multilanguage interpreter to write semantics more quickly and compactly - Memory Object Management System (MOMS).

MOMS operates with some basic memory objects such as name of object, reference (handle to memory object), value (handle to byte stream that contains a value of an object), constructor (handle to object type description). Constructor may be primitive constructor or combination of primitive constructors. By using of constructors we can describe the structure and features of any memory object, for instance, type of variable, function arguments, procedure, etc. MOMS also operates with more complex memory objects such as dictionaries, tables, memory blocks, stacks, collections.

MOMS functions we use for describing the semantics of the subject language. These functions provide definition of the features of program running environment, description of the source language basic data types, description of the source language basic operations (+, -, *, /, <, >, min(), max(), substr(), etc.), definition of scope for memory objects, sets of functions that allow to create easier user defined data type, various operations with variables, constants, functions for realizing the source language procedures and functions, and other useful functions.

The advantage of MLI/MSI is easiness and simplicity of various semantics describing, modifying of them and linking of them to syntax during execution.

We have created conventional semantics for the simple poor language PAM as well as special semantics that perform symbolic execution along the chosen program path. MOMS can be used by other tools too. It is already used as central part of MOSAIC (CASE tool for business modeling).

Jānis Bārzdiņš (Ed.)

Databases and Information Systems

Proceedings of the
Third International Baltic Workshop
Rīga, April 15-17, 1998

Volume 1

Rīga 1998

Application generation for the simple database browser based on the ER diagram

Guntis Arnicans

University of Latvia
Faculty of Physics and Mathematics
Rainis Blvd. 19, Riga LV-1459, Latvia
garnican@lanet.lv

Abstract

This paper describes a development technique for the rough browser of a database. The offered data browser or data management system can be generated automatically from the a physical data model represented by an ER diagram. The ER diagram used to generate a target application source text is described by common simple concepts and by some additional attributes with default changeable values. All the ER diagram elements are mapped to standard screen object groups and are the main components in the target system screens. Various screen templates for generated applications are defined depending on the entities, the relationships between them and an acceptable user interface. The generated application can be used for database browsing, data manipulating, system prototyping, fast developing of simple information systems and data analyzing.

1. Introduction

There are many strategies for information system development and project management in nowadays. The development of very advanced CASE tools lets us use the Rapid Application Development (RAD) methodology. This approach includes several steps - business modeling, data modeling, process modeling, application generation, testing and turnover [1]. In this paper the simple technique is described that allows to develop specific information system - database browser and data manager. The main attention is turned to the generation of application.

Many powerful tools already exist to assist in system development with RAD technology, for instance, Oracle Designer/2000 [2], [3]. But practice shows that these tools sometimes are not useful. The reasons are that they are expensive and require high educated and trained specialists to work with them. And we have to work hard for some time. The quality of the information system mostly depends on the data model. Especially this data model (object model) is critical when we use Object Modeling Technique (OMT) [4]. Let us assume that we already have designed the physical data model for our database. Like a conceptual data model the physical data model can be described by Entity-Relationship Diagram (ER diagram). This is popular instrument to describe data model or database and these diagrams are known for most programmers.

Our goal is offer to the user a technique that allows to create a database browser from the physical data model described by the ER diagram. What does the developer have to do? He has to create a simple ER diagram for the existing or the planned database. We do not care in whether he makes a serious analysis and design, whether he creates the ER diagram "on the fly". He obtains quickly generated database browser, a simple information analysis and filtering tool, a data entering and editing tool, a prototype for the most serious business application, simple database testing tool. Thus, while the real system is developed, a robust information system is obtained.

2. ER diagram - the source for generation

2.1 The elements of the ER model

The ER diagram is a source for application generation. We consider only the ER diagrams that represent physical data models. The main objects we manipulate are the ER diagram descriptor (describing common features of database), the entities (representing tables in our database), the relationships (representing the relations between the entities), the fields (representing the data fields in the record of the physical table).

Developers use various variants of the ER diagrams. Let us take a diagram that is not too simple and not very complex. The ER diagram can be described by the object model shown in the Figure 1. We choose the following elements in the ER diagram:

- **diagram descriptor** - *DiagramName*;
- **entity** - *EntityName*, [*EntityType*], *PrimaryKey*, *UniqueKey*, *Index*;
- **field** - *FieldName*, [*FieldType*], *DataType*, [*Visibility*], [*ShortView*], [*LongView*];
- **relationship** - *EndEntity_1*, *EndEntity_2*, *Cardinality_1*, *Cardinality_2*, *Role_1*, *Role_2*, *ForeignKey_1*, *ForeignKey_2*.

The attributes in brackets are introduced for generation better applications. For simplicity we assume that primary key and foreign key are represented only by one field. In our simplified model we assume that an index is created from a field without using any function. It is not so hard to expand the model to use a combination of fields as the primary key (as the foreign key respectively) and a function of fields as the index.

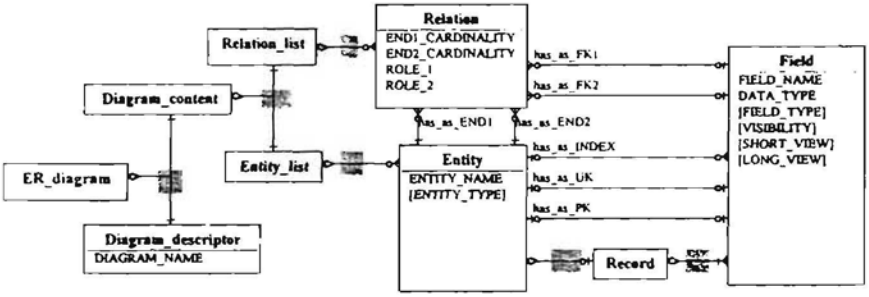


Figure 1 The conceptual object model for the ER diagram

2.2 Additional elements of the ER diagram

Let us introduce several new attributes for the ER diagram. These attributes provide the additional information for the application generation program to generate a more convenient application.

Entity type is a special attribute of an entity that allows us to generate application screens with a specific information layout and data manipulation means. This attribute is stated automatically and depends on the relationships between the entities. The user can correct it while automatic type fixing.

Field type is an automatically calculated attribute of the field. If the field is defined as the primary key of *Entity* via relationship *has as PK* (Figure 1) then the field has type ***PK***. Similarly we define type ***UK*** (via relationship *has as UK*) and type ***FK*** (via relationship *has as FK1* or *has as FK2*). Otherwise the field type is ***Attribute***.

Visibility is a feature of a field. It states whether the information associated with the field is or is not displayed to the user. The default value of *visibility* is TRUE for fields with types *UK*, *FK* and *Attribute* but FALSE for type *PK*.

ShortView and **LongView** are field attributes that define how the entity record can be best displayed on the screen.

2.3 Entity types

Entity type is an important concept in our application generation ideology. Let us define the following *entity types*.

- ***Domain*** - list of standard data elements, allowed values for an attribute or an object property.
- ***SimpleEntity*** - simple object that is determined by a set of attributes or standard data elements.
- ***ComplexEntity*** - complex object is similar to *SimpleEntity* but it includes other simple or complex objects.
- ***Link*** - logical relation between at least tv o simple or complex objects.

2.4 Algorithm for entity type determination

1. Scan through all the entities and fix those that have no field with type *FK* (foreign key) and all the incoming ends of whose relationships are either of cardinality 1 or 0..1. We have to assign the type *Domain* or *SimpleEntity* to the fixed entities. The type *Domain* is assigned by default.
2. Scan through all entities without a fixed type and fix any one that has fields with type *FK* referenced only to entities with type *Domain* and all the incoming ends of whose remaining relationships are either of cardinality 1 or 0..1. The type *SimpleEntity* is assigned to the fixed entities.
3. Scan through all entities without a fixed type and fix each one which at that moment is referenced by a foreign key from any entity with type *SimpleEntity*, *ComplexEntity* or undefined type. The entity can also reference to itself. The type *ComplexEntity* is assigned to the fixed entities.
4. Scan through all entities without a fixed type and fix any one that has at least two fields with type *FK* that reference to entities with type *SimpleEntity* or *ComplexEntity*. We have to assign the type *Link* or *ComplexEntity* to the fixed entities. The type *Link* is assigned by default.
5. The type *ComplexEntity* is assigned to the any remaining entity without fixed type.

The user decides on the entity type (1. and 4. step) according to the semantics of the entity and on how he wants to see the information on the screen.

2.5 Textual visualization of entity record

We need to define several textual visualizations of an entity record in our system. A textual visualization of an entity record is mapping the field values to text. These texts (or entity views) are used to display an entity on the screen. Let us define three functions: *shortView()*, *longView()*, ***allFields()***. The *shortView()* displays some of the record fields in an ordered sequence. The order is defined by assigned an order number to attribute *ShortView*. If the field is not included in short view the 0 is assigned to *ShortView*. The similar approach is used for *longView()*. The *allFields()* displays all fields.

2.6 ER diagram for example

The ER diagram for example is given in Figure 2. The attributes of each field are given in the following sequence - ***field name**, **data type**, **field type**, **visibility*** (T for TRUE, F for FALSE), *ShortView*, *LongView*.

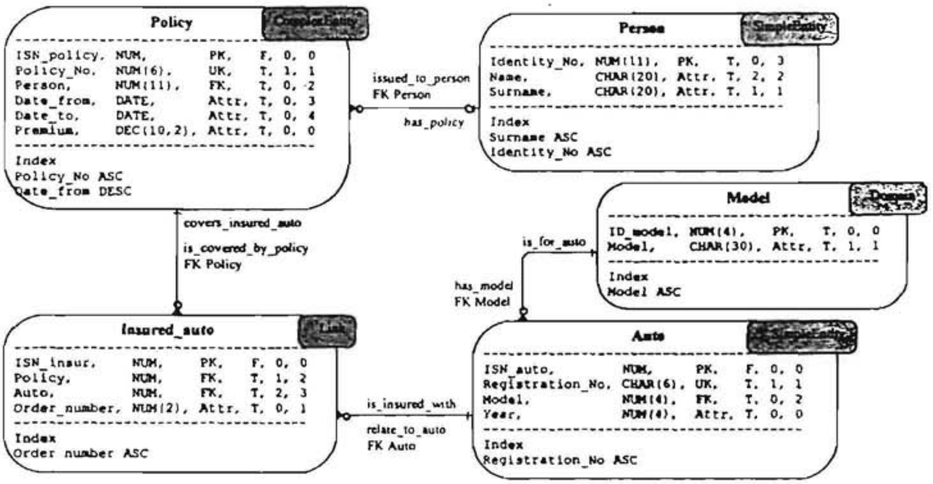

Policy
ComplexEntity
ISN_policy, NUM, PK, F, 0, 0
Policy_No, NUM(6), UK, T, 1, 1
Person, NUM(11), FK, T, 0, 2
Date_from, DATE, Attr, T, 0, 3
Date_to, DATE, Attr, T, 0, 4
Premium, DEC(10,2), Attr, T, 0, 0
Index
Policy_No ASC
Date_from DESC
issued_to_person
FK Person
has_policy
Person
SimpleEntity
Identity_No, NUM(11), PK, T, 0, 3
Name, CHAR(20), Attr, T, 2, 2
Surname, CHAR(20), Attr, T, 1, 1
Index
Surname ASC
Identity_No ASC
covers_insured_auto
is_covered_by_policy
FK Policy
Model
Domain
ID_model, NUM(4), PK, T, 0, 0
Model, CHAR(30), Attr, T, 1, 1
Index
Model ASC
is_for_auto
has_model
FK Model
Insured_auto
Link
ISN_insur, NUM, PK, F, 0, 0
Policy, NUM, FK, T, 1, 2
Auto, NUM, FK, T, 2, 3
Order_number, NUM(2), Attr, T, 0, 1
Index
Order number ASC
is_insured_with
relate_to_auto
FK Auto
Auto
ISN_auto, NUM, PK, F, 0, 0
Registration_No, CHAR(6), UK, T, 1, 1
Model, NUM(4), FK, T, 0, 2
Year, NUM(4), Attr, T, 0, 0
Index
Registration_No ASC

Figure 2 ER diagram for example

3. Application generation

The general idea of generation is to create a system with a predefined user interface and functionality. The features of the system depend on the generator. We can generate the whole application for the ER diagram or only some components for this application.

3.1 Screens and menu system

The quality and usefulness of the generated system depends mainly on the generated screen system. Let us define several standard screens that allow us to handle data in the database. The basic generation principle in our approach is to generate the specific data editor for one or several tables connected by relationships. We generate a set of related screens and this enables direct transition from one screen to another.

The primary objects in our system are entities and relationships between them. We define some screen types with different user interface and different functionality for any entity or relationship. For instance, we can display on the screen entity, links to other entities (relationships in the ER model), information about related entities or display related entities by some relationship. The screens of all types are generated for each entity (accordingly to entity type) and for each relationship if the generation options do not define another behavior.

The menu provides access to any generated screen and standard operation defined for any application. The screens are organized in a hierarchy for easy orientation.

3.2 Screen components

Every screen logically consists of two large sets with different screen objects.

- **Information group** - screen objects that display information stored in the database and objects that are generated from the ER model. This group mainly consists from table fields and relations between entities. The basic *information subgroups* are: **Field group** (screen objects that display visible record fields), **Entity presentation group** (screen objects that display the record of the entity or the list of records), **Relationship presentation group** (screen objects that display relationship between entities), **Order button group** (radio button group that determines the order in what records are ordered).
- **Management group** - screen objects that provide additional management over the data stored in the database. Their generation depends on the screen type. The following *management subgroups* are defined: **Edit button group** (a group of buttons for entity record editing - *New*, *Edit*, *Save*, *Delete*, *Cancel* buttons), **Locate button group** (a group of buttons for locating the desirable record - *First*, *Next*, *Previous*, *Last*, *Find* buttons), **Print button** (a button for printing the current record

or a record list), **OK button** (a button for leaving the window), **Control button group** (specific buttons included in the screens of specific type).

4. Mapping ER model objects to application objects

4.1 Mapping sequence

A rough algorithm for application generation is the following.

- Map the ER diagram name to application name.
- Generate the screens for each entity (all screen types allowed for given entity type):

1. Generate screen name from entity name.
2. Generate each information subgroup needed for the given screen type. The layout of this group (horizontal, vertical, tabular or other) is not the subject of this paper.
    a) Generate all field groups for given entity.
    b) Generate information about all related entities via foreign keys.
    c) Generate information about all related entities via relationship without a foreign key at the end of given entity.
    d) Generate order button group for given entity.
3. Generate all management subgroups necessary for the given screen type.

- Generate screens of all allowed types for each relationship.

1. Generate screen name from related entity names and role names.
2. Generate information group as for the entity screens.
3. Generate management group as for the entity screens.

- Generate menu system organized by screen types. The deepest menu items are generated from the entity name (for entity based screen) or from two entity names and role names (for relationship based screen). The menu item will open the window for the specified screen type and entity (or relationship) as the central object.
- Generate additional menu items for standard operations.

4.2 Field mapping

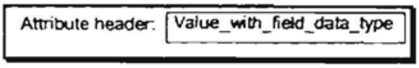

Attribute header: Value_with_field_data_type

Figure 3 Screen object group AttributeInfo

A field with type *Attribute* maps to screen object group ***AttributeInfo*** (Figure 3).

Attribute header is TextBox object with value generated from *FieldName* and EditBox object contains the value with type defined by field *DataType*. The EditBox length depends on the data type but is limited by some reasonable maximal length. If necessary scrolling through field is provided.

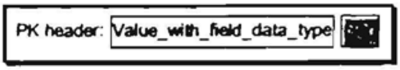


Figure 4 Screen object group PkInfo

A field with type *PK* (primary key) maps to screen object group ***PkInfo*** (Figure 4). It is similar to *AttributeInfo* but it also has the button ***Gen***. The user can manually enter a value for the record primary key or generate a value automatically by pressing the button *Gen*. Key generation depends on the selected default rule. When the user leaves EditBox the system checks whether the value is unique.

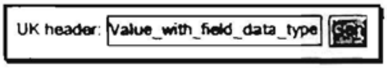


Figure 5 Screen object group UkInfo

A field with type *UK* (unique key) maps to screen object group ***UkInfo*** (Figure 5). This group is similar to the screen object group *PkInfo*.

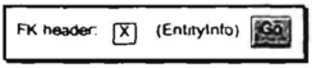


Figure 6 Screen object group FkInfo

A field with type *FK* (foreign key) maps to screen object group ***FkInfo*** (Figure 6). The content of this group depends on the screen type, relationship type and connected entity type. *Fk header* is TextBox object with value generated from *FieldName*. CheckBox is an optional element and is generated when corresponding relationship at the opposite end has cardinality 0..1, otherwise (cardinality is 1) CheckBox is not generated. We can assign an empty value to the foreign key field by turning off CheckBox. *EntityInfo* is another screen object group (see Entity presentation). The button ***Go*** is optional and its generation depends on the screen type.

4.3 Entity presentation

An instance of entity is represented by screen object group ***EntityInfo***. Let us define several subgroups for *EntityInfo*.

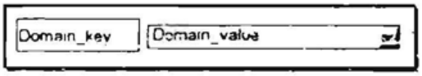


Figure 7 Screen object group DomainInfo

Entity with type *Domain* is represented by ***DomainInfo*** (Figure 7). ComboBox provides the selection of domain value and shows the current value. This screen object is obligatory part of the group. EditBox *Domain_key* is optional. It is generated by the following rules. At first, if entity has the visible unique key then *Domain_key* gets the data type from this field.

Otherwise, if entity has the visible primary key then it gets the data type from this field. If entity has no visible unique or primary key then EditBox is not generated. Both objects always reference to the same table record. EditBox can be used for selecting the domain value by entering the key in the EditBox. *Domain_value* is the entity representing text depending on the default text function.

Entity_text

Figure 8 Screen object EntityTextInfo

Entity with type *SimpleEntity* or *ComplexEntity* is presented by ***EntityTextInfo*** (Figure 8). TextBox contains the entity representing text depending on the default text generation function.

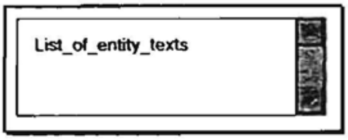


Figure 9 Screen object EntityListInfo

Several similar entities with type *SimpleEntity* or *ComplexEntity* are presented by ***EntityListInfo*** (Figure 9). ListBox contains entities representing texts depending on the default text generation function. *EntityListInfo* represents also entities with the type *Link* but in the text generation function can exclude one field with type *FK*.

4.4 Relationship representation

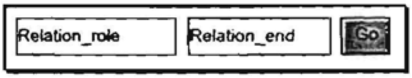


Figure 10 Screen object RelationshipEndInfo

One direction of relationship is represented by ***RelationshipEndInfo*** (Figure 10). Let us suppose that we represent relationship from *Entity_1* to *Entity_2* with role *Role_1*. TextBox *Relation_role* contains text 'Role_1' and TextBox *Relation_end* contains text 'Entity_2'. Button ***Go*** provides going to the screen that represents entity *Entity_2*.

4.5 Mapping of the indexes

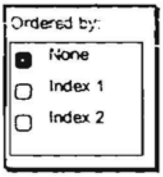


Figure 11 OrderButtonGroup

If the entity has at least one index then all indexes are mapped to Order button group represented by ***OrderButtonGroup*** (Figure 11). The first button *None* allows to remove any previously used record sequence. Every next button corresponds to some index.

4.6 Traversing through screens

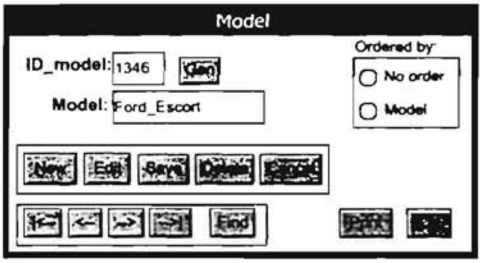
Traversing through screens is performed by button ***Go***. This button is usually attached to the screen object group representing the entity. The button brings us to another screen that belongs to this entity. The button can work in two modes - with filter or without one. If the filter option is chosen then in the newly opened screen we can access only those records that are logically tied with the record or records in the previous screen. For instance, if we fix any car model in the table Model then in the table Auto only cars with this model are accessible.

5. Screen types

The design of screen types depends on the user's needs. Let us define screen templates that can be regarded as basic screen types. The screen examples correspond to Figure 2.

- **Simple entity view**

This screen type can be used for the representation of any entity (Figure 12). The information group contains all visible field groups and *OrderButtonGroup* if any index is defined. The management group contains Edit button group, Locate button group, Print button, OK button.



Model

ID_model: 1346

Model: Ford_Escort

Ordered by

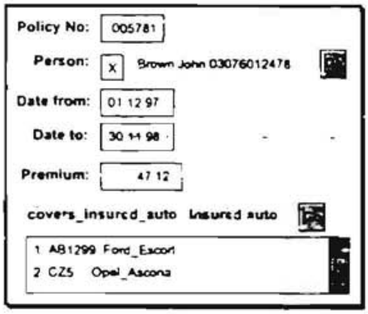
○ No order

○ Model

Figure 12 Screen for entity Model

- **Entity view extension with links**

This screen extension can be added to screens of entities with type *SimpleEntity* or *ComplexEntity*. All entities that have type *Link* and are directly connected via relationship with the given entity are shown on the screen. The presentation is performed by screen object groups *RelationshipEndInfo* and *EntityListInfo*. Figure 13 contains a screen fragment for the entity Policy with insured cars (*LongView* option is used) for the current policy.



Policy No: 005781

Person: X Brown John 03076012478

Date from: 01 12 97

Date to: 30 11 98

Premium: 47 12

covers_insured_auto Insured auto

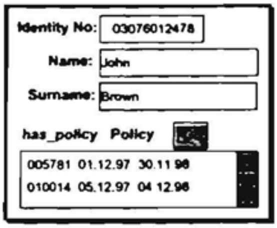
1 AB1299 Ford_Escort

2 CZ5 Opel_Ascona

Figure 13 Fragment of screen for entity Policy

- **Entity view extension with relations**

This screen extension can be added to the screens with types *Domain, SimpleEntity* or *ComplexEntity.* We represent all relationships that have foreign key at the opposite end (it means that the other entity references to the given entity via foreign key) and the corresponding entities. For the presentation screen object groups *RelationshipEndInfo*, *EntityTextInfo* or *EntityListInfo* are used. All the policies (*LongView* option is used) are displayed for the current person in Figure 14.



Identity No: 03076012478

Name: John

Surname: Brown

has_policy Policy

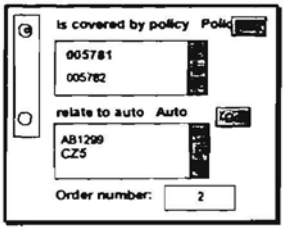
005781 01.12.97 30.11 98

010014 05.12.97 04 12.98

Figure 14 Fragment of screen for entity Person

- **Simple link view**

This screen is the special view to the entities with the type *Link* that links together exactly two entities with type *SimpleEntity* or *ComplexEntity* (Figure 15). The *ShortView* option is used to represent linked entities in *EntityListInfo*. A special *Control button group* determines the main ListBox. In this case for a fixed policy 005781 all insured cars are displayed in the other ListBox with label Auto.



● is covered by policy Polic

005781

005782

○ relate to auto Auto

AB1299

CZ5

Order number: 2

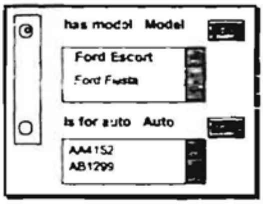
Figure 15 Fragment of screen for entity Insured auto

- **Embedded entities view**

This screen type is useful only for the entity with type *ComplexEntity*. Let us take *Simple entity view* as a base for such an entity. Instead of each field group *FkInfo* we incorporate all visible fields from the related entity. We can imagine each embedded entity as a subwindow where it is displayed with *Embedded entities view* for complex entity or *Simple entity view* for the simple entity. E.g., in Figure 13 the objects group with header *Person* is replaced by three screen object groups - *Identity_No*, *Name*, *Surname* from entity *Person*. During the generation process we must beware of cyclic embedding and stop embedding when we discover the cycle.

- **Relationship view**

This screen provides a special view to relationship and entities connected by it. Related entity is represented by *RelationshipEndInfo* and *EntityListInfo*. The main entity is selected by the radio button. Figure 16 shows the fragment for relationship [Model] is_for_auto /



● has model Model

Ford Escort

Ford Fiesta

○ is for auto Auto

AA4152

AB1299

Figure 16 Fragment of screen for relationship between entities Model and Auto

has_model [Auto] with *ShortView* option.

6. Conclusion and future directions

This approach is based on the common ER diagram elements mapping to some screen object constructs. It is not hard to create templates for generation - the standard code for the whole screen and the standard SQL based code fragments for each generated screen object group. Generation basically is code compiling from prepared code templates. This technique partly is applied in practice - the real business applications are developed but screen code is written by hand.

This approach has several future directions that seem very interesting. The screens can be generated dynamically while application is running. E.g., the appropriate HTML page can be generated and displayed. This improvement enables the information view to be changed dynamically.

An ER diagram can be described by context-free grammar (E.g., in BNF notation). The generator is an interpreter that reads the ER diagram as a program and creates a source code for the screens [6]. Other graphical tools, e.g., GRADE [5] can be used to prepare the ER diagram as input statements according to this grammar for the generator.

7. References

[1] Tucker, A.: The Computer Science and Engineering Handbook, CRC PRESS, 1997.

[2] Billings, C., Billings, M., Tower, J.: Rapid Application Development with Oracle Designer/2000, Addison-Wesley, 1997.

[3] Anderson, W., Wendelken, D.: The Oracle Designer/2000 Handbook. Addison-Wesley, 1997.

[4] Rumbaugh, J.: Object-Oriented modeling and Design, Prentice-Hall, 1991.

[5] Barzdins, J., Kalnins, A., Podnicks, K. et al.: GRADE Windows: an Integrated CASE Tool for Information System Development, Proceedings of SEKE'94, pp.54-61, 1994.

[6] Arnicane, V., Arnicans, G., Bicevskis, J.: Multilanguage Interpreter, Proceedings of the Second International Baltic Workshop, pp.173-174, 1996.

International Conference "Information Technologies and Telecommunications in the Baltic States"

ABSTRACTS OF PAPERS FROM THE BALTIC IT&T '99 CONFERENCE

RĪGA, APRIL 28-30, 1999

RĪGA CONGRESS PALACE

The Information Technology Committee of the Baltic Council of Ministers

DT Media Group

The Concept of Setting Up a Communications Server

Mr. Guntis Arnicāns, Dr. Jānis Bičevskis, Mr. Ģirts Karnītis, Faculty of Physics and Mathematics, University of Latvia

A communications server is a set of software and computer equipment that allows a wide range of users, both domestically and internationally, to receive information from a variety of sources (government registers, data bases, information systems) through a single contact point. A communications server identifies users, authorizes the use of the respective data, fulfills a request that involves several information sources, and evaluates the cost of the process so that the appropriate financial transaction can be made. A communications server allows users to learn where information is being stored and what kind of information it is, as well as to request and to receive information from various registers without any need for in-depth knowledge about the technical aspects of its storage. The need to establish a communications server became evident when the governments of the Baltic States were setting up their joint data transmission network. In order to allow institutions in one country to obtain information about objects registered in another (enterprises, persons, motor vehicles, etc.), it is useful to receive the necessary data from a single information source, without having to study the data base structures of the other country. The use of the communications server, as has been seen through the elaboration of an integrated state significance information systems project, is also of significance within one country, because it provides a universal resource for information exchange among various information systems.

A communications server is a set of software and computer equipment that allows a wide range of users (both in Latvia and in other countries) to receive information from a variety of sources (government registers, data bases, information systems) through a single contact point. A communications server identifies users, authorizes the use of the respective data, ~~fulfills a request that involves several information~~ sources, and evaluates the cost of the process so that the appropriate financial transaction can be made. A communications server allows users to learn where information is being stored and what kind of information it is, as well as to request and to receive information from various registers without any need for in-depth knowledge about the technical aspects of its storage.

The need to establish a communications server became apparent when the governments of the Baltic States were setting up their joint data transmission network. In order to allow institutions in one country to obtain information about objects registered in another (enterprises, persons, motor vehicles, etc.), it is useful to receive the necessary data from a single information source, without having to study the data base structures of the other country. The use of the communications server, as has been seen through the elaboration of an integrated state significance information systems project, is also of significance within one country, because it provides a universal resource for information exchange among various information systems.

PROBLEM IDENTIFICATION

The need to establish a communications server was noted in the national program "Informatics" [1 and 2], as well as during the elaboration of two major projects – the Baltic States Government Data Transmission Network (hereafter in the text – the Network) [3 and 4] and the Integrated State Significance Information System (hereafter – the Megasystem) [5]. The goal in establishing the network is to provide fundamental improvements in the exchange of telecommunications and data among the administrative institutions of the Baltic States. During the first phase of the project (1998 and 1999), universal solution is being set up to provide for the exchange of data among Latvia's Company Register, Motor Vehicles Register and Lost Motor Vehicles Register, as well as between these registers and the related international information structures. So far this has involved three concrete activities:

1) Accession of the Latvian Company Register to the European Business Register (EBR);

2) Cooperation between the Motor Vehicles Register and the related European-level structure EuCaris, as well as the establishment of a motor vehicles insurance system in Latvia (the so-called "green cards");

3) Improvements to the system whereby lost and stolen motor vehicles are registered in Latvia, including a connection to the international data bases of Interpol in this area.

During the second phase of this project, between 2000 and 2002, more work will be done to include Latvian registers into the Network and to integrate them into international information structures. In the second phase, the plan is to place the Population Register, the Lost Persons Register, the Lost Personal Documents Register, the Educational Documents Data Base, the Visas Data Base, the State Statistics Information System, the Consular Information System, the Health Care Information System and the Narcotics Information System on the Network.

In a situation where information from various sources is available on the Network, but users have no knowledge about the technical details of storing that information, there is an obvious need for a universal solution, and that is where the communications server comes in. The main requirement for a communications server is that it must allow users to formulate their information requests in a simple way and to receive responses to those requests without having to understand the technical aspects of the process. Users are not, after all, informatics specialists; they are employees of other administrative structures of the state, and there is no reason to think that they know anything about the way in which data objects are distributed among the registers of another country. We can expect both standardized and wholly unpredictable requests in this process. In terms of the urgency of requests, we can expect demands for on-line responses that require rapid response, as well as requests for off-line responses that can take hours or even days to fulfill. Needless to say, in setting up the communications system we must provide for all aspects of information confidentiality and user authorization.

The setting up of the communications system is important not only in the context of the Network, but also in the context of the Megasystem, which is a universal resource for the exchange of information among various information systems within a single country.

THE CONCEPT OF THE SOLUTION

The communications server, which is illustrated in Figure 1, is an Internet resource point. Users of the server can access it via various protocols – HTTP, CORBA, DCOM, SMTP (E-mail) and FTP. The server provides users with an opportunity to find out where information is stored and what kind of information is available, and then to request and receive information from various registers without studying their structure. Because users may have access to sensitive information, users are identified with certificates, and all data transmissions are coded.

Users who wish to have access to sensitive information before work with the system is begun must receive a certificate that corresponds to the X.509 standard. The certificate must issued for a specific period of time (usually one year) by a specialized institution (presumably in Latvia this would occur under the supervision of the Constitutional Defense Bureau). Certificates of this kind contain information that identifies the user, and they are virtually impossible to forge. The certificates are used to code data and to identify the user. Latvia's communications server will use a standard coding protocol such as SSL.

A user of the communications server sends information requests to it and receives responses from it. This can happen both on-line (HTTP, CORBA, DCOM) and off-line (HTTP, E-mail, FTP).

In the on-line regime, work with the communications server is based on the following structure: At the beginning of the process the user is identified. This means that the user sends his or her certificate to the communications server, which reviews it and specifies the user's rights. If the user does not have a certificate, then he or she can access the communications server as a guest and receive a limited amount of information from it. Next the user requests information. The communications server once again identifies the user and, on the basis of the level of the user's authorization, makes the appropriate requests to the data registers, sending the response to the user when it is received. The register receives not only the information request from the communications server, but also the user's certificate, which means that the register itself can identify the user and the user's level of authorization. The result of this is that the register provides only that information to the communications server for which the user is cleared.

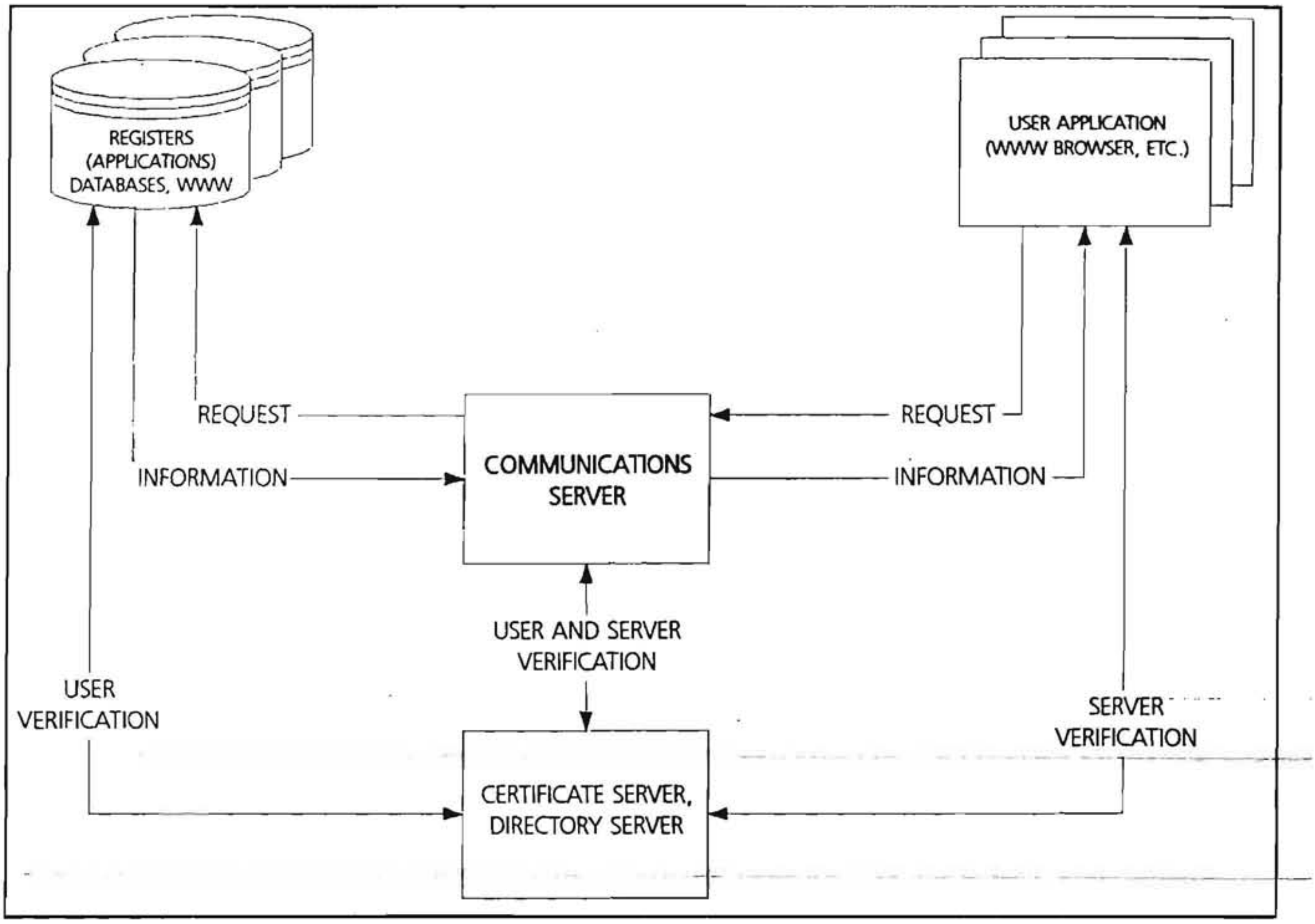


Figure 1. The operational structure of the communications server

In an off-line regime, the user requests information via HTTP, E-mail or FTP. During periods of time when it is less busy (usually at night), the communications server processes the request – identifies and verifies the user and then requests the respective information from the information registers. The response is sent to the user via E-mail, or it is stored until the user asks for it on-line.

The main advantage of an on-line regime in this process is that information can be obtained immediately when the need arises. This system can be used in cases when the speed at which a response is received is of importance, either from the point of view of the system (e.g., at border control facilities), or from the point of view of the operation (e.g., an application in which the registration number of an automobile is entered and information is received about the automobile from the Road Traffic Safety Department so that it need not be entered a second time).

The advantage of the off-line regime is that registers can even out the volume of work that is required, given that at night there should be relative few on-line requests for information. Off-line requests can also be sent in by users who have dial-up Internet connections, thus reducing costs. It is advisable to make off-line requests less expensive than on-line ones so that users are motivated to use the off-line system.

THE FUNCTIONS OF THE COMMUNICATIONS SERVER

We can specify five main functions for a communications server:

- User identification
- Authorization with respect to the use of information
- Management of user rights
- Fulfillment of requests that involve several information sources
- Evaluation of the costs of each request for billing purposes

USER IDENTIFICATION IN A COMMUNICATIONS SERVER

As was noted before, user identification involves X.509-standard certificates. In order to ensure that the certificate mechanism is operational, a communications system needs both a certificate server and a directory server. The former is a server that belongs to the certifying organization, generating and maintaining electronic certificates – both server certificates (issued to the server) and client certificates (issued to the user). The latter is a server in which the public keys of the certificates are stored, along with information about certificates that have been issued – when a certificate has been issued, to whom it was issued, and whether the certificate is valid or has been revoked.

The directory server is available to any interested party. For example, if a WWW server has been issued a certificate, any WWW user can ascertain that the server is secure. If a WWW client has been issued a certificate, in turn, the WWW server can ascertain that the client is authorized to work with the server. Both the client and the server can check the validity of the submitted certificates by looking them up in the directory server.

Work with certificates in WWW applications involves SSL (Secure Socket Layer) technologies, which are supported by most WWW servers, as well as the main WWW browsers – Netscape Navigator and Microsoft Internet Explorer. SSL technologies provide the following components of secure communications:

1) **WWW server approval:** A user can ascertain the fact that the WWW server is secure and that it can be entrusted with confidential information;

2) **The privacy of information:** The entire information flow between the client and the server is coded, using a unique session key. The session key is coded by the server with the client's public key in order to send the respective information to the client in a secure way. Each session key is used in only one session, which makes it difficult to decode the information without authorization. The information, in other words, cannot be viewed by unauthorized persons, even if it is intercepted on its way between the server and the client.

3) **The integrity of the information:** Both the server and the client calculate the control code on the basis of the content of the information, and if the information has been changed en route, the codes do not match. This means that the receiver of the information sees precisely the same information that was sent by the sender.

Secure data exchange between the WWW server and the client occurs in the following way when SSL technologies are used:

1) The client sends a request for data exchange to the WWW server;

2) The server in response sends its certificate to the client, asking for the client's certificate if appropriate;

3) The client checks the validity of the server certificate through the digital signature of the certificate server, sending the client's own certificate to the server if necessary;

4) When the authorization process is complete, the client sends the session key to the server, coding it with the public key of the server;

5) Both the server and the client know the session key, and further data flow between the server and the client during the respective session is coded with the session key.

The certificates of the server and the client are exchanged quickly and without any involvement by the user. The same is true with respect to an exchange of certificates among other applications.

When information is requested from the communications server (through the WWW or otherwise), the process occurs in the following way:

1) The user is identified through the aforementioned protocol, and the communications server checks the user in the directory server.

2) The communications server has a data base which records user rights, and the server uses this data base to specify the authorization level of the specific user. In carrying out the user's request, the communications server checks the user's rights in its own data base and, if the necessary level of authorization is there, then the request is sent along to the concrete register.

3) The register is also sent identification data about the user who has requested the information.

4) The software in the register checks the information in the directory server and authorizes the user.

5) According to the level of the user's authorization, either the request is carried out and the result is returned to the communications, server, or the communications server is told that the user does not have the right to carry out the request.

6) The communications server returns the result to the user.

A user can also request information from the register directly, without passing through the communications server. In that case the operational mechanism is similar:

1) When the information is requested from the register, the user must supply identifying information (a certificate).

2) The software in the register checks the information in the directory server and authorizes the user.

3) On the basis of the user's authorization and the level of his or her access rights, either the request is fulfilled and the result is sent back to the user, or the user is sent information saying that he or she does not have the right to receive the data.

This mechanism ensures that there is no need for the user to reintroduce identification each time a new request is made. In each session, the user is identified on the first occasion that a request is made with respect to a confidential data source, and in later requests the information is sent on to all of the respective information sources. Another advantage of the mechanism is that there is a centralized method for distributing user rights, as well as a unified policy with respect to this. It's also true that the user's rights do not change depending on the way in which he or she accesses the information – via the WWW, via a different application, or through some other method.

MANAGEMENT OF USER RIGHTS

The rights of users can be divided into several categories:

- The right to obtain information about what is stored in a concrete register – provided that the information is publicly available;
- The right to obtain information about one entry in one table in one register, based on the unique identifier of that particular entry;
- The right to obtain a list of data from one table in one register, selected on the basis of specific criteria;
- The right to obtain a list of data from several tables in a single register (whether the link exists or not);
- The right to obtain information from several tables in one register that are linked through a specific relation, the data being chosen on the basis of specific criteria;
- The right to obtain information about one object from several registers on the basis of the primary key of the object;
- The right to obtain information about the existence of a link among specific objects from various registers;
- The right to obtain a list of data that are selected on the basis of criteria entered by the user, the data coming from several tables in several registers that are mutually linked.
- The obtaining of information can be differentiated at four levels:
- A response as to whether the requested information has been found or has not been found;
- A response as to how many entries have been found;
- The primary keys of objects;
- The data that is being requested.

Each of these levels provides a different volume of information, and there are instances when the jump between proximate levels is quantitative, while in other instances it is qualitative. We could consider four different requests here:

"Does individual X own an automobile?"

"How many automobiles does individual X own?"

"What automobiles does individual X own?"

"Does individual X own automobile Y?"

The management of user rights is intentionally divided up so that it occurs in several places. The communications server has its own user management module, in which it stores information about the right of users to make various kinds of complex requests. Information about the right of a user to receive data from a specific register is stored either in the communications server or in a concrete register. The place where information about user rights is stored is harmonized between the communications server and the register. Because it is expected that before a register issues information, it will want to check the user's rights to use the information, then information about the user's rights with respect to a specific register will usually be stored in that register. From the perspective of centralized management, it would be better if information about user rights with respect to all registers were stored in the communications server. For various organizational reasons, unfortunately, this is either impossible on only partly possible. Information about user rights is stored both in the communications server and in the registers themselves.

The communications server is designed to work with both of these options, as well as with a combination of them, and the following scheme emerges:

- The communications server checks the right of the user to make a request in the first place, as well as the right of the user to seek out a link between objects in various registers;
- The communications server checks whether the user rights with respect to the concrete register are stored in the communications server or the register;
- If the rights are stored in the communications server, then it checks the rights before it sends the request to the register;
- If the information is stored in the register, then the register checks the user rights before it fulfills the request;
- If the rights are not stored in the register, then the register can, if necessary, receive information about the rights from the communications server in order to be able to check the rights of the respective individual to make the request.

Because it is possible for users to connect to the registers not only via the communications server, but also directly from an application, and because it should be true that in both instances the user has the same authorization to obtain information, then the check of whether a user has the right to obtain information from a specific register should occur not in the communications server, but in the register itself.

INFORMATION REQUESTS AND THE OBTAINING AND DEPICTION OF INFORMATION

The basic mission of the communications server is to provide users with access to various information sources so that they can obtain data from them. Let us take a look at the problems that arise in this process, devoting particular attention to the submission of requests and the obtaining of responses, and leaving aside the issue of user authorization, control over data access, registration of who has asked for information and what information has been requested, billing issues and such matters.

INFORMATION SOURCES

An information source or resource facility can be any information system or data base from any organization. There are administrative regulations concerning the organizations, information systems and data bases that are included in the communications server's network of services.

Over the course of time, the number of information sources can reach into the tens or even hundreds of sources. In Latvia alone there are already several dozen government registers, and their number may increase. Communications servers should also provide access to certain foreign information sources, as well as to the data bases of various other organizations in Latvia; these, too, could be included in the range of services provided by the communications server.

The communications server itself does not have an information sources. Each information source is primarily meant to carry out concrete and specific functions inside the respective organization Information systems and data bases that are used in an organization are chosen, designed and optimized specifically for the needs of the respective organization. They may not be aimed at providing information to other entities, but if such an opportunity is intended, then it can be very specific, and many limitations can be applied to it. This means that the communications server must adapt to the information sources, and not vice-versa. Of course some information sources can upgrade their information systems and optimize their data exchange procedures in order to meet the communications server's requirements.

Information sources that are part of the communications server's network can differ in terms of significance and volume. The more significant a data base, the better must be cooperation with it. The size of data bases must also be taken into account, because it has much to do with the respective data processing mechanisms.

Another key issue is the quality and stability of information sources. Information systems can involve a wide variety of technologies, and they are of varying ages. Depending on the resources that have been invested, some are of a higher quality and some – of a lower quality. Of course, it is easier to make contact with a high-quality information system and data base that have been designed with modern technologies than with systems that are old and of a lower quality level. A communications server must certainly be ready to deal with information sources that are unstable, that make errors and that in some instances are not even accessible.

Information systems can be designed with various systems, they may have various data bases, and their use may involve various operating systems and computer technologies. A communications server must be prepared to handle these problems, although this is no longer the worst possible difficulty, given that many different solutions are in existence.

Information can be stored in a wide variety of formats – that is the next issue. The most popular method for data storage is still relation data bases. Object-oriented data bases, static WEB pages and dynamic WEB pages that are generated from an internal format are becoming rapidly more influential. We must not, however, forget other information storage methods such as files of many different structures.

A concrete information unit and a logical group of information units can be doubled, stored in various formats, coded in various ways and stored in such a way that some of the information is kept secret. Information can be contradictory either within a single information system or among various information sources. This means that in the future the field of communications servers will have to involve various laws and data processing algorithms that are based on the technologies or artificial intelligence.

All of these aspects serve to demonstrate how serious is the issue of various information sources being highly varied. It should also be added that this heterogeneity exists among more than just information sources. The same situation can exist within a single register or a single organization.

It must also be remembered that each information source exists fairly independently. It can be updated, changed or liquidated, it can be created anew, its operations can be suspended for a while, or it can be withdrawn from cooperation with a communications server. This means that a communications server must exist in an environment that is not only highly varied, but also is extremely changeable.

USERS

For our purposes, we will say that a communications server user is any subject that wishes to obtain a service from the server.

Users are usually differentiated on the basis of their level of authorization to obtain specific information from specific information sources. These rights are regulated by law and by other normative acts, and they are managed by a specific user management bloc within the communications server.

From the perspective of the communications server, another very important user classification is based on a different aspect – the way in which the user requests information and the way in which the user receives a response. A communications server should be operated on the basis of the principle that it is there for the convenience of users, not vice-versa. This principle means that the server must be ready to receive information requests of a great many varieties and forms, and it must be ready, every time, to provide a response that is convenient for the user in terms of its type and form.

REQUESTS AND RESPONSES

A communications server must be ready to accept information requests that are stated in various ways and forms. The main operational regime for communications servers is an on-line connection, but this can involve a dedicated line to the communications server, dial-up access to the server, or a connection through informational networks (the Internet, the Latvian State Significance Data Transmission Network (VNDPT), or the networks of other national, global or organizational networks). We must also remember other ways to submit a request – E-mail, a request submitted on an electronic information carrier such as a diskette, a written request submitted on paper, or even an oral request.

Responses to various requests can be prepared in the same format as the original request. It should be added, however, that the user must have the right to select the method of response, irrespective of the way in which the request was submitted. Limitations on the ways in which requests and responses are formatted can be specified by administrative regulations, but in terms of technologies, a communications server must be prepared for all kinds of cooperation methods.

The forms of requests and responses can be highly varied. The most popular cooperation form is probably a WEB page, both for requests and for responses. This form of cooperation can be highly varied, and this is underpinned by existing WEB-type applications. The use of special procedures and functions may also be important when the procedure itself has parameters that specify the request and its result (i.e., the response to the desired request as specified by the parameters). Cooperation can also occur in the following forms:

1) Special applications that can work with the communications server;
2) Active objects that can work with the communications server and can be used in the client's applications;
3) Files with requests that are recorded in a specific format or response files in a specific format;
4) A group of files (including even data bases) for the requests and the responses;
5) Paper documents in an agreed format for requests and responses;
6) E-mail, which can be seen as a modification of items 3, 4 and 5 on this list.

It is commonly held that requests from a user can come in a dialogue regime from a human user and in an automated regime where the user is an application on the user's computer.

There must also be plans to work in a synchronous regime (request-wait-response) and in an asynchronous regime (request-processing over a specific period of time-report to the user about the availability of a result-response), because this ensures more efficient work for the user and the communications server alike, especially when it comes to processing large and complex requests.

In work with the user thought must also be given to such aspects as the various levels of preparedness among users, the language of communication, the respective text coding formats, the abilities of the user's computer equipment, operating systems and applications, and limitations in all of these things.

In other words, the main mission and, at the same time, the main problem that a communications server must handle is the way in which many different kinds of requests can be handled, submitting processed information from various information sources that sometimes are not compatible, and submitting a result to the user in the desired type and form.

INFORMATION ABOUT INFORMATION

As the number of information sources available through the communications server increases, an overabundance of information can quickly occur – one in which even the administrators of the communications server can get lost. It is necessary to classify all of the information sources and the information that is contained therein, keeping firmly in mind that information sources can change.

Communications servers must have data source repositories that contain formal descriptions of the sources, their properties, the data that are contained within them and the properties of the data. These repositories must be very flexible, it must be able to change them easily and quickly so that changes in the surrounding environment can be monitored. If there is to be a proper reaction to user requests, other parts of the communications system must be able to adapt to changes in the repository in a dynamic way.

The repository is not, however, meant only for internal use in the communications server. The user, too, must know where and what he can receive (of course, within the limitations of the user's authorization). This means that the communications server must also, so to speak, provide information about information. Using forms and terms that the user can understand, the server must describe the information that can be obtained and the ways in which it can be requested. There must also be efforts to link the various request formulation mechanisms as closely as possible to the repository, thus making easier the work of a user who takes advantage of the communications server's services only seldom.

Users often don't care where and how the desired information is stored. This means that the communications server must satisfy requests that concern information from many different sources. The repository, therefore, must also describe the links between the sources, as well as the ways in which various contradictions among the sources can be resolved, data be converted, etc. The repository must be an entity that makes it possible to consider all of the sources in a communications server to be one, big data base.

THE ABILITIES OF THE COMMUNICATIONS SERVER

A communications server is a dynamic system which must work in a highly changeable external environment. A communications server must be much more flexible and dynamic than a day-to-day system, because it must work with highly heterogeneous external information systems that keep up with rapid technological changes. When it comes to technologies, communications servers must be a step ahead of other systems, because otherwise it may turn out that the communications server ends up unable to perform its functions.

The goal of this paper is not to describe the internal architecture and ideology of communications servers precisely. The establishment of such systems is a very serious process throughout the world these days, and various solutions are being sought out that are linked to the following technologies:

- Distributed Dynamic Systems
- Distributed and Dynamic Objects
- Dynamic Object-Oriented Programming
- Reflection
- Domain Specific Programming Languages
- Artificial Intelligence

Many of these technologies are still quite new, and they are still being developed. This means that not all of them have ready-made tools that support various properties or functions of the technologies. Some tools exist, some are at the prototype stage, while some have already become popular among professionals (this is particularly true of prototype tools that are designed at universities and research laboratories in order to test the latest technologies). In the design of a communications server it is worthwhile to such modern technologies and research results as the Multilanguage Interpreter [6] and the Database Browser Generator [7].

EVALUATION OF REQUESTS FOR BILLING PURPOSES

A billing system is part and parcel of the mechanism whereby a communications server fulfills requests. When a specific request is fulfilled, the system not only does what has been requested, but it also automatically calculates the resources that are used in the process. Within the communications server, a price has been attached to every resource, and it can change on the basis of the volume of information that has been requested, the time of day when the request is filed, etc. The price of each request is calculated automatically and stored in a journal that then is used for billing purposes.

A resource is an information request to a register. The price of resources changes on the basis of the type of the request, the complexity of the request, the register that is involved, etc.

USES OF A COMMUNICATIONS SERVER

There are three major ways to use a communications server:

- As an international resource facility that can be used to access information from Latvian registers;
- As an internal resource facility that can be used to search for information in registers;
- As a way of setting up cooperation among various registers.

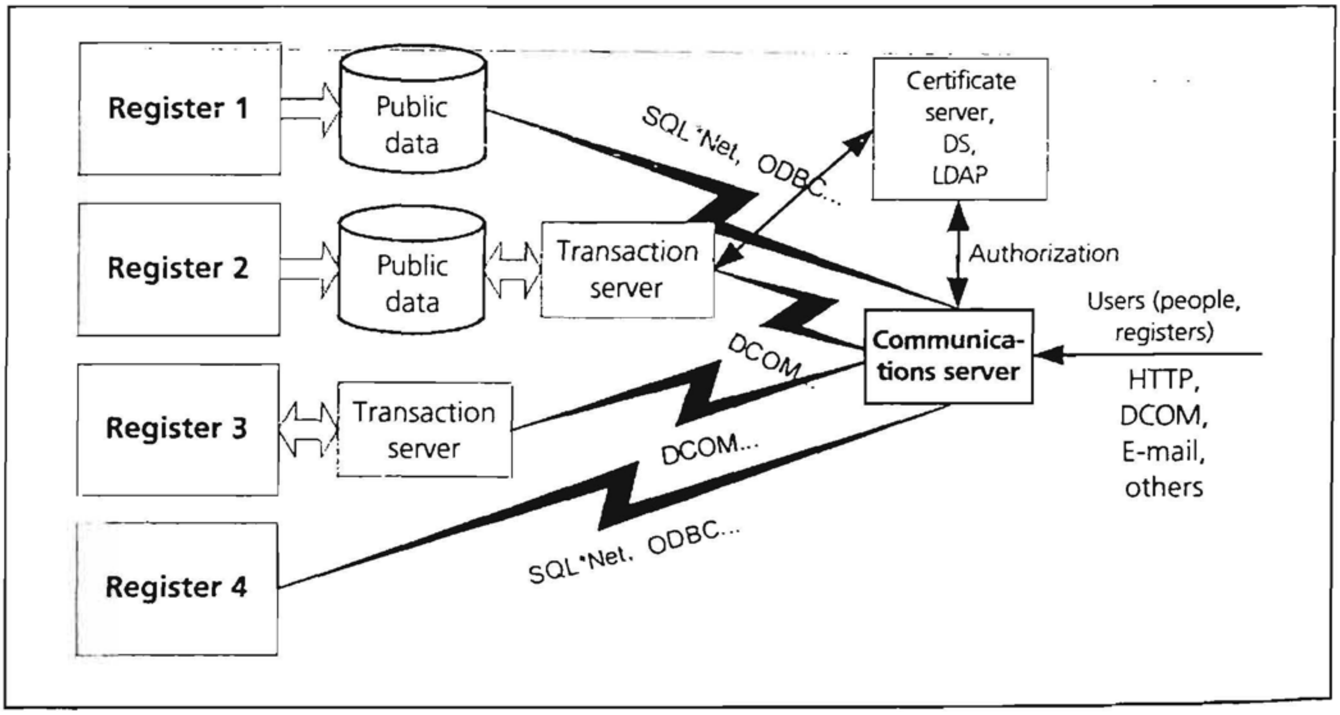


Figure 2. The structure of cooperation between a communications server and other entities

The need to access information from Latvian registers via a single contact facility is the main reason for elaborating the communications server. Of course, this is more than just a trivial solution in which a single Internet application is designed for connection to other registers via their Internet addresses. This simplified design does not deal with the main issue – the ability to collect information from various sources (i.e., various registers) without the user having to hook up to each register separately. The information that a user needs is collected from the various registers by the communications server, and the user himself may be completely unaware of the technical details of this process. Thus the communications server is needed by employees of foreign institutions in order to obtain information that is stored in Latvia's registered.

A second use for the communications server is the fulfillment of domestic information requests in Latvia. The previously described situation in which users do not want to or are unable to understand the technical details of information storage is typical among the personnel of Latvia's administrative structures. Of course, given the fact that access rights to authorization may vary for foreign users and Latvian users, the communications server sets out a unified set of requirements in this area, and solutions are the same for both groups of users.

The third way of using a communications server is to use it in order to exchange information among various registers. It is obviously irrational to maintain communications channels and to conduct information exchange individually with each of many registers that are mutually linked. It is much more rational to set up a centralized contact facility – the communications server – which is linked to all of the registers and through which information is exchanged among them. The general process of information exchange among registers via a communications server is shown at Figure 2.

This diagram shows four ways in which a register can be connected to a communications server. Every register that participates in the data exchange procedure can have its own data base in which those data that are intended for transfer to other registers and for publication can be separated out. The data base can be maintained by a separate computer or server so that approaches to the public data base do not hamper work with the basic data base of the register. Data from the basic data base are regularly copied to the public data base (an automatic replication mechanism). This solution is rational not only from the perspective of using communications channels; it also ensures:

- That the fulfillment of external requests does not hamper the work of the register;
- That there is higher security, i.e., that in the case of unauthorized access, the basic data base is not damaged.

The link between the communications server and the public data base can be implemented on the basis of various technologies, such as DCOM object calls, MS Transaction servers and Oracle SQL*NET. User authorization is provided via a certificate server, a directory server and the Lightweight Directory Access Protocol (LDAP).

REFERENCES

1. The Latvian national program "Informatics", Ministry of Transport, 1998, 211 pp.

2. The Latvian national program "Informatics" (summary), Ministry of Transport, 1998, 60 pp.

3. "The Baltic States Government Data Transmission Network: Conceptual and Methodological Considerations", Riga, 1998, 11 pp.

4. "The Baltic States Government Data Communications Network. Feasibility Study for a Data Networking Concept to Improve the Interchange of Information Among the Baltic States", Riga, 1998, 83 pp.

5. "The Integrated State Significance Information System (Megasystem): Conceptual and Methodological Considerations", Rīga, 1998, 16 pp.

6. Arnicāne, V., Arnicāns, G. and J. Bičevskis. "Multilanguage Interpreter", in Proceedings of the Second International Baltic Workshop, 1996, pp. 173-174.;

7. Arnicāns, G. "Application Generation for the Simple Database Browser Based on the ER Diagram", in Proceedings of the Third International Baltic Workshop, 1998, pp. 198-209.

Contact information:
University of Latvia
Raiņa bulv. 29-331, Riga, LV-1050, Latvia
Tel.: +371 7228226
Fax: +371 7820153
E-mail: bics@lanet.lv

Baltic IT&T 2000

4th INTERNATIONAL CONFERENCE INFORMATION TECHNOLOGIES AND TELECOMMUNICATIONS IN THE BALTIC STATES

The Information Society: The Future for the Baltic Region

Radisson SAS Daugava Hotel, April 6 – 7, Riga, Latvia

Information Technology Committee of the Baltic Council of Ministers
Data Media Group

Abstracts of papers from the Baltic IT&T 2000 Conference

The Unified Megasystem of Latvian Registers: Development of a Communications Server – the First Results and Conclusions

Mr. Ģirts Karnītis, assistant, Mr. Guntis Arnicāns, lecturer, Prof. Jānis Bičevskis, Head of Department of Computer Science, Faculty of Physics and Mathematics, University of Latvia

This paper describes a development of Communications Server, the first realization version and conclusions. A communications server is a set of software and computer equipment that allows a wide range of users (both in Latvia and in other countries) to receive information from a variety of sources (government registers, data bases, information systems) through a single contact point. A communications server identifies users, authorizes the use of the respective data, fulfills a request that involves several information sources, and evaluates the cost of the process so that the appropriate financial transaction can be made. A communications server allows users to learn where information is being stored and what kind of information it is, as well as to request and to receive information from various registers without any need for in-depth knowledge about the technical aspects of its storage.

INTRODUCTION

The need to establish a Communications Server became apparent when the governments of the Baltic States were setting up their joint data transmission network[6]. One of the main tasks is to obtain information about objects (enterprises, persons, motor vehicles, etc.) without having to study the data base structures in any country. One year ago the concept of Communications Server was defined [1] and project of Communications Server was started in Latvia.

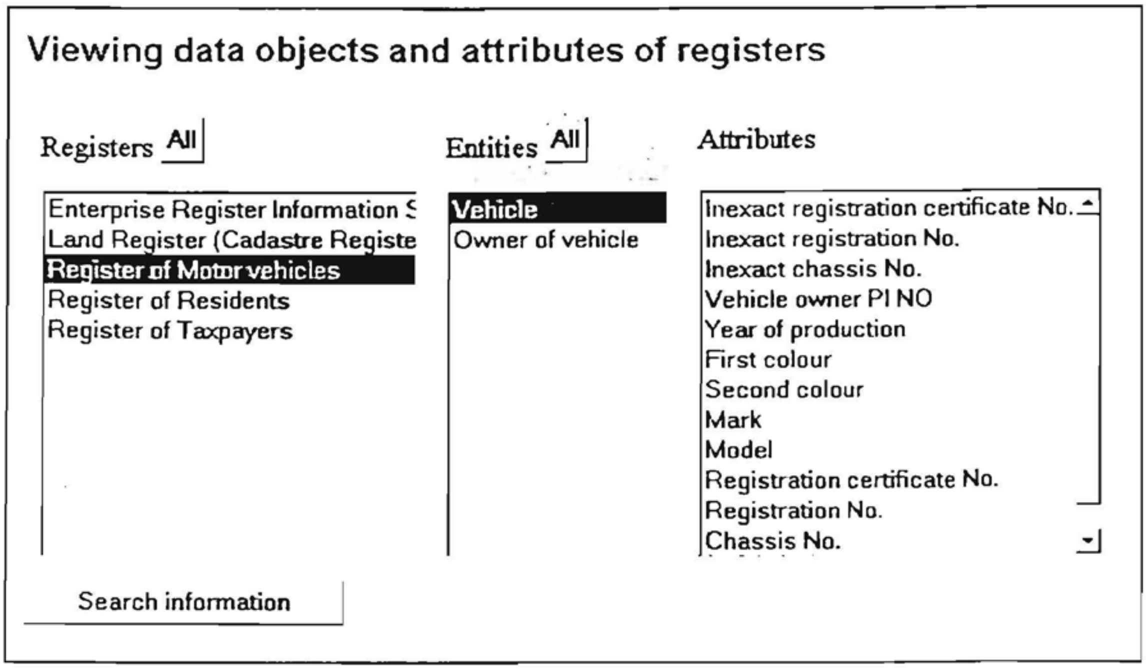
Data retrieval from different autonomous sources has become a hot topic during the last years not only in Latvia but also in all countries or large enterprises. The problem is very complicated and its solution can takes several years and many high-qualified specialists to solve it [2][3][4]. There was made the choice to develop Communications Server step by step in Latvia. Latvia has several dozens of registers and information sources (public and with restricted access). To develop all system at once it is too complex due to, for example, various organizational and technical problems. Design and implementation of all functionality for the Communications Server also takes much time.

CORE OF THE COMMUNICATION SERVER

The main functions for a Communications Server are:

1. User identification
2. Authorization with respect to the use of information
3. Management of user rights
4. Fulfillment of requests that involve several information sources
5. Evaluation of the costs of each request for billing purposes

It is more or less clear how to implement the first three functions, but the largest problems arise to develop last two functions. The original technology was developed to search and obtain data from various data sources during the design phase. This technology bases on WEB technologies and Meta models of data sources[5].



Viewing data objects and attributes of registers

Registers All

- Enterprise Register Information S
- Land Register (Cadastre Registe
- Register of Motor vehicles
- Register of Residents
- Register of Taxpayers

Entities All

- Vehicle
- Owner of vehicle

Attributes

- Inexact registration certificate No.
- Inexact registration No.
- Inexact chassis No.
- Vehicle owner PI NO
- Year of production
- First colour
- Second colour
- Mark
- Model
- Registration certificate No.
- Registration No.
- Chassis No.

Search information

Figure 1. Registers and data objects

For the first version of Communications Server was determined several principles or requirements:

- It is possible define new source in couple days
- It is possible to access any type of data source
- It is easy and quickly create primitive services (wrappers) to search and obtain needed data from source
- It is possible to tie related data from various data sources
- It is easy maintain all system (make changes, add new possibilities, etc.)
- The program code have to be simple and small to reduce the possibility to make mistakes
- Initially data is retrieved only from WWW (from end-user point of view)

REGISTER OF REGISTERS

The Register of registers is the information system that contains information of other information systems maintained in Latvia. There is much useful information, such as IS name, content, owner, data model, relations with data objects in other information systems, in database of the Register of registers.

The first version of Communications Server widely uses information stored in Register of registers. For instance, the information searching starts with high level representation of data sources and objects stored in them. See the Figure 1.

We can see what data sources are available, what data objects are available from these sources and what attributes describe each data object. We can start browsing from any data source or data object.

BASIC ADDITIONAL REQUIREMENTS FOR COMMUNICATION SERVER

Various additional aspects and requirements were taken to create first version of Communications Server:

- some data are very sensible (only for authorized and restricted use)
- some data are available for money

For these reasons we keep a close attention to security, to log all activities and to accounting of all retrieved information to calculate accounts between information providers and consumers.

Security is designed to fulfill requirements determined by law, government and information source provider. At present for each user are defined: what data objects (register, information from register, etc.) are accessible, what operations can be done (searching and retrieving) and what templates of WWW

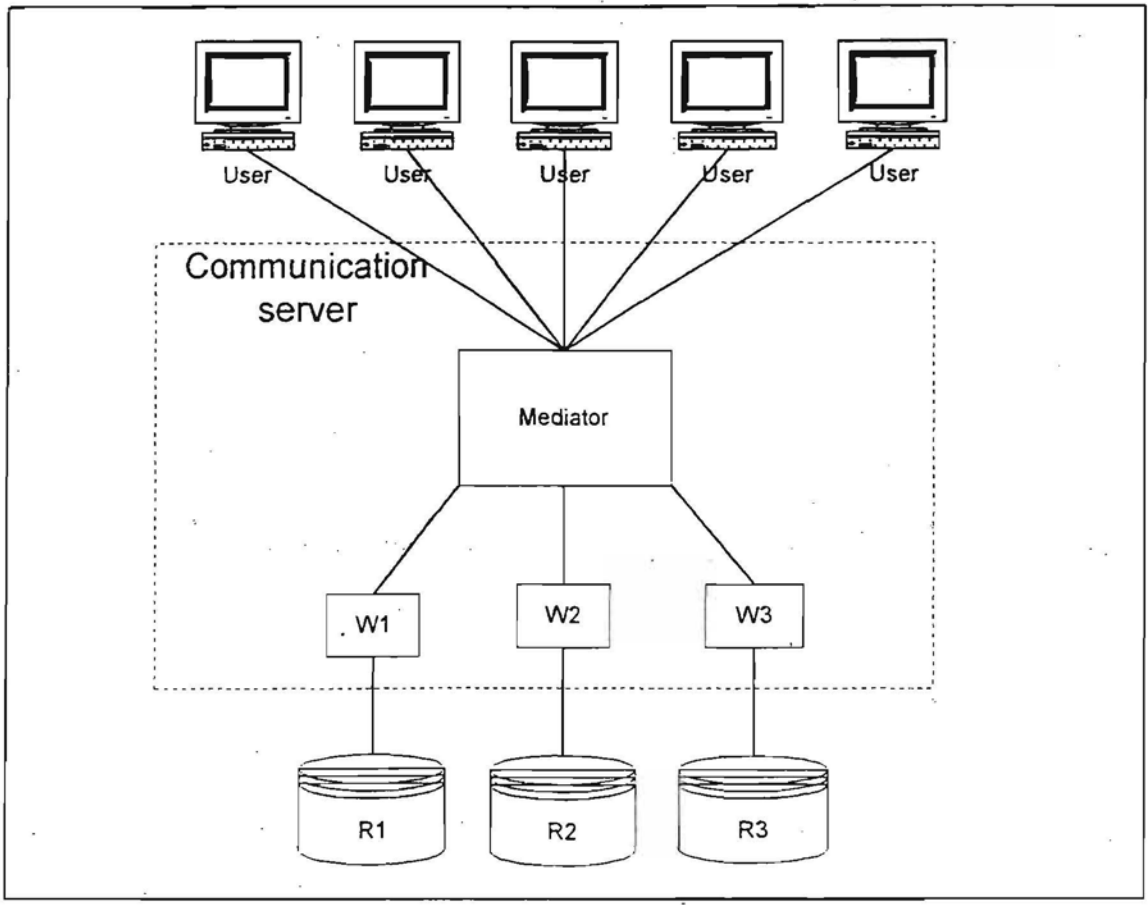


Figure 2. Conceptual shema of Communication Server

pages (data retrieving, combining from various registers and presentation) are available.

All user activities are logged in special journals. The system saves not only the type of activity and user who have done it, but also the request is stored. It is possible to track for any data object (person, for instance) all history - who asked what and what data objects and its attributes were displayed.

We can account costs for information consumers if the cost is defined for some information. Since we are logging any request with details then we can calculate overall accounts for any user and provider.

TECHNICAL SOLUTION

The main task for a Communications Server is to retrieve information from data sources. Let us see the rough view to the implementation principles (Figure 2). User asks the Mediator for information. The Mediator translates requests to set of internal small requests to data sources through wrappers. When the wrapper returns data, the Mediator forms the information presentation and sends the www page to user.

To retrieve information from data source, we have to create special small programs – data wrappers. This approach has the following advantages:

- It allows access data source via different protokols and methds – ODBC, OLE DB, SQL*Net, DCOM, etc.
- Data source usually is made to well suit for specific business tasks, it is not primary made for data access from other system (Communications Server). The access is limited, it is allowed execute some stored procedures to query data. Wrapper allows us to execute only authorized functions.
- Querying data source via functions allows us to have easy transfer real data from data source physical data model to our logical data model (stored in meta database) that is more understandable for the user.
- If the data source changes we need only correct the appropriate wrappers.

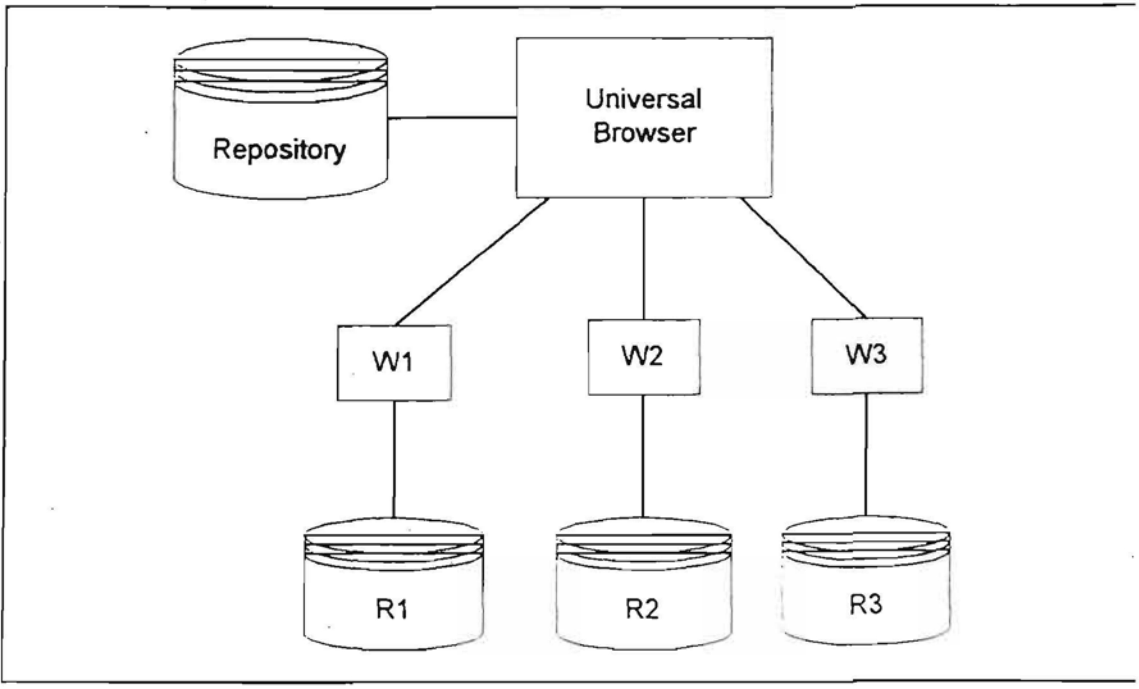
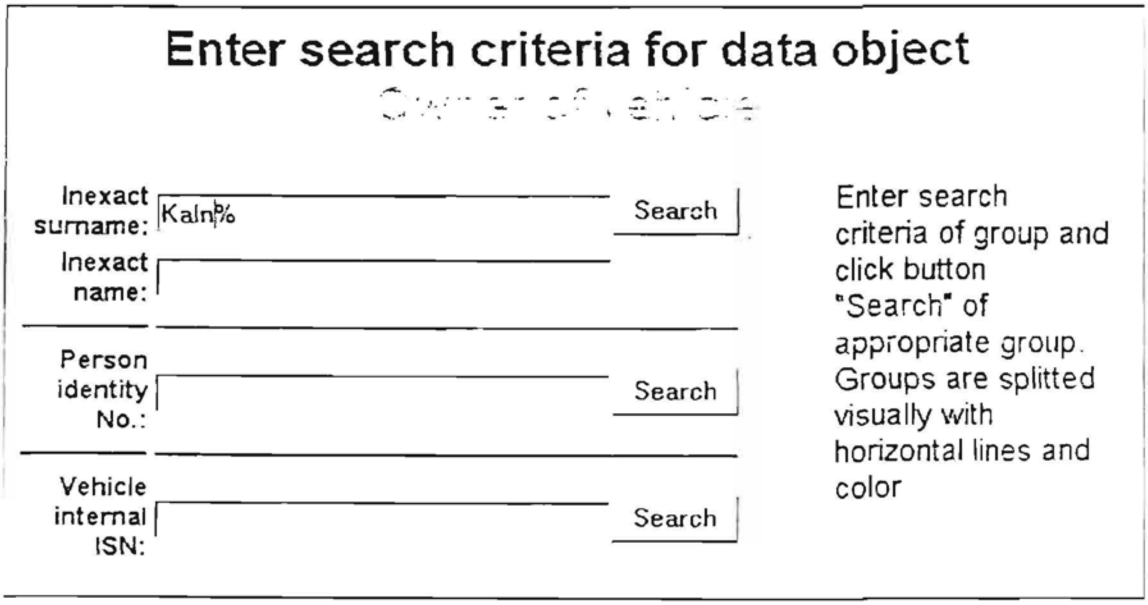


Figure 3. Universal browser

To communicate with user via Internet the special browser is designed that bases on a meta model of data sources. The browser takes the information stored in meta model, generate www pages to communicate with user. We can image browser as Driver and Repository (Figure 3).

Repository is database that stores information about data sources, data objects in sources and relations between them, functions that allows us query source, screen templates (www page structure) and other useful information.

Driver is special program that generate www pages to manage querying at high level and display information. The Driver can analyze relations between data sources and merge together all related information.



Enter search criteria for data object

Owner of vehicle

Inexact surname: Kaln%

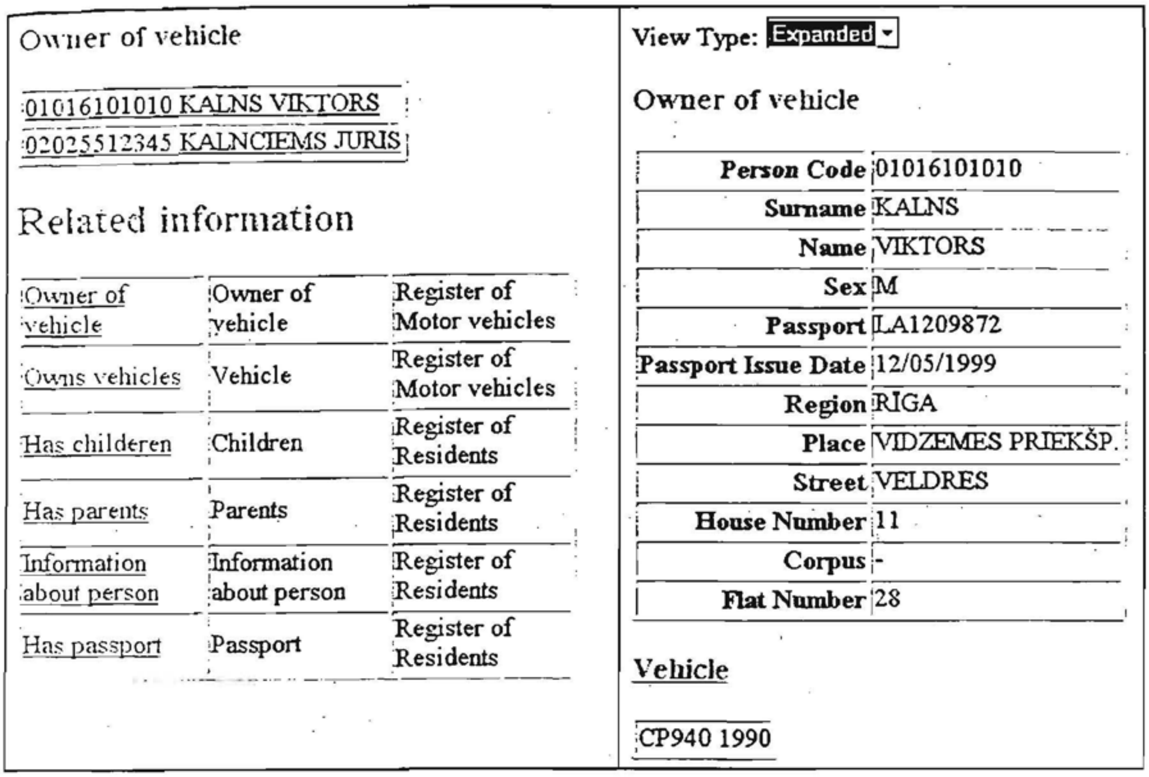
Inexact name:

Person identity No.:

Vehicle internal ISN:

Enter search criteria of group and click button "Search" of appropriate group. Groups are splitted visually with horizontal lines and color

Figure 4. Search criteria input window



Owner of vehicle

01016101010 KALNS VIKTORS
02025512345 KALNCIEMS JURIS

Related information

Owner of vehicle	Owner of vehicle	Register of Motor vehicles
Owns vehicles	Vehicle	Register of Motor vehicles
Has childeren	Children	Register of Residents
Has parents	Parents	Register of Residents
Information about person	Information about person	Register of Residents
Has passport	Passport	Register of Residents

View Type: Expanded

Owner of vehicle

Person Code	01016101010
Surname	KALNS
Name	VIKTORS
Sex	M
Passport	LA1209872
Passport Issue Date	12/05/1999
Region	RĪGA
Place	VIDZEMES PRIEKŠP.
Street	VELDRES
House Number	11
Corpus	-
Flat Number	28

Vehicle

CP940 1990

Figure 5. Information about car owners

DATA SEARCHING AND BROWSING SCENARIO

Let us look at small example how the Communications Server works from end-user point of view. First step is to choose from which register and which data object information will be quered (Figure 1). Then system asks search criteria for the choosen object (Figure 4).

User fills in search criteria and pushes button 'Search', system searches in appropriate the register for necessary information and results are showed (Figure 5).

From this screen user can easily get related information from other registers, for example, if user wants Information about Person from Register of Residents, user needs only to click on appropriate link and appropriate information are showed (Figure 6).

CONCLUSIONS AND FURTHER DIRECTIONS

The prototype of Communications Server was made in the middle of 1999 [7]. 4 registers (with test data) were connected for testing purposes. 2 of them use Oracle as DBMS and 2 others use Microsoft SQL Server. The prototype has shown the effectiveness of designed approach. The prototype of the system was much more powerful, than we expected and can be used as the real system. At present additional improvements is made and the first version of the real system is developed. This version is introduced in real exploitation now.

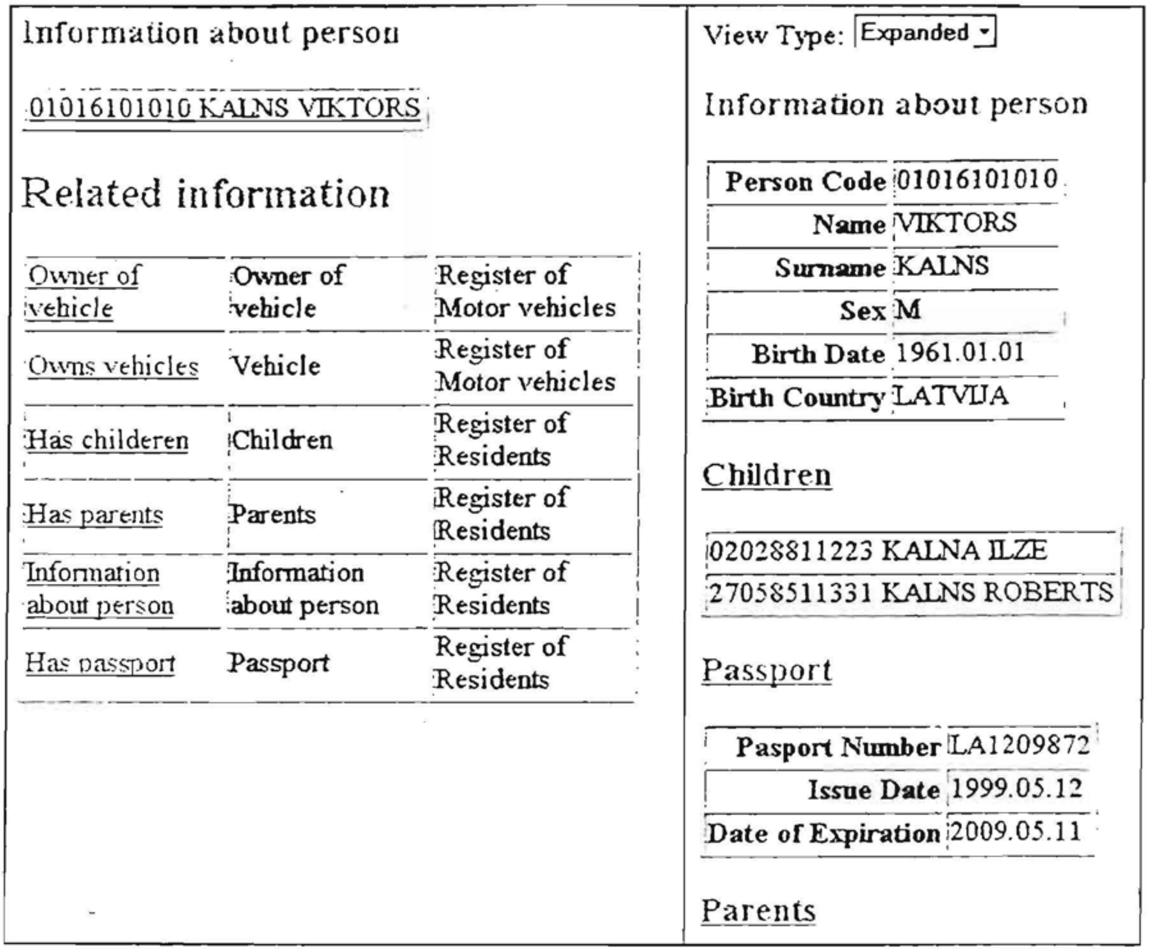
Future direction of our work is to develop a query processor that can take as input SQL-like query and return as output the result queried from multiple data source.

Other direction that already is partially developed – to make Communications Server available not only from WWW browsers, but also from custom programs using XML to query data and return answers.

REFERENCES

[1] Arnicans G, Bicevskis J, Karnitis G, "The Concept of Setting Up a Communications Server", in Abstracts of Papers of 3rd International Conference "Information Technologies and Telecommunications in the Baltic States", pp. 48-57, 1999

[2] Tomasic A, Amouroux R, Bonnet P, Kapitskaia O, Naacke H, and Raschid L, "The distributed information search component (disco) and the World Wide Web" in Proceedings of ACM SIGMOD International



Information about person

01016101010 KALNS VIKTORS

Related information

Owner of vehicle	Owner of vehicle	Register of Motor vehicles
Owns vehicles	Vehicle	Register of Motor vehicles
Has childeren	Children	Register of Residents
Has parents	Parents	Register of Residents
Information about person	Information about person	Register of Residents
Has passport	Passport	Register of Residents

View Type: Expanded ▾

Information about person

Person Code	01016101010
Name	VIKTORS
Surname	KALNS
Sex	M
Birth Date	1961.01.01
Birth Country	LATVIJA

Children

02028811223 KALNA ILZE
27058511331 KALNS ROBERTS

Passport

Pasport Number	LA1209872
Issue Date	1999.05.12
Date of Expiration	2009.05.11

Parents

Figure 6. Information about person

Conference on Management of Data, Tuscon, Arizona, 1997, Prototype Demonstration.

[3] Haas L. M, Miller R. J, Niswonger B, Tork Roth M, Schwarz P. M, Wimmers E. L, "Transforming Heterogeneous Data with Database Middleware: Beyond Integration", Data Engineering Bulletin 1999

[4] Hammer J, Garcia-Molina H, Ireland K, Papakonstantinou Y, Ullman J, Widom J, "Information translation, mediation, and Mosaic-based browsing in the TSIMMIS system", in Proceedings of ACM SIGMOD International Conference on Management of Data, 1995, Project Demonstration.

[5] Arnicans G, "Application generation for the simple database browser based on the ER diagram", Proceedings of the Third International Baltic Workshop, pp.198-209, 1998.

[6] "The Baltic States Government Data Transmission Network: Conceptual and Methodological Considerations", Riga, 1998, 11 pp.

[7] www.mega.lv

Contact information:
Datorikas Institūts
Raiņa bulv. 29-220
Rīga, LV-1050
Latvia
Tel.: +371 7503383
Fax: +371 7503531
E-mail:girts@di.lv

Vilnius Gediminas Technical University
Institute of Mathematics and Informatics
Lithuanian Computer Society

DATABASES& INFORMATION SYSTEMS

PROCEEDINGS
OF THE

4th
IEEE
INTERNATIONAL
BALTIC
WORKSHOP

Edited by
Albertas ČAPLINSKAS

Vol. 1

Vilnius
Lithuania
May 1-5
2000

Vilnius "Technika" 2000

8. References

[1] Burdett, D. Internet Open Trading Protocol Version 0.9.9. The Open Trading Protocol Consortium, 1998.

[2] Christoffel, M. Pulkowski, S., Schmitt, B., Lockemann, P. Electronic Commerce: The roadmap for university libraries and their members to survive in the infonnation jungle. *ACM Sigmod Record*, 27(4), 1998, pp. 68-73.

[3] Christoffel, M. A Trader for Services in a Scientific Literature Market. In *Proceedings of the 2nd International Workshop on Engineering Federated Information Systems (EFIS'99)*, Kühlungsborn, 1999, pp. 123-130.

[4] Hewlett Packard. *E-Speak - the platform for E-services*. http://www,e-speak.hp.com.

[5] IBM. *DB2 Digital Library*. http://www-4.ibm.com/software/is/dig-lib/about.html .

[6] JavaSoft. Java Remote Method Invocation Specification. Technical Report, Sun Microsystems, 1997. http://www.javasoft.com/products/jdk/1.1/docs/guide/rmi/.

[7] Karlsruher Virtueller Katalog. http://www.ubka.uni-karlsruhe.de/kvk.html.

[8] MeDoc- The Online Computer Science Library. http://medoc.informatik.tu-muenchen.de/english/medoc.html.

[9] Microsoft. DCOM Technical Overview. Technical Report, Microsoft Corporation, Redmond, 1996.

[10] Object Management Group. CORBA 2.0/IIOP Specification. Technical Report PTC/96-03-04, Framingham Corporate Center, Framingham (MA), USA, 1996.

[11] Pulkowski, S. Making Information Sources Available for a New Market in an Electronic Commerce Environment. In *Proceedings of the International Conference on Management of Information and Communication Technology (MICT'99)*, Copenhagen, 1999.

[12] Pulkowski, S.: Intelligent Wrapping of Information Sources: Getting Ready for the Electronic Market. In *Proceedings of the 10th VALA Conference on Technologies for the Hybrid Library*, Melbourne, 2000.

[13] Rachlevsky-Reich, B., Ben-Shaul, I. et. al. GEM: A Global Electronic Market System. In *Information Systems*, 24(6), 1999, pp. 495-518.

[14] Schmitt, B., Schmidt, A. METALICA: An Enhanced Meta Search Engine for Literature Catalogs. In *Proceedings of the 2nd Asian Digital Library Conference (ADL'99)*, Taipei, 1999.

[15] Stanford Digital Library Project. http://www-diglib.stanford.edu/diglib/.

[16] Stevens, W.R. *TCP/IP Illustrated, Volume 1, The Protocols*. Addison Wesley, Reading, 1995.

[17] Wang Baldonado, W., Winogred, T. Hi-Cites: dynamically created citations with active highlighting. In *Proceedings of the International Conference on Human factors in computing systems (CHI '98)*, Los Angeles, 1998, pp. 408-415.

[18] World Wide Web Consortium. Extensible Markup Language Recommendation. 1998. http://www.w3.org/TR/1998/REC-xml-19980210.

Heterogeneous Database Browsing in WWW Based on Meta Model of Data Sources

Guntis Arnicans, Girts Karnitis

University of Latvia
Faculty of Physics and Mathematics
Rainis Blvd. 19, Riga LV-1459, Latvia
garnican@lanet.lv, girts@di.lv

Abstract

This paper describes a development principle and technique for a simple universal multiple database browser. The browser operates by getting information from metamodel of data sources and actual data from legacy data sources. Every element such as entity, field, relation is mapped to some component of HTML page with appropriate structure and layout. Many templates of information layouts can be created allowing to dynamically change HTML page to acceptable user interface. The wrappers are used to provide browser with actual data and to act as mediators between data sources and browser. This approach allows to quickly describing new data sources, creating wrappers, making modifications later and managing data browsing in a simple unified style. The browser architecture is flexible enough to incorporate data sources with a variety of data models and query capabilities by various protocols. It is possible to select logically tied information from all available legacy data sources.

Keywords: Web-based information system, distributed information system, metamodels, database browsing.

1. Introduction

Data retrieval from different autonomous sources has become a hot topic during the last years. For instance, there are such data sources as enterprise register, register of pledges, register of state orders. When some state institution wants to order something from private business, civil servants are interested to know whether applicants are registered, whether they have pledges and what is their financial situation. Civil servants need information system that can collect related information from different Data Sources (DS) and show it.

We have found some such systems [2], [3], [4] that allow to do data querying from different data sources. All those systems are very complex, with their own query processor, but without universal user end. We decided to make a simple Universal Browser (UB) that acts on DS model during development of Megasystem and Communication server [5], [6].

Main ideas of the UB are described in [1], where the idea of database browsing based on the ER model is described. Our approach is a modified UB, that can browse multiple DS, which can be made in different technologies and with limited access rights and possibilities. Access to the DS is made via wrappers.

2. Repository of conceptual data models of data sources

Repository is a database that contains information about data sources (DS) and the links between them – the specific ER model. Repository also contains a description of functions that can be executed by DS.

2.1 Metamodel of repository

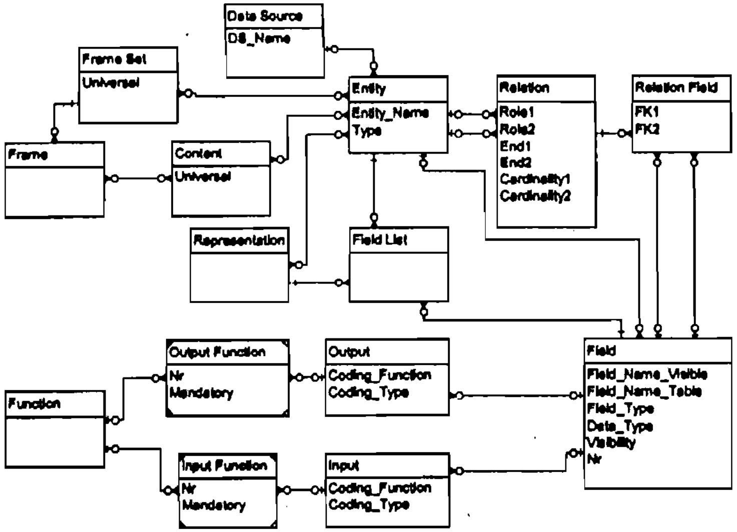


Figure 1. Metamodel of repository

Figure 1 shows an ER model of Universal Browser's (UB) repository. There are different parts in this model that are used for different purposes:

- Entities Data Source, Entity, Field, Relation, Relation Field contain DS models and information about entities and relations.
- Entities Function, Input, Input Function, Output, Output Function contain information about functions that query information from DS and input and output fields of these functions.

- Entities Representation and Field List contain information about visual representations for each entity i.e. what fields in what order have to be shown. For instance, let us take the entity Citizen that contains information about a person. In *short* representation fields PK, Name, Surname are visible, but in *long* representation fields PK, Name, Surname, Address have to be shown.
- Entities Frame Set, Frame and Content contain information about visual representation.

2.2 Conceptual model of data source

DS is a real existing legacy data source that exposes its data to other systems. Any DS can be made with different technologies, and expose its data in different ways. Any DS has some functions that can be executed to get information from DS. It is not necessary for the user to know technical details of DS to get information from it. The user needs a simple and understandable logical information representation that is related to the objects from the real world.

For example, information about cars can be stored in many tables in the real system. We are interested in conceptual data model, without technical details. It means a car can be represented with one entity in the conceptual model.

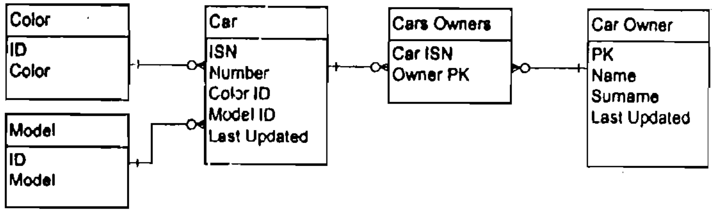


Figure 2. Example of physical data model

There can be such technological fields in the real database, which are necessary for the real system functioning, but they are not interesting for user and are not shown in the conceptual model.

There are two types of fields in the conceptual data model of DS:

- Fields that can be queried with some function,
- Fields from which we cannot query information. It means there are no functions where any of those fields are outputs. Usually these fields are not showed to the user, and they are used as input fields for some function. These fields are also used to link different entities.

There are links between DS entities, which means that, if you know information from one entity, you can get information from the other entity. There are links between entities, if such functions exist, which can query information from DS, using as an input information from other entity. For example, if you know some information about the person (especially person's PK (person code)), you can query the information about the person's passport. It means there is a link from person to passport. This function returns the passport number and the issue date. On the other hand, if you know the passport number, you can't get the passport's owner PK, because there is no function that returns this information. It also means that there can't be a link from passport to citizen.

2.3 Logical links between the data sources

There are entities of different types used to link together infonnation from different DS. These entities are used as base class of DS entities and do not belong to any DS. For instance, the entity *Person with PK* is such a base class. This base class has only one field PK. This field is primary key for similar objects that concern person for most of DS. If you know the PK you can get the information related to the person information from the appropriate DS. For instance, ***Person with PK*** links together information from the entities *Citizen*, *Passport*, *Car* and *Car Owner* (**Figure 3**).

2.4 An example of repository

Two DS and one base class are given in **Figure 3**.

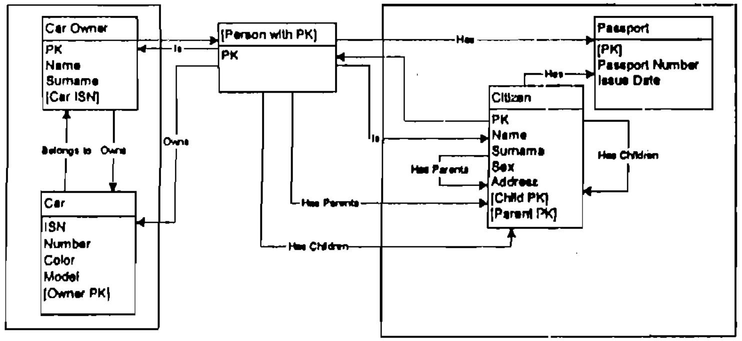


Figure 3. Example of conceptual model of data sources

Fields in square brackets are invisible fields used for search purposes only. Solid line with arrows means if you know information from the entity that is a starting point of the arrow, you can get the related information from the entity that is at the opposite end of the arrow. Interrupted line shows the relation between *normal entity* and *base class entity*. The values of arrows are shown in **Table 1**.

Table 1. Description of Relations

End1	PK1	End2	PK2	Relation name
Citizen	PK	Citizen	Child PK	Has Parents
Citizen	PK	Citizen	Parent PK	Has Children
Citizen	PK	Passport	PK	Has
Citizen	PK	Person With PK	PK	
Car Owner	PK	Car	Owner PK	Owns
Car Owner	PK	Person With PK	PK	
Car	ISN	Car Owner	Car ISN	Belongs To
Person With PK	PK	Car Owner	PK	IS
Person With PK	PK	Car	PK	Owns
Person With PK	PK	Passport	PK	Has
Person With PK	PK	Citizen	PK	Is
Person With PK	PK	Citizen	Child PK	Has Parents
Person With PK	PK	Citizen	Parent PK	Has Children

3. Browsing principles

General idea for dynamic browsing of various data sources is to generate Web pages with predefined information layout and functionality, get data from data sources and put them into page.

A web page consists of a set of frames (Frame) – FrameSet. The Frameset has a prefixed count of Frames, its layout and sizes. We can define as many as we need different FrameSets to organize and display information for the user. The FrameSet is a view to related data from one or many data sources. One of the Frames is the main Frame. The information in any other Frame is logically connected with data in the main Frame. The Frames can contain controls to manage the content in the other Frame.

The layout of the Frame is defined by rule, lets call it Content. Theoretically the Content is a formula or function: ***Content(frameEntity, filterExpr)*** where ***frameEntity*** is any entity from the metamodel of data sources and ***filterExpr*** is logical expression that filters data from appropriate data source. The Content defines: 1) what is the structure and principles of layout, 2) what data from metamodel and from actual legacy data sources are required to display information, 3) what actual instances of the defined entity are retrieved, 5) what controls are used to manage the content of the other Frame or to open the other FrameSet and 5) what related entities are involved from the same or any other data source. If we have various predefined Contents, then we can dynamically apply any Content to the Frame and get another data presentation for the same ***frameEntity*** and ***filterExpr***.

4. Defining the Content of Frame

Let us assume that Content is the function ***Content(frameEntity, filterExpr)***. Let us determine the means how we can define Content. We introduce the following data types:

- **entity** - determines the entity from the metamodel,
- **field** - determines the field of the entity from the metamodel,
- **relation** - determines the relation for two entities from the metamodel,
- **record** - determines the actual data from the data source for one fixed instance of the entity,
- **value** - determines the actual data of the field for one fixed instance of the entity,
- **string** - determines the character string,
- **list** - determines the list of elements with any other allowed data type, we denote such types by the element type followed by postfix "List",
- **updateAction** - determines the action that updates Frame
- **navigateAction** - determines the action that navigates browsing to another FrameSet
- **sObject** - determines the HTML object that contains string to display,
- **aObject** - determines the HTML object with assigned some action to perform,
- **fObject** - determines the HTML object that is formatted for displaying,
- **frame** - determines the Frame,
- **frameSet** - determines the FrameSet,
- **view** - determines the list of fields that must be displayed.

Let us rewrite the Content as a function ***Content(entity, expr(entity))***.

Let us introduce several additional functions to work with the metamodel and data sources, and to format HTML page.

<u>Functions to work with the metamodel</u>:

1. ***SourceName(entity)→string*** – returns the source name the entity belongs to
2. ***EntityName(entity)→string*** – returns the entity name
3. ***RelationList(entity)→relationList*** – returns all direct relations from the given entity to another entity (including itself) from the same data source
4. ***MetaRelationList(entity)→relationList*** – returns all indirect relations from the given entity to another entity from all available data sources
5. ***FieldList(entity, view)→fieldList*** – returns the list of all the fields of the entity
6. ***RelationName(relation)→string*** – returns the name (role) of the relation
7. ***FieldName(field)→string*** – returns the name of the field
8. ***RelationEntity(relation))→entity*** – returns the entity at the opposite end of relation

<u>Functions to work with data sources through wrappers</u>:

1. ***RecordList(entity, expr(entity))→recordList*** – returns the list of instances (records) of the entity according to the given filtering expression
2. ***ValueList(record, view)→valueList*** – returns the list values of the given entity instance (record)
3. ***Value(value)→string*** – returns the field value as character string

<u>Functions to work with the list</u>:

1. ***List(element_1, element_2, ..., element_i)→list_1*** – returns the list of given elements and the list type *list_1* is appropriate to the element type
2. ***IterateList(n%list_1, function(n%))→list_2*** – returns the list *list_2* that has as elements the results applying the given function. The function is executed with each parameter *n%* that is taken from the list *list_1* denoted by the identifier *n%* (n is any unique integer) and the list type *list_2* is appropriate to the function return type
3. ***Concatenate(list_1, list_2)→list_3*** – returns the concatenation of two lists with the same element type.

<u>Functions to format HTML page</u>:

1. ***SO(string)→sObject*** – creates sObject from the character string
2. ***StringListObject(stringList, separatorString)→sObject*** – creates sObject from the list of character strings separated by *separatorString*
3. ***Update(frame, entity, expr(entity), content)→updateAction*** – activates information update into the frame with the given entity, filter expression and layout
4. ***Clear(frame)→updateAction*** – clears the given frame
5. ***Navigate(frameSet, entity, expr(entity), content)→navigateAction*** – navigates to another FrameSet and update main Frame with the given entity, filter expression and layout
6. ***Link(sObject, navigateAction, updateActionList)→aObject*** – converts sObject into aObject and assign the navigation action and set of update actions to it. Any of action parameters may be empty.
7. ***AO(sObject)→aObject*** – converts sObject into aObject with empty action
8. ***FO(aObject)→fObject*** – converts aObject into fObject without any special formating
9. ***HorizontalTable(aObjectListList)→fObject*** – creates fObject from the list of lists, this frame object is displayed as table, and internal lists are placed in rows
10. ***VerticalTable(aObjectListList)→fObject*** – creates fObject from the list of lists, this frame object is displayed as table, and internal lists are placed in columns

11. *ListBox(aObjectList)→fObject* – creates fObject from the list, this frame object is displayed as listbox

12. *Horizontal(fObjectList)→fObject* – creates new fObject by arranging the given list horizontally

13. *Vertical(fObjectList)→fObject* – creates new fObject by arranging the given list vertically.

Only frame objects with the type fObject may be displayed in the Web page.

5. Data wrappers

Function *RecordList* must be implemented to get information from DS. The technology we use is simple, but effective. UB gets information from DS via Wrappers. This approach has the following advantages:

- It allows to access DS via different protocols and methods - ODBC, OLE DB, SQL*Net, DCOM.
- DS usually are made well suited for specific business tasks. DS are not primary made for data access from UB. The access to DS data usually is limited, it is allowed to execute some stored procedures to query data. Wrapper allows us to execute only authorized functions.
- Querying DS via functions allows us to have easy transfer real data from DS physical data model to logical data model that is more understandable for the user.

Information about functions is stored in the UB meta database: defined input and output fields for each function. Each input field may be mandatory or optional.

During development of the prototype, we discovered some rules for function implementation and developing conceptual model of DS.

- First rule - it is desirable to have input and output fields from one entity. It simplifies development of DS model and wrappers.
- Second rule - two approaches possible for making DS model and functions. One approach is that we already have functions, and we make conceptual data model of DS using the first rule. In case DS is a system we maintain and own, it is often possible to make functions according to conceptual data model of DS. In such a case we make conceptual data model of DS at first and then we make data access functions according to conceptual data model and the first rule. It is helpful to make two types of functions:

1. The function gets information identifying the object from DS by some search criteria. For example, get person's PK by its name and surname (might be partial). The answer usually is a list of person's identifying information according to search criteria.
2. The function gets information about one object from one entity by its identifier. An example - get all information about the citizen by its PK.

For instance, we have two functions for the entity Citizen:

1. Input data – Name, Surname (might be partial). Output data – PK, Name, Surname (full).
2. Input data – PK. Output data – PK, Name, Surname, Address.

There are also 2 functions to get information about the citizen's parents and children:

3. Input data – Parent PK. Output data – Children PK, Name, Surname.
4. Input data – Child PK. Output data – Parents PK, Name, Surname.

There is a procedure that implements the function *RecordList*. This procedure gets the entity and filter expression as input and returns data from DS as output. In our implementation this procedure gets information from the meta database about functions that can be executed over entity from which we need information. In our implementation the filter expression is fields and corresponding values for these fields, e.g. PK="123456-111111". There is "brute force" algorithm that finds functions we can execute e.g. those are functions that have enough input data from the filter expression to be executed, executes these functions and returns result. There can be, of course, other implementations.

DS data access via wrappers allows to connect new DS to our system easily and quickly. We have to write a new wrapper and add information about new DS to the meta database. With some experience the writing of wrappers is easy and fast process, and there is no need to make any modification in DS.

6. Templates for Web page structure and functionality

The design of FrameSet and Frames is based on template principle. With some experience the new FrameSets and Frames can be developed quickly. The design has two main steps – FrameSet structure planning and creating formulas for Frame Contents. We give some templates and ideas how the Web pages can be designed. The above given functions are used.

6.1 Simple entity instance presentation in table

The first column contains field names and the second – field values

```
A(entity, record) = VerticalTable(List(A1, A2))
A1 = IterateList(1%FieldList(entity, view), AO(SO(FieldName(1%))))
A2 = IterateList(2%ValueList(record, view), AO(SO(Value(2%))))
```

PK	12121211111
Name	Andris
Surname	Kalns
Sex	M
Address	Riga, Liepu 1-12, LV-1000

Figure 4. Example of entity instance presentation

6.2 Entity instance presentation as text

Instance field values are concatenated according to select *view*.

```
B(record) = FO(AO(SO(StringListObject(B1, " "))))
B1 = IterateList(3%ValueList(record, view), Value(3%))
```

12121211111 Andris Kalns M Riga, Liepu 1-12, LV-1000

6.3 Entity relations presentation in vertical list

Each relation is represented as relation name concatenated with entity name at the opposite relation end.

```
C(entity) = Vertical(IterateList(4%RelationList(entity), C1))
C1 = Horizontal(List(C2, FO(AO(SO(" "))), C3))
C2 = FO(AO(SO(RelationName(4%))))
C3 = FO(AO(SO(EntityName(RelationEntity(4%)))))
```

Has Passport
Has Parents Citizen
Has Children Citizen

Figure 5. Example of relations presentation

6.4 All relation presentation in table

The data about all relations (relation name, entity name and data source) are placed in table with headings.

```
D(entity, expr(entity))=HorizontalTable(Concatenate(D1,D2))
D1 = AO(StringListObject("Relation", "Entity name", "Data source"))
D2 = IterateList(5%MetaRelationList(entity),List(D3, D4, D5))
D3 = AO(SO(RelationName(5%)))
D4 = AO(SO(EntityName(RelationEntity(5%))))
D5 = AO(SO(SourceName(RelationEntity(5%))))
```

Relation	Entity name	Data source
Is	Citizen	Register of Residents
Has	Passport	Register of Residents
Has Parents	Citizen	Register of Residents
Has Children	Citizen	Register of Residents
Is	Car Owner	Register of Motor vehicles
Owns	Car	Register of Motor vehicles

Figure 6. Example of relation presentation

6.5 An example of FrameSet

Let us look how a FrameSet can be built. Let us assume FrameSet *FRS_1* with 4 Frames – *FR_1, FR_2, FR_3, FR_4*. FR_1 is used to list instances of entity, FR_2 – to show details of fixed instance in FR_1, FR_3 – to list all relations to other entities in all data sources, FR_4 – to show details of another related entity instances for FR_2 or FR_4. See Figure 7.

At first let us create three presentations or Contents (E, F, G) for viewing entities. We use formulas created before in this paper.

- Content formula E() for Frame FR_4 (from FR_4 we can update all Frames in FRS_1)

```
E(entity, expr(entity)) = Vertical(E1, E5)
E1 = Horizontal(List(FO(E2), FO(AO(SO(" "))), FO(AO(SO(SourceName(entity))))))
E2 = Link(SO(EntityName(entity)), E3, E4)
E3 = Navigate("FRS_1", entity, expr(entity), "" )
E4 = List(Clear("FR_2"), Update("FR_3", entity, expr(entity), ""), Clear(FR_4))
E5 = Vertical(IteateList(6%RecordList(entity, expr(entity)), A(entity, 6%)))
```

- Content formula F() and G() for Frame FR_2 (from FR_2 we can update this frame or update FR_4)

```
F(entity, expr(entity)) = Vertical(H, FO(AO(SO(" "))), E5)

G(entity, expr(entity)) = Vertical(H, FO(AO(SO(" "))), Vertical(E5, G1))
G1 = C(entity), where C3 is substitute with G2 in all places (we have added the action)
G2 = FO(Link(SO(EntityName(RelationEntity(4%))), NULL, G3))
G3 = List(Update("FR_4", RelationEntity(4%), expr(RelationEntity(4%)), "E"))

H = ListBox(List(Link("Presentation F", NULL, H1), Link("Presentation G", NULL, H2)))
H1 = Update("FR_2", entity, expr(entity), "F")
H2 = Update("FR_2", entity, expr(entity), "G")
```

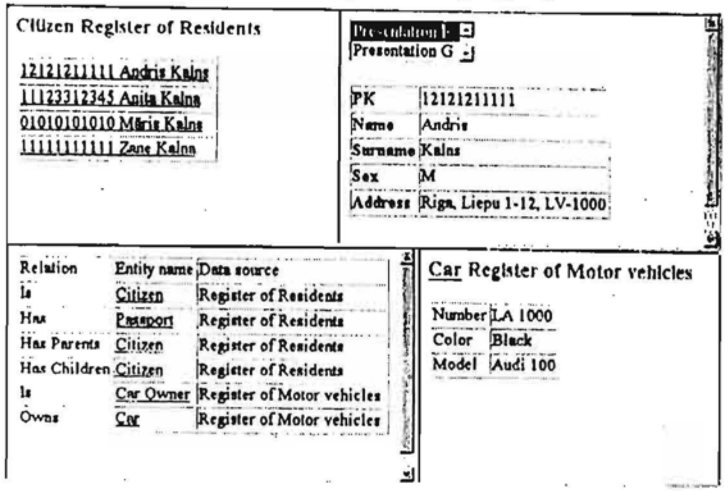
- Content formula I() for Frame FR_1 (from FR_1 we can update FR_2, FR3, FR_4)

```
I(entity, expr(entity)) = Vertical(I1, I2)
I1 = Horizontal(List(FO(EntityName(entity)), FO(AO(SO(" "))), FO(AO(SO(SourceName(entity))))))
I2 = HorizontalTable(IterateList(7%RecordList,Link(B(7%), NULL, I3))
I3 = List(Update("FR_2", entity, expr(entity) and expr(7%), "F"),
        Update("FR_3", entity, expr(entity) and expr(7%), ""), Clear("FR_4"))
```

- Content formula J() for Frame FR_3 (from FR_3 we can update FR_4)

```
J(entity, expr(entity)) = D(entity, expr(entity)), where D4 is substitute with J1 in all places (we have add the action)
```

```
J1 = Link(SO(EntityName(RelationEntity(5%))), NULL, J2)
J2 = List(Update("FR_4", RelationEntity(5%), expr(RelationEntity(5%)), "E"))
```



Citizen Register of Residents

12121211111 Andris Kalns
11123312345 Anita Kalna
01010101010 Māris Kalns
11111111111 Zane Kalna

PK	12121211111
Name	Andris
Surname	Kalns
Sex	M
Address	Riga, Liepu 1-12, LV-1000

Relation	Entity name	Data source
Is	Citizen	Register of Residents
Has	Passport	Register of Residents
Has Parents	Citizen	Register of Residents
Has Children	Citizen	Register of Residents
Is	Car Owner	Register of Motor vehicles
Owns	Car	Register of Motor vehicles

Car Register of Motor vehicles

Number	LA 1000
Color	Black
Model	Audi 100

Figure 7. Example of WWW page

7. Conclusions and future directions

The prototype of the UB is made during developing Megasystem and Communication Server. Four registers test databases are connected to the UB for testing purposes. Two of them use Oracle as DBMS, other two use Microsoft SQL Server.

The UB prototype shows the effectiveness of our approach and is being initiated as first version of the real system.

There are many aspects that are very important in real life application, but not covered in this article – security, user authorization, logging, query cost calculation. All these features are incorporated in the UB.

The UB is useful in many large organizations having many autonomous data sources as a browser for these systems with integrated view.

Future direction of our work is to develop a query processor that can take as input SQL-like query and return as output the result queried from multiple data sources. Other directions of future work – to make CS available not only from WWW browsers, but also from custom programs using XML to query data and return answers.

8. References

[1] Arnicans G, "Application generation for the simple database browser based on the ER diagram", Proceedings of the Third International Baltic Workshop, pp.198-209, 1998.

[2] Tomasic A, Amouroux R, Bonnet P, Kapitskaia O, Naacke H, and Raschid L, "The distributed information search component (disco) and the World Wide Web" in Proceedings of ACM SIGMOD International Conference on Management of Data, Tuscon, Arizona, 1997, Prototype Demonstration.

[3] Haas L. M, Miller R. J, Niswonger B, Tork Roth M, Schwarz P. M, Wimmers E. L, "Transforming Heterogeneous Data with Database Middleware: Beyond Integration", Data Engineering Bulletin 1999.

[4] Hammer J, GarciaMolina H, Ireland K, Papakonstantinou Y, Ullman J, Widom J, "Information translation, mediation, and Mosaicbased browsing in the TSIMMIS system", in Proceedings of ACM SIGMOD International Conference on Management of Data, 1995, Project Demonstration.

[5] Arnicans G, Bicevskis J, Karnitis G, "The Concept of Setting Up a Communications Server", in Abstracts of Papers of 3rd International Conference "Information Technologies and Telecommunications in the Baltic States", pp. 48-57, 1999.

[6] www.mega.lv

Databases and Information Systems

Fourth International Baltic Workshop, Baltic DB&IS 2000 Vilnius, Lithuania, May 1–5, 2000 Selected Papers

Edited by

Janis Barzdins

Institute of Mathematics and Computer Science,
University of Latvia, Riga

and

Albertas Caplinskas

Institute of Mathematics and Informatics,
Vilnius

KLUWER ACADEMIC PUBLISHERS
DORDRECHT / BOSTON / LONDON

Heterogeneous Database Browsing in WWW Based on Meta Model of Data Sources

Guntis Arnicans, Girts Karnitis

University of Latvia
Faculty of Physics and Mathematics
Raina Blvd. 19, Riga LV-1586, Latvia
garnican@lanet.lv, girts@di.lv

Abstract

This paper describes a development principle and technique for a simple universal multiple database browser. The browser operates by getting information from metamodel of data sources and actual data from legacy data sources. Every element such as entity, field, and relation is mapped to some component of HTML page with appropriate structure and layout. Many templates of information layouts can be created allowing to dynamical changing of HTML page to acceptable user interface. The wrappers are used to provide browser with actual data and to act as mediators between data sources and browser. This approach allows to quickly describing new data sources, creating wrappers, making modifications later and managing data browsing in a simple unified style. The browser architecture is flexible enough to incorporate data sources with a variety of data models and query capabilities by various protocols. It is possible to select logically tied information from all available legacy data sources.

Keywords: Web-based information system, distributed information system, metamodels, database browsing.

1. Introduction

Organisations, both governmental and business, have to manage large amount of information stored in some form of databases or files. One of the main problems to deal with information managing is the weak interoperability between various databases and information systems. Especially this problem is serious when we want organise collaboration between the information systems of various organisations.

In nowadays a significant fraction of new information systems or services bases on the Web solutions. Usually developers use Web applications to organise communications between data source and data consumer (user) but data sources sometimes remain the old ones from the current or previous information systems. This leads to the operation with very heterogeneous data. To deal with problems the metadata of the data sources (data structure, content, attributes, etc.) are used to describe the heterogeneous information models. This approach supports the creating of very dynamical systems and it is easy to maintain system in the rapidly changing world.

In this paper we describe some results achieved during the development of two projects - the Integrated State Significance Information System (Megasystem) and the Baltic States Government Data Transmission Network (Network) [2, 5]. The goal of these projects is to provide fundamental improvements in the exchange of telecommunications and data among the administrative institutions of the Baltic States. The principles described in this paper were used to build up the first implementation of *Communication server.* A Communication server is a set of software and

J. Barzdins and A. Caplinskas (eds.), Databases and Information Systems, 167–178.

computer equipment that allows a wide range of users to receive information from variety of sources (governmental registers, databases, information systems) through a single contact point. Among the other significant functions the Communication server fulfils a requests that involves several information sources, merges together information, allows users to learn where information is stored and what kind of information it is, and to receive information from various registers without any need for in-depth knowledge about the technical aspects of its storage.

Data retrieval from different autonomous sources has become a hot topic during the last years in the other countries and large enterprises also. There are many different approaches to deal with this task. For instance, the systems described in [3, 4, 6, 7] allow data querying from different Data Sources (DS). All those systems are very complex, with their own query processor, but without universal user end. The development of these systems consumes many resources (time, money, people).

Our first aim was to make a simple Universal Browser (UB) that acts on model of data sources and is very useful in practice (relative to consumed development resources). Main ideas of the UB are described in [1], where the idea of database browsing based on the ER model is described. Our approach is a modified UB that can browse multiple DS, which can be made in different technologies and with limited access rights and possibilities. Access to the DS is made via wrappers. Information retrieval bases on logical data models, information between different data model are tied via special logical data entities. The simple means are offered to obtain information and display it on WWW page – the set of functions that allows to create executable formulas.

2. Repository of Conceptual Data Models of Data Sources

Repository is a database that contains information about data sources (DS) and the links between them – the specific ER model. Repository also contains a description of functions that can be executed by DS.

2.1 METAMODEL OF REPOSITORY

Figure 1 shows an ER model of Universal Browser's (UB) repository. There are different parts in this model that are used for different purposes:

- Entities *Data Source, Entity, Field, Relation, Relation Field* contain DS models and information about entities and relations.
- Entities *Function, Input, Input Function, Output, Output Function* contain information about functions that query information from DS and input and output fields of these functions.
- Entities *Representation* and *Field List* contain information about visual representations for each entity i.e. what fields in what order have to be shown. For instance, let us take the entity *Citizen* that contains information about a person. In *short* representation fields *PK, Name, Surname* are visible, but in *long* representation fields *PK, Name, Surname, Address* have to be shown.
- Entities *Frame Set, Frame* and *Content* contain information about visual representation.

2.2 CONCEPTUAL MODEL OF DATA SOURCE

DS is a real existing legacy data source that exposes its data to other systems. Any DS can be made with different technologies, and expose its data in different ways. Any DS has some functions that can be executed to get information from DS. It is not necessary for the user to know technical details of DS to get information from it. The user needs a simple and understandable logical information representation that is related to the objects from the real world.

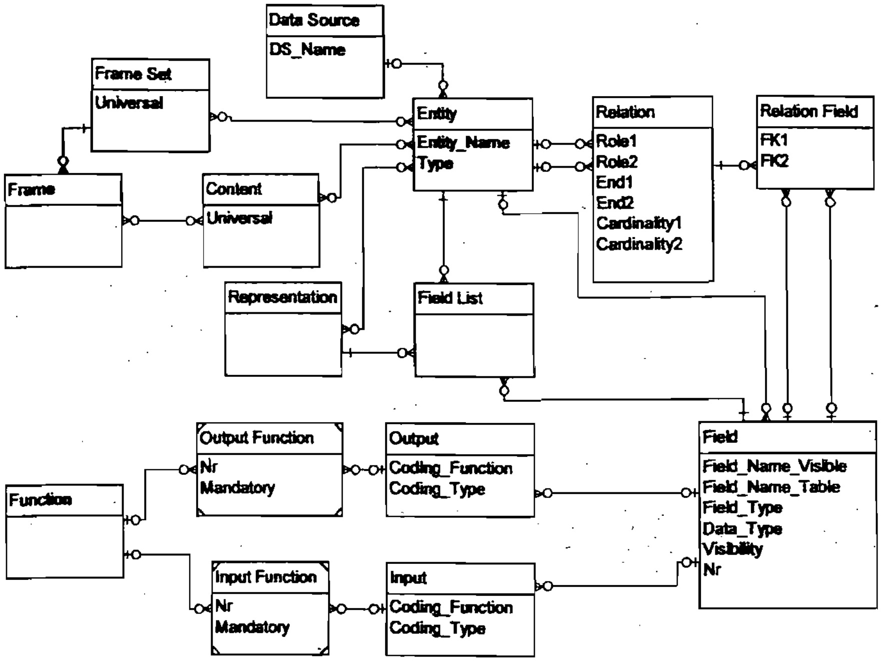


Figure 1. Metamodel of repository

For example, information about cars can be stored in many tables in the real system. We are interested in conceptual data model, without technical details. It means a car can be represented with one entity in the conceptual model.

There can be such technological fields in the real database, that are essential for the real system functioning, but they are not necessary for user and are not shown in the conceptual model.

There are two types of fields in the conceptual data model of DS:

- Fields that can be queried with some function,
- Fields from which we cannot query information. It means there are no functions where any of those fields are outputs. Usually these fields are not showed to the user, and they are used as input fields for some function. These fields are also used to link different entities.

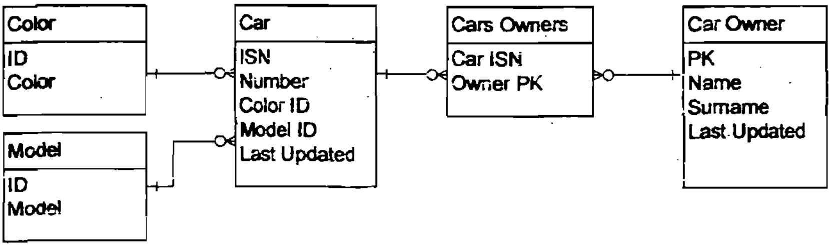


Figure 2. Example of physical data model

There are links between DS entities, which means that, if you know information from one entity, you can get information from the other entity. There are links between entities, if such functions exist, which can query information from DS, using as an input information from other entity. For example, if you know some information about the person (especially person's PK (person code)), you can query the information about the person's passport. It means there is a link from person to passport. This function returns the passport number and the issue date. On the other hand, if you know the passport number, you cannot get the passport's owner PK, because there is no function that returns this information. It also means that there cannot be a link from passport to citizen.

2.3 LOGICAL LINKS BETWEEN THE DATA SOURCES

There are entities of different types used to link together information from different DS. These entities are used as base class of DS entities and do not belong to any DS. For instance, the entity *Person with PK* is such a base class. This base class has only one field PK. This field is primary key for similar objects that concern person for most of DS. If you know the PK you can get the information related to the person information from the appropriate DS. For instance, *Person with PK* links together information from the entities *Citizen, Passport, Car* and *Car Owner* (Figure 3).

2.4 AN EXAMPLE OF REPOSITORY

Two Data Sources and one base class are given in Figure 3.

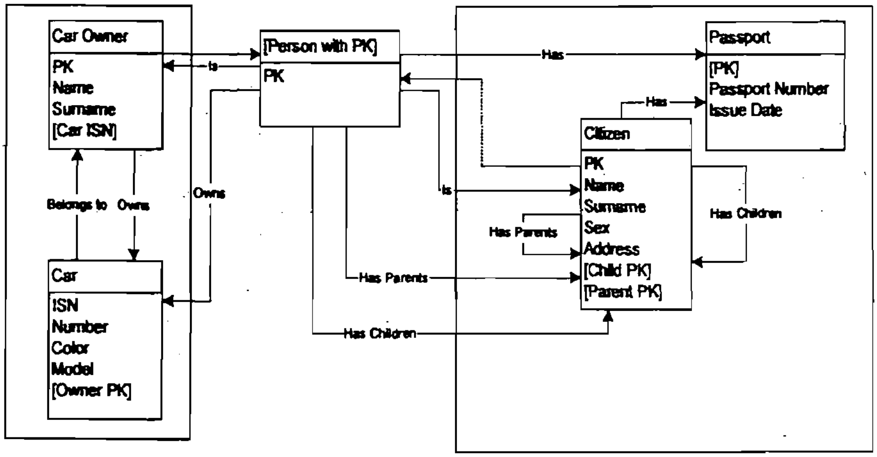


Figure 3. Example of conceptual model of data sources

Fields in square brackets are invisible fields used for search purposes only. Solid line with arrows means if you know information from the entity that is a starting point of the arrow, you can get the related information from the entity that is at the opposite end of the arrow. Interrupted line shows the relation between *normal entity* and *base class entity*. The values of arrows are shown in Table 1.

Table 1. Description of relations

End1	PK 1	End2	PK2	Relation name
Citizen	PK	Citizen	Child PK	Has Parents
Citizen	PK	Citizen	Parent PK	Has Children
Citizen	PK	Passport	PK	Has
Citizen	PK	Person With PK	PK	
Car Owner	PK	Car	Owner PK	Owns
Car Owner	PK	Person With PK	PK	
Car	IS N	Car Owner	Car ISN	Belongs To
Person With PK	PK	Car Owner	PK	IS
Person With PK	PK	Car	PK	Owns
Person With PK	PK	Passport	PK	Has
Person With PK	PK	Citizen	PK	Is
Person With PK	PK	Citizen	Child PK	Has Parents
Person With PK	PK	Citizen	Parent PK	Has Children

3. Browsing Principles

General idea for dynamic browsing of various data sources is to generate Web pages with predefined information layout and functionality, get data from data sources and put them into page.

A Web page consists of a set of frames (Frame) – FrameSet. The FrameSet has a prefixed count of Frames, its layout and sizes. We can define as many as we need different FrameSets to organise and display information for the user. The FrameSet is a view to related data from one or many data sources. One of the Frames is the main Frame. The information in any other Frame is logically connected with data in the main Frame. The Frames can contain controls to manage the content in the other Frame.

The layout of the Frame is defined by rule, lets call it Content. Theoretically the Content is a formula or function: ***Content(frameEntity, filterExpr)*** where *frameEntity* is any entity from the metamodel of data sources and *filterExpr* is logical expression that filters data from appropriate data source. The Content defines:

1) what is the structure and principles of layout,
2) what data from metamodel and from actual legacy data sources are required to display information,
3) what actual instances of the defined entity are retrieved,
4) what controls are used to manage the content of the other Frame or to open the other FrameSet,
5) what related entities are involved from the same or any other data source. If we have various predefined Contents, then we can dynamically apply any Content to the Frame and get another data presentation for the same *frameEntity* and *filterExpr*.

4. Defining the Content of Frame

Let us assume that Content is the function ***Content(frameEntity, filterExpr)***. Let us determine the means how we can define Content.

We introduce the following data types:

- **entity** - determines the entity from the metamodel,
- **field** - determines the field of the entity from the metamodel,

- **relation** - determines the relation for two entities from the metamodel,
- **record** - determines the actual data from the data source for one fixed instance of the entity,
- **value** - determines the actual data of the field for one fixed instance of the entity,
- **string** - determines the character string,
- **list** - determines the list of elements with any other allowed data type, we denote such types by the element type followed by postfix "List",
- **updateAction** - determines the action that updates Frame,
- **navigateAction** - determines the action that navigates browsing to another FrameSet,
- **sObject** - determines the HTML object that contains string to display,
- **aObject** - determines the HTML object with assigned some action to perform,
- **fObject** - determines the HTML object that is formatted for displaying,
- **frame** - determines the Frame,
- **frameSet** - determines the FrameSet,
- **view** - determines the list of fields that must be displayed.

Let us rewrite the Content as a function ***Content(entity, expr(entity))***.

Let us introduce several additional functions to work with the metamodel and data sources, and to format HTML page.

- Functions to work with the metamodel:
  1. ***SourceName(entity) →string*** – returns the source name the entity belongs to.
  2. ***EntityName(entity) →string*** – returns the entity name.
  3. ***RelationList(entity) →relationList*** – returns all direct relations from the given entity to another entity (including itself) from the same data source.
  4. ***MetaRelationList(entity) →relationList*** – returns all indirect relations from the given entity to another entity from all available data sources.
  5. ***FieldList(entity, view) →fieldList*** – returns the list of all the fields of the entity.
  6. ***RelationName(relation) →string*** – returns the name (role) of the relation.
  7. ***FieldName(field) →string*** – returns the name of the field.
  8. ***RelationEntity(relation)) →entity*** – returns the entity at the opposite end of relation.

- Functions to work with data sources through wrappers:
  1. ***RecordList(entity, expr(entity)) →recordList*** – returns the list of instances (records) of the entity according to the given filtering expression.
  2. ***ValueList(record, view) →valueList*** – returns the list values of the given entity instance (record).
  3. ***Value(value) →string*** – returns the field value as character string.

- Functions to work with the list:
  1. ***List(element_1, element_2, ..., element_i) →list_1*** – returns the list of given elements and the list type *list_1* is appropriate to the element type.
  2. ***IterateList(n%list_1, function(n%)) →list_2*** – returns the list *list_2* that has as elements the results applying the given function. The function is executed with each parameter *n%* that is taken from the list *list_1* denoted by the identifier *n%* (n is any unique integer) and the list type *list_2* is appropriate to the function return type.
  3. ***Concatenate(list_1, list_2) →list_3*** – returns the concatenation of two lists with the same element type.

- Functions to format HTML page:
  1. ***SO(string)→sObject*** – creates sObject from the character string.
  2. ***StringListObject(stringList, separatorString)→sObject*** – creates sObject from the list of character strings separated by *separatorString*.
  3. ***Update(frame, entity, expr(entity), content)→updateAction*** – activates information update into the frame with the given entity, filter expression and layout.
  4. ***Clear(frame)→updateAction*** – clears the given frame .
  5. ***Navigate(frameSet , entity, expr(entity), content)→navigateAction*** – navigates to another FrameSet and update main Frame with the given entity, filter expression and layout.
  6. ***Link(sObject, navigateAction, updateActionList)→aObject*** – converts sObject into aObject and assign the navigation action and set of update actions to it. Any of action parameters may be empty.
  7. ***AO(sObject)→aObject*** – converts sObject into aObject with empty action.
  8. ***FO(aObject)→fObject*** – converts aObject into fObject without any special formatting.
  9. ***HorizontalTable(aObjectListList)→ fObject*** – creates fObject from the list of lists, this frame object is displayed as table, and internal lists are placed in rows.
  10. ***VerticalTable(aObjectListList)→fObject*** – creates fObject from the list of lists, this frame object is displayed as table, and internal lists are placed in columns.
  11. ***ListBox(aObjectList)→fObject*** – creates fObject from the list, this frame object is displayed as listbox.
  12. ***Horizontal(fObjectList)→ fObject*** – creates new fObject by arranging the given list horizontally.
  13. ***Vertical(fObjectList)→fObject*** – creates new fObject by arranging the given list vertically. Only frame objects with the type fObject may be displayed in the Web page.

5. Data Wrappers

Function ***RecordList*** must be implemented to get information from DS. The technology we use is simple, but effective. UB gets information from DS via Wrappers. This approach has the following advantages:

- It allows accessing DS via different protocols and methods – ODBC™, OLE DB™, SQL*Net™, DCOM™, COM+™, XML, HTTP.
- DS usually are made well suited for specific business tasks. DS are not primary made for data access from UB. The access to DS data usually is limited, it is allowed to execute some stored procedures to query data. Wrapper allows us to execute only authorised functions.
- Querying DS via functions allows us to have easy transfer real data from DS physical data model to logical data model that is more understandable for the user.

Information about functions is stored in the UB meta database: defined input and output fields for each function. Each input field may be mandatory or optional.

During development of the prototype, we discovered some rules for function implementation and developing conceptual model of DS.

- First rule - it is desirable to have input and output fields from one entity. It simplifies development of DS model and wrappers.
- Second rule - two approaches possible for making DS model and functions. One approach is that we already have functions, and we make conceptual data model of DS using the first rule. In case DS is a system we maintain and own, it is often possible to make functions according to conceptual data model of DS. In such a case we make conceptual data model of DS at first and

then we make data access functions according to conceptual data model and the first rule. It is helpful to make two types of functions:

1. The function gets information identifying the object from DS by some search criteria. For example, get person's PK by its name and surname (might be partial). The answer usually is a list of person's identifying information according to search criteria.
2. The function gets information about one object from one entity by its identifier. An example - get all information about the citizen by its PK.

For instance, we have two functions for the entity Citizen:

- Input data – Name, Surname (might be partial). Output data – PK, Name, Surname (full).
- Input data – PK. Output data – PK, Name, Surname, Address.

There are also 2 functions to get information about the citizen's parents and children:

- Input data – Parent PK. Output data – Children PK, Name, Surname.
- Input data – Child PK. Output data – Parents PK, Name, Surname.

There is a procedure that implements the function *RecordList*. This procedure gets the entity and filter expression as input and returns data from DS as output. In our implementation this procedure gets information from the meta database about functions that can be executed over entity from which we need information. In our implementation the filter expression is fields and corresponding values for these fields, e.g. PK="123456-11111". There is "brute force" algorithm that finds functions we can execute e.g. those are functions that have enough input data from the filter expression to be executed, executes these functions and returns result. There can be, of course, other implementations.

DS data access via wrappers allows connecting new DS to our system easily and quickly. We have to write a new wrapper and add information about new DS to the meta database. With some experience the writing of wrappers is easy and fast process, and there is no need to make any modification in DS.

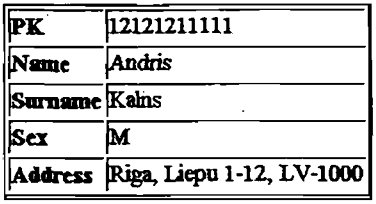
6. Templates for Web Page Structure and Functionality

The design of FrameSet and Frames is based on template principle. With some experience the new FrameSets and Frames can be developed quickly. The design has two main steps – FrameSet structure planning and creating formulas for Frame Contents. We give some templates and ideas how the Web pages can be designed. The above given functions are used. Formulas are logically divided into several subparts only for easier understanding. Some formulas use subparts of other previously defined formulas. The example of visual presentation for each formula is given.

6.1 SIMPLE ENTITY INSTANCE PRESENTATION IN TABLE

The first column contains field names and the second – field values. The field values are retrieved according to the selected *view*.

```
A(entity, record) = VerticalTable(List(A1, A2))
A1 = IterateList(1%FieldList(entity, view), AO(SO(FieldName(1%))))
A2 = IterateList(2%ValueList(record, view), AO(SO(Value(2%))))
```



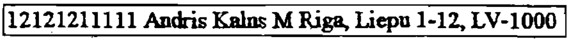
PK	12121211111
Name	Andris
Surname	Kalns
Sex	M
Address	Riga, Liepu 1-12, LV-1000

Figure 4. Example of entity instance presentation

6.2 ENTITY INSTANCE PRESENTATION AS TEXT

Instance field values are concatenated according to the order of the selected *view*.

```
B(record) = FO(AO(SO(StringListObject(B1, " "))))
B1 = IterateList(3%ValueList(record, view), Value(3%))
```



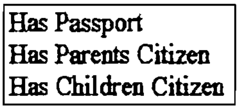
12121211111 Andris Kalns M Riga, Liepu 1-12, LV-1000

Figure 5. Example of entity instance presentation as text

6.3 ENTITY RELATIONS PRESENTATION IN VERTICAL LIST

Each relation is represented as relation name concatenated with entity name at the opposite relation end.

```
C(entity) = Vertical(IterateList(4%RelationList(entity), C1))
C1 = Horizontal(List(C2, FO(AO(SO(" "))), C3))
C2 = FO(AO(SO(RelationName(4%))))
C3 = FO(AO(SO(EntityName(RelationEntity(4%)))))
```



Has Passport
Has Parents Citizen
Has Children Citizen

Figure 6. Example of relations presentation

6.4 ALL RELATION PRESENTATION IN TABLE

The data about all relations (relation name, entity name and data source) are placed in table with headings.

```
D(entity, expr(entity))=HorizontalTable(Concatenate(D1,D2))
Di = AO(StringListObject("Relation", "Entity name", "Data source"))
D2 = IterateList(5%MetaRelationList(entity),List(D3, D4, D5))
D3 = AO(SO(RelationName(5%)))
D4 = AO(SO(EntityName(RelationEntity(5%))))
D5 = AO(SO(SourceName(RelationEntity(5%))))
```

Relation	Entity name	Data source
Is	Citizen	Register of Residents
Has	Passport	Register of Residents
Has Parents	Citizen	Register of Residents
Has Children	Citizen	Register of Residents
Is	Car Owner	Register of Motor vehicles
Owns	Car	Register of Motor vehicles

Figure 7. Example of relation presentation

6.5 AN EXAMPLE OF FRAMESET

Let us look how a FrameSet can be built. Let us assume FrameSet *FRS_1* with four Frames – *FR_1, FR_2, FR_3, FR_4*. See Figure 8.

FR_1 (upper left) – to list instances of entity,
FR_2 (upper right) – to show details of fixed instance in FR_1,
FR_3 (lower left) – to list all relations to other entities in all data sources,
FR_4 (lower right) – to show details of another related entity instances of FR_2.

At first let us create three presentations or Contents (E, F, G) for viewing entities. We use formulas created before in this paper.

- Content formula E() for Frame FR_4 (from FR_4 we can update all Frames in FRS_1)

```
E(entity, expr(entity)) = Vertical(E1, E5)
E1 = Horizontal(List(FO(E2), FO(AO(SO(" ")), FO(AO(SO(SourceName(entity))))))
E2 = Link(SO(EntityName(entity)), E3, E4)
E3 = Navigate("FRS_1", entity, expr(entity), "" )
E4 = List(Clear("FR_2"), Update("FR_3", entity, expr(entity), ""), Clear(FR_4))
E5 = Vertical(IteateList(6%RecordList(entity, expr(entity)), A(entity, 6%)))
```

- Content formula F() and G() for Frame FR_2 (from FR_2 we can update this frame or update FR_4)

```
F(entity, expr(entity)) = Vertical(H, FO(AO(SO(" "))), E5)
G(entity, expr(entity))  = Vertical(H, FO(AO(SO(" "))), Vertical(E5, G1))
G1 = C(entity), where C3 is substitute with G2 in all places (we have added the action)
G2 = FO(Link(SO(EntityName(RelationEntity(4%))), NULL, G3))
G3 = List(Update("FR_4", RelationEntity(4%), expr(RelationEntity(4%)), "E"))
H = ListBox(List(Link("Presentation F", NULL, H1), Link("Presentation G", NULL, H2)))
H1 = Update("FR_2", entity, expr(entity), "F")
H2 = Update("FR_2", entity, expr(entity), "G")
```

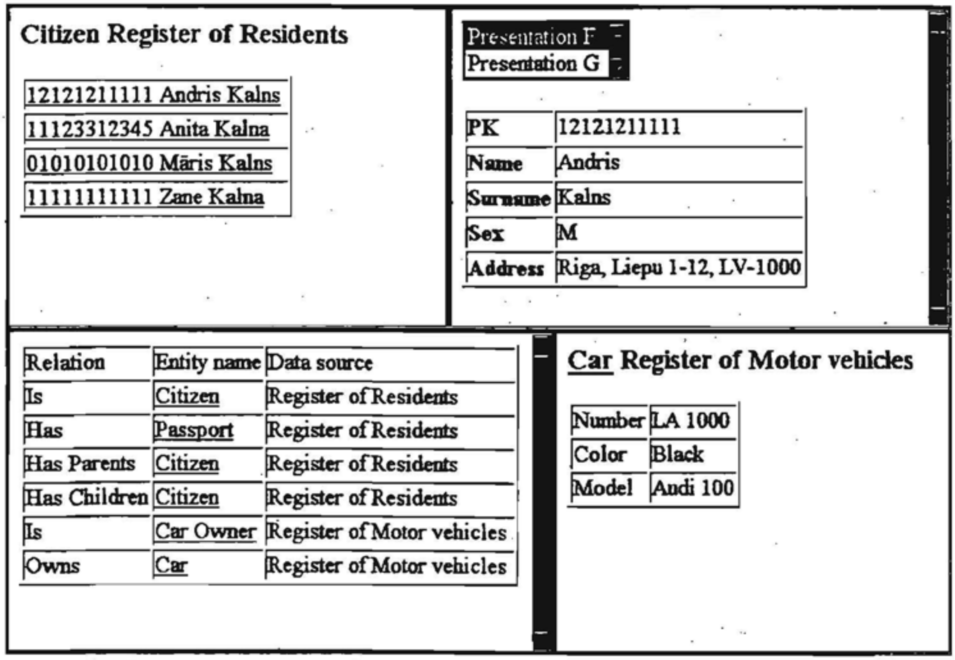
- Content formula I() for Frame FR_1 (from FR_1 we can update FR_2, FR3, FR_4)

```
I(entity, expr(entity))  = Vertical(I1, I2)
I1 = Horizontal(List(FO(EntityName(entity)), FO(AO(SO(" "))), FO(AO(SO(SourceName(entity))))))
I2 = HorizontalTable(IterateList(7%RecordList.Link(B(7%), NULL, I3))
I3 = List(Update("FR_2", entity, expr(entity) and expr(7%), "F"),
Update("FR_3", entity, expr(entity) and expr(7%), ""), Clear("FR_4"))
```

- Content formula J() for Frame FR_3 (from FR_3 we can update FR_4)

J(entity, expr(entity)) = D(entity, expr(entity)), *where D4 is substitute with J1 in all places (we have add the action)*

J1 = Link(SO(EntityName(RelationEntity(5%))), NULL, J2)
J2 = List(Update("FR_4", RelationEntity(5%), expr(RelationEntity(5%)), "E"))



Citizen Register of Residents

12121211111 Andris Kalns
11123312345 Anita Kalna
01010101010 Māris Kalns
11111111111 Zane Kalna

Presentation F
Presentation G

PK	12121211111
Name	Andris
Surname	Kalns
Sex	M
Address	Riga, Liepu 1-12, LV-1000

Relation	Entity name	Data source
Is	Citizen	Register of Residents
Has	Passport	Register of Residents
Has Parents	Citizen	Register of Residents
Has Children	Citizen	Register of Residents
Is	Car Owner	Register of Motor vehicles
Owns	Car	Register of Motor vehicles

Car Register of Motor vehicles

Number	LA 1000
Color	Black
Model	Audi 100

Figure 8. Example of WWW page

7. Conclusions and Future Directions

The prototype of the UB is made during developing Megasystem and Communication Server. Four state significance registers test databases are connected to the UB for testing purposes. Two of them use Oracle™ as DBMS, other two use Microsoft SQL Server™.

The UB prototype shows the effectiveness of our approach and is being initiated as first version of the real system at present time.

Our approach differs from other systems by several aspects:

- We have developed simple universal user-end that still allows us to show to users information in many different ways. We achieved this goal by implementing user-end using formal formulas.
- UB operates using logical models of DS. Related objects from these models are bound together with base classes that do not belong to any particular DS.
- We transfer physical model of DS to our internal logical representation which is much more comfortable for end-user. We do it by using of data wrappers.
- Our approach allows us to maintenance system and to connect new DS or modify existing one without interrupting operation of Communication server.

There are many aspects that are very important in real life application, but not covered in this article – security, user authorisation, logging, query cost calculation. All these features also are incorporated in the UB. The UB is useful in many large enterprises having many autonomous data sources as a browser for these systems with integrated view.

Future direction of our work is to develop a query processor that can take as input SQL-like query and return as output the result queried from multiple data sources. Other directions of future work – to make Communication server available not only from WWW browsers, but also from custom programs using XML to query data and return answers.

References

1. Arnicans, G. Application generation for the simple database browser based on the ER diagram. *Proceedings of the Third International Baltic Workshop Databases and Information Systems*, Riga, 1998, pp. 198-209.
2. Arnicans, G., Bicevskis, J., Karnitis, G. The concept of setting up a communications server. *Abstracts of Papers of 3rd International Conference Information Technologies and Telecommunications in the Baltic States*, 1999, pp. 48-57.
3. Haas, L. M., Miller, R. J., Niswonger, B., Tork Roth, M., Schwarz, P. M., Wimmers, E. L. Transforming heterogeneous data with database middleware: beyond integration. *Data Engineering Bulletin*, 1999.
4. Hammer, J., Garcia-Molina, H., Ireland, K., Papakonstantinou, Y., Ullman, J., Widom, J. Information translation, mediation, and Mosaic-based browsing in the TSIMMIS system. *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1995, Project Demonstration.
5. *Megasystem - Integrated State Significance Information System.* http://www.mega.lv.
6. Singh, N. Unifying heterogeneous information models. *Communications of the ACM*, **41**(5), 1998, pp. 37-44.
7. Tomasic, A., Amouroux, R., Bonnet, P., Kapitskaia, O., Naacke, H., and Raschid, L. The distributed information search component (disco) and the World Wide Web. *Proceedings of ACM SIGMOD International Conference on Management of Data*, Tuscon, Arizona, 1997, Prototype Demonstration.

Zbornik D 5. mednarodne multi-konference
Proceedings D of the 5th International Multi-Conference

INFORMACIJSKA DRUŽBA IS'2002
INFORMATION SOCIETY IS'2002

Vzgoja in izobraževanje v informacijski družbi
Education in Information Society

Uredili / Edited by
Vladislav Rajkovič
Tanja Urbančič
Mojca Bernik

Razvoj in prenovitev informacijskih sistemov
Development and Reengineering of Information Systems

Uredil / Edited by
Ivan Rozman

Sodelovanje in informacijska družba
Collaboration and Information Society

Uredila / Edited by
Marjan Heričko
Matjaž B. Jurič

Upravljanje v informacijski družbi
Management in Information Society

Uredil / Edited by
Cene Bavec

http://is.ijs.si

14. - 18. oktober 2002 / 14 - 18th October 20
Ljubljana, Slovenia

SMART INTEGRATED MEGA-SYSTEM AS A BASIS FOR E-GOVERNANCE

Guntis Arnicans, Prof. Janis Bicevskis, Prof. Edvins Karnitis, Girts Karnitis
Department of Computer Science
University of Latvia
Raina blv. 19, LV-1050 Riga, Latvia
Tel: +371 7228226; fax: +371 7820153
e-mail: garnican@lanet.lv, bics@di.lv,
Edvins.Karnitis@sprk.gov.lv, girts@di.lv

ABSTRACT

Principles and basic informatics tools for modernization of governance in Latvia are described in the paper. Ensuring access to well-developed information services for everyone should be envisaged as a tool for democratic development and functioning of society. Development of ICT directly affects political/governance procedures also, usage and management of public sector information become the base for all governance procedures. Interconnection and interoperation of public information systems, development of smart Mega-system, integration of national information resources of Latvia in Transeuropean telematic networks become components of unified process.

UNIVERSAL INFORMATION SERVICE

Public sector information, its usage and management is the real base for all governance (both G2G and G2C) procedures [1]. It ranks high among various types of information by its amount and significance, the public sector is the most important (and very often the single) collector and producer of information content.

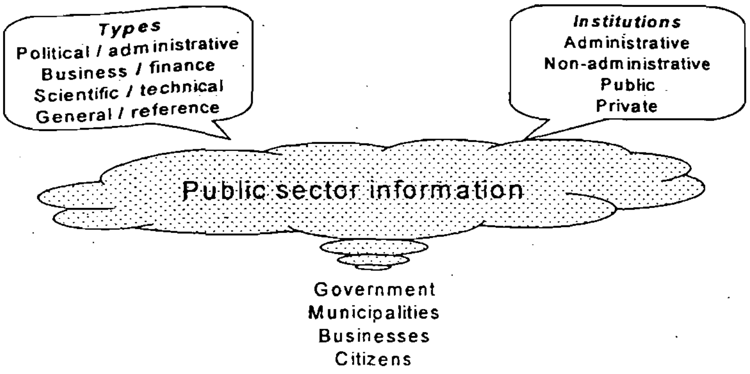


Figure 1. *Public sector information*

Processing and usage of information that can be deemed as being the *information of national significance* is the most significant component [2]. Within this term we shall mean various information that is necessary for state or regional administration, for the development of national economy, for management of financial, educational and social processes. The main subjects of this type of information are real estate and movable property, legal and private persons, substantial for the country processes (legislation, statistics, finances, health care, etc.).

In order to ensure wide and active usage of public sector information, Latvia's approach is developed by the National Program *Informatics* and several other more detailed conceptual documents [3, 4]. In principle it is similar to EU concept, but more extended and methodologically more advanced. The Program implies under the *universal information service* a general access to information services for everybody in an order as set by normative acts without any discrimination, a long-time service of a defined quality at an affordable price.

Ensuring technical access to telecommunications and data transmission networks is well known as the *universal telecommunications service*. Exactly inclusion of the Internet access shows clear forward-looking vision for Latvia: developed data transmission services, convergence of all kinds of information and communications services.

In addition universal information service means ensured access to all types of public sector information (and first of all to information of national significance). An electronic delivery of information is envisaged (on-line or broadcasting, magnetic, optical or another carrier, etc.). Services can be provided on demand or to be interactive. A number of different information services are components of the universal information service – full set of business and finance information services, availability of data collected in national and municipality information systems (IS), library and reference information services, reference and entertainment services, etc.

A number of bounded up and interdependent subprograms of the National Program *Informatics* are directed to development of the universal information service. The Program includes both macro level strategy (policy of the development) and micro level measures (a number of applications and projects).

Although there are number of common principles in provision of the public sector information to all end-users, many important differences exist too. General access of any citizen to the public sector information (G2C) differs from utilization of the information for state governance (G2G), there are different information compositions, level of confidentiality, demands for completeness, correctness, updating of information. Therefore side by side with common methodological principles, different approaches are used for development of information processing and provision systems and services.

THE MEGA-SYSTEM: ADVANCED TOOL FOR ADMINISTRATION OF THE COUNTRY

Creation of corporative sectoral IS for interconnecting related institutions on national and international scale (e.g., EU Programme IDA [5]) are important activities that are going on in number of countries. The next step – interoperability of IS, interchange of data between sectoral information systems/networks and handling requests that require processing data from various IS.

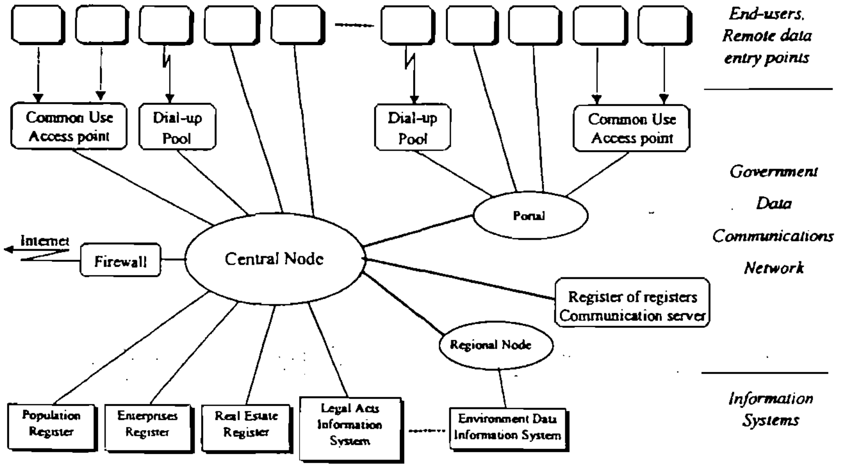


Figure 2. *The Mega-system*

Because a drastic improvement of quality and full interoperability of all IS are vital for the development of e-Government, all set of public IS in Latvia is being developed as a logically unified and technologically distributed information processing Mega-system with a common data field as well as unified user's interface, access principles and authorization procedures. Several basic principles are implemented into the Mega-system:

- the Mega-system is a set of separately functioning harmonized IS;
- all objects of national significance (persons, cars, real estate, legal entities, etc.) must be registered in IS;
- data must be fixed electronically in the place where they are originated; each object is allowed to be registered only in one of primary registers: the source of the information on the object as well as responsibility for its quality must be defined;
- all IS must use information on particular object from corresponding primary register, it is not allowed to duplicate data entry; it is allowed only to keep copy of the data from the basic register for improvement of access;

- the registration certificate of any object (passport, certificate of legal entity, etc.) must be issued only as the result of registration of objects; it is not allowed to repeat manual information input from registration certificate or other documents.

Creation of the Mega-system is not only technological decision, in fact it means solving of number of various informative, legal, organizational problems first of all, among them:

- to analyze existing data flows, to formulate functions of the Mega-system and to distribute them among IS, to formulate demands on systems and their data structure;
- to define subjects of various IS and the amount of stored information, as well as institutions that are responsible for the collection, processing and distribution of data;
- to define a unified user interface, access principles and authorization procedures;
- to elaborate several intercompatible informative models for implementation by local authorities;
- to ensure data quality and security as well as interoperability with EU IS; to elaborate a methodology for data verification;
- to determine the principles of electronic archives.

The integration of primary registers has realized. In addition to various IS the Mega-system includes a portal as a gateway to information resources, a ***register of registers*** for collection and distribution of meta information (formal and informal description of objects, data models, data flows, etc.) on all components of the Mega-system as well as ***communication server*** -- common central access point to information resources of the Mega-system. Other IS are being attached to the developed central core of the Mega-system gradually as far as they are prepared. For this purpose development of the IS is being continued, and primary data entry is taking place in many systems, even as other data are already being used.

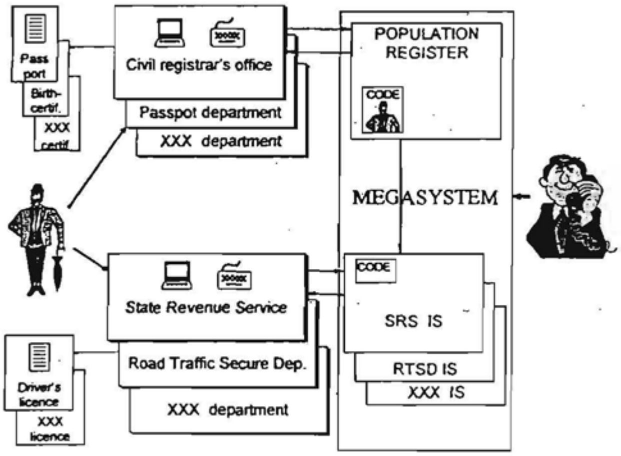


Figure 3. *The Mega-system: data flows*

All end-systems (various IS, their remote data entry and access points, end-users of information) are interconnected through a high speed *Government Data Communications Network*, that is an essential communications element for development of the Mega-system. This Network must provide operative and reliable interoperability of all interconnected systems, therefore requirements to the Network include:

- high security and reliability level -- there must be uninterrupted action time, undistorted data transmission, a guarantee of several levels of confidentiality and security of information;
- high speed data transmission, some of real time systems need guaranteed channel capacity;
- presence of a common gateway to public data transmission network (the Internet environment) which contains a reliable security system.

On the basis of the Mega-system during following years the G2G usage of traditional paper documents will be changed to usage of data base files, when an event or fact is assured not by a paper document, but by a record in a database. Each record in the IS will become legally approved document.

COMMUNICATION SERVER - A CENTRAL ACCESS POINT TO INFORMATION RESOURCES

Communication server is a set of hardware and software that provides a universal resource for information exchange among various information systems and other G2G transactions within the Mega-system as well as allows a wide range of users to receive information from a variety of public IS through a single contact point. The need to establish a communication server becomes apparent when it is necessary to interconnect a lot of IS and to retrieve information from number of systems in unified way.

Information becomes available on the Network, but users (most of them are employees of administrative structures) shouldn't have no knowledge about the technical details of information storage. There is an obvious need for a universal solution, and that is where the communication server comes in. The main requirement for the communication server is that it must allow users to formulate their information requests in a simple way and to receive responses to those requests without necessity to understand the technical aspects of the process and knowledge on distribution of data objects among the IS (by interconnection with the register of registers).

For these purposes the communication server identifies users, authorizes the use of the respective data, manages users rights, fulfills requests that involves usage of several information sources. It allows users to learn where information is being stored and what kind of information it is, as well as to request and to receive information from various registers without any need for in-depth knowledge about the technical aspects of its storage. National ID card is a crucial part to implement person's identification in communication process, there might be desirable to implement not only ID cards for citizens but also ID card for legal entities.

The communications server is an Internet resource point. Users of the server can access it via various protocols – HTTP, XML, CORBA, DCOM, SMTP and FTP. The server provides users an opportunity to find out where information is stored and what kinds of data are available, to request and receive information from various IS without studying their structure. Because users may have access to confidential and sensitive information, they are identified with certificates, and all data transmissions are coded.

THE MEGA-SYSTEM AS A TOOL FOR INTERNATIONAL INTEROPARABILITY

Multilingual and multicultural Europe has a particular interest in international cooperation and united activities, because individual national markets for information services are mostly small and ineffective. Competition on the global scale also requires a common European strategy and intercoordinated development. For this reason, the European Commission sees collaboration among all European states – including the associated Central and Eastern European Countries as an important component of integrated information policy [6]. Integration of national information resources of Latvia in Transeuropean information systems and networks is going on.

In order to further develop national IS of Estonia, Latvia and Lithuania, to prepare their future informative and technological connection with European IS, prime ministers of the Baltic States in 1997 made the decision to create a *Baltic Governmental Data Communications Network*. The Baltic Network is considered as expanding of the Mega-system and the Government Data Communications Network on international scale.

The concept envisages to develop the Baltic Network as a pilot stage for integration of national IS in Transeuropean systems. All principles of the Mega-system structure and user access, data structure and interfaces are being developed so as to allow for integration of the Latvia's IS into the Transeuropean corporate telematic networks. International expansion of the Mega-system involves the creation of resources-points and interfaces for international interconnection of IS, while maintaining the basic principles of the Mega-system.

Centralized resources of the Mega-system -- register of registers and communication server support both local and international information services. Data on international resources, that are available for Latvia's end-users, are included in the register of registers. Common communication server can be used for authorization and access to national information resources for foreign end-users equally with local users.

Such approach corresponds to basic principles of the IDA Programme, it is the basis for successful participation of Latvia in the Programme. The Programme consists of one central network connecting countries and local networks for each country, it request one central access point for each country. The Mega-system serves as Latvia's local network and communication server serves as Latvia's single access point.

A number of national IS are already participating in activities of international systems, they are pioneers among CEES at present. The Enterprise Register has been joined to the European Business Register in order to support international financial relations and investment processes, as well as business cooperation and foreign trade. Vehicles Register has already been connected to European Car Register. A number of another national IS are participating in the activities of international systems.

REGUIREMENTS TO THE NATIONAL PORTAL: TO BRING ADMINISTRATION CLOSER TO CITIZENS AND BUSINESSES

Latvia has adopted the EU recommended approach to the *level of electronization* of all G2B, G2C and G2G services identifies four different levels [7]:

- level 1: the provision of information – data on services are available on the Internet;
- level 2: interactivity – forms and documents can be downloaded;

- level 3: multi-directional interactivity – client authorization is enabled, and forms and information can be submitted electronically;
- level 4: processing of transactions – full handling services, including the taking relevant decisions and the making payments.

National portals in many countries ensure the first level, although the quality of the information that is provided is not always guaranteed. The possibility to download forms and documents is also fairly common (e.g., United States, France, Estonia) because this does not demand excessively complicated technologies.

There are different situations at the third and fourth levels. There are only very few countries that have resolved the client authorization problem, national laws on digital signatures and electronic documents have not been adopted yet. E-transactions are most commonly offered through Great Britain's *UK Online* and Singapore's *eCitizen* programs.

Usually national portals contain more than one way of looking of stored information. The following organizational types of information can be identified:

- around everyday themes (UKOnline, Danmark.dk, eCitizen Singapore);
- by regions (Danmark.dk);
- by sectors in a catalogue-type principle (in nearly all national portals);
- around the country's administrative structure (Bundesregierung, FirstGov, etc.);
- separately for citizens, businesses and foreigners (Canada).

In general e-governance means that the government shifts to a more advanced model of functioning. There are changes in the structure of the government and in relations among government institutions. The portal must be seen as an instrument in e-government, but by no means it cannot be seen as the tool that actually implements e-government. The Mega-system (including the portal as an interface) will provide government *back office* functionality.

The Latvia's national portal has been developed as a unified access point for information services what are provided by public agencies and institutions [8]. The first version of portal has been developed and provides first level services. The first and second levels services is provided directly by portal, while the third and fourth levels – in cooperation with the communication server.

The Latvia's national portal is being established defining three groups of individuals with different needs – citizens, business people and officials. A system have reciprocal links for all of them. It is important also to define the links between all categories of users as well as various levels of officials. Several organizational types of information are being developed at present [9].

As a result of analysis of the Latvia's situation and the possible demands of users, the basic principles have been defined:

- decentralization of information;
- the national portal is a portal of links;
- high quality (completeness, correctness, actuality) of the content;
- access to the public IS (in cooperation with the register of registers and the communication server);
- opportunities for contacts with officials;
- possibility to download forms and documents;
- autentification of users and personalization of content;
- availability of services in several languages (Latvian is mandatory);
- development of tools for support and maintenance of portal.

The development of the Latvia's portal is going on, the first version is available at the WWW [10]. The process of fulfilling portal with actual data is in progress at this moment.

OPENNESS AND TRANSPARENCY - THE MODEL OF E-GOVERNANCE

Rapid evolution of ICT make possible to change general requirements to the governance principles and procedures in line with development of e-democracy.

It is very hard task to control state institutions nowadays. Usage of ICT will help to introduce number of important for democratic society principles:

- openness – quantity and quality of information that any public institution provides to society, especially on-line;
- transparency – possibility to track how the institution acts and how it is making decisions;
- interactivity – possibilities that citizens and businesses have to contact with any level institution, its readiness to the fast reaction and dialog; possibility to offer opinions as well as to influent decisions;
- control, audit, inspection – public possibility to monitor and to control institution from outside.

Governance will become fully democratic only if every member of society will have possibility to get easy detailed information on administrative processes that are important for welfare of citizens and efficiency of businesses [11]. Citizens or officials initiate any process. While developing the Mega-system, it becomes possible to define and identify set of processes any person

Databases and Information Systems

Edited by Hele-Mai Haav and Ahto Kalja

Proceedings of the
Fifth International Baltic Conference,
BalticDB&IS 2002,
Tallinn, June 3-6, 2002

Volume 1

SEMANTICS FOR MANAGING SYSTEMS IN HETEROGENEOUS AND DISTRIBUTED ENVIRONMENT

Guntis Arnicans and Girts Karnitis
University of Latvia

Abstract: The problem of legacy systems collaboration is being solved. Particularly we look at collaboration as a workflow in a distributed and heterogeneous environment. Attention is paid to the description of semantics for workflow process definition languages. There are many solutions how semantics can be decomposed into logical fragments, but the problem of obtaining reusable components, that are easy to compile into desired specific semantics, remains. We evolve the dividing of semantics by semantic aspects, which description bases on abstract data types (pre-build components) and connectors (meta-programs to produce glue code) between them. This paper offers a way, in which semantic aspects are linked with the intermediate representation of a program, and performing of semantics is provided. We mix together various semantics aspects to get a desirable semantics.

Keywords: workflow, programming language specifications, semantics, interpreter, compiler, reusable components, domain specific languages, tool generation.

1. Introduction

Nowadays new technologies are emerging into a government sector, allowing speak about e-Government. The processes are one of the core components of e-Government [1]. We stated that practically no automation of processes in the governmental institutions that organizes collaboration between legacy systems among various organizations and institutions. Document flows are manual or by email. The automated workflows have to be introduced to make a document turnover faster and to improve a service for citizens.

In [2] workflow is defined as "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules."

Workflow management system is defined as "a system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and where required, invoke the use of IT tools and applications."

In the latest years many researchers and developers pay attention to a problem how organize collaboration between legacy systems, and the exploitation of workflow is one of the most popular solutions [3,4]. Various workflow process definition languages have been created which can be considered as domain specific languages.

The workflow implementations commonly base on the one fixed semantics like most of the programming languages. We are interested in various semantics for a particular workflow, for example, a common workflow semantics, a statistical data gathering semantics, a semantics for debugging and simulating purposes or its composition. We need not only a compiler or an interpreter, but the necessity for specific supporting tools becomes a burning question due to demands for high software quality. Interesting topic is changing of semantics for active instance of workflow on the fly.

In our approach semantics are connected to syntax elements via semantic connectors that naturally allow linking legacy systems into collaborative workflow and allow define or execute multiple different semantics simultaneously. Actually each semantic implementation is a tool, similarly to the principle in [5]. We present fragments of semantic description for simple programming language to demonstrate usefulness of this approach for wide class of programming languages, and ideas how to implement a simple workflow description language.

2. Implementation of Domain-Specific Language

According to Kinnersley's investigation [6], there were more then 2000 exploited languages in 1995, and most of them were classified as domain-specific language (DSL). Together with growth of DSL many implementations and maintenance problems arise (e.g. [7] analysis of common problems and large annotated bibliography; [8] particular languages and problems). Unfortunately formal semantics descriptions loose their position because of weak support to solve practical problems [9, 10,11].

In [2] workflow is defined as "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules."

Workflow management system is defined as "a system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and where required, invoke the use of IT tools and applications."

In the latest years many researchers and developers pay attention to a problem how organize collaboration between legacy systems, and the exploitation of workflow is one of the most popular solutions [3,4]. Various workflow process definition languages have been created which can be considered as domain specific languages.

The workflow implementations commonly base on the one fixed semantics like most of the programming languages. We are interested in various semantics for a particular workflow, for example, a common workflow semantics, a statistical data gathering semantics, a semantics for debugging and simulating purposes or its composition. We need not only a compiler or an interpreter, but the necessity for specific supporting tools becomes a burning question due to demands for high software quality. Interesting topic is changing of semantics for active instance of workflow on the fly.

In our approach semantics are connected to syntax elements via semantic connectors that naturally allow linking legacy systems into collaborative workflow and allow define or execute multiple different semantics simultaneously. Actually each semantic implementation is a tool, similarly to the principle in [5]. We present fragments of semantic description for simple programming language to demonstrate usefulness of this approach for wide class of programming languages, and ideas how to implement a simple workflow description language.

2. Implementation of Domain-Specific Language

According to Kinnersley's investigation [6], there were more then 2000 exploited languages in 1995, and most of them were classified as domain-specific language (DSL). Together with growth of DSL many implementations and maintenance problems arise (e.g. [7] analysis of common problems and large annotated bibliography; [8] particular languages and problems). Unfortunately formal semantics descriptions loose their position because of weak support to solve practical problems [9, 10,11].

Like a natural language the programming language definition consists of three components or aspects [12, 13]: syntax deals with questions of superficial form of a language, semantics deals with underlying meaning of a language, pragmatics deals with practical use of a language. A language's syntax and semantics can be formalized, and both formalizations together form formal specification of a programming language.

The formalisms for dealing with syntax aspect of a programming language are well developed. The theory of scanning, parsing and attribute analysis provides not only means to perform syntactical analysis but to generate a whole compiler as well. There are a lot of problems with a practical use of semantics formalisms. Recently the criticism of classical formalism arises from the difficulty of using formal methods. The main problem to use the formal specifications of programming languages widely in practice is that specifications become too complex, too abstruse to manage them, often it is impossible to express all needs, and in the end – who verifies and proves the correctness of specification?

Summarizing the best practices in compiler construction we can declare that most of commercial compilers (interpreters or other tools that deal with programs) are written without using any formalisms or only the first phases (scanning and parsing) exploit some formalisms [10].

Let us look at the language description again, try to divide it into smaller parts and see, what we can obtain from that. Traditionally the first decision is to separate syntax from semantics, and semantics consists of two parts: static semantics and dynamic (run-time) semantics. But we should divide syntax and semantic further, eliminate reusable components and provide a mechanism to stick all things together.

Syntax components are more or less visible: basic elements (for instance, terminals and nonterminals, if we parse program) connected with some relations (for instance, edges in the parse tree or abstract syntax tree).

To divide semantics into pieces we offer to split it by semantic aspects. Here are some examples of semantic aspects: program control flow management (e.g. loops, conditional branching), execution of commands or statements (e.g. basic operations, assigning), dealing with symbols (e.g. variables, constants), environment management (e.g. scopes of visibility), pretty printing of program, dynamic accounting of statistic, symbolic execution, specific program instrumentation, etc.

We are interested in any formalism to deal with syntax, because we want to make intermediate representation (IR) of program or structured information. It is a clear situation in dealing with a conventional programming language. But our goal is a workflow implementation and we have to take into account other languages, for instance diagrammatic visual languages (e.g. Petri nets, E-R diagrams, Statecharts) and state of art in this field (e.g. [14]).

Our approach borrows some principles from attribute grammars, for instance, the ways to link semantics with syntax [11], modular decomposition and reuse of specification [15], distributed computing in a real time (e.g. Communicating Timed AG [16]).

We founded that many formalisms of semantics use abstract data types (ADT). ADT is collection of data type and value definitions and operations on those definitions, which behaves as a primitive data type. This software design approach decomposes problem into components by identifying the public interface and private implementation. Typical example is a Stack, a Queue, a Symbol table [17, 13].

Recently one of the simple and popular methods to build some simple tool for a programming language is parse and traverse principle [18] that means to build intermediate representation (IR) of program or information, traverse IR and make appropriate computations at each node. This method is similar to the Visitor Pattern [19]. Another useful patterns are also developed [20, 21]. Besides nontraditional traversal strategies exists [e.g. 22]. Many solutions can be obtained from Component collaboration [e.g. 23].

We take into account our experience building prototypes of multi-language interpreter [24]. A Multi-Language Interpreter (MLI) is a program, which receives source language syntax, source language semantics and a program written in the source language, then performing the operations on the basis of the program and the relevant semantics.

3. Principles of Semantic Definition and Implementation

3.1 Runtime Principles

Let us assume that we have fixed some formalism to describe the syntax of our language (e.g. BNF). Now we can define the language syntax and develop a language parser (e.g. by using Lex/Yacc). The parser creates intermediate representation (IR) of program (e.g. Parse tree), and IR is based on a desirable structure and contains any needed information about the syntax (e.g. node type (nonterminal), name (name of nonterminal), value (terminal value), etc.).

To perform semantics at runtime we choose a principle of parse and traverse. The Traverser that realizes our chosen traversing strategy (e.g. left-depth tree traversing) has to be created for our IR representation. The computations, that have to be done at each node visited by the Traverser, are defined in Semantic Connectors (SC). They use predefined data structures with operations to establish cooperation between Legacy Systems (LS) (Figure 1).

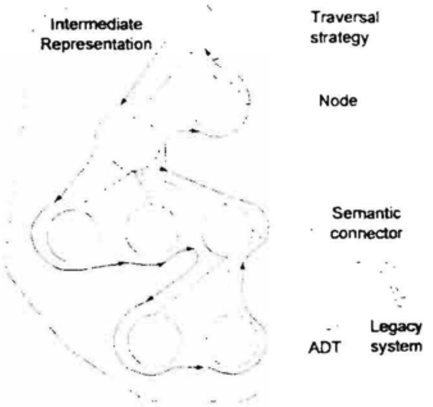


Figure 1. Runtime correspondence between syntax and semantics

Actually the most of work performing semantics is done via operations over various Abstract Data Types (ADT). In such way we hide most of implementation details and concentrate mainly on logic of semantic aspect. The consequence of this approach is that we can choose the best physical implementation of ADT for given task. For instance, Stack can be implemented in a contiguous memory or in a linked memory. The instances of ADT can be distributed objects in a heterogeneous computing network.

A concept of a *semantic connector* or simply *connector* is introduced to connect the instances of ADT and LS in a desirable environment. Connector is a meta-program that introduces a concrete communication connection into a set of components, i.e., it generates the adaptation and communication glue code for a specific connection. This concept is adopted from similar problem: how to connect pre-build components in distributed and heterogeneous environment [25].

3.2 Semantic Definition Principles

Similarly to patterns in [11] we choose a correspondence *Nonterminal with visiting aspect* = *Semantic connector* to establish relationships between syntax and semantics. *Nonterminal with visiting aspect* means that we distinguish computations performed at nonterminal node considering an aspect of node visiting (e.g. PreVisit or PostVisit). Any connector can see any instance of ADT or LS of the semantic aspect it (connector) belongs to.

The main problem is to find a good way to define semantics and obtain semantic connectors for the definition. After exploring various approaches how semantics can be described and organized, we suppose that semantic aspect is good basic component for constructing whole semantics according to our goals. The conceptual components of a semantics description and relationships with other concepts are represented in Figure 2.

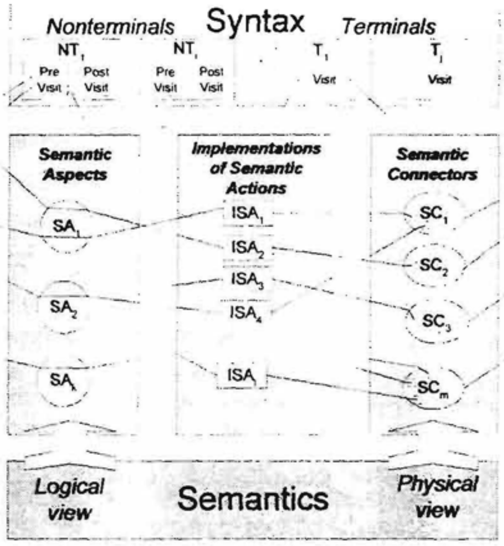
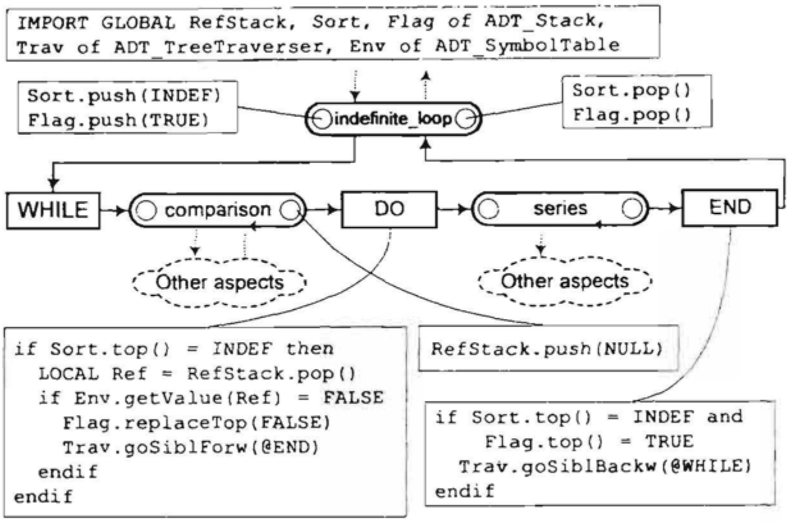


Figure 2. The conceptual schema of semantics.

Semantics can be observed from two different sides – a logical definition view and a physical runtime view. From the logical viewpoint semantics consists from Semantic Aspects (SA). The SA states what syntax elements (terminals, nonterminals) are involved in and what actions have to be performed traversing internal representation and visiting the corresponding node to realize semantic aspect. Let us define concept *Semantic Action* that denotes the action performed to realize SA while visiting a corresponding node, and – concept *Implementation of Semantic Action* (ISA) that denotes meta program which implements semantic action. In our example SA_1 involves nonterminal NT_1 and terminal T_1, and ISA_1 is performed while *previsiting* NT_1 and ISA_2 – while *visiting* T_1.

The example of semantic aspect INDEFINITE LOOP is given in Figure 3. There are various nonterminals and terminals organized by some syntax description. The arrows represent a traversal strategy. The small circles represent the semantic actions and the rectangles connected to the circles contain implementation of semantic action (meta program). A left circle into nonterminal stands for PreVisit and a right circle - for PostVisit. All used abstract data types (ADT) are defined within semantic aspect. Another example of semantic aspect is given in Figure 4.



```
IMPORT GLOBAL RefStack, Sort, Flag of ADT_Stack,
Trav of ADT_TreeTraverser, Env of ADT_SymbolTable
```

```
Sort.push(INDEF)
Flag.push(TRUE)
```

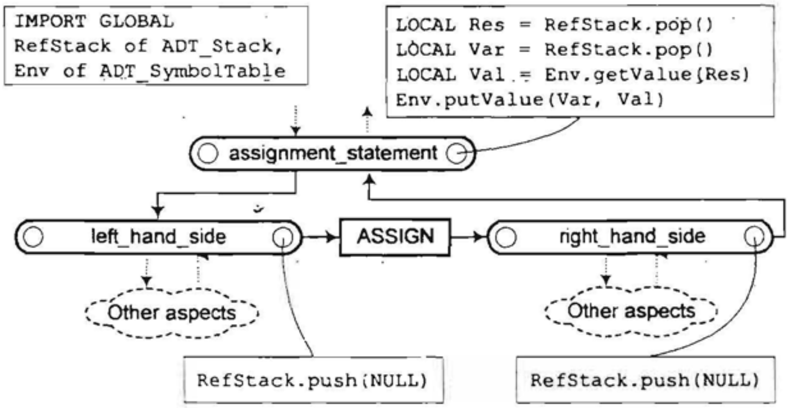
```
Sort.pop()
Flag.pop()
```

```
if Sort.top() = INDEF then
  LOCAL Ref = RefStack.pop()
  if Env.getValue(Ref) = FALSE
    Flag.replaceTop(FALSE)
    Trav.goSiblForw(@END)
  endif
endif
```

```
RefStack.push(NULL)
```

```
if Sort.top() = INDEF and
     Flag.top() = TRUE
   Trav.goSiblBackw(@WHILE)
endif
```

Figure 3. Semantic aspect INDEFINITE LOOP. It “goes through” series and back to WHILE until comparison sets NULL reference or reference with value FALSE



```
IMPORT GLOBAL
RefStack of ADT_Stack,
Env of ADT_SymbolTable
```

```
LOCAL Res = RefStack.pop()
LOCAL Var = RefStack.pop()
LOCAL Val = Env.getValue(Res)
Env.putValue(Var, Val)
```

```
RefStack.push(NULL)
```

```
RefStack.push(NULL)
```

Figure 4. Semantic aspect ASSIGNMENT. It takes reference to a variable and reference to a value from the stack, and assigns the value to the variable. Pushing of references is simulated, real references will be pushed by other aspects and simulating will be excluded

Let us look at the physical view. Semantic connector contains all corresponding semantic actions having to be executed while visiting syntax element (IR node). For instance, visiting any node with name T_1 we have to execute the semantic connector SC_2 that contains the implementation of the semantic action ISA_2. Similarly SC_1 is some composition of ISA_1 and ISA_4.

3.3 Obtaining Semantics from Semantic Aspects

From the logical point of view semantics is a composition of semantic aspects with concrete linking to instances of abstract data types, legacy systems and traverser that performs a traversal strategy over fixed intermediate representation of program or structured information. We cannot simply stick all SA together risking to get senseless semantics. A composition of semantic aspects is operations over set of implementations of semantic actions with aim to get one set of connectors that correspond to the new mixed semantic aspect (Table 1).

Table 1. Fragment of semantics description for simple imperative language

```
compatible with ir_type ParseTree, traverser_type ParseTreeTraverser
...
syntax elements (program, expression, VARIABLE, ...)
semantic actions (<PROGRAM> program PreVisit {ENV.prepareProgEnv()},
          <PROGRAM> program PostVisit {...}, ...)
...
global Trav of ADT_TreeTraverser, Env of ADT_SymbolTable
create DataStack, OperatorStack, CanCreateVar, LoopSortStack, LoopCounterStack,
LoopFlagStack, IfFlagStack of ADT_Stack, InputFile, OutputFile of ADT_FILE
...
compose aspect <COMPOSED SA>  // composes semantic aspects from predefined aspects
(<PROGRAM>)
append (<ELEMENT>
  replace RefStack with DataStack // replaces stack for collaborating work
  rename INTEGER Visit with CONSTANT Visit) // renames according to PAM syntax
append (<ASSIGNMENT>
  replace RefStack with DataStack
  rename left_hand_side PostVisit with VARIABLE Visit,
      right_hand_side PostVisit with expression PostVisit
  ignore left_hand_size PostVisit)    // ignore pushing of NULL reference
append (<INDEFINITE LOOP>
  replace RefStack with DataStack,
    Sort with LoopSortStack, Flag with LoopFlagStack)
...
end compose aspect
... // other aspect are defined and composed together
link for <COMPOSED SA> Trav to TreeTraverser, Env to SymbolTable
use aspect <COMPOSED SA> with traverser TreeTraverser
```

The obtaining of semantics for the fixed syntax is achieved in several steps: 1) select predefined semantic aspects or define new ones for desired semantics, 2) rename syntax elements and traversing aspect in the selected semantic aspects with names from fixed syntax and traversing strategy, 3) rename instances of abstract components to organize collaboration between semantic aspects, 4) make composition from semantic aspects, 5) specify the runtime environment and translate the meta-code to the code of the target programming language, and 6) compile the semantics.

After obtaining meta-semantics (Table 1) meta-code is translated to the target programming language, taking into account the target language (e.g. C++), the implementation of abstract components (e.g. Stack), the operating system (e.g. Unix), the communications between components (e.g. CORBA), runtime components type (e.g. DLL), etc. The translation may be done by hand or automatically (desirable in common cases).

By replacing ADT names we achieve independent working for some semantic aspects or collaborating work between them through common instances of ADT. Another way to get new semantics is to combine semantics aspects as whole black-box unit. Self-evident method is to execute several semantic aspects sequentially, for instance, we perform static semantic first and dynamic one after that. Instances of ADT can be shared and one semantic aspect can use results of others. More complex is a parallel executing of many semantics where we need to organize synchronization via instances of ADT.

4. Workflow Case Study

To demonstrate our approach we use very simple workflow definition language that syntax is described with BNF (Table 2). We have two types of generic statements for describing tasks in a workflow – universal statements and specific statements.

Table 2. Fragment of BNF for simple workflow definition language

workflow	-> series
series	-> statement | series ; statement
statement	-> generic_stm | cond_stm
cond_stm	-> IF compar THEN series ELSE series FI
compar	-> expr relation expr
expr	-> const | var
generic_stm	-> universal_stm | specific_stm
universal_stm	-> u_stm_type name
u_stm_type	-> DCOM | CORBA | WEBSERVICE | MANUAL
specific_stm	-> s_stm_type name
s_stm_type	-> ASK | ANSW

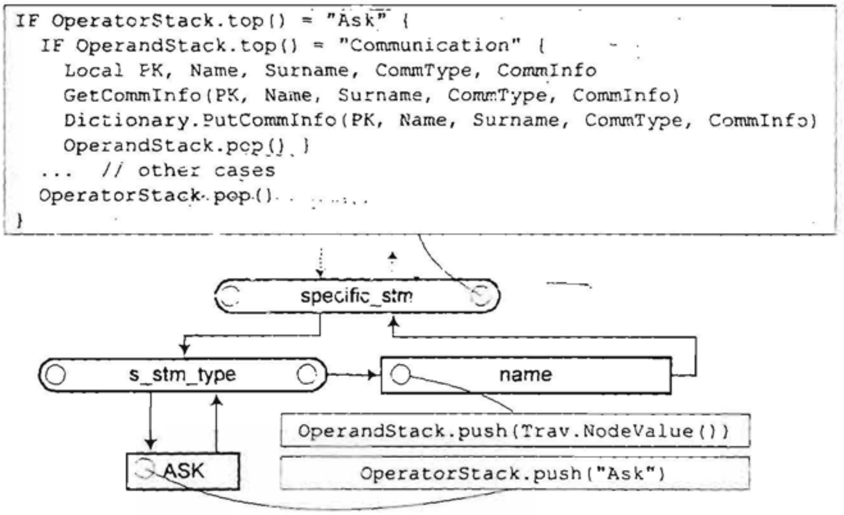
The *universal statements* are used to collaborate with external applications. The universal statement type describes connection type:

DCOM, CORBA, WEBSERVICE means automatic processing but MANUAL - that human handles this operation. The *specific statement* is used to communicate with a person – usually with a citizen who uses the particular service. There are two types of specific statements. ASK gets information from a person, ANSW sends some information to a person.

Lets take a look at the following simple workflow:

```
WEBSERVICE Application_writing_and_submitting
ASK Communication
DCOM Application_data_control_and_update
IF Is_data_control_and_updating_successful = True THEN
   DCOM Printing_of_passport
   ANSW Positive_answer
   MANUAL Passport_handing_out
ELSE
   ANSW Negative_answer
FI
```

The purpose of this workflow is to issue a new passport for a person. Workflow has the following activities - citizen fills an application form and submits it to official. It can be a paper form or a web based application. The official or the application asks from person a communication kind and address, and records data into workflow environment. Then the official verifies correctness of the citizen's fulfilled form with the data in Population Register, and if all data is correct, then a passport is issued and delivered to citizen. Otherwise negative answer is sent to citizen. An example of one semantic aspect of this workflow is given in Figure 5.



```
IF OperatorStack.top() = "Ask" {
  IF OperandStack.top() = "Communication" {
    Local PK, Name, Surname, CommType, CommInfo
    GetCommInfo(PK, Name, Surname, CommType, CommInfo)
    Dictionary.PutCommInfo(PK, Name, Surname, CommType, CommInfo)
    OperandStack.pop() }
  ...  // other cases
  OperatorStack.pop()
}
```

```
OperandStack.push(Trav.NodeValue())
OperatorStack.push("Ask")
```

Figure 5. Semantic aspect SPECIFIC STATEMENT ASK

5. Conclusions

We have presented ideas for establishing a framework to deal with different collaboration problems between legacy systems. The problem is reduced to describing the collaboration (e.g. workflow) by DSL and building various tools (various semantics) for this DSL.

There are many application generators that automatically produce conventional compiler and interpreter, but we need not only those ones. It is necessary to obtain various supporting tools that base on language text processing. Existing formal semantics are not well accepted by language or tool designers. We have made attempt to search for a compromise to minimize this gap. The latest related works in this field, to establish tool-oriented approach, are mentioned in [5], [26], [27].

We have offered ideas how semantics can be decomposed into reusable parts and specific semantics can be composed from it, and how execution is organized. We delegate the most of semantic actual work to pre-built components (ADT). Our approach allows minimize semantics descriptions for easiest management and provides good implementation in a possible parallel, distributed and heterogeneous environment. Our approach bases on an experience received by constructing prototypes of multi-language interpreter for conventional programming languages.

By doing a favor to practical needs, we lose something from precision and benefits of classical semantic formalisms. The next step is to finish formalization of our approach and to compare it with other formalisms, especially with attribute grammars. Another activities have to be the designing of useful collection of abstract data types.

References

[1] E. Karnitis. E-Government: An Innovative Model of Governance in the Information Society. *Baltic IT&T Review*, 1, 2001

[2] Layna Fischer, editor. The Workflow Handbook 2001, Published in association with the Workflow Management Coalition, 2000

[3] Workflow Standards and Associated Documents, http://www.wfmc.org/standards/docs/Stds_diagram.pdf

[4] Workflow/BPR Tools Vendors http://www.waria.com/databases/wfvendors-A-L.htm

[5] J. Heering and P. Klint. Semantics of Programming Languages: A Tool-Oriented Approach. *ACM SIGPLAN Notices*, 35(3):39-48, March 2000.

[6] W.Kinnersley, ed., The Language List. 1995. http://wuarchive.wustl.edu/doc/misc/lang-list.txt

[7] A Deursen, P. Klint, and J. Visser. Domain-Specific Languages: An Annotated Bibliography. *ACM SIGPLAN Notices*, 35(6):26-36, June 2000.

,[8] Special issue on domain-specific languages. *IEEE Transactions on Software Engineering*, 25(3), May/June1999.

[9] David A. Schmidt. Programming Language Semantics. In Tucker [28], pp.2237-2254.

[10] Kenneth C. Louden. Compilers and Interpreters. In Tucker [28], pp.2120-2147.

[11] J. Paakki. Attribute Grammar Paradigms – A High-Level Methodology in Language Implementation. *ACM Computing Surveys*, 27(2):196-255, June 1995.

[12] Frank G. Pagan. *Formal Specification of Programming Languages: A Panoramic Primer*. Prentice-Hall, 1981.

[13] K. Slonneger and B. L. Kurtz. *Formal Syntax and semantics of Programming Languages: A Laboratory Based Approach*. Addison-Wesly, 1995.

[14] F. Ferrucci, F. Napolitano, G. Tortora, M. Tucci, and G. Vitiello. An Interpreter for Diagrammatic Languages Based on SR Grammars. *Proceedings of the 1997 IEEE Symposium on Visual Languages (VL '97)*, pages 292-299, 1997.

[15] U. Kastens and W. M. Wait. Modularity and reusability in attribute grammars. *Acta Informatica 31*, pages 601-627,1994.

[16] T. Matsuzaki and T. Tokuda. CTAG Software Generator Model for Constructing Network Applications. *Proceedings of the Asia Pacific Software Engineering Conference*, pp.120-127, 1998.

[17] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Technigues, and Tools*. Addison-Wesley, 1986.

[18] C. Clark. Build a Tree – Save a Parse. *ACM SIGPLAN Notices*, 34(4):19-24, April 1999.

[19] E. Gamma, R. Helm, R. Johnson, and J. Vlisides. *Design Patterns: Elements of Reusable Software*, pages 331-334. Addison-Wesley, 1995.

[20] J. Ovlinger and M. Wand. A Language for Specifying Recursive Traversals of Object Structures. SIGPLAN Notices, 34(10):70-81, 1999. *Proceedings of the 1999 ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages & Applications (OOPSLA '99)*.

[21] T. Kühne. The Translator Pattern – External Functionality with Homomorphic Mappings. *Proceedings of the Tools-23: Technology of Object-Oriented Languages and Systems*, pp. 48-59, 1997.

[22] B. Biswas and R. Mall. Reverse Execution of Programs. *ACM SIGPLAN Notices*, 34(4):61-69, April 1999.

[23] M. Mezini and K. Lieberherr. Adaptive Plug-and-Play Components for Evolutionary Software Development. SIGPLAN Notices, 33(10):97-116, 1998. *Proceedings of the 1998 ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages & Applications (OOPSLA '98)*.

[24] 24. V. Arnicane, G. Arnicans, and J. Bicevskis. Multilanguage interpreter. In H.-M. Haav and B. Thalheim, editors, *Proceedings of the Second International Baltic Workshop on Databases and Information Systems (DB&IS '96)*, Volume 2: Technology Track, pages 173-174. Tampere University of Technology Press, 1996.

[25] U. Aßmann, T. Genßler, and H. Bär. Meta-programming Grey-box Connectors. *Proceedings of the Technology of Object-Oriented Languages and Systems (TOOLS 33)*, pp.300-311, 2000.

[26] M. Mernik, M. Lenič, E. Avdičaušević, and V. Žumer. Compiler/Interpreter Generator System LISA. *Proceedings of the 33rd Hawaii International Conference on System Sciences - 2000*, pp.10, 2000.

[27] A. M. Sloane. Generating Dynamic Program Analysis Tools. *Proceedings of the Autralian Software Endineering Conference (ASWEC'97)*, pp.166-173, 1997.

[28] Allen B. Tucker, editor. *The computer science and engineering handbook*. CRC Press, 1997.

Databases and Information Systems II

Fifth International Baltic Conference, Baltic DB&IS'2002
Tallinn, Estonia, June 3–6, 2002
Selected Papers

Edited by

HELE-MAI HAAV

Institute of Cybernetics at Tallinn Technical University,
Tallinn, Estonia

and

AHTO KALJA

Department of Computer Engineering of Tallinn Technical University,
Tallinn, Estonia

KLUWER ACADEMIC PUBLISHERS
DORDRECHT / BOSTON / LONDON

SEMANTICS FOR MANAGING SYSTEMS IN HETEROGENEOUS AND DISTRIBUTED ENVIRONMENT

Guntis Arnicans and Girts Karnitis
University of Latvia, Riga, Latvia

Abstract The problem of legacy systems collaboration is being solved. Particularly we look at the collaboration as workflow in a distributed and heterogeneous environment. Attention is paid to the description of semantics for workflow process definition languages. There are many solutions how semantics can be decomposed into logical fragments, but the problem of obtaining reusable components that are easy to compile into desired specific semantics still remains. We evolve the division of semantics by semantic aspects whose description is based on abstract data types (pre-built components) and connectors (meta-programs to produce the glue code) between them. This paper offers a way in which semantic aspects are linked with the intermediate representation of a program, and performing of semantics is provided. We mix together various semantics aspects to get a desirable semantics.

Keywords: workflow, programming language specifications, semantics, interpreter, compiler, reusable components, domain specific languages, tool generation.

1. Introduction

Nowadays new technologies are emerging in the government sector allowing to speak about the e-Government. The processes are one of the core components of e-Government [11]. We stated that there is practically no automation of processes in the governmental institutions that organize collaboration between legacy systems among various organizations and institutions. Document flows are manual, or by email. The automated workflows have to be introduced to make the document turnover faster and to improve the provided service for citizens.

H.-M. Haav and A. Kalja (eds.), Databases and Information Systems II, 149–160.

In [8] workflow is defined as "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules."

The workflow management system is defined as "a system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines which is able to interpret the process definition, interact with workflow participants and where required, invoke the use of IT tools and applications."

In the latest years many researchers and developers have paid attention to a problem how to organize the collaboration between legacy systems, and the exploitation of workflow is one of the most popular solutions [27, 28]. Various workflow process definition languages have been created which can be considered as domain specific languages.

The workflow implementations commonly are based on one fixed semantics, like most of the programming languages. We are interested in various semantics for a particular workflow, for example, common workflow semantics, a statistical data gathering semantics, a semantics for debugging and simulating purposes or its composition, therefore we need not only a compiler or an interpreter, but the necessity for specific supporting tools becomes a burning question due to demands for high software quality. An interesting topic is changing of semantics for active instance of workflow on the fly.

In our approach semantics are connected to syntax elements via semantic connectors that naturally allow linking legacy systems into collaborative workflow and allow to define or to execute multiple different semantics simultaneously. Actually, each semantic implementation is a tool, similarly to the principle in [10]. We present fragments of semantic description for simple programming language to demonstrate usefulness of this approach for a wide class of programming languages, and ideas how to implement a simple workflow description language.

2. Implementation of Domain-Specific Language

According to Kinnersley's investigation [13], there were more than 2000 exploited languages in 1995, and most of them were classified as domain-specific language (DSL). Together with the growth of DSL many implementations and maintenance problems arise (e.g. [6] analysis of common problems and large annotated bibliography; [25] particular languages and problems). Unfortunately, formal semantics descriptions lose their position because of a weak support to solve practical problems [15, 20, 22].

Like natural language, the programming language definition consists of three components or aspects [21, 24]: syntax deals with questions of superficial form of a language, semantics deals with the underlying meaning of a language, pragmatics deals with the practical use of a language. The syntax and semantics of a language can be formalized, and both formalizations together form formal specification of a programming language.

The formalisms for dealing with the syntax aspect of a programming language are well developed. The theory of scanning, parsing and attribute analysis provides not only means to perform syntactical analysis, but to generate a whole compiler as well. There is a lot of problems with practical use of semantics formalisms. Recently the criticism of classical formalism has arisen from the difficulty of using formal methods. The main problem to use widely in practice the formal specifications of programming languages is that specifications become too complex, too abstruse to manage them, often it is impossible to express all needs, and in the end – who verifies and proves the correctness of the specification?

Summarizing the best practices in compiler construction we can declare that most of commercial compilers (interpreters or other tools that deal with programs) are written without using any formalisms or only the first phases (scanning and parsing) to exploit some formalisms [15].

Let us look at the language description again, try to divide it into smaller parts and see, what we can obtain from that. Traditionally the first decision is to separate syntax from semantics, and semantics consists of two parts: static semantics and dynamic (run-time) semantics. But we should divide syntax and semantics further, eliminate reusable components and provide a mechanism to stick all things together.

Syntax components are more or less visible: basic elements (for instance, terminals and nonterminals, if we parse program) connected with some relations (for instance, edges in the parse tree or abstract syntax tree).

To divide semantics into pieces we offer to split it by semantic aspects. Here are some examples of semantic aspects: program control flow management (e.g. loops, conditional branching), execution of commands or statements (e.g. basic operations, assigning), dealing with symbols (e.g. variables, constants), environment management (e.g. scopes of visibility), pretty printing of program, dynamic accounting of statistic, symbolic execution, specific program instrumentation, etc.

We are interested in any formalism to deal with syntax, because we want to make intermediate representation (IR) of program or structured information. The situation is clear what refers to conventional programming languages. But our goal is a workflow implementation, and we have to take into account other languages, for instance, diagrammatic visual languages (e.g. Petri nets, E-R diagrams, Statecharts) and the state-of-art in this field (e.g. [7]).

Our approach has borrowed some principles from attribute grammars, for instance, the ways to link semantics with syntax [20], modular decomposition and reuse of specification [12], distributed computing in real time (e.g. Communicating Timed AG [16]).

We found out that many formalisms of semantics use abstract data types (ADT). ADT is a collection of data type and value definitions and operations on those definitions which behave as primitive data type. This software design approach decomposes problem into components by identifying the public interface and private implementation. A typical example is Stack, Queue, Symbol table [1, 24].

Recently one of the simple and popular methods to build some simple tool for a programming language has become parse and traverse principle [5] that means to build intermediate representation (IR) of program or information, traverse IR and make appropriate computations at each node. This method is similar to the Visitor Pattern [9]. Other useful patterns are also developed [14, 19]. Besides, there exist also nontraditional traversal strategies [e.g. 4]. Many solutions can be obtained from Component collaboration [e.g. 18].

We have taken into account our experience in building prototypes of multi-language interpreter [2]. A Multi-Language Interpreter (MLI) is a program which receives source language syntax, source language semantics and a program written in the source language, and then it performs the operations on the basis of the program and the relevant semantics.

3. Principles of Semantic Definition and Implementation

3.1 Runtime Principles

Let us assume that we have fixed some formalism to describe the syntax of our language (e.g. BNF). Now we can define the language syntax and develop a language parser (e.g. by using Lex/Yacc). The parser creates intermediate representation (IR) of program (e.g. Parse tree), and IR is based on a desirable structure and contains any needed information about the syntax (e.g. node type (nonterminal), name (name of nonterminal), value (terminal value) etc.).

To perform semantics at runtime we choose a principle of parse and traverse. The Traverser that realizes our chosen traversing strategy (e.g. left-depth tree traversing) has to be created for our IR representation. The computations that have to be done at each node visited by the Traverser are defined in Semantic Connectors (SC). They use predefined data structures with operations to establish the cooperation between Legacy Systems (LS) (Figure 1).

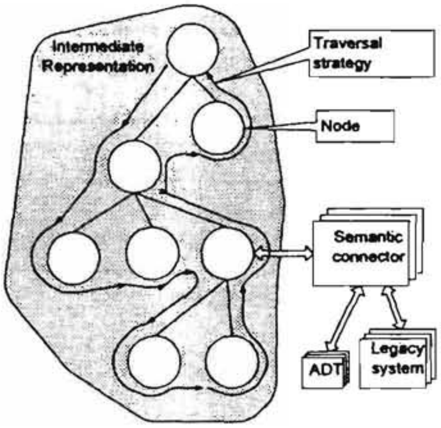


Figure 1. Runtime correspondence between syntax and semantics

Actually, most of work performing semantics is done via operations over various Abstract Data Types (ADT). In such a way we hide most of implementation details and concentrate mainly on logic of semantic aspect. The consequence of this approach is that we can choose the best physical implementation of ADT for the given task. For instance, Stack can be implemented in a contiguous memory or in a linked memory. The instances of ADT can be distributed objects in a heterogeneous computing network.

A concept of a *semantic connector* or simply *connector* is introduced to connect the instances of ADT and LS in a desirable environment. Connector is a meta-program that introduces a concrete communication connection into a set of components, i.e. it generates the adaptation and communication glue code for a specific connection. This concept is adopted from similar problem: how to connect pre-built components in a distributed and heterogeneous environment [3].

3.2 Semantic Definition Principles

Similarly to patterns in [20] we choose a correspondence *Nonterminal with visiting aspect = Semantic connector* to establish the relationship between syntax and semantics. *Nonterminal with visiting aspect* means that we distinguish computations performed at nonterminal node considering an aspect of node visiting (e.g. PreVisit or PostVisit). Any connector can see the instance of ADT or LS of the semantic aspect it (connector) belongs to.

The main problem is to find a good way to define semantics and obtain semantic connectors for the definition. After exploring various approaches how semantics can be described and organized, we suppose that semantic aspect is a good basic component for constructing whole semantics according to our goals. The conceptual components of a semantics description and relationships with other concepts are represented in Figure 2.

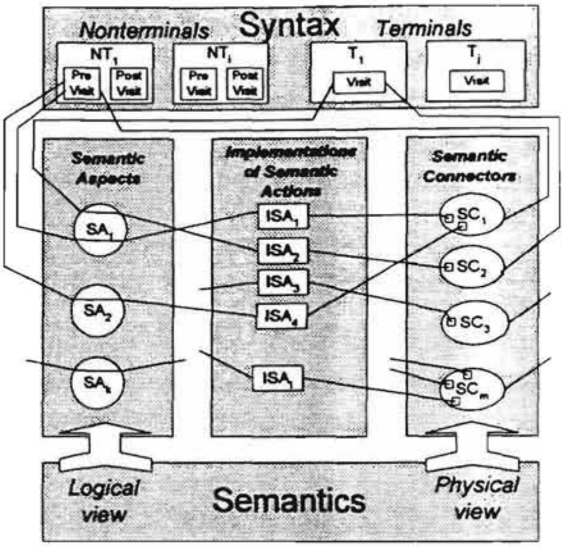
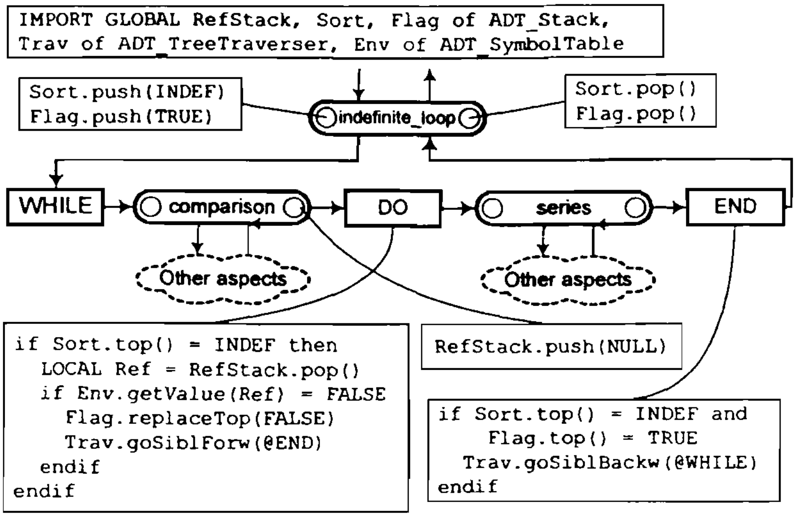


Figure 2. The conceptual schema of semantics.

Semantics can be observed from two different sides – a logical definition view and a physical runtime view. From the logical viewpoint semantics consists of Semantic Aspects (SA). The SA states what syntax elements (terminals, nonterminals) are involved, and what actions have to be performed traversing internal representation and visiting the corresponding node to implement the semantic aspect. Let us define the concept *Semantic Action* that denotes the action performed to implement SA while visiting a corresponding node, and the concept *Implementation of Semantic Action* (ISA) that denotes a meta program which implements the semantic action. In our example SA_1 involves nonterminal NT_1 and terminal T_1, and ISA_1 is performed while *previsiting* NT_1 and ISA_2 – while *visiting* T_1.

The example of semantic aspect INDEFINITE LOOP is given in Figure 3. There are various nonterminals and terminals organized by some syntax description. The arrows represent a traversal strategy. The small circles represent the semantic actions, and the rectangles connected to the circles contain the implementation of semantic action (meta program). A left circle into nonterminal stands for PreVisit, and a right circle - for PostVisit. All used abstract data types (ADT) are defined within semantic aspect. Another example of semantic aspect is given in Figure 4.



```
IMPORT GLOBAL RefStack, Sort, Flag of ADT_Stack,
Trav of ADT_TreeTraverser, Env of ADT_SymbolTable
```

```
Sort.push(INDEF)
Flag.push(TRUE)
```

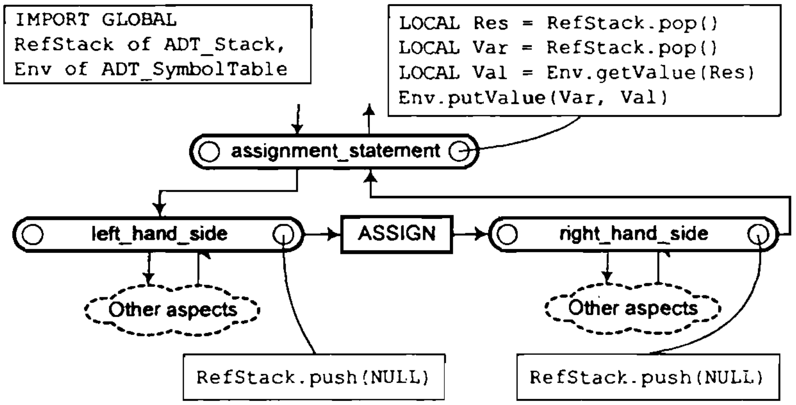
```
Sort.pop()
Flag.pop()
```

```
if Sort.top() = INDEF then
  LOCAL Ref = RefStack.pop()
  if Env.getValue(Ref) = FALSE
    Flag.replaceTop(FALSE)
    Trav.goSiblForw(@END)
  endif
endif
```

```
RefStack.push(NULL)
```

```
if Sort.top() = INDEF and
    Flag.top() = TRUE
  Trav.goSiblBackw(@WHILE)
endif
```

Figure 3. Semantic aspect INDEFINITE LOOP. It "goes through" series and back to WHILE until the comparison sets NULL reference or reference with value FALSE



```
IMPORT GLOBAL
RefStack of ADT_Stack,
Env of ADT_SymbolTable
```

```
LOCAL Res = RefStack.pop()
LOCAL Var = RefStack.pop()
LOCAL Val = Env.getValue(Res)
Env.putValue(Var, Val)
```

```
RefStack.push(NULL)
```

```
RefStack.push(NULL)
```

Figure 4. Semantic aspect ASSIGNMENT. It takes reference to a variable and reference to a value from the stack, and assigns the value to the variable. Pushing of references is simulated, real references will be pushed by other aspects and simulating will be excluded

Let us look at the physical view. The semantic connector contains all corresponding semantic actions having to be executed while visiting syntax element (IR node). For instance, visiting any node with the name T_1 we have to execute the semantic connector SC_2 that contains the implementation of the semantic action ISA_2. Similarly, SC_1 is some composition of ISA_1 and ISA_4.

3.3 Obtaining Semantics from Semantic Aspects

From the logical point of view semantics is a composition of semantic aspects with concrete linking to instances of abstract data types, legacy systems and traverser that performs a traversal strategy over fixed intermediate representation of program or structured information. We cannot simply stick all SA together risking to get senseless semantics. A composition of semantic aspects is operations over a set of implementations of semantic actions with the aim to get one set of connectors that correspond to the new mixed semantic aspect (Table 1).

Table 1. Fragment of semantics description for simple imperative language

```
compatible with ir_type ParseTree, traverser_type ParseTreeTraverser
...
syntax elements (program, expression, VARIABLE, ...)
semantic actions (<PROGRAM> program PreVisit {ENV.prepareProgEnv()},
           <PROGRAM> program PostVisit {...}, ...)
...
global Trav of ADT_TreeTraverser, Env of ADT_SymbolTable
create DataStack, OperatorStack, CanCreateVar, LoopSortStack, LoopCounterStack,
LoopFlagStack, IfFlagStack of ADT_Stack, InputFile, OutputFile of ADT_FILE
...
compose aspect <COMPOSED SA>  // composes semantic aspects from predefined aspects
(<PROGRAM>)
append (<ELEMENT>
  replace RefStack with DataStack // replaces stack for collaborating work
  rename INTEGER Visit with CONSTANT Visit)  // renames according to PAM syntax
append (<ASSIGNMENT>
  replace RefStack with DataStack
  rename left_hand_side PostVisit with VARIABLE Visit,
      right_hand_side PostVisit with expression PostVisit
  ignore left_hand_size PostVisit)    // ignore pushing of NULL reference
append (<INDEFINITE LOOP>
  replace RefStack with DataStack,
   Sort with LoopSortStack, Flag with LoopFlagStack)
...
end compose aspect
... // other aspect are defined and composed together
link for <COMPOSED SA> Trav to TreeTraverser, Env to SymbolTable
use aspect <COMPOSED SA> with traverser TreeTraverser
```

The obtaining of semantics for the fixed syntax is achieved in several steps: 1) select predefined semantic aspects or define new ones for desired semantics, 2) rename syntax elements and traversing aspect in the selected semantic aspects with names from fixed syntax and traversing strategy, 3) rename instances of abstract components to organize the collaboration between semantic aspects, 4) make composition from semantic aspects, 5) specify the runtime environment and translate the meta-code to the code of the target programming language, and 6) compile the semantics.

After obtaining meta-semantics (Table 1) meta-code is translated into the target programming language, taking into account the target language (e.g. C++), the implementation of abstract components (e.g. Stack), the operating system (e.g. Unix), the communications between components (e.g. CORBA), runtime components type (e.g. DLL), etc. The translation may be done by hand or automatically (desirable in common cases).

By replacing ADT names we achieve an independent working for some semantic aspects or collaboration between them through common instances of ADT. Another way to get a new semantics is to combine semantic aspects as whole black-box unit. Self-evident method is to execute several semantic aspects sequentially, for instance, we perform static semàntics first and dynamic one after that. Instances of ADT can be shared and one semantic aspect can use the results of others. More complex is a parallel execution of any semantics where we need to organize synchronization via instances of ADT.

4. Workflow Case Study

To demonstrate our approach we use a very simple workflow definition language that syntax is described with BNF (Table 2). We have two types of generic statements for describing tasks in a workflow – universal statements and specific statements.

Table 2. Fragment of BNF for simple workflow definition language

workflow	-> series
series	-> statement | series ; statement
statement	-> generic_stm | cond_stm
cond_stm	-> IF compar THEN series ELSE series FI
compar	-> expr relation expr
expr	-> const | var
generic_stm	-> universal_stm | specific_stm
universal_stm	-> u_stm_type name
u_stm_type	-> DCOM | CORBA | WEBSERVICE | MANUAL
specific_stm	-> s_stm_type name
s_stm_type	-> ASK | ANSW

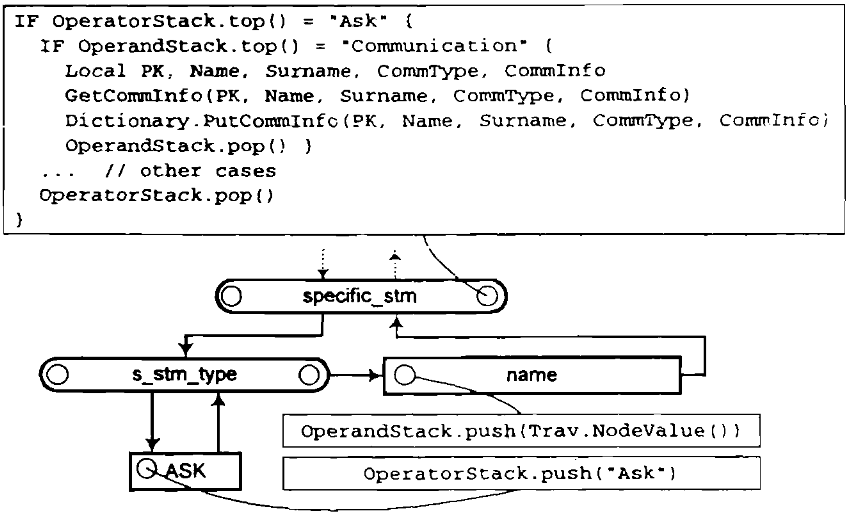
The *universal statements* are used to collaborate with external applications. The universal statement type describes the connection type: DCOM, CORBA, WEBSERVICE means automatic processing, but

MANUAL means, that a human handles this operation. The *specific statement* is used to communicate with a person – usually with a citizen who uses the particular service. There are two types of specific statements. ASK gets information from a person, ANSW sends some information to a person.

Lets us take a look at the following simple workflow:

```
WEBSERVICE Application_writing_and_submitting
ASK Communication
DCOM Application_data_control_and_update
IF Is_data_control_and_updating_successful = True THEN
   DCOM Printing_of_passport
   ANSW Positive_answer
   MANUAL Passport_handing_out
ELSE
   ANSW Negative_answer
FI
```

The purpose of this workflow is to issue a new passport for a person. The workflow has the following activities - a citizen fills in an application form and submits it to the official. It can be a paper form or a web based application. The official or the application asks from the person the communication type and address, and records data into workflow environment. Then the official verifies the correctness of the citizen's filled in form with the data in the Population Register, and if all data are correct, then the passport is issued and delivered to the citizen. Otherwise a negative answer is sent to the citizen. An example of one semantic aspect of this workflow is given in Figure 5.



```
IF OperatorStack.top() = "Ask" {
  IF OperandStack.top() = "Communication" (
    Local PK, Name, Surname, CommType, CommInfo
    GetCommInfo(PK, Name, Surname, CommType, CommInfo)
    Dictionary.PutCommInfo(PK, Name, Surname, CommType, CommInfo)
    OperandStack.pop() )
  ...  // other cases
  OperatorStack.pop()
}
```

```
OperandStack.push(Trav.NodeValue())
```

```
OperatorStack.push("Ask")
```

Figure 5. Semantic aspect SPECIFIC STATEMENT ASK

5. Conclusions

We have presented ideas for establishing a framework to deal with different collaboration problems between legacy systems. The problem is reduced to describing the collaboration (e.g. workflow) by DSL and building various tools (various semantics) for this DSL.

There are many application generators that automatically produce onventional compiler and interpreter, but we need not only those ones. It is necessary to obtain various supporting tools that are based on the language text processing. The existing formal semantics are not well accepted by language or tool designers. We have made an attempt to search for a compromise to minimize this gap. The latest related works in this field to establish tool-oriented approach are mentioned in [10, 17, 23].

We have offered ideas how semantics can be decomposed into reusable parts, and specific semantics can be composed from them, and how execution organized. We delegate most of the actual work for semantics to pre-built components (ADT). Our approach allows minimize semantics descriptions for an easier management and provides good implementation in, possible parallel, distributed and heterogeneous environment. Our approach is based on the experience received by constructing prototypes of a multi-language interpreter for conventional programming languages.

Due to practical needs, we lose some precision and benefits of classical semantic formalisms. The next step is to finish the formalization of our pproach and to compare it with other formalisms, especially with attribute grammars. Other activities have to be the designing of useful collection of abstract data types.

References

[1] Aho, A.V., Sethi, R., and Ullman, J. D. Compilers: Principles, Technigues, and Tools. Addison-Wesley, 1986.

[2] Arnicane, V., Arnicans, G., and Bicevskis, J. Multilanguage interpreter. In H.-M. Haav and B. Thalheim, editors, Proceedings of the Second International Baltic Workshop on Databases and Information Systems (DB&IS '96), Volume 2: Technology Track, pages 173-174. Tampere University of Technology Press, 1996.

[3] Aßmann, U., Genßler, T., and Bär, H. Meta-programming Grey-box Connectors. Proceedings of the Technology of Object-Oriented Languages and Systems (TOOLS 33), 2000, pp. 300-311.

[4] Biswas, B. and Mall, R. Reverse Execution of Programs. ACM SIGPLAN Notices, April 1999, 34(4):61-69.

[5] Clark, C. Build a Tree – Save a Parse. ACM SIGPLAN Notices, 34(4):19-24, April 1999.

[6] Deursen, A., Klint, P. and Visser, J. Domain-Specific Languages: An Annotated Bibliography. ACM SIGPLAN Notices, June 2000, 35(6):26-36.

[7] Ferrucci, F., Napolitano, F., Tortora, G., Tucci, M., and Vitiello, G. An Interpreter for Diagrammatic Languages Based on SR Grammars. Proceedings of the 1997 IEEE Symposium on Visual Languages (VL '97), 1997, pp. 292-299.

[8] Fischer, L. (ed) The Workflow Handbook 2001, Published in association with the Workflow Management Coalition, 2000

[9] Gamma, E., Helm, R., Johnson, R., and Vlisides, J. Design Patterns: Elements of Reusable Software. Addison-Wesley, 1995, pp. 331-334.

[10] Heering, J. and Klint, P. Semantics of Programming Languages: A Tool-Oriented Approach. ACM SIGPLAN Notices, March 2000, 35(3):39-48

[11] Karnitis, E. E-Government: An Innovative Model of Governance in the Information Society. Baltic IT&T Review, 1, 2001

[12] Kastens, U. and Wait, W. M. Modularity and reusability in attribute grammars. Acta Informatica 31, 1994, pp. 601-627.

[13] Kinnersley, W. (ed), The Language List. 1995. http://wuarchive.wustl.edu/doc/misc/lang-list.txt

[14] Kühne, T. The Translator Pattern – External Functionality with Homomorphic Mappings. Proceedings of the Tools-23: Technology of Object-Oriented Languages and Systems, 1997, pp. 48-59.

[15] Louden, K.C. Compilers and Interpreters. In Tucker [28], pp. 2120-2147.

[16] Matsuzaki, T. and Tokuda, T. CTAG Software Generator Model for Constructing Network Applications. Proceedings of the Asia Pacific Software Engineering Conference, 1998, pp.120-127.

[17] Mernik, M., Lenič, M., Avdičauševič, E., and Žumer, V. Compiler/Interpreter Generator System LISA. Proceedings of the 33rd Hawaii International Conference on System Sciences – 2000, 2000, pp. 10.

[18] Mezini, M. and Lieberherr, K. Adaptive Plug-and-Play Components for Evolutionary Software Development. SIGPLAN Notices, 33(10):97-116, 1998. Proceedings of the 1998 ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages & Applications (OOPSLA '98).

[19] Ovlinger, J. and Wand, M. A Language for Specifying Recursive Traversals of Object Structures. SIGPLAN Notices, 34(10):70-81, 1999. Proceedings of the 1999 ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages & Applications (OOPSLA '99).

[20] Paakki, J. Attribute Grammar Paradigms – A High-Level Methodology in Language Implementation. ACM Computing Surveys, June 1995, 27(2):196-255.

[21] Pagan, G.P. Formal Specification of Programming Languages: A Panoramic Primer. Prentice-Hall, 1981.

[22] Schmidt, D.A. Programming Language Semantics. In Tucker [28], pp. 2237-2254.

[23] Sloane, A.M. Generating Dynamic Program Analysis Tools. Proceedings of the Autralian Software Endineering Conference (ASWEC'9), 1997, pp. 166-173.

[24] Slonneger, K., and Kurtz, B.L. Formal Syntax and semantics of Programming Languages: A Laboratory Based Approach. Addison-Wesly, 1995.

[25] Special issue on domain-specific languages. IEEE Transactions on Software Engineering, 25(3), May/June 1999.

[26] Tucker, A.B. (ed) The computer science and engineering handbook. CRC Press, 1997.

[27] Workflow Standards and Associated Documents,
http://www.wfmc.org/standards/docs/Stds_diagram.pdf

[28] Workflow/BPR Tools Vendors http://www.waria.com/databases/wfvendors-A-L.htm

Multilanguage interpreter

Vineta Arnicâne, Guntis Arnicâns, Jânis Bièevskis
University of Latvia
Faculty of Physics and Mathematics
Rainis Blvd. 19, Riga LV-1459, Latvia
e-mail: varnican@lanet.lv, garnican@lanet.lv
and
Riga Institute of Information Technology
Skanstes 13, LV-1013 Riga, Latvia
e-mail: bicevskis@swh.lv

Abstract

The concept of multilanguage interpreter that allows with a unified method to solve popular problems of program complexity, analysis and automated testing is offered. The traditional program syntax analysis tools such as LEX and combined with specific sets of commands for semantics description are used. The source program's execution according to the given semantics is accomplished traversing parse tree and executing instructions corresponding to node type in this semantics. The given method is demonstrated on a small example. The possibility to use multilanguage interpreter for prototyping logically complicated systems is demonstrated.

Introduction

The idea of the multilanguage interpreter has risen from a plenty of problems that require to analyze the given program text and do something with it. Let us look at some of such problems.

Program translation to another programming language. To deal with this problem we use the appropriate translator or compiler [2].

Dynamic program execution. For this task we usually take interpreter. An interpreter is a program (tool) that reads a program written in source language, translates it into intermediate representation and immediately performs the operations implied by the source program.

Program beautifying and clarifying. That means that program text is transformed in a such way that user can easy catch its meaning. For instance, to display the keywords, numbers, variables in different fonts, forms or color, or show structure of the program, etc.

Determining program complexity. Frequently we need to determine program complexity according to some criteria. This task requires the program analysis [5, 6].

Creation of cross-reference tables. It is hard to imagine the building of a serious system without the various cross-reference tables. These tables can contain the following information: functions called by a given function or either functions that call a given function; variables used in a given function or either functions that use a given variable, etc.

Static program testing. Some problems in static testing are searching for the common semantics errors (syntax errors usually is caught by compilers), finding the relations between variables [5, 6, 8].

Symbolic testing. Instead conventional execution we can execute the chosen program path symbolically (real variable values are substituted by symbols). This can help to prove program correctness and is the essential step forward to program verification [5, 6, 8].

Automatic testing. Any interpreter for given programming language can be modified to record user defined tests to the data base, later automatically replay tests and compare results with correct results stored while test accumulation [1]. The other approach is to create tree from program text and automatically generate the test cases for some criteria (usually for C1) [7].

Program instrumentation with additional text. There are special code insertion is made into original program text to force program do some additional work, for instance, dynamically control arrays boundaries [5, 6, 7].

Dynamic testing supporting. Interpreters has advantage compare with compiler that the first one easier can deal with many of problems witch arise during the program execution [1, 8].

Any program is written in some source language (programming language). Conventional compiler or interpreter is designed for only the one predefined programming language. The realization of these tools are very similar based on compiler construction theory. Differences between implementations mainly are stated by the features of the programming language.

The main characteristics that describe programming language are syntax, semantics and pragmatic. “Roughly speaking, syntax deals with questions of superficial form of a language, semantics with its underlying meaning, and pragmatic with its practical use”. [3] In our case only the first two characteristics are meaningful because they dictate the behavior of the tool. In most of the implementations of interpreters and compilers both of these parts depend on each other and it is very hard to change anything in one of them without changing the other.

Our goal is to reduce efforts in creating such tools that do something with the text in source language. Especially it may be useful for researchers - quickly to build prototype or simulator for a new tool. We try to create a tool that acts like the mentioned tools and call it the multilanguage interpreter (MLI). If we reduce the problem by fixing one syntax and use it with various semantics then we could call it as multisemantic interpreter (MSI). From our point of view there is not essential difference between MLI and MSI in this paper.

The multilanguage interpreter is a program that receives the source language syntax, the source language semantics and the program written in source language and performs the operations implied by this program and given semantics.

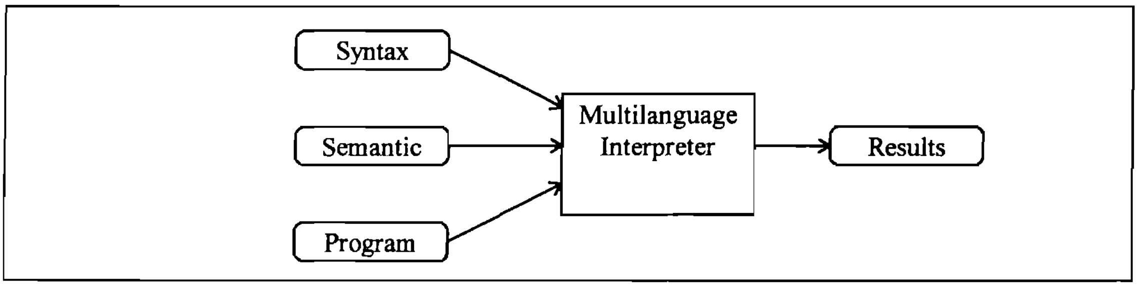


Figure 1

This interpreter can interpret programs written in many source languages. To our understanding, language is a pair <*syn,sem*> where *syn* is some syntax and *sem* is some semantics. Both are defined in a formal way specially for MLI. Some pair <syn,sem> may be senseless for a user but interpreter tries to execute the program with this syntax and semantics.

As the most useful technique for describing the language in MLI we have chosen attribute grammars. The syntax we describe by using of BNF (Backus-Naur Form), but semantics - with rules connected with terminals and nonterminals of grammar. The execution of the program is traversing through parse tree according to semantics rules and performing of rules connected with each node of tree.

The description of language is broken into two parts: syntax and semantics. The syntax of a language is the set of rules that determine which constructs are correctly formed in program and which are not. The semantics of a language is the description of the way a syntactically correct program is interpreted.

We use attribute grammars as basic concept in MLI. As metalanguage for expressing grammars we will use BNF. The syntactic structure of a given source program as generated by the grammar, can be depicted as a parse tree. Most programming languages cannot be completely specified by a context-free grammar. So we will specify in syntax part of MLI only the context-free portion of a language syntax.

Syntax part of MLI have to support functions for navigation through the parse tree (such as - getOldestChild, getNextChild, getParent) and functions for gathering the information about each node (terminal or nonterminal, name of it, if terminal - token of it).

Conceptually we parse the input token stream, build the parse tree, and then traverse the tree as needed to evaluate the semantics rules at the parse tree nodes. Once an explicit parse tree is available, we are free to visit the children of each node in any order.

'Semantics part' of attribute grammar used by multilanguage interpreter is not pure attribute grammar. It is modified for practical considerations but the basic ideas follow the attribute grammar theory. We cannot take the attribute grammar without modifications because it is oriented to building of compilers, not to building process of interpreters. From various approaches of language semantics describing in MLI is preferable to use the operational approach. This approach allows to describe the semantics in terms of such devices as abstract machine with discrete states and more-or-less explicit sequences of the computational operations. The most known representative of this approach is VDL (Vienna Definition Language). According the

VDL the abstract machine interprets a program by passing through a sequence of discrete states. The allowable state transitions are defined by a set of instruction definitions written in a special notation.

From VDL we have taken the idea of abstract machine that is represented by special data structures and set of operations with them. Abstract machine operations can be used as part of metalanguage that describes semantics. Each semantics instructions can change the state of abstract machine. The VDL forces us to put into attribute grammar high level functions and complicated data structures that allow to easier understand “semantics rules”.

In practice the semantics for MLI is the set of programs written in some metalanguage and some real programming language. These programs are written in such a way that they can be executed on computer. The evaluation rule is not associated with some grammar production rule but it is connected with the terminal or nonterminal of BNF. Each attribute is not directly associated with each distinct symbol of BNF but global stack is used to manage the values of attribute at the execution time (it is the common approach to realize attribute grammar in practical tools). The special support tool is designed for multilanguage interpreter to write semantics more quickly and compactly - the system for memory object managing.

An essential function of a compiler is recording of identifiers used in the source program and collecting of information about various attributes of each identifier. The attribute may contain the information about the storage allocated for the identifier, its type, its scope, number and types of procedure arguments, etc. The special data structure - symbol table - is used for that purpose [2]. There is a similar data structure created for MLI, too. This data structure not only can store the information but also its functions can be used as part of metalanguage in defining of the semantics rules. We have called it as Memory Object Management System (MOMS). We suppose that MOMS is one of the essential parts of the MLI that allows to make the description of semantics more quickly and understandably.

MOMS operates with some basic memory objects such as name of object, reference (handle to memory object), value (handle to byte stream that contains a value of an object), constructor (handle to object type description). Constructor may be primitive constructor or combination of primitive constructors. By using of constructors we can describe the structure and features of any memory object, for instance, type of variable, function arguments, procedure, etc.

MOMS also operates with more complex memory objects such as dictionaries, tables, memory blocks, stacks, collections.

MOMS functions we use for describing the semantics of the subject language. These functions provide definition of the features of program running environment, description of the source language basic data types, description of the source language basic operations (+, -, *, /, <, >, min(), max(), substr(), etc.), definition of scope for memory objects, sets of functions that allow to create easier user defined data type, various operations with variables, constants, functions for realizing the source language procedures and functions, and other useful functions.

MOMS is designed specially for MLI, but it can be used by other tools too.

As it was mentioned above, for interpreting the source program we need the description of syntax and semantics of object language.

We describe syntax by BNF. According this description we to generate the code for lexical and syntactic analysis of source program with goal to built the parse tree of source program. This code includes the functions for navigation through tree, too.

We describe semantics by using our own metalanguage. Our metalanguage is C like language which includes parse tree navigation functions and specific functions for memorizing any memory object in MLI. From this description we can generate code in C++ which can interpret the source program according to this semantics.

For practical considerations we decide that sometimes attribute of some grammar symbol can serve as inherited or synthesized attribute. The role of attribute depends on direction we arrive into the node. For this reason we divide semantics rule for nonterminal symbol into two functions: Pre_function that is executed if we visit node from parent or sibling node and Post_function if we visit node from child. In all Pre_functions attributes are inherited but in Post_functions - synthesized. For terminal symbols we write only one function.

So at the beginning of its work MLI makes parse tree of source program, takes the root of it and interprets the program by carrying out semantics rules.

Example

As example we will use a statement in simple poor language PAM [3]. There is only one data type - int (integer) - in this language, there are all arithmetic operations with integers (except unary minus), input, output operators, conditional and loop statements in PAM. There is semantics difference - in the language there are no declarations of the variables. Variable is declared when the value is assigned to it first.

Let us look at a small example how we can interpret the ordinary command in various programming languages - assignment statement. It looks in the following way:

z := y + 1.

Useful statements of BNF for this statement are:

```
<assignment_statement> ::= K_VARIABLE K_ASSIGN <expression>
<expression> ::= <term> | <expression> K_WEAK_OPERATOR <term>
<term> ::= <element> | <term> K_STRONG_OPERATOR <element>
<element> ::= K_CONSTANT | K_VARIABLE |
              K_LEFT_BRACKET <expression> K_RIGHT_BRACKET
```

Figure 2

In this grammar the names that consist of upper case letters are terminals (the actual terminal symbols are substituted by words starting with 'K_'). The parse tree of example statement is seen in Figure 3. The visiting order of tree nodes is shown by arrows and numbers at them.

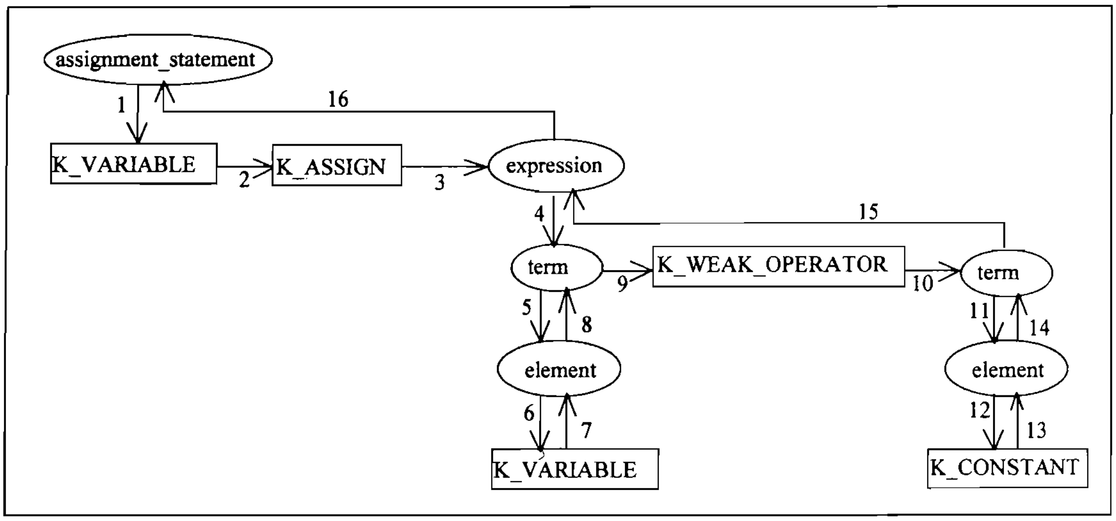


Figure 3

Executing of this statement starts by visiting the node of nonterminal 'assignment_statement'. At first the function 'assignment_statement_PRE' (Figure 4) is called. The function 'assignment_statement_POST' is called after all the subtree of this node will be visited (arrow 16). We used some attributes in the description of semantics of PAM -*flagDontCreateVars* (because in subtree function K_VARIABLE should know whether it can create this variable or not), *OperatorStack* (for storing of operations) and *DataStack* (for storing of references to variable definitions and values).

```
int assignment_statement_PRE() {  flagDontCreateVars = FALSE; return 1;}
```

Figure 4

As next the function ‘K_VARIABLE’ is called. It takes the name of variable from parse tree node and creates variable in MOMS with this name because the *flagDontCreateVars* is FALSE (was assigned in function ‘assignment_statement_PRE’). Then the function takes the reference to variable from MOMS and pushes it in *DataStack.*

```
int K_VARIABLE() {
        char* varName = valueFromTree();
        if (flagDontCreateVars)
                if(! MOMS.findVariable(variableName)) {
                        MessageBox(0,”Noninitialized variable”, varName,MB_OK);
                        return 0;}
        else
                MOMS.createVariable(variableName,”int_”);
        Ref variableRef = MOMS.getRef(variableName);
        push(DataStack,Ref,variableRef); return 1;}
```

Figure 5

The next called function ‘K_ASSIGN’ does nothing (it is empty).

```
int K_ASSIGN() {return 1; }
```

Figure 6

Function ‘expression_PRE’ pushes the label NOP into the *OperatorStack.* It will be useful in function ‘expression_POST’ as indicator of the last operation in this expression.

```
int expression_PRE()
        push(OperatorStack, char*, NOP); return 1;}
```

Figure 7

Function ‘term_PRE’ pushes into the *OperatorStack* the NOP.

```
int term_PRE() push(OperationStack, char*,NOP); return 1;}
```

Figure 8

The function ‘element_PRE’ changes *flagDontCreateVars* value to TRUE. It means that the variables existing in subtree (in the right part of assignment statement) have to be created and initialized before.

```
int element_PRE() flagDontCreateVars = TRUE; return 1;}
```

Figure 9

The function ‘K_VARIABLE’ works like it was in the previous call. The single exception is that instead of the creating of the variable the function searches for it in MOMS. If it is not in MOMS the error message will be given.

The function ‘element_POST’ changes*flagDontCreateVars* value to FALSE.

```
int element_POST() flagDontCreateVars = FALSE; return 1;}
```

Figure 10

Function ‘term_POST’ pops operations from *OperatorStack* and calls for the execution for each operation. In our case the first operation is NOP.

```
int term_POST()
        char* operat;       pop(OperatorStack, char*, operat);
        while(strcmp(operat,NOP)!=0) {
                if (executeOperator(operat,2)==0) return 0;
                delete operat;
                pop(OperationStack,char*,operat); }return 1;}
```

Figure 11

The function ‘K_WEAK_OPERATOR’ takes the operation symbol (‘+’ in our case) from parse tree node and pushes it into the *OperatorStack.*

```
int K_WEAK_OPERATOR(){
        char* value=valueFromTree();
        push(OperatorStack, char*, value); return 1;}
```

Figure 12

Then the functions ‘term_PRE’ and ‘element_PRE’ are carried out. After that the function ‘K_CONSTANT’ is called. It takes the constant from tree, creates unique name for it, stores it as atomic value in MOMS, get its reference from MOMS and pushes the reference in *DataStack.*

```
int K_KONSTANT(){
        char* text = valueFromTree();
        int value; atoi(text, value, 10);
        char* litName = new char[strlen(“ int_”)+ strlen(text)+1];
        strcpy(litName,” int_);      strcat(litName,text);
        if (MOMS.createLiteral(litName,”int_”))
                MOMS.putValue(MOMS.getRef(litName),(char*)&value));
        Ref litRef = MOMS.getRef(litName);
        push(DataStack, Ref, litRef); return 1;}
```

Figure 13

Then the functions 'term_POST' and 'element_POST' are carried out. After that the function 'expression_POST' is called. It is the same as 'term_POST'. At this moment the operator in *OperatorStack* is '+' and the function 'executeOperator' is called.

The function 'executeOperator' was created during the describing the semantics as supplement function for clarifying the description. It takes the 'argNum' references from *DataStack* and calls the corresponding function. In our case it is the function 'PLUS' which is carrying out the adding, creates reference of the result and stores it in *DataStack*.

```
#define callbasefn( fnsign, fnname )        \
        if (strcmp(funcName, fnsign) == 0)    \
                return fnname(intArg);

int executeOperator(const char* funcName, const Uint argNum) {
        Ref refArg[2]; int_ intArg[2];
        for (Uint i=0;i<argNum;i++) {
                pop(DataStack, Ref, refArg[i]);
                intArg[argNum-i-1]=*(int_*)MOMS.getValue(refArg[i]);}
        callbasefn ( "+", PLUS );
        callbasefn ( "-", MINUS ) ;         callbasefn ( "*", MULTIPLY );
        callbasefn ( "/", DIVIDE );         callbasefn ( ">", G );
        callbasefn ( "<", L );      callbasefn ( ">=", GE );
        callbasefn ( "<=", LE );    callbasefn ( "==", EQ );
        callbasefn ( "<>", NE );
        MessageBox(NULL,"Nondefined base function","executeOperator",MB_OK);
        return 0; }

int PLUS(const int_ *intArg)   {  //intArg - array of parameters
        int_ rezult=intArg[0] + intArg[1];
        Ref rezultRef=MOMS.createRef("int_");
        mm.putValue(rezultRef,(char*)(&rezult));
        push(DataStack, Ref, rezultRef); return 1; }
```

Figure 14

At last the function 'assignment_statement_POST' is called. It takes references to expression result and variable from *DataStack,* and assigns the result to variable. It turns the value of*flagDontCreateVars* to TRUE, too.

```
int assignment_statement_POST {
        Ref rezRef, varRef;
        pop(DataStack, Ref, rezRef);
        char* value = MOMS.getValue(rezRef);
        pop(DataStack, Ref, varRef);
        MOMS.putValue(varRef, value);
        flagDontCreateVars = TRUE; return 1;}
```

Figure 15

It can seem too hard and complex for such simple statement but even if the expression on the right side of statement would be very complex, it can be executed by the very same set of functions.

Practical results

Some practical results are achieved. Some various modifications of PAM language syntax is created. We use some tools created by our colleagues. One tool allows us to transfer BNF into some internal representation. This internal representation can be translated to program text that serves as input for LEX and YACC. These tools are available on many computer platforms and are very popular. The result of LEX and YACC linked together with special library that allows to navigate through parse tree we consider as syntax of given language. If the BNF is already given then creating syntax for MLI takes some hours.

The first version of Memory Object Management System and a simple MLI kernel is created. A special tool from BNF internal representation can create templates for semantics instructions. Different semantics are created for PAM. As examples we have created conventional semantics as well as special semantics that perform symbolic execution along the chosen program path.

References

1. Boris Beizer *Black-Box Testing Techniques for Functional Testing of Software and Systems*, _John Wiley & Sons, Inc, _USA, _1995, _294 p.

2. Aho A., Sethi R., Ullman J.D. *Compilers. Principles, Techniques, and Tools* _ADDISION-WESLEY PUBLISHING COMPANY, _USA, 1988. _795 p.

3. Pagan F.G. *Formal specification of programming languages* _New Jersey, PRENTICE-HALL, 1981. _241 p.

4. Herbert L. Dershem, Michael J.Jipping *Programming Languages: Structures and Models,* _Wadsworth Publishing Company, _USA, 1990, _413 p.

5. *Software Testing* Vol1, Analysis and Bibliography, _Berkshire, _England, 1979. _305 p.

6. *Software Testing* Vol2, Invited papers, _Berkshire, _England, 1979. _371 p.

7. *Conference Proceedings Eighth International Software Quality Week 1995,* _Software Research Institute, _USA, 1995.

8. J.Bicevskis, J.Borzovs, U.Straujums, A.Zarins, and E.F.Miller. *SMOTL- a system to construct samples for data processing program debugging.* IEEE Transactions on Software Engineering, SE-5, No. 1,1979, pp. 60-66.

BALTIC IT REVIEW

Journal for the Information Society

OVERVIEW: E-BUSINESS

Baltic IT&T 2000

BALTIC STATES GOVERNMENT DATA COMMUNICATIONS NETWORK

DISCUSSION OF THE EVOLUTION TOWARD NEXT-GENERATION NETWORKS

INTERNET LEARNING IN THE BALTIC STATES

www.dtnet.lv

ISSN 1407-2912
04

Development of a Communications Server: First Results and Conclusions

Guntis Arnicāns, Ģirts Karnītis, Prof. Jānis Bičevskis, Faculty of Physics and Mathematics, University of Latvia

The authors describe a program to develop a communications server, which is a set of software and hardware that allows a wide range of users in Latvia and in other countries to receive information from a variety of sources (government registers, databases, information systems) through a single contact point. A communications server identifies users, authorizes the use of the respective data, fulfills requests which involve several information sources, and assesses the costs of the process so that the appropriate financial transaction can be made. A communications server also allows users to find out where information is being stored and what kind of information it is. Users can also request and receive information from various registers without having to have any in-depth knowledge about the technical aspects of its storage.

INTRODUCTION

The need for a communications server became evident when the governments of the Baltic States were establishing their joint data transmission network [6]. One of the main tasks in this process is to obtain information about objects such as enterprises, persons, motor vehicles, etc., without having to study the database structures in any one, specific country. The concept for the communications server [1] was defined a year ago, and the project to set up the server in Latvia was begun.

Data retrieval from different, autonomous sources has been an important issue not only in Latvia, but also in other countries and even in large enterprises in recent years. The problem is a very complicated one, and solutions may require years of time and many highly qualified specialists [2, 3, 4]. The process of developing a communications server in Latvia has been a step-by-step one. Latvia has several dozen registers and information sources – some public, others with restricted access. It would have been too complex to de-

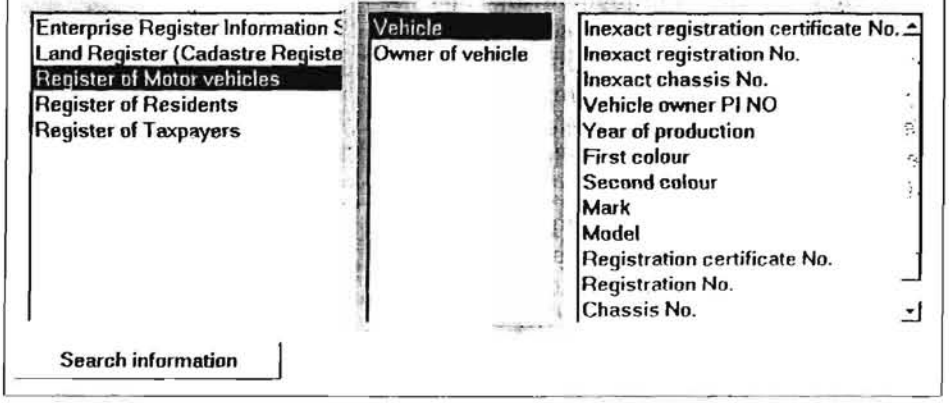


Figure 1. Registers and data objects

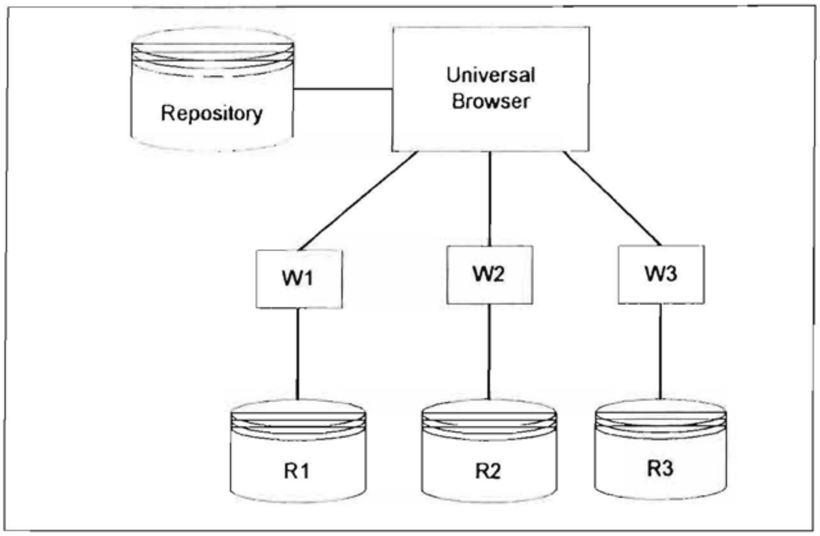


Figure 2. Conceptual structure of the communications server.

velop the system all at once, and there would have been various organizational and technical problems. The design and implementation of all of the functions of the communications server takes a long time.

THE CORE OF THE COMMUNICATIONS SERVER

The main functions of a communications server are:

1) User identification;

2) Authorization to use the information;

3) Management of user rights;

4) Fulfillment of requests which involve several information sources;

5) Evaluation of the costs of each request for billing purposes.

The implementation of the first three functions is more or less uncomplicated, but there have been considerable problems in implementing the latter two. Technologies to search for and extract data from various data sources were developed during the design phase, and they are based on Web technologies and Meta models of data sources [5].

Several principles and requirements were determined for the first version of the communications server:

- It must be possible to define a new source in a couple of days;
- It must be possible to access any type of data source;
- It must be possible to create primitive services (wrappers) to search and obtain the needed data from the source quickly and easily;
- It must be possible to tie together related data from various data sources;
- It must be easy to maintain the entire system (make changes, add new possibilities, etc.);
- The program code must be simple and short so as to reduce the possibility of mistakes;
- Initially data must be retrieved only from the WWW (from the end-user's point of view).

THE REGISTER OF REGISTERS

The register of registers is an information system which contains information from other information systems that are maintained in Latvia. It contains a great deal of useful information – IS name, content, owner, data model, relations with data objects in other information systems and in the database of the register of registers, etc.

The first version of the communications server makes extensive use of information from the register of registers. Information searches, for example, start with a high-level representation of data sources and objects stored within them (Figure 1).

Here we can see which data sources are available, which data ob-

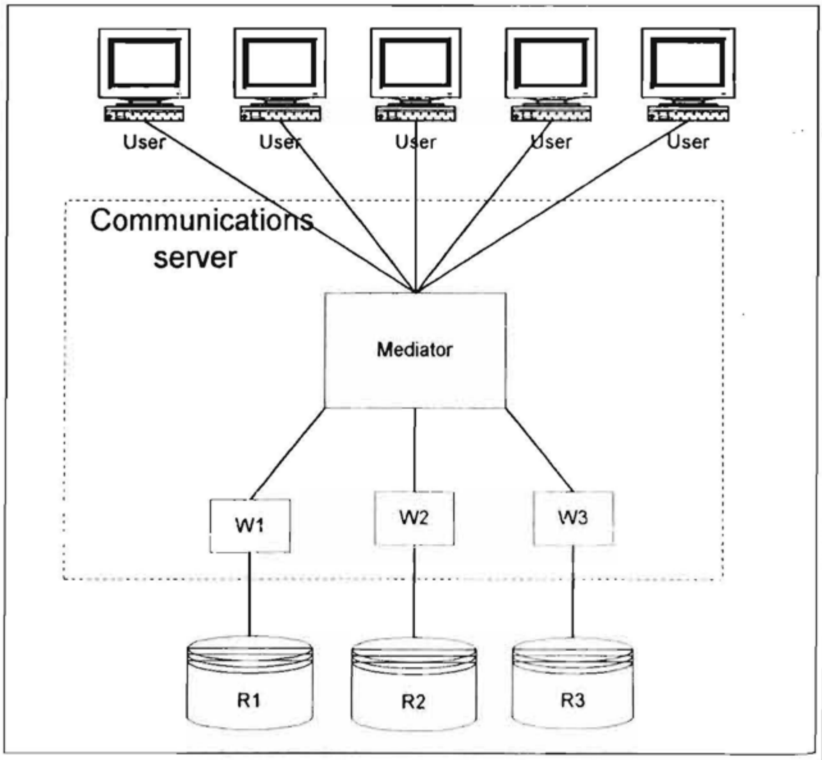


Figure 3. A universal browser.

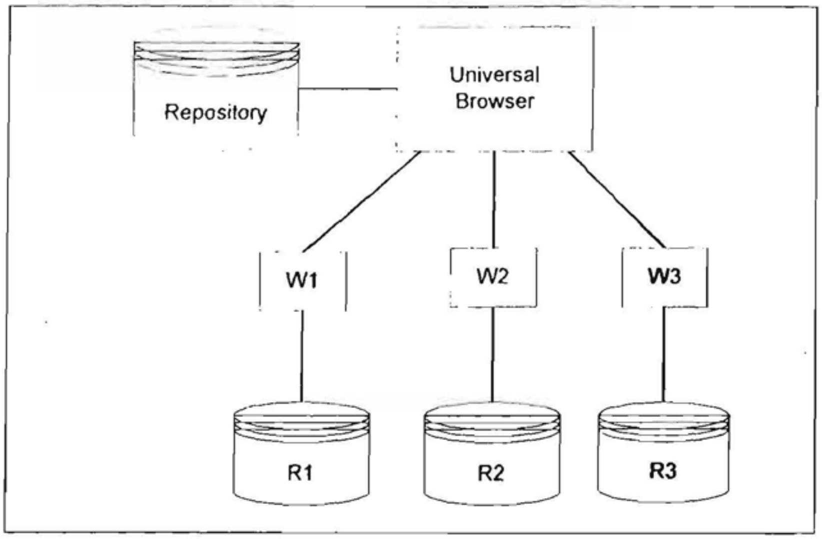


Figure 2. Conceptual structure of the communications server.

velop the system all at once, and there would have been various organizational and technical problems. The design and implementation of all of the functions of the communications server takes a long time.

THE CORE OF THE COMMUNICATIONS SERVER

The main functions of a communications server are:

1) User identification;

2) Authorization to use the information;

3) Management of user rights;

4) Fulfillment of requests which involve several information sources;

5) Evaluation of the costs of each request for billing purposes.

The implementation of the first three functions is more or less uncomplicated, but there have been considerable problems in implementing the latter two. Technologies to search for and extract data from various data sources were developed during the design phase, and they are based on Web technologies and Meta models of data sources [5].

Several principles and requirements were determined for the first version of the communications server:

- It must be possible to define a new source in a couple of days;
- It must be possible to access any type of data source;
- It must be possible to create primitive services (wrappers) to search and obtain the needed data from the source quickly and easily;
- It must be possible to tie together related data from various data sources;
- It must be easy to maintain the entire system (make changes, add new possibilities, etc.);
- The program code must be simple and short so as to reduce the possibility of mistakes;
- Initially data must be retrieved only from the WWW (from the end-user's point of view).

THE REGISTER OF REGISTERS

The register of registers is an information system which contains information from other information systems that are maintained in Latvia. It contains a great deal of useful information – IS name, content, owner, data model, relations with data objects in other information systems and in the database of the register of registers, etc.

The first version of the communications server makes extensive use of information from the register of registers. Information searches, for example, start with a high-level representation of data sources and objects stored within them (Figure 1).

Here we can see which data sources are available, which data ob-

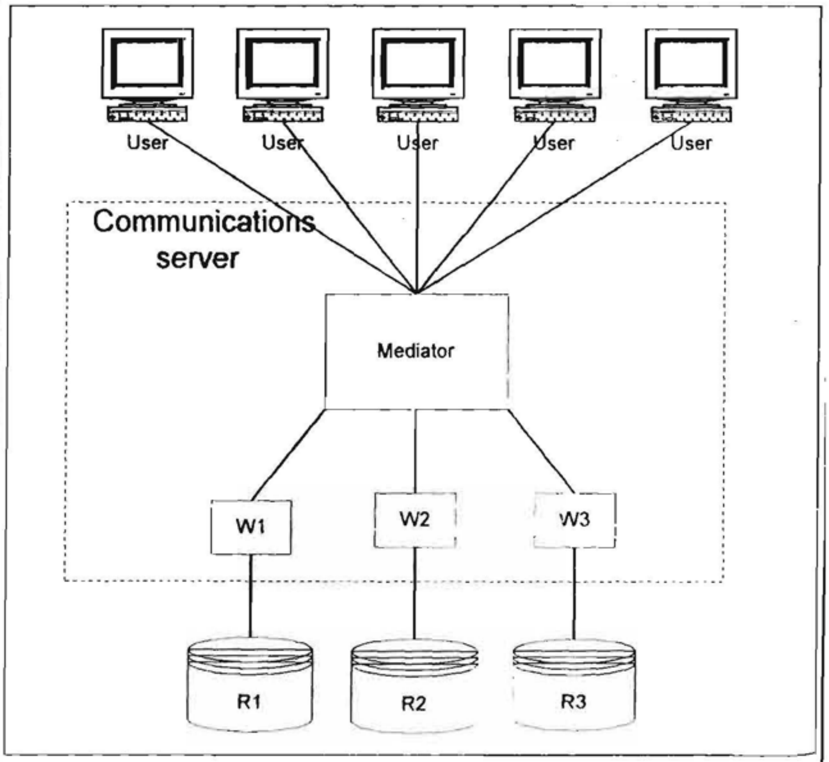


Figure 3. A universal browser.

jects are available from those sources, and which attributes can be used to describe each data object. Then we can start browsing from any data source or data object.

BASIC ADDITIONAL REQUIREMENTS FOR COMMUNICATIONS SERVERS

There are various other aspects and requirements to be taken into account in creating the first version of a communications server:

- Some data are very sensitive (only for authorized and restricted use);
- Some data are available for money.

For these reasons, we have maintained close attention over security, logging in all activities and accounting for all information that is received, the point being to enable us to settle accounts between information providers and consumers.

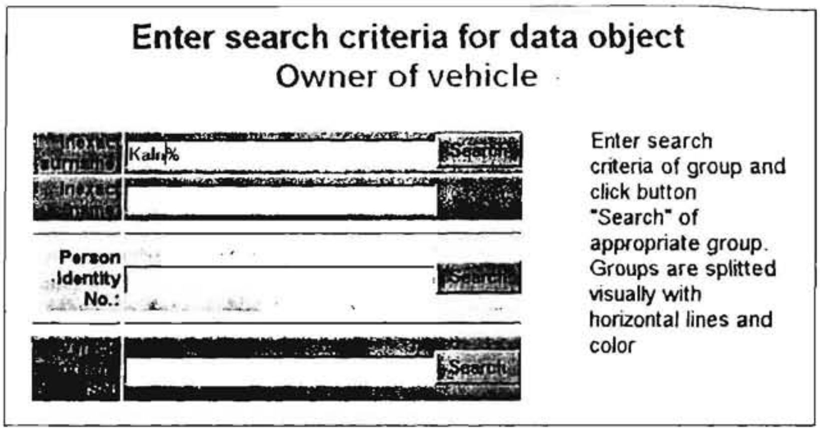
Our security system is designed to fulfil the requirements of the law, government regulations and information providers. With respect to each user, we currently define the data objects (registers, information from registers, etc.) are available, what operations (searching, retrieving) can be done, and what WWW page templates (data retrieval, combining of data from various registers, presentation of data) are available.

All user activities are logged in to special journals. The system saves information about the activity, the user who engaged in the activity, and also the related request. It is possible to track the entire history of any data object – who asked what, and what data objects and attributes were displayed.

We can calculate the costs for information consumers in those cases where the cost of information is defined Because we log in any request in some detail, we can calculate the costs of any user or provider.

TECHNICAL SOLUTIONS

The main task for a communications server is to retrieve information from data sources. A rough view of the implementation principles is shown in Figure 2. The user asks the mediator for information. Th mediator translates the request into a set of small, internal requests to data sources through wrappers. When the wrapper returns the data, the mediator establishes the information presentation and sends the WWW page to the user.



Enter search criteria for data object

Owner of vehicle

Kaln%

Person Identity No.:

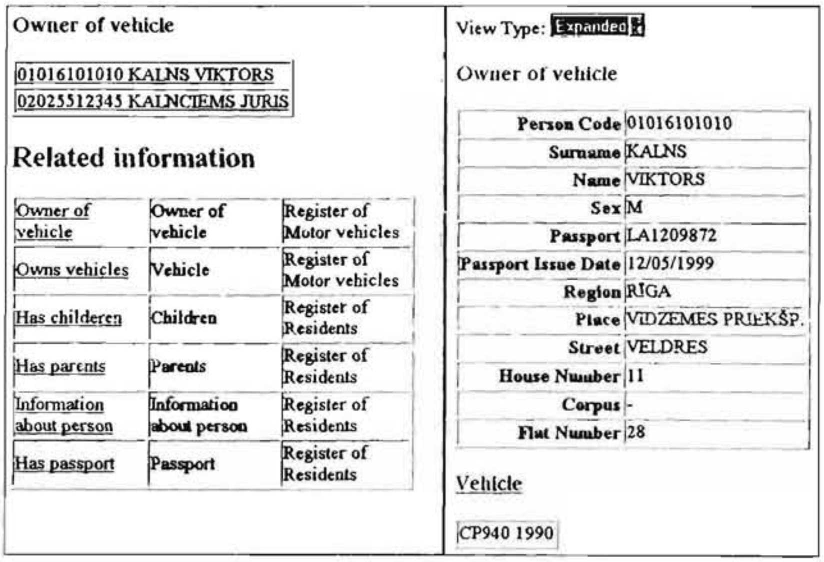
Enter search criteria of group and click button "Search" of appropriate group. Groups are splitted visually with horizontal lines and color

Figure 4. Search criteria input window.

In order to retrieve information from a data source, we must create special small programs known as data wrappers. This approach has the following advantages:

- It allows us to access the data source via different protocols and methods – ODBC, OLE DB, SOL*NET, DCOM, etc.;
- Data sources are usually suited to specific business tasks, and their primary function is not to provide access to data as requested from another system (communications server); access is limited, and only stored procedures can be used to query data; the wrapper allows us to execute only authorized functions;
- Querying the data source via functions allows us to transfer real data from the data source's physical data model to our logical data model (stored in the meta database) easily and in a form that is more easily understood by the user;
- If the data source changes, we need only to adjust the relevant wrappers.

A special browser has been de-



Owner of vehicle

01016101010 KALNS VIKTORS

02025512345 KALNCIEMS JURIS

Related information

Owner of vehicle	Owner of vehicle	Register of Motor vehicles
Owns vehicles	Vehicle	Register of Motor vehicles
Has childeren	Children	Register of Residents
Has parents	Parents	Register of Residents
Information about person	Information about person	Register of Residents
Has passport	Passport	Register of Residents

View Type: Expanded

Owner of vehicle

Person Code	01016101010
Surname	KALNS
Name	VIKTORS
Sex	M
Passport	LA1209872
Passport Issue Date	12/05/1999
Region	RĪGA
Place	VIDZEMES PRIEKŠP.
Street	VELDRES
House Number	11
Corpus	-
Flat Number	28

Vehicle

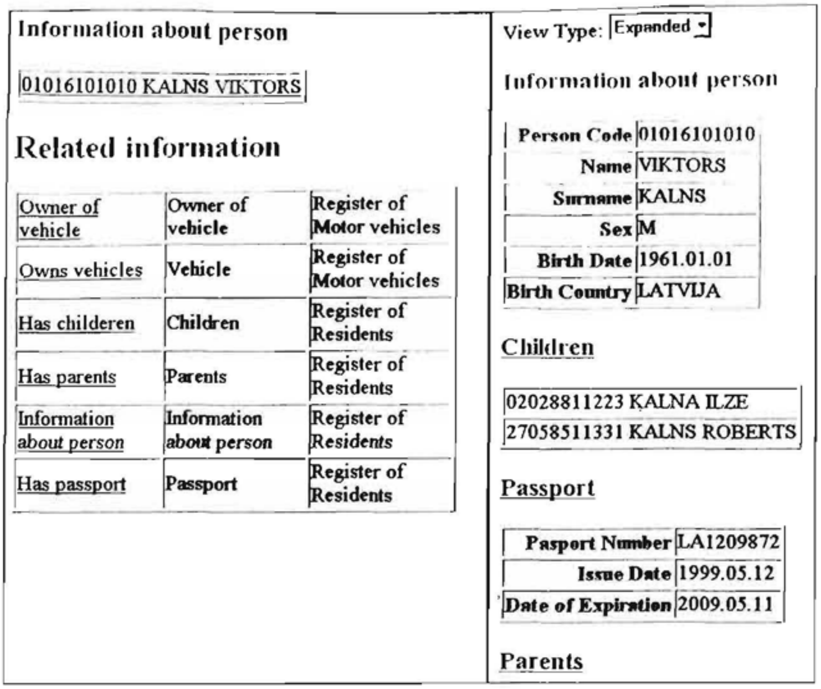
CP940 1990

Figure 5. Information about car owners.

signed for communications with a user via the Internet, and it is based on a meta model of data sources. This browser retrieves information that is stored in a meta model and generates WWW pages to communicate with the user. We can think of the browser as a driver and a repository (Figure 3).

The repository is a database which stores information about data sources, data objects in sources and relationships among them. It has functions which allow us to query a source, to screen templates (the WWW page structure) and to obtain other useful information.

The driver is a special program which generates WWW pages to handle queries at the higher level and to display information. The driver can analyze relations between data sources and merge all relevant information.



Information about person

01016101010 KALNS VIKTORS

Related information

Owner of vehicle	Owner of vehicle	Register of Motor vehicles
Owns vehicles	Vehicle	Register of Motor vehicles
Has childeren	Children	Register of Residents
Has parents	Parents	Register of Residents
Information about person	Information about person	Register of Residents
Has passport	Passport	Register of Residents

View Type: Expanded

Information about person

Person Code	01016101010
Name	VIKTORS
Surname	KALNS
Sex	M
Birth Date	1961.01.01
Birth Country	LATVIJA

Children

02028811223 KALNA ILZE
27058511331 KALNS ROBERTS

Passport

Pasport Number	LA1209872
Issue Date	1999.05.12
Date of Expiration	2009.05.11

Parents

Figure 6. Information about the person.

DATA SEARCHING AND BROWSING SCENARIOS

Let us look at a brief example of the way in which the communications server works from the end-user's point of view. The first step is to select the register and data object from which the information will be queried (Figure 1). The system asks for search criteria for the chosen object (Figure 4).

The user fills in the search criteria and pushes the "Search" button. The system searches the appropriate register for the necessary information, and the results are displayed (Figure 5).

From this screen the user can easily obtain related information from other registers, too. If, for example, the user wants information about a person from the Population Register, he needs only to click on the appropriate link, and the relevant information will be displayed (Figure 6).

CONCLUSIONS AND FURTHER MOVEMENT

The prototype of the communications server was created in mid-1999 [7]. Four registers (with test data) were connected for testing purposes. Two of them have Oracle as the DBMS, while two others use the Microsoft SQL Server. The prototype has demonstrated the effectiveness of the designed approach. The prototype of the system was much more powerful than we expected, and it can be used as the real system. At present, additional improvements have been made, and the first version of the real system has been developed. It is already in use.

Further work will lead to the development of a queries processor which can take an SQL-like query as input and return the result, as queried from multiple data sources, as the output.

Another area of work in which he have made advances is making the communications server available not only from WWW browsers, but also from custom software which uses XML to query data and return answers. ❑

REFERENCES

1. Arnicāns, G., J. Bičevskis and G. Karnitis. "The Concept of Setting Up a Communications Server", in Abstracts of Papers from the 3rd International Conference "Information Technologies and Telecommunications in the Baltic States", 1999, pp. 48-57.

2. Tomasic, A., R. Amoroux, et. al. "The Distributed Information Search Component (Disco) and the World Wide Web", in Proceedings of the ACM SIGMOD International Conference on Management of Data, Tucson, Arizona, 1997, prototype demonstration.

3. Haas, L., R.J. Miller, et al. "Transforming Heterogeneous Data with Database Middleware: Beyond Integration", *Data Engineering Bulletin,* 1999.

4. Hammer, J., H. Garcia-Molina, et. al. "Information, Translation, Mediation and Mosaic-Based Browsing in the TSIMMIS System", in Proceedings of the ACM SIGMOD International Conference on Management of Data, 1995, project demonstration.

5. Arnicāns, G. "Application Generation for the Simple Database Browser Based on the ER Diagram", in Proceedings of the Third International Baltic Workshop, 1998, pp. 198-209.

6. "The Baltic States Government Data Transmission Network: Conceptual and Methodological Considerations", Riga, 1998, 11 pp.

7. http://www.mega.lv.

G. Arnicans, J. Bicevskis, G. Karnitis, E. Karnitis

The Mega-system: integration of National information systems

Conceptual and Methodological Baselines of the Megasystem

Because a drastic improvement of quality and full interoperability of all National Information Systems are vital for the development of the country, all set of systems is being developed as a logically unified and technologically distributed information processing Mega-system with a common data field as well as unified user's interface, access principles and authorisation procedures.

All end-systems, irrespective of their ownership (various information systems, their remote data entry and access points, end-users of information) will all be interconnected through a high speed data communications network (see Fig. 1). The unified Mega-system will be spread to all regional and rural administrative centers and to number of cities, border checkpoints, ports, etc. Local authorities will be connected to the Mega-system in order to conduct direct data entry into all components of the Mega-system and to use information from all systems for local needs. Special terminals and access points (*information kiosks*) are envisaged for public access to information that is specified for general use.

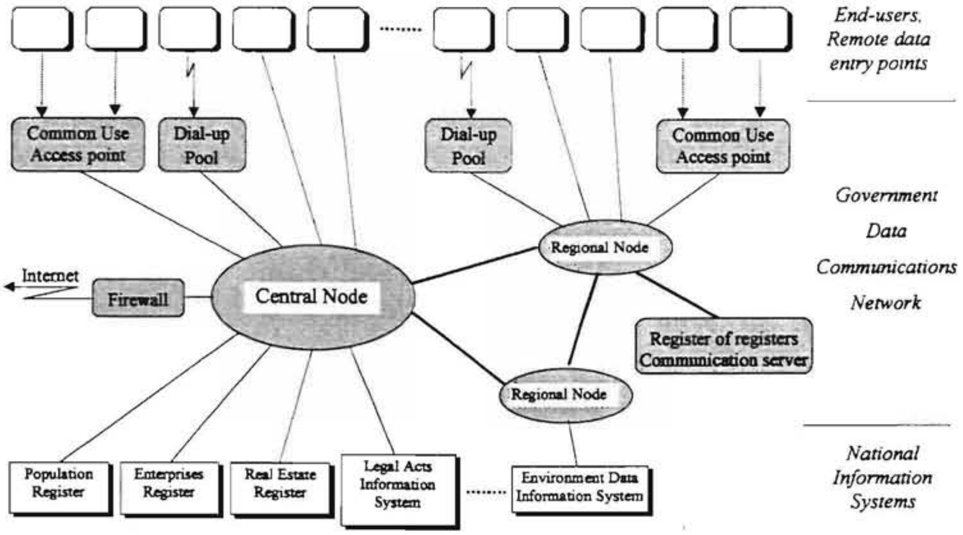

End-users, Remote data entry points
Common Use Access point
Dial-up Pool
Dial-up Pool
Common Use Access point
Government Data Communications Network
Regional Node
Internet
Firewall
Central Node
Register of registers Communication server
Regional Node
National Information Systems
Population Register
Enterprises Register
Real Estate Register
Legal Acts Information System
Environment Data Information System

Figure 1 The Megasystem

This means that it will be possible to move basic data entry and utilization procedures to places where the information has been originated or exploited as well as to provide direct access to information for everyone who has the proper authorisation (see Fig.2). It will avoid duplication of records and coincidence of records in documents and databases as well will provide united and user-friendly access to information. In addition to various information systems the Mega-system will include register of registers for collection and distribution of information on all components of the Mega-system as well as the communication server -- common central access point to information resources of the Mega-system. Conceptual and methodological propositions of the Mega-system and corresponding action plan has been accepted by special direction of the Cabinet of Ministers.

In order to realize all plans and to achieve the aforementioned goals on both state and municipal levels information systems for local authorities will be elaborated and implemented on qualitative new advanced level and connected to the Megasystem as soon as possible:

- to conduct direct data entry into all National Information Systems;
- to use information collected in all components of the megasystem for satisfaction of local needs;
- to provide electronic document exchange throughout the country;
- to envisage general access to public information and electronic contacts of population with State and local authorities.

Creation of the Mega-system is not only technological decision, in fact it means solving of number of various informative, legal, organisational, financial and qualification problems first of all. It was necessary among other issues:

- to analyse existing data flows, to formulate the functions of the Mega-system and to distribute them among information systems, to formulate demands on systems and their data structure;
- to define the subjects of various information systems and the amount of stored information, as well as the institutions that are responsible for the collection, processing and distribution of data;
- to formulate the tasks and subjects of information systems for local authorities and to elaborate several intercompatible informative models for implementation by local authorities;
- to define a unified user interface, access principles and authorisation procedures;
- to ensure data quality and security as well as interoperability with EU information systems;
- to elaborate a methodology for data verification;
- to determine the principles of electronic archives.

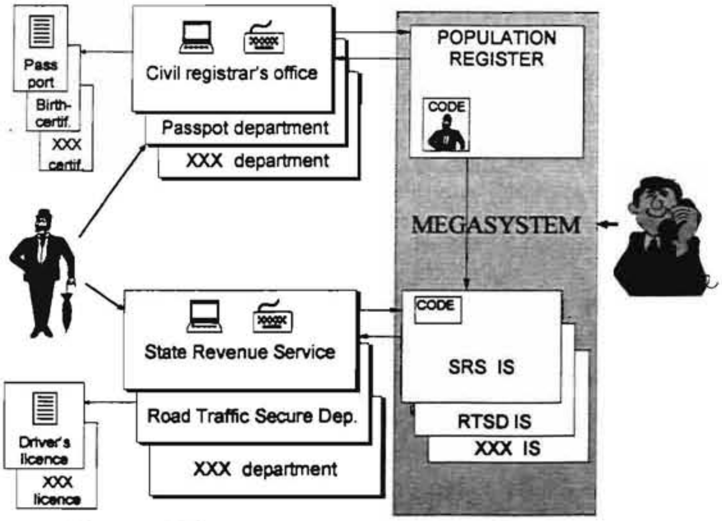


Figure 2 The Mega-system: data flows

With this conception emphasised was necessity for the country to put in order during the first stage of the Mega-system's project its main subjects registration which should go ahead of other systems elaboration: private persons (population), legal persons (enterprises, establishments, organisations), real estate (land, buildings, owners) and movable property (transport vehicles, owners), as well as state finances (taxes). In compliance with these principles five relatively primary NIS were proposed for the first stage as to-be-integrated systems:

- Population Register;
- Enterprise Register ;
- Real Property Register;
- State Vehicles Register;
- State Revenue Service Information System.

The integration of the primary NIS as well as elaboration and installation of the central body (the register of registers and the communication server) were realised during 1998-1999.

In the same time mentioned primary NIS are not declared as the only state significance information keepers. At present in Latvia operating are over 30 branch information systems by what understood are those NIS settling one branch, ministry, region or one problematic issues. These information systems will be attached to the central core of the Mega-system gradually as far as they will be prepared. It is planned to develop during the second stage connection of the Unified Information System for Local Governments, Education Informatization System and several information systems that deals with real estate.

The Government Data Communications Network for the government's and local authorities needs is an essential communications element in establishing of the Mega-system. This Network at the moment is the major part of integrated voice/data network, developed by the non-profit organisation state joint stock company *State Information Network Agency VITA* on a common transport network basis. The Cabinet of Ministers approved a complex contemporary development concept for the network in 1999.

The Government Data Communications Network must provide close and operative interoperability of all interconnected systems. Various but similar requirements to the network can be separated into several groups:

- reliability; there must be uninterrupted action time, undistorted data transmission, a guarantee of several levels of confidentiality and security of information;
- high speed data transmission; some of real time systems need guaranteed channel capacity (e.g., Vessels Traffic Management Information System);
- presence of a gateway to public data transmission network (the Internet environment) which contains a reliable firewall system.

On-line access is becoming a basic one for data transmission, but on-line connections by means of separate communications channels, however, must not be an end in itself, their usage should be well grounded both technically and economically. Connection of end-users depends on real traffic, e.g., access points of common use for several branches or dial-up connections would have to be established in cases where the traffic level is low. Connection of rural centers (villages) will have to be done on a selective basis, and in many cases local centers will be able to participate via dial-up connections or by use of diskettes to exchange and update information.

The Concept of Communications Server

A communications server is a set of software and computer equipment that allows a wide range of users (both in Latvia and in other countries) to receive information from a variety of sources (government registers, data bases, information systems) through a single contact point. A communications server identifies users, authorizes the use of the respective data, fulfills a request that involves several information sources, and evaluates the cost of the process so that the appropriate financial transaction can be made. A communications server allows users to learn where information is being stored and what kind of information it is, as well as to request and to receive information from various registers without any need for in-depth knowledge about the technical aspects of its storage.

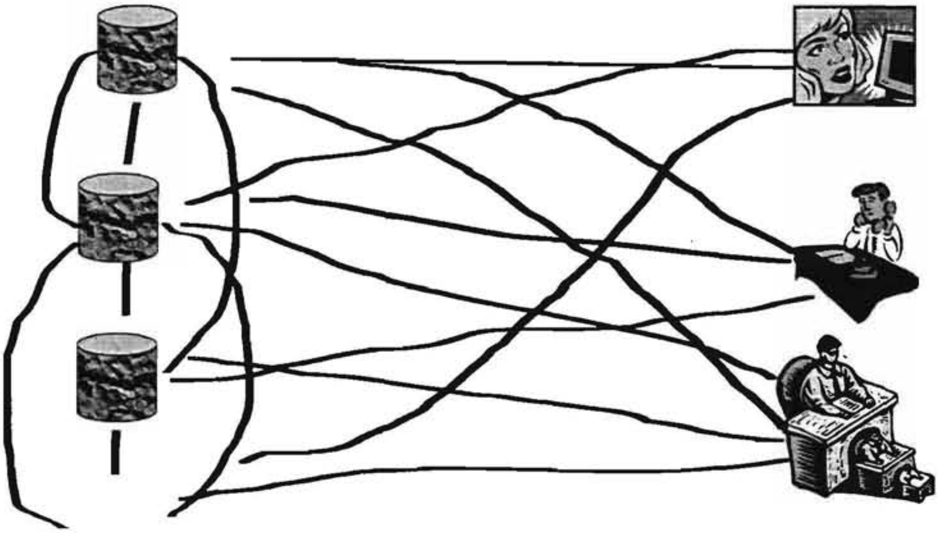


Figure 3. Communications between many registers and many users

The need to establish a communications server became apparent when the governments of the Baltic States were setting up their joint data transmission network. In order to allow institutions in one country to obtain information about objects registered in another (enterprises, persons, motor vehicles, etc.), it is useful to receive the necessary data from a single information source, without having to study the data base structures of the other country. The use of the communications server, as has been seen through the elaboration of an integrated state significance information systems project, is also of significance within one country, because it provides a universal resource for information exchange among various information systems.

Problem identification

The need to establish a communications server was noted in the national program "Informatics", as well as during the elaboration of two major projects – the Baltic States Government Data Transmission Network (hereafter in the text – the Network) and the Integrated State Significance Information System (hereafter – the Megasystem). The goal in establishing the network is to provide fundamental improvements in the exchange of telecommunications and data among the administrative institutions of the Baltic States. During the first phase of the project (1998 and 1999), universal solution is being set up to provide for the exchange of data among Latvia's Company Register, Motor Vehicles Register and Lost Motor Vehicles Register, as well as between these registers and the related international information structures. So far this has involved three concrete activities:

1) Accession of the Latvian Company Register to the European Business Register (EBR);
2) Cooperation between the Motor Vehicles Register and the related European-level structure EuCaris, as well as the establishment of a motor vehicles insurance system in Latvia (the so-called "green cards");
3) Improvements to the system whereby lost and stolen motor vehicles are registered in Latvia, including a connection to the international data bases of Interpol in this area.

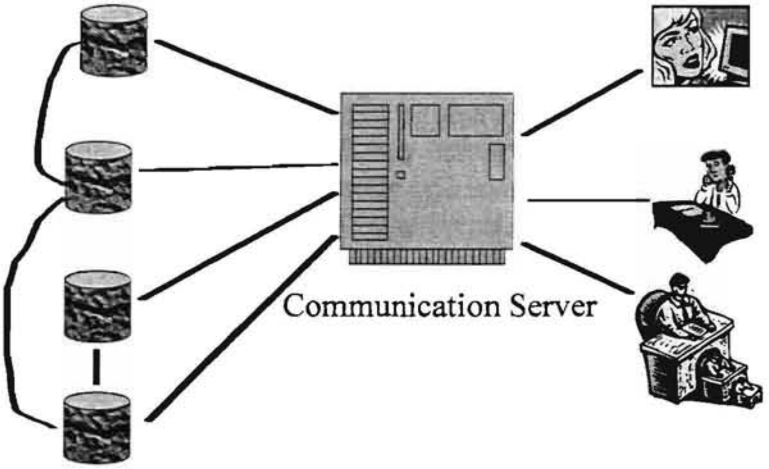


Figure 4 Communication server: the principle

During the second phase of this project, between 2000 and 2002, more work will be done to include Latvian registers into the Network and to integrate them into international information structures. In the second phase, the plan is to place the Population Register, the Lost Persons Register, the Lost Personal Documents Register, the Educational Documents Data Base, the Visas Data Base, the State Statistics Information System, the Consular Information System, the Health Care Information System and the Narcotics Information System on the Network.

In a situation where information from various sources is available on the Network, but users have no knowledge about the technical details of storing that information, there is an obvious need for a universal solution, and that is where the communications server comes in. The main requirement for a communications server is that it must allow users to formulate their information requests in a simple way and to receive responses to those requests without having to understand the technical aspects of the process. Users are not, after all, informatics specialists; they are employees of other administrative structures of the state, and there is no reason to think that they know anything about the way in which data objects are distributed among the registers of another country. We can expect both standardized and wholly unpredictable requests in this process. In terms of the urgency of requests, we can expect demands for on-line responses that require rapid response, as well as requests for off-line responses that can take hours or even days to fulfill. Needless to say, in setting up the communications system we must provide for all aspects of information confidentiality and user authorization.

The setting up of the communications system is important not only in the context of the Network, but also in the context of the Megasystem, which is a universal resource for the exchange of information among various information systems within a single country.

The concept of the solution

The communications server, which is illustrated in Figure 5, is an Internet resource point. Users of the server can access it via various protocols – HTTP, CORBA, DCOM, SMTP (E-mail) and FTP. The server provides users with an opportunity to find out where information is stored and what kind of information is available, and then to request and receive information from various registers without studying their structure. Because users may have access to sensitive information, users are identified with certificates, and all data transmissions are coded.

Users who wish to have access to sensitive information before work with the system is begun must receive a certificate that corresponds to the X.509 standard. The certificate must issued for a specific period of time (usually one year) by a specialized institution (presumably in Latvia this would occur under the supervision of the Constitutional Defense Bureau). Certificates of this kind contain information that identifies the user, and they are virtually impossible to forge. The certificates are used to code data and to identify the user. Latvia's communications server will use a standard coding protocol such as SSL.

A user of the communications server sends information requests to it and receives responses from it. This can happen both on-line (HTTP, CORBA, DCOM) and off-line (HTTP, E-mail, FTP).

In the on-line regime, work with the communications server is based on the following structure: At the beginning of the process the user is identified. This means that the user sends his or her certificate to the communications server, which reviews it and specifies the user's rights. If the user does not have a certificate, then he or she can access the communications server as a guest and receive a limited amount of information from it. Next the user requests information. The communications server once again identifies the user and, on the basis of the level of the user's authorization, makes the appropriate requests to the data registers, sending the response to the user when it is received. The register receives not only the information request from the communications server, but also the user's certificate, which means that the register itself can identify the user and the user's level of authorization. The result of this is that the register provides only that information to the communications server for which the user is cleared.

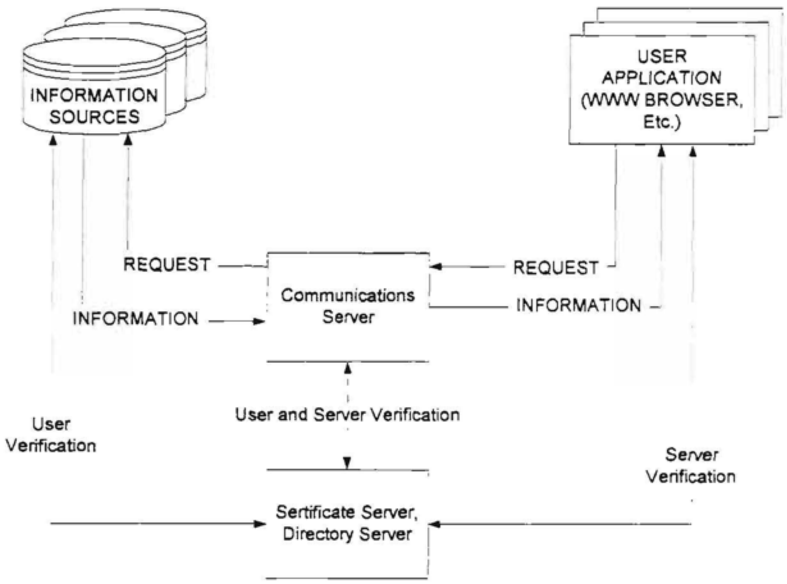


Figure 5 The operational structure of the communications server

In an off-line regime, the user requests information via HTTP, E-mail or FTP. During periods of time when it is less busy (usually at night), the communications server processes the request – identifies and verifies the user and then requests the respective information from the information registers. The response is sent to the user via E-mail, or it is stored until the user asks for it on-line.

The main advantage of an on-line regime in this process is that information can be obtained immediately when the need arises. This system can be used in cases when the speed at which a response is received is of importance, either from the point of view of the system (e.g., at border control facilities), or from the point of view of the operation (e.g., an application in which the registration number of an automobile is entered and information is received about the automobile from the Road Traffic Safety Department so that it need not be entered a second time).

The advantage of the off-line regime is that registers can even out the volume of work that is required, given that at night there should be relative few on-line requests for information. Off-line requests can also be sent in by users who have dial-up Internet connections, thus reducing costs. It is advisable to make off-line requests less expensive than on-line ones so that users are motivated to use the off-line system.

The functions of the communications server

We can specify five main functions for a communications server:

- User identification
- Authorization with respect to the use of information
- Management of user rights
- Fulfillment of requests that involve several information sources
- Evaluation of the costs of each request for billing purposes

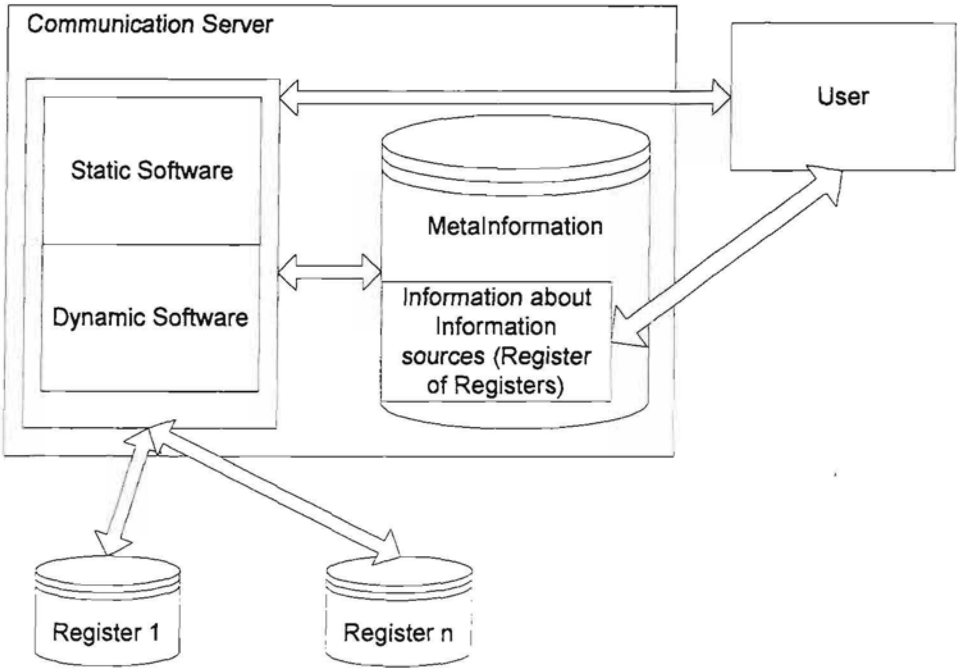


Figure 6 Architecture of communication server.

User identification in a communications server

As was noted before, user identification involves X.509-standard certificates. In order to ensure that the certificate mechanism is operational, a communications system needs both a certificate server and a directory server. The former is a server that belongs to the certifying organization, generating and maintaining electronic certificates – both server certificates (issued to the server) and client certificates (issued to the user). The latter is a server in which the public keys of the certificates are stored, along with information about certificates that have been issued – when a certificate has been issued, to whom it was issued, and whether the certificate is valid or has been revoked.

The directory server is available to any interested party. For example, if a WWW server has been issued a certificate, any WWW user can ascertain that the server is secure. If a WWW client has been issued a certificate, in turn, the WWW server can ascertain that the client is authorized to work with the server. Both the client and the server can check the validity of the submitted certificates by looking them up in the directory server.

Work with certificates in WWW applications involves SSL (Secure Socket Layer) technologies, which are supported by most WWW servers, as well as the main WWW browsers – Netscape Navigator and Microsoft Internet Explorer. SSL technologies provide the following components of secure communications:

1) **WWW server approval**: A user can ascertain the fact that the WWW server is secure and that it can be entrusted with confidential information;

2) **The privacy of information**: The entire information flow between the client and the server is coded, using a unique session key. The session key is coded by the server with the client's public key in order to send the respective information to

the client in a secure way. Each session key is used in only one session, which makes it difficult to decode the information without authorization. The information, in other words, cannot be viewed by unauthorized persons, even if it is intercepted on its way between the server and the client.

3) **The integrity of the information**: Both the server and the client calculate the control code on the basis of the content of the information, and if the information has been changed en route, the codes do not match. This means that the receiver of the information sees precisely the same information that was sent by the sender.

Secure data exchange between the WWW server and the client occurs in the following way when SSL technologies are used:

1) The client sends a request for data exchange to the WWW server;

2) The server in response sends its certificate to the client, asking for the client's certificate if appropriate;

3) The client checks the validity of the server certificate through the digital signature of the certificate server, sending the client's own certificate to the server if necessary;

4) When the authorization process is complete, the client sends the session key to the server, coding it with the public key of the server;

5) Both the server and the client know the session key, and further data flow between the server and the client during the respective session is coded with the session key.

The certificates of the server and the client are exchanged quickly and without any involvement by the user. The same is true with respect to an exchange of certificates among other applications.

When information is requested from the communications server (through the WWW or otherwise), the process occurs in the following way:

1) The user is identified through the aforementioned protocol, and the communications server checks the user in the directory server.

2) The communications server has a data base which records user rights, and the server uses this data base to specify the authorization level of the specific user. In carrying out the user's request, the communications server checks the user's rights in its own data base and, if the necessary level of authorization is there, then the request is sent along to the concrete register.

3) The register is also sent identification data about the user who has requested the information.

4) The software in the register checks the information in the directory server and authorizes the user.

5) According to the level of the user's authorization, either the request is carried out and the result is returned to the communications, server, or the communications server is told that the user does not have the right to carry out the request.

6) The communications server returns the result to the user.

A user can also request information from the register directly, without passing through the communications server. In that case the operational mechanism is similar:

1) When the information is requested from the register, the user must supply identifying information (a certificate).

2) The software in the register checks the information in the directory server and authorizes the user.

3) On the basis of the user's authorization and the level of his or her access rights, either the request is fulfilled and the result is sent back to the user, or the user is sent information saying that he or she does not have the right to receive the data.

This mechanism ensures that there is no need for the user to reintroduce identification each time a new request is made. In each session, the user is identified on the first occasion that a request is made with respect to a confidential data source, and in later requests the information is sent on to all of the respective information sources. Another advantage of the mechanism is that there is a centralized method for distributing user rights, as well as a unified policy with respect to this. It's also true that the user's rights do not change depending on the way in which he or she accesses the information – via the WWW, via a different application, or through some other method.

Management of user rights

The rights of users can be divided into several categories:

- The right to obtain information about what is stored in a concrete register – provided that the information is publicly available;
- The right to obtain information about one entry in one table in one register, based on the unique identifier of that particular entry;
- The right to obtain a list of data from one table in one register, selected on the basis of specific criteria;
- The right to obtain a list of data from several tables in a single register (whether the link exists or not);
- The right to obtain information from several tables in one register that are linked through a specific relation, the data being chosen on the basis of specific criteria;
- The right to obtain information about one object from several registers on the basis of the primary key of the object;
- The right to obtain information about the existence of a link among specific objects from various registers;
- The right to obtain a list of data that are selected on the basis of criteria entered by the user, the data coming from several tables in several registers that are mutually linked.

The obtaining of information can be differentiated at four levels:

- A response as to whether the requested information has been found or has not been found;

- A response as to how many entries have been found;
- The primary keys of objects;
- The data that is being requested.

Each of these levels provides a different volume of information, and there are instances when the jump between proximate levels is quantitative, while in other instances it is qualitative. We could consider four different requests here:

> "Does individual X own an automobile?"
>
> "How many automobiles does individual X own?"
>
> "What automobiles does individual X own?"
>
> "Does individual X own automobile Y?"

The management of user rights is intentionally divided up so that it occurs in several places. The communications server has its own user management module, in which it stores information about the right of users to make various kinds of complex requests. Information about the right of a user to receive data from a specific register is stored either in the communications server or in a concrete register. The place where information about user rights is stored is harmonized between the communications server and the register. Because it is expected that before a register issues information, it will want to check the user's rights to use the information, then information about the user's rights with respect to a specific register will usually be stored in that register. From the perspective of centralized management, it would be better if information about user rights with respect to all registers were stored in the communications server. For various organizational reasons, unfortunately, this is either impossible on only partly possible. Information about user rights is stored both in the communications server and in the registers themselves.

The communications server is designed to work with both of these options, as well as with a combination of them, and the following scheme emerges:

- The communications server checks the right of the user to make a request in the first place, as well as the right of the user to seek out a link between objects in various registers;
- The communications server checks whether the user rights with respect to the concrete register are stored in the communications server or the register;
- If the rights are stored in the communications server, then it checks the rights before it sends the request to the register;
- If the information is stored in the register, then the register checks the user rights before it fulfills the request;
- If the rights are not stored in the register, then the register can, if necessary, receive information about the rights from the communications server in order to be able to check the rights of the respective individual to make the request.

Because it is possible for users to connect to the registers not only via the communications server, but also directly from an application, and because it should be true that in both instances the user has the same authorization to obtain information, then the check of whether a user has the right to obtain information from a specific register should occur not in the communications server, but in the register itself.

Information requests and the obtaining and depiction of information

The basic mission of the communications server is to provide users with access to various information sources so that they can obtain data from them. Let us take a look at the problems that arise in this process, devoting particular attention to the submission of requests and the obtaining of responses, and leaving aside the issue of user authorization, control over data access, registration of who has asked for information and what information has been requested, billing issues and such matters.

Information sources

An information source or resource facility can be any information system or data base from any organization. There are administrative regulations concerning the organizations, information systems and data bases that are included in the communications server's network of services.

Over the course of time, the number of information sources can reach into the tens or even hundreds of sources. In Latvia alone there are already several dozen government registers, and their number may increase. Communications servers should also provide access to certain foreign information sources, as well as to the data bases of various other organizations in Latvia; these, too, could be included in the range of services provided by the communications server.

The communications server itself does not have an information sources. Each information source is primarily meant to carry out concrete and specific functions inside the respective organization Information systems and data bases that are used in an organization are chosen, designed and optimized specifically for the needs of the respective organization. They may not be aimed at providing information to other entities, but if such an opportunity is intended, then it can be very specific, and many limitations can be applied to it. This means that the communications server must adapt to the information sources, and not vice-versa. Of course some information sources can upgrade their information systems and optimize their data exchange procedures in order to meet the communications server's requirements.

Information sources that are part of the communications server's network can differ in terms of significance and volume. The more significant a data base, the better must be cooperation with it. The size of data bases must also be taken into account, because it has much to do with the respective data processing mechanisms.

Another key issue is the quality and stability of information sources. Information systems can involve a wide variety of technologies, and they are of varying ages. Depending on the resources that have been invested, some are of a higher quality and some – of a lower quality. Of course, it is easier to make contact with a high-quality information system and data base that have been designed with modern technologies than with systems that are old and of a lower quality level. A communications server must certainly be ready to deal with information sources that are unstable, that make errors and that in some instances are not even accessible.

Information systems can be designed with various systems, they may have various data bases, and their use may involve various operating systems and computer technologies. A communications server must be prepared to handle these problems, although this is no longer the worst possible difficulty, given that many different solutions are in existence.

Information can be stored in a wide variety of formats – that is the next issue. The most popular method for data storage is still relation data bases. Object-oriented data bases, static WEB pages and dynamic WEB pages that are generated from an internal format are becoming rapidly more influential. We must not, however, forget other information storage methods such as files of many different structures.

A concrete information unit and a logical group of information units can be doubled, stored in various formats, coded in various ways and stored in such a way that some of the information is kept secret. Information can be contradictory either within a single information system or among various information sources. This means that in the future the field of communications servers will have to involve various laws and data processing algorithms that are based on the technologies or artificial intelligence.

All of these aspects serve to demonstrate how serious is the issue of various information sources being highly varied. It should also be added that this heterogeneity exists among more than just information sources. The same situation can exist within a single register or a single organization.

It must also be remembered that each information source exists fairly independently. It can be updated, changed or liquidated, it can be created anew, its operations can be suspended for a while, or it can be withdrawn from cooperation with a communications server. This means that a communications server must exist in an environment that is not only highly varied, but also is extremely changeable.

Users

For our purposes, we will say that a communications server user is any subject that wishes to obtain a service from the server.

Users are usually differentiated on the basis of their level of authorization to obtain specific information from specific information sources. These rights are regulated by law and by other normative acts, and they are managed by a specific user management bloc within the communications server.

From the perspective of the communications server, another very important user classification is based on a different aspect – the way in which the user requests information and the way in which the user receives a response. A communications server should be operated on the basis of the principle that it is there for the convenience of users, not vice-versa. This principle means that the server must be ready to receive information requests of a great many varieties and forms, and it must be ready, every time, to provide a response that is convenient for the user in terms of its type and form.

Requests and responses

A communications server must be ready to accept information requests that are stated in various ways and forms. The main operational regime for communications servers is an on-line connection, but this can involve a dedicated line to the communications server, dial-up access to the server, or a connection through informational networks (the Internet, the Latvian State Significance Data Transmission Network (VNDPT), or the networks of other national, global or organizational networks). We must also remember other ways to submit a request – E-mail, a request submitted on an electronic information carrier such as a diskette, a written request submitted on paper, or even an oral request.

Responses to various requests can be prepared in the same format as the original request. It should be added, however, that the user must have the right to select the method of response, irrespective of the way in which the request was submitted. Limitations on the ways in which requests and responses are formatted can be specified by administrative regulations, but in terms of technologies, a communications server must be prepared for all kinds of cooperation methods.

The forms of requests and responses can be highly varied. The most popular cooperation form is probably a WEB page, both for requests and for responses. This form of cooperation can be highly varied, and this is underpinned by existing WEB-type applications. The use of special procedures and functions may also be important when the procedure itself has parameters that specify the request and its result (i.e., the response to the desired request as specified by the parameters). Cooperation can also occur in the following forms:

1) Special applications that can work with the communications server;
2) Active objects that can work with the communications server and can be used in the client's applications;
3) Files with requests that are recorded in a specific format or response files in a specific format;
4) A group of files (including even data bases) for the requests and the responses;
5) Paper documents in an agreed format for requests and responses;
6) E-mail, which can be seen as a modification of items 3, 4 and 5 on this list.

It is commonly held that requests from a user can come in a dialogue regime from a human user and in an automated regime where the user is an application on the user's computer.

There must also be plans to work in a synchronous regime (request-wait-response) and in an asynchronous regime (request-processing over a specific period of time-report to the user about the availability of a result-response), because this ensures more efficient work for the user and the communications server alike, especially when it comes to processing large and complex requests.

In work with the user thought must also be given to such aspects as the various levels of preparedness among users, the language of communication, the respective text coding formats, the abilities of the user's computer equipment, operating systems and applications, and limitations in all of these things.

In other words, the main mission and, at the same time, the main problem that a communications server must handle is the way in which many different kinds of requests can be handled, submitting processed information from various information sources that sometimes are not compatible, and submitting a result to the user in the desired type and form.

Information about information

As the number of information sources available through the communications server increases, an overabundance of information can quickly occur – one in which even the administrators of the communications server can get lost. It is necessary to classify all of the information sources and the information that is contained therein, keeping firmly in mind that information sources can change.

Communications servers must have data source repositories that contain formal descriptions of the sources, their properties, the data that are contained within them and the properties of the data. These repositories must be very flexible, it must be able to change them easily and quickly so that changes in the surrounding environment can be monitored. If there is to be a proper reaction to user requests, other parts of the communications system must be able to adapt to changes in the repository in a dynamic way.

The repository is not, however, meant only for internal use in the communications server. The user, too, must know where and what he can receive (of course, within the limitations of the user's authorization). This means that the communications server must also, so to speak, provide information about information. Using forms and terms that the user can understand, the server must describe the information that can be obtained and the ways in which it can be requested. There must also be efforts to link the various request formulation mechanisms as closely as possible to the repository, thus making easier the work of a user who takes advantage of the communications server's services only seldom.

Users often don't care where and how the desired information is stored. This means that the communications server must satisfy requests that concern information from many different sources. The repository, therefore, must also describe the links between the sources, as well as the ways in which various contradictions among the sources can be resolved, data be converted, etc. The repository must be an entity that makes it possible to consider all of the sources in a communications server to be one, big data base.

The abilities of the communications server

A communications server is a dynamic system which must work in a highly changeable external environment. A communications server must be much more flexible and dynamic than a day-to-day system, because it must work with highly heterogeneous external information systems that keep up with rapid technological changes. When it comes to technologies, communications servers must be a step ahead of other systems, because otherwise it may turn out that the communications server ends up unable to perform its functions.

The goal of this paper is not to describe the internal architecture and ideology of communications servers precisely. The establishment of such systems is a very serious process throughout the world these days, and various solutions are being sought out that are linked to the following technologies:

- Distributed Dynamic Systems
- Distributed and Dynamic Objects
- Dynamic Object-Oriented Programming

- Reflection
- Domain Specific Programming Languages
- Artificial Intelligence

Many of these technologies are still quite new, and they are still being developed. This means that not all of them have ready-made tools that support various properties or functions of the technologies. Some tools exist, some are at the prototype stage, while some have already become popular among professionals (this is particularly true of prototype tools that are designed at universities and research laboratories in order to test the latest technologies). In the design of a communications server it is worthwhile to such modern technologies and research results as the Multilanguage Interpreter and the Database Browser Generator.

Evaluation of requests for billing purposes

A billing system is part and parcel of the mechanism whereby a communications server fulfills requests. When a specific request is fulfilled, the system not only does what has been requested, but it also automatically calculates the resources that are used in the process. Within the communications server, a price has been attached to every resource, and it can change on the basis of the volume of information that has been requested, the time of day when the request is filed, etc. The price of each request is calculated automatically and stored in a journal that then is used for billing purposes.

A resource is an information request to a register. The price of resources changes on the basis of the type of the request, the complexity of the request, the register that is involved, etc.

Uses of a communications server

There are three major ways to use a communications server:

- As an international resource facility that can be used to access information from Latvian registers;
- As an internal resource facility that can be used to search for information in registers;
- As a way of setting up cooperation among various registers.

The need to access information from Latvian registers via a single contact facility is the main reason for elaborating the communications server. Of course, this is more than just a trivial solution in which a single Internet application is designed for connection to other registers via their Internet addresses. This simplified design does not deal with the main issue – the ability to collect information from various sources (i.e., various registers) without the user having to hook up to each register separately. The information that a user needs is collected from the various registers by the communications server, and the user himself may be completely unaware of the technical details of this process. Thus the communications server is needed by employees of foreign institutions in order to obtain information that is stored in Latvia's registered.

A second use for the communications server is the fulfillment of domestic information requests in Latvia. The previously described situation in which users do not want to or are unable to understand the technical details of information storage is typical among the personnel of Latvia's administrative structures. Of course, given the fact that access rights to authorization may vary for foreign users and Latvian users, the communications server sets out a unified set of requirements in this area, and solutions are the same for both groups of users.

The third way of using a communications server is to use it in order to exchange information among various registers. It is obviously irrational to maintain communications channels and to conduct information exchange individually with each of many registers that are mutually linked. It is much more rational to set up a centralized contact facility – the communications server – which is linked to all of the registers and through which information is exchanged among them.

Register can be connected to a communications server via different ways. Every register that participates in the data exchange procedure can have its own data base in which those data that are intended for transfer to other registers and for publication can be separated out. The data base can be maintained by a separate computer or server so that approaches to the public data base do not hamper work with the basic data base of the register. Data from the basic data base are regularly copied to the public data base (an automatic replication mechanism). This solution is rational not only from the perspective of using communications channels; it also ensures:

- That the fulfillment of external requests does not hamper the work of the register;
- That there is higher security, i.e., that in the case of unauthorized access, the basic data base is not damaged.

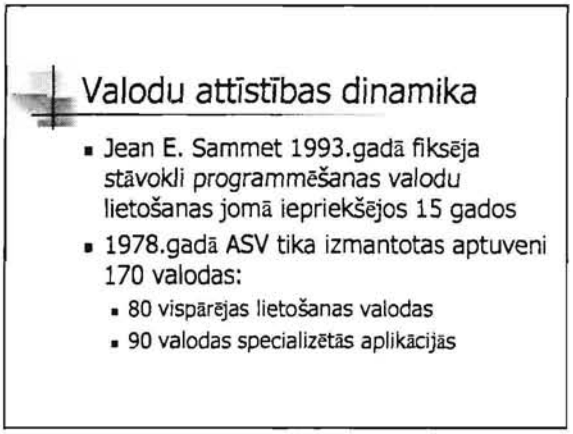
The link between the communications server and the public data base can be implemented on the basis of various technologies, such as DCOM object calls, MS Transaction servers and Oracle SQL*NET. User authorization is provided via a certificate server, a directory server and the Lightweight Directory Access Protocol (LDAP).

Domēnspecifisko valodu izmantošanas iespējas

Guntis Arnicāns
Latvijas Universitāte
Fizikas un matemātikas fakultāte

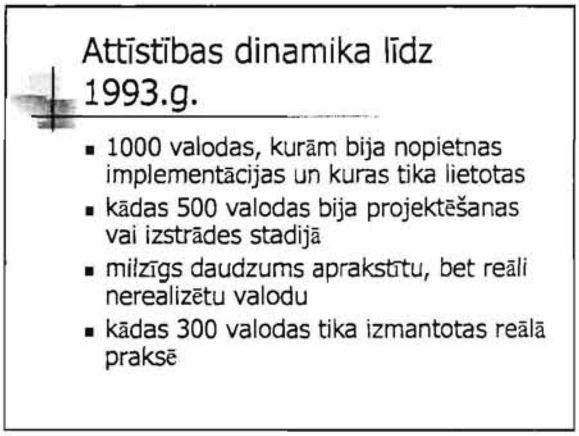
Valodu sadalījums

- **Vispārējā pielietojuma valodas**
  - daudziem aplikāciju domēniem (PL/1, Algol68)
- **Problēmorientētās valodas**
  - noteiktam lielam aplikāciju domēnam, var izmantot arī citiem domēniem (Smalltlak, C, Prolog, ML, Visual Basic, tcl, Pascal, Postscript, LaTex)
- **Domēnspecifiskās valodas**
  - konkrētam šaurākam aplikāciju domēnam, ir domēnam raksturīgas abstrakcijas un operācijas (HTML, XML, LEX, YACC, SQL)



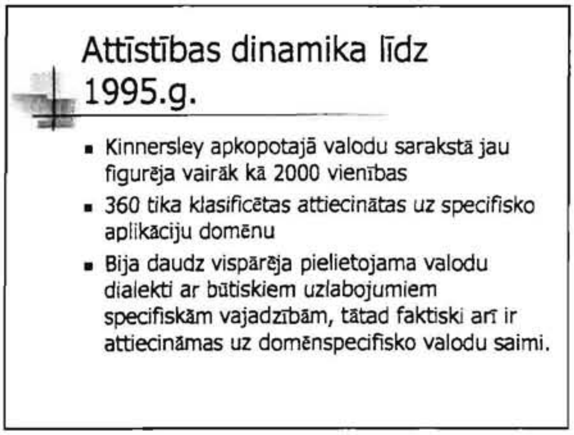
Valodu attīstības dinamika

- Jean E. Sammet 1993.gadā fiksēja stāvokli programmēšanas valodu lietošanas jomā iepriekšējos 15 gados
- 1978.gadā ASV tika izmantotas aptuveni 170 valodas:
  - 80 vispārējas lietošanas valodas
  - 90 valodas specializētās aplikācijās



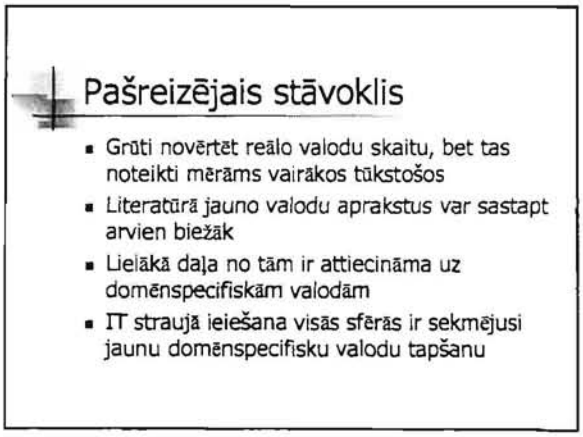
Attīstības dinamika līdz 1993.g.

- 1000 valodas, kurām bija nopietnas implementācijas un kuras tika lietotas
- kādas 500 valodas bija projektēšanas vai izstrādes stadijā
- milzīgs daudzums aprakstītu, bet reāli nerealizētu valodu
- kādas 300 valodas tika izmantotas reālā praksē



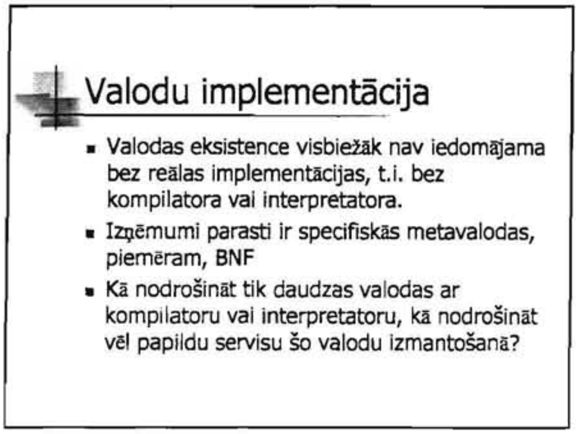
Attīstības dinamika līdz 1995.g.

- Kinnersley apkopotajā valodu sarakstā jau figurēja vairāk kā 2000 vienības
- 360 tika klasificētas attiecinātas uz specifisko aplikāciju domēnu
- Bija daudz vispārēja pielietojama valodu dialekti ar būtiskiem uzlabojumiem specifiskām vajadzībām, tātad faktiski arī ir attiecināmas uz domēnspecifisko valodu saimi.



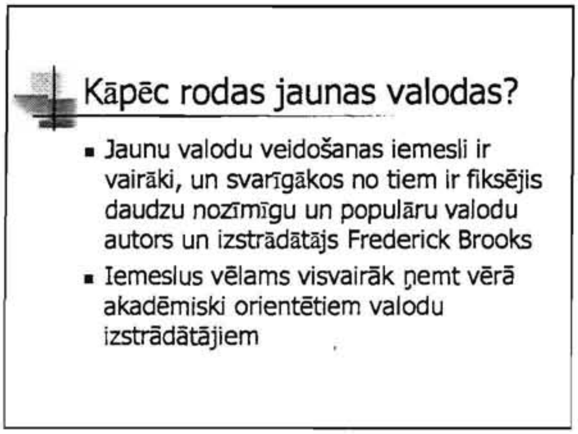
Pašreizējais stāvoklis

- Grūti novērtēt reālo valodu skaitu, bet tas noteikti mērāms vairākos tūkstošos
- Literatūrā jauno valodu aprakstus var sastapt arvien biežāk
- Lielākā daļa no tām ir attiecināma uz domēnspecifiskām valodām
- IT straujā ieiešana visās sfērās ir sekmējusi jaunu domēnspecifisku valodu tapšanu



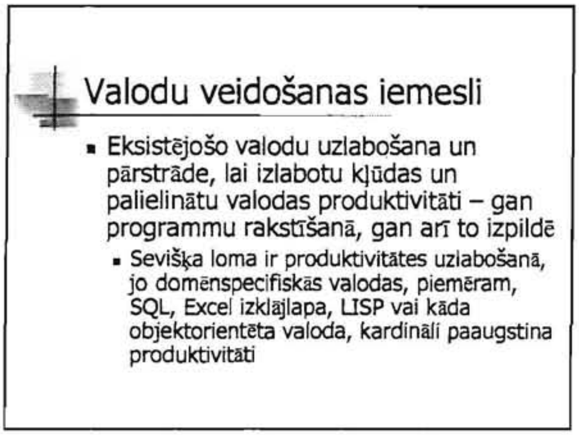
Valodu implementācija

- Valodas eksistence visbiežāk nav iedomājama bez reālas implementācijas, t.i. bez kompilatora vai interpretatora.
- Izņēmumi parasti ir specifiskās metavalodas, piemēram, BNF
- Kā nodrošināt tik daudzas valodas ar kompilatoru vai interpretatoru, kā nodrošināt vēl papildu servisu šo valodu izmantošanā?



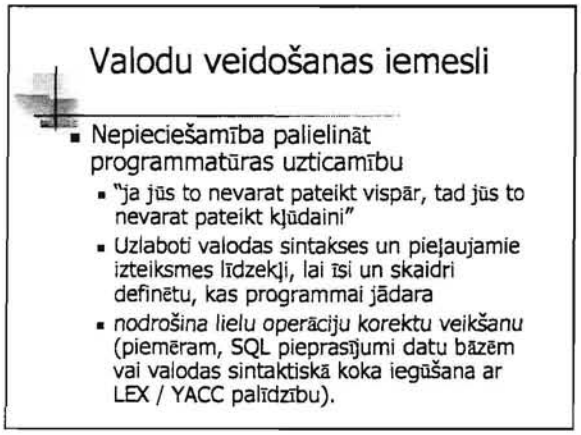
Kāpēc rodas jaunas valodas?

- Jaunu valodu veidošanas iemesli ir vairāki, un svarīgākos no tiem ir fiksējis daudzu nozīmīgu un populāru valodu autors un izstrādātājs Frederick Brooks
- Iemeslus vēlams visvairāk ņemt vērā akadēmiski orientētiem valodu izstrādātājiem



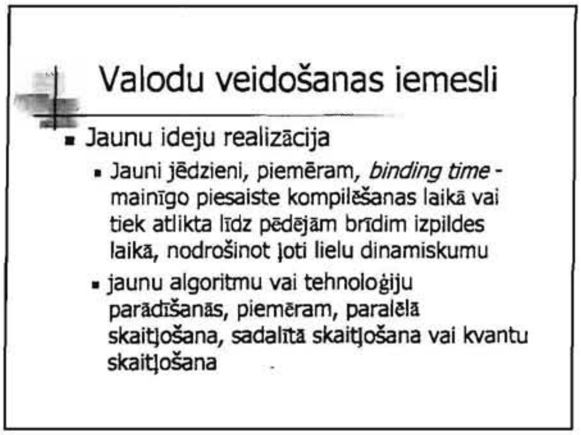
Valodu veidošanas iemesli

- Eksistējošo valodu uzlabošana un pārstrāde, lai izlabotu kļūdas un palielinātu valodas produktivitāti – gan programmu rakstīšanā, gan arī to izpildē
  - Sevišķa loma ir produktivitātes uzlabošanā, jo domēnspecifiskās valodas, piemēram, SQL, Excel izklājlapa, LISP vai kāda objektorientēta valoda, kardināli paaugstina produktivitāti



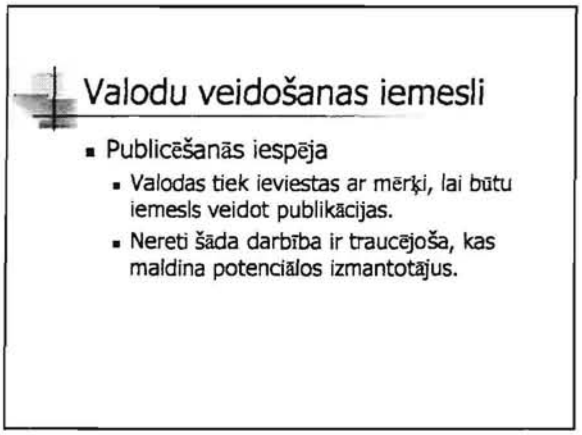
Valodu veidošanas iemesli

- Nepieciešamība palielināt programmatūras uzticamību
  - "ja jūs to nevarat pateikt vispār, tad jūs to nevarat pateikt kļūdaini"
  - Uzlaboti valodas sintakses un pieļaujamie izteiksmes līdzekļi, lai īsi un skaidri definētu, kas programmai jādara
  - *nodrošina lielu operāciju korektu veikšanu* (piemēram, SQL pieprasījumi datu bāzēm vai valodas sintaktiskā koka iegūšana ar LEX / YACC palīdzību).



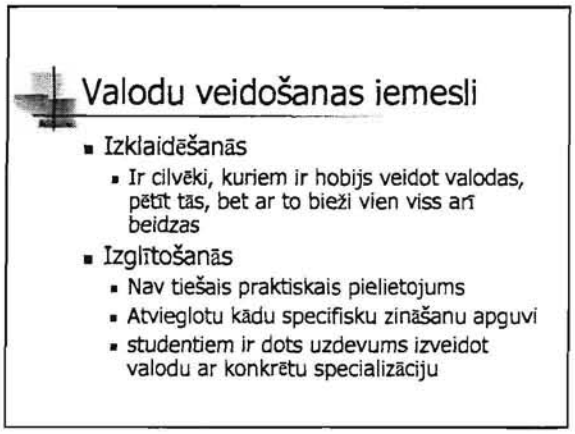
Valodu veidošanas iemesli

- Jaunu ideju realizācija
  - Jauni jēdzieni, piemēram, *binding time* - mainīgo piesaiste kompilēšanas laikā vai tiek atlikta līdz pēdējām brīdim izpildes laikā, nodrošinot ļoti lielu dinamiskumu
  - jaunu algoritmu vai tehnoloģiju parādīšanās, piemēram, paralēlā skaitļošana, sadalītā skaitļošana vai kvantu skaitļošana



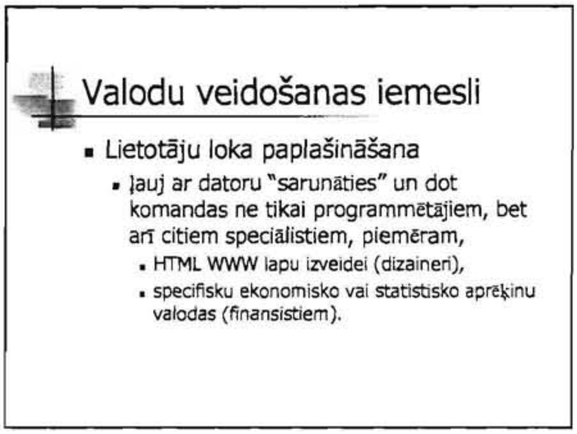
Valodu veidošanas iemesli

- Publicēšanās iespēja
  - Valodas tiek ieviestas ar mērķi, lai būtu iemesls veidot publikācijas.
  - Nereti šāda darbība ir traucējoša, kas maldina potenciālos izmantotājus.



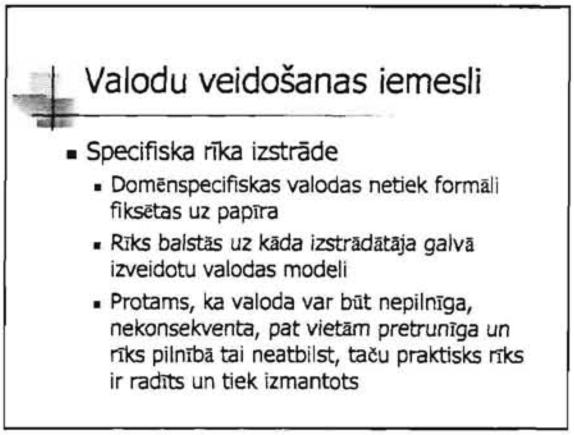
Valodu veidošanas iemesli

- Izklaidēšanās
  - Ir cilvēki, kuriem ir hobijs veidot valodas, pētīt tās, bet ar to bieži vien viss arī beidzas
- Izglītošanās
  - Nav tiešais praktiskais pielietojums
  - Atvieglotu kādu specifisku zināšanu apguvi
  - studentiem ir dots uzdevums izveidot valodu ar konkrētu specializāciju



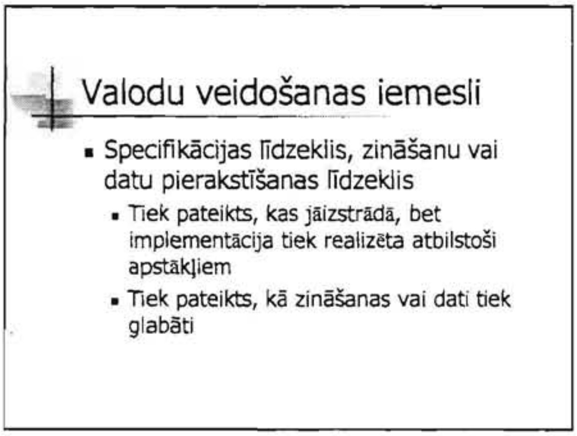
Valodu veidošanas iemesli

- Lietotāju loka paplašināšana
  - ļauj ar datoru "sarunāties" un dot komandas ne tikai programmētājiem, bet arī citiem speciālistiem, piemēram,
    - HTML WWW lapu izveidei (dizaineri),
    - specifisku ekonomisko vai statistisko aprēķinu valodas (finansistiem).



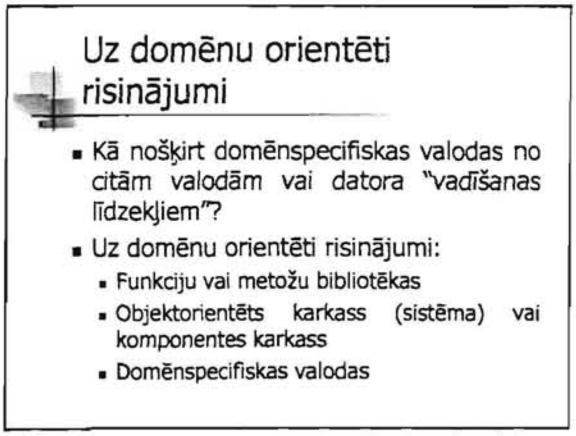
Valodu veidošanas iemesli

- Specifiska rīka izstrāde
  - Domēnspecifiskas valodas netiek formāli fiksētas uz papīra
  - Rīks balstās uz kāda izstrādātāja galvā izveidotu valodas modeli
  - Protams, ka valoda var būt nepilnīga, nekonsekventa, pat vietām pretrunīga un rīks pilnībā tai neatbilst, taču praktisks rīks ir radīts un tiek izmantots



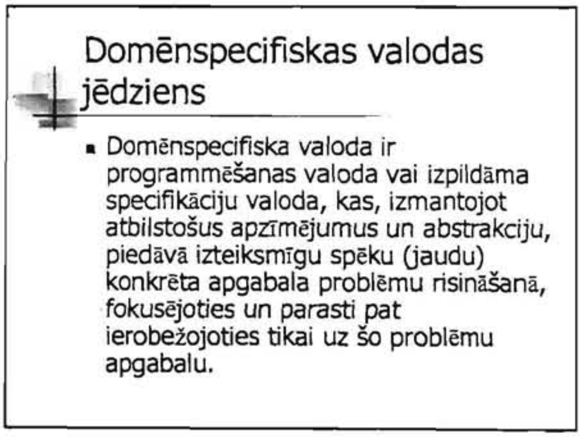
Valodu veidošanas iemesli

- Specifikācijas līdzeklis, zināšanu vai datu pierakstīšanas līdzeklis
  - Tiek pateikts, kas jāizstrādā, bet implementācija tiek realizēta atbilstoši apstākļiem
  - Tiek pateikts, kā zināšanas vai dati tiek glabāti



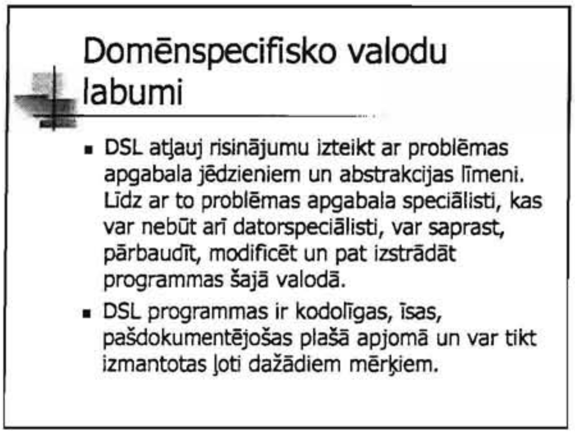
Uz domēnu orientēti risinājumi

- Kā nošķirt domēnspecifiskas valodas no citām valodām vai datora "vadīšanas līdzekļiem"?
- Uz domēnu orientēti risinājumi:
  - Funkciju vai metožu bibliotēkas
  - Objektorientēts karkass (sistēma) vai komponentes karkass
  - Domēnspecifiskas valodas



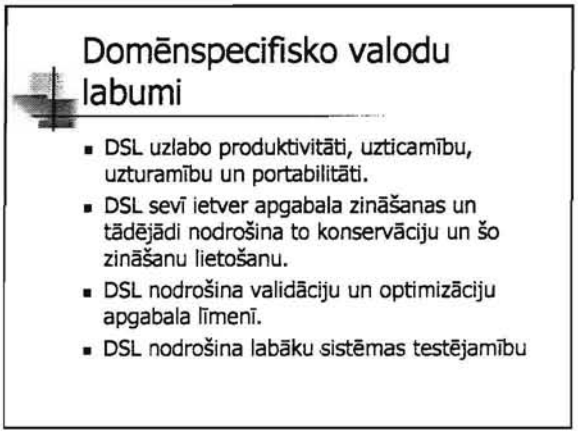
Domēnspecifiskas valodas jēdziens

- Domēnspecifiska valoda ir programmēšanas valoda vai izpildāma specifikāciju valoda, kas, izmantojot atbilstošus apzīmējumus un abstrakciju, piedāvā izteiksmīgu spēku (jaudu) konkrēta apgabala problēmu risināšanā, fokusējoties un parasti pat ierobežojoties tikai uz šo problēmu apgabalu.



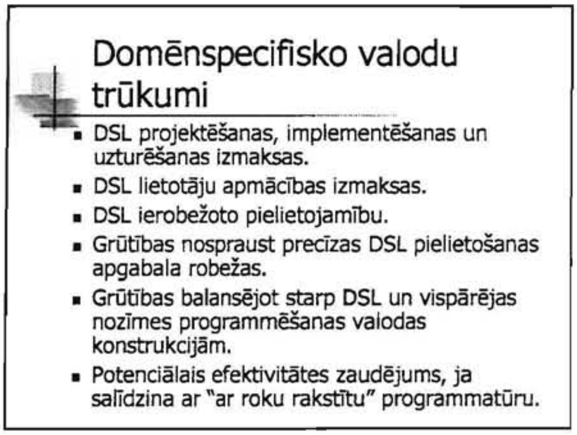
Domēnspecifisko valodu labumi

- DSL atļauj risinājumu izteikt ar problēmas apgabala jēdzieniem un abstrakcijas līmeni. Līdz ar to problēmas apgabala speciālisti, kas var nebūt arī datorspeciālisti, var saprast, pārbaudīt, modificēt un pat izstrādāt programmas šajā valodā.
- DSL programmas ir kodolīgas, īsas, pašdokumentējošas plašā apjomā un var tikt izmantotas ļoti dažādiem mērķiem.



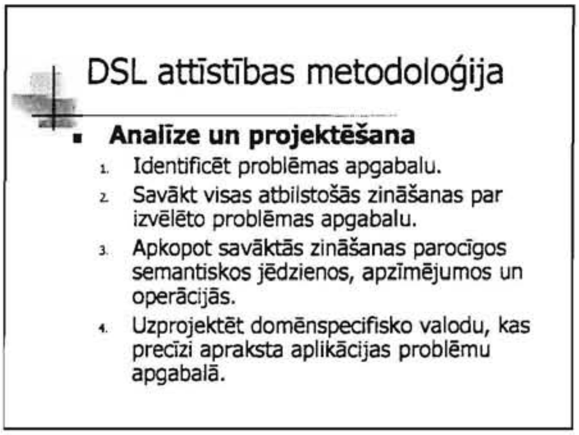
Domēnspecifisko valodu labumi

- DSL uzlabo produktivitāti, uzticamību, uzturamību un portabilitāti.
- DSL sevī ietver apgabala zināšanas un tādējādi nodrošina to konservāciju un šo zināšanu lietošanu.
- DSL nodrošina validāciju un optimizāciju apgabala līmenī.
- DSL nodrošina labāku sistēmas testējamību



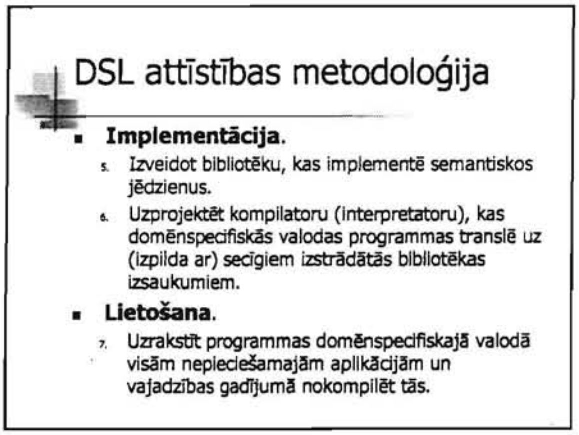
Domēnspecifisko valodu trūkumi

- DSL projektēšanas, implementēšanas un uzturēšanas izmaksas.
- DSL lietotāju apmācības izmaksas.
- DSL ierobežoto pielietojamību.
- Grūtības nospraust precīzas DSL pielietošanas apgabala robežas.
- Grūtības balansējot starp DSL un vispārējas nozīmes programmēšanas valodas konstrukcijām.
- Potenciālais efektivitātes zaudējums, ja salīdzina ar "ar roku rakstītu" programmatūru.



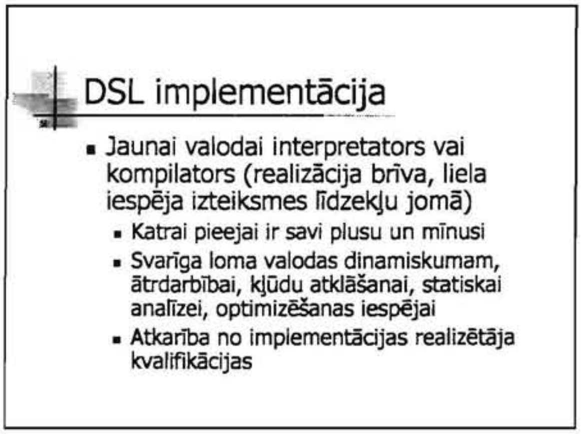
DSL attīstības metodoloģija

- **Analīze un projektēšana**
  1. Identificēt problēmas apgabalu.
  2. Savākt visas atbilstošās zināšanas par izvēlēto problēmas apgabalu.
  3. Apkopot savāktās zināšanas parocīgos semantiskos jēdzienos, apzīmējumos un operācijās.
  4. Uzprojektēt domēnspecifisko valodu, kas precīzi apraksta aplikācijas problēmu apgabalā.



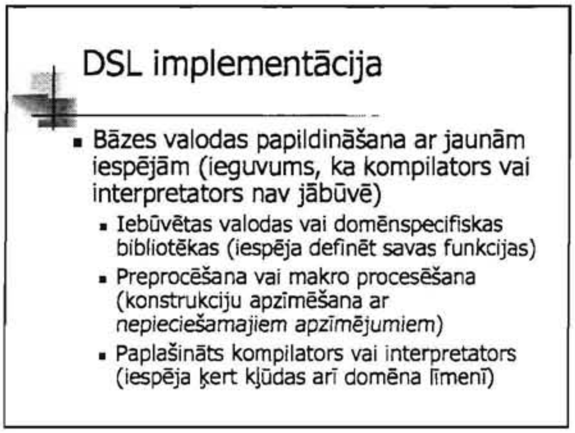
DSL attīstības metodoloģija

- **Implementācija**.
  5. Izveidot bibliotēku, kas implementē semantiskos jēdzienus.
  6. Uzprojektēt kompilatoru (interpretatoru), kas domēnspecifiskās valodas programmas translē uz (izpilda ar) secīgiem izstrādātās bibliotēkas izsaukumiem.
- **Lietošana**.
  7. Uzrakstīt programmas domēnspecifiskajā valodā visām nepieciešamajām aplikācijām un vajadzības gadījumā nokompilēt tās.



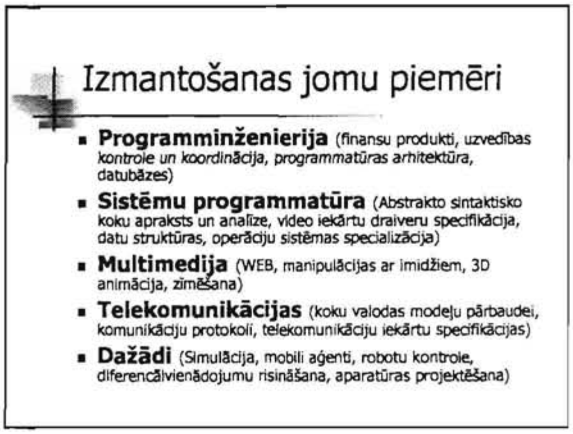
DSL implementācija

- Jaunai valodai interpretators vai kompilators (realizācija brīva, liela iespēja izteiksmes līdzekļu jomā)
  - Katrai pieejai ir savi plusu un mīnusi
  - Svarīga loma valodas dinamiskumam, ātrdarbībai, kļūdu atklāšanai, statiskai analīzei, optimizēšanas iespējai
  - Atkarība no implementācijas realizētāja kvalifikācijas



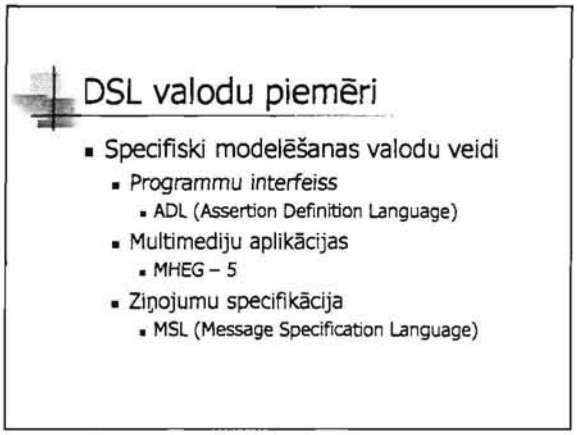
DSL implementācija

- Bāzes valodas papildināšana ar jaunām iespējām (ieguvums, ka kompilators vai interpretators nav jābūvē)
  - Iebūvētas valodas vai domēnspecifiskas bibliotēkas (iespēja definēt savas funkcijas)
  - Preprocēšana vai makro procesēšana (konstrukciju apzīmēšana ar *nepieciešamajiem apzīmējumiem*)
  - Paplašināts kompilators vai interpretators (iespēja ķert kļūdas arī domēna līmenī)



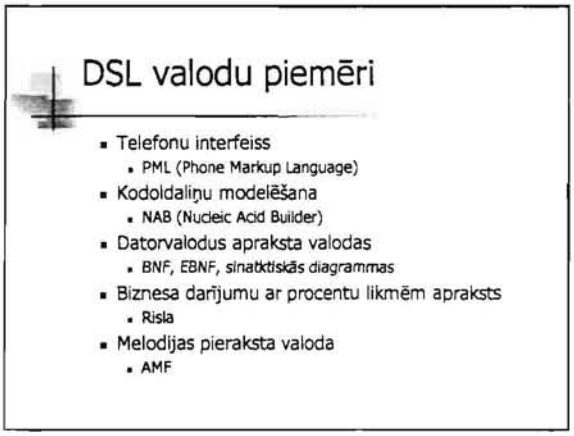
Izmantošanas jomu piemēri

- **Programminženierija** (finansu produkti, uzvedības *kontrole un koordinācija, programmatūras arhitektūra,* datubāzes)
- **Sistēmu programmatūra** (Abstrakto sintaktisko koku apraksts un analīze, video iekārtu draiveru specifikācija, datu *struktūras*, operāciju *sistēmas* specializācija)
- **Multimedija** (WEB, manipulācijas ar imidžiem, 3D animācija, zīmēšana)
- **Telekomunikācijas** (koku valodas modeļu pārbaudei, komunikāciju protokoli, telekomunikāciju iekārtu specifikācijas)
- **Dažādi** (Simulācija, mobili aģenti, robotu kontrole, diferencālvienādojumu risināšana, aparatūras projektēšana)



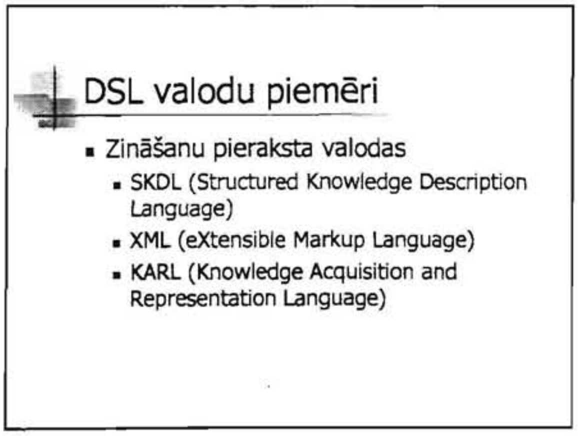
DSL valodu piemēri

- Specifiski modelēšanas valodu veidi
  - *Programmu interfeiss*
    - ADL (Assertion Definition Language)
  - Multimediju aplikācijas
    - MHEG – 5
  - Ziņojumu specifikācija
    - MSL (Message Specification Language)



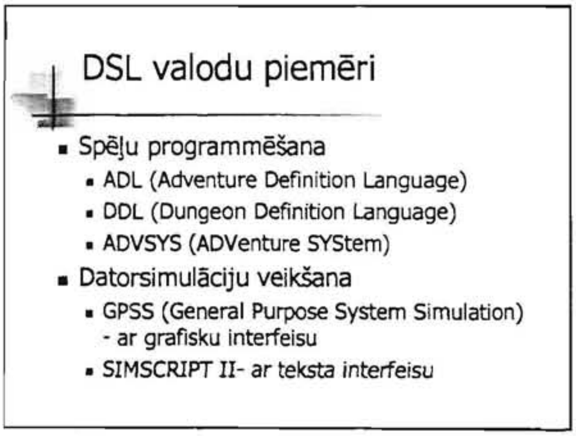
DSL valodu piemēri

- Telefonu interfeiss
  - PML (Phone Markup Language)
- Kodoldaliņu modelēšana
  - NAB (Nucleic Acid Builder)
- Datorvalodus apraksta valodas
  - *BNF, EBNF, sinatktiskās diagrammas*
- Biznesa darījumu ar procentu likmēm apraksts
  - Risla
- Melodijas pieraksta valoda
  - AMF



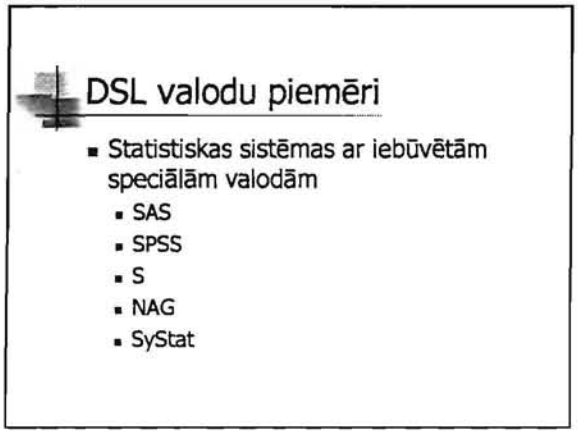
DSL valodu piemēri

- Zināšanu pieraksta valodas
  - SKDL (Structured Knowledge Description Language)
  - XML (eXtensible Markup Language)
  - KARL (Knowledge Acquisition and Representation Language)



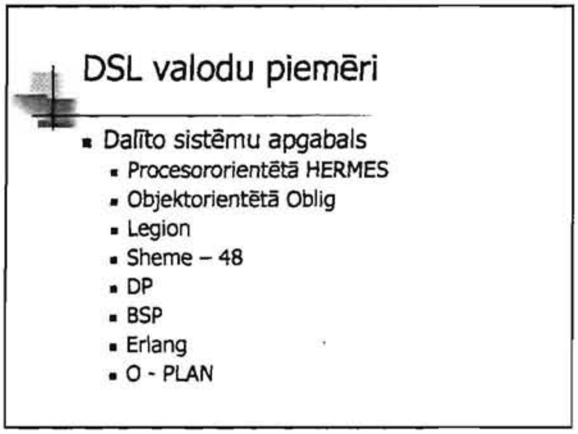
DSL valodu piemēri

- Spēļu programmēšana
  - ADL (Adventure Definition Language)
  - DDL (Dungeon Definition Language)
  - ADVSYS (ADVenture SYStem)
- Datorsimulāciju veikšana
  - GPSS (General Purpose System Simulation) - ar grafisku interfeisu
  - SIMSCRIPT II- ar teksta *interfeisu*



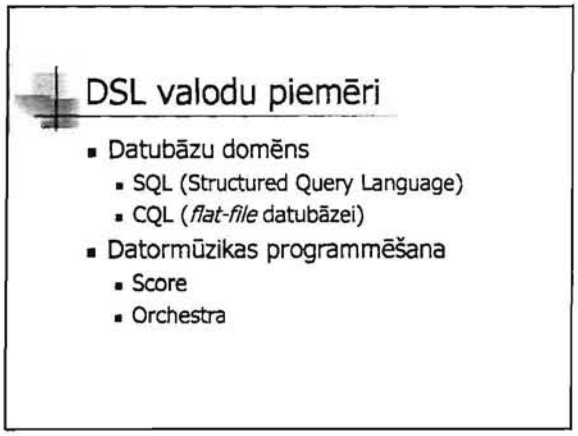
DSL valodu piemēri

- Statistiskas sistēmas ar iebūvētām speciālām valodām
  - SAS
  - SPSS
  - S
  - NAG
  - SyStat



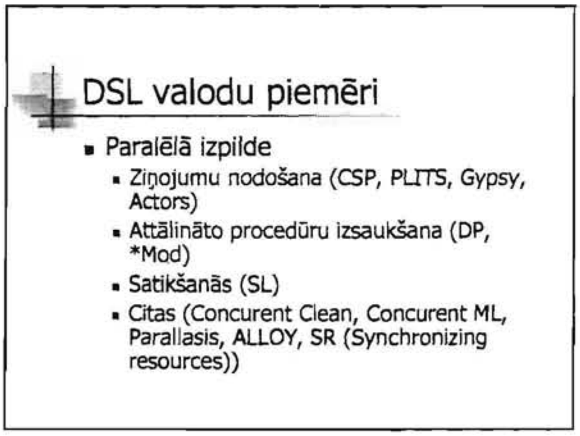
DSL valodu piemēri

- Dalīto sistēmu apgabals
  - Procesororientētā HERMES
  - Objektorientētā Oblig
  - Legion
  - Sheme – 48
  - DP
  - BSP
  - Erlang
  - O - PLAN



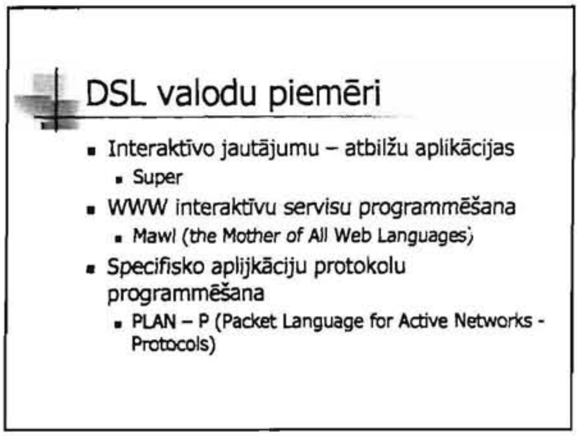
DSL valodu piemēri

- Datubāzu domēns
  - SQL (Structured Query Language)
  - CQL (*flat-file* datubāzei)
- Datormūzikas programmēšana
  - Score
  - Orchestra



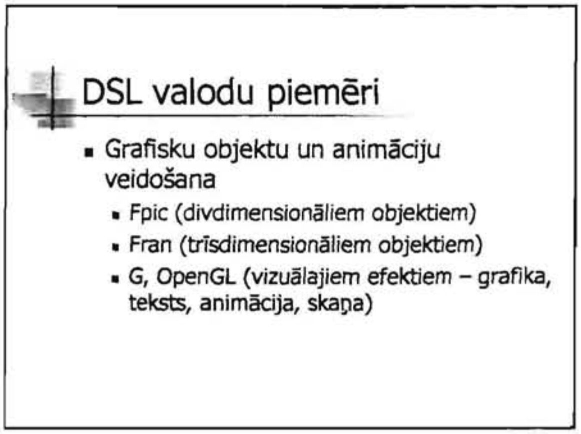
DSL valodu piemēri

- Paralēlā izpilde
  - Ziņojumu nodošana (CSP, PLITS, Gypsy, Actors)
  - Attālināto procedūru izsaukšana (DP, *Mod)
  - Satikšanās (SL)
  - Citas (Concurent Clean, Concurent ML, Parallasis, ALLOY, SR (Synchronizing resources))



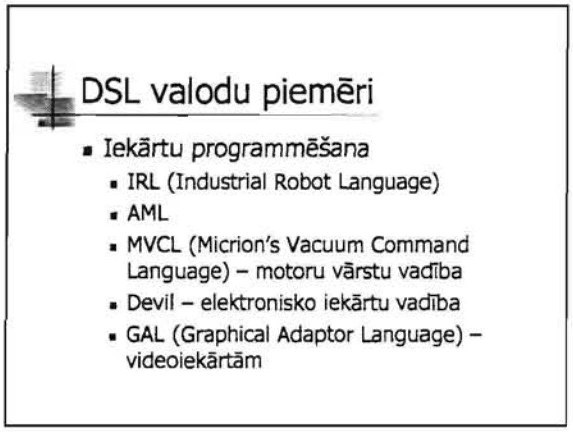
DSL valodu piemēri

- Interaktīvo jautājumu – atbilžu aplikācijas
  - Super
- WWW interaktīvu servisu programmēšana
  - Mawl (the Mother of All Web Languages)
- Specifisko aplijkāciju protokolu programmēšana
  - PLAN – P (Packet Language for Active Networks - Protocols)



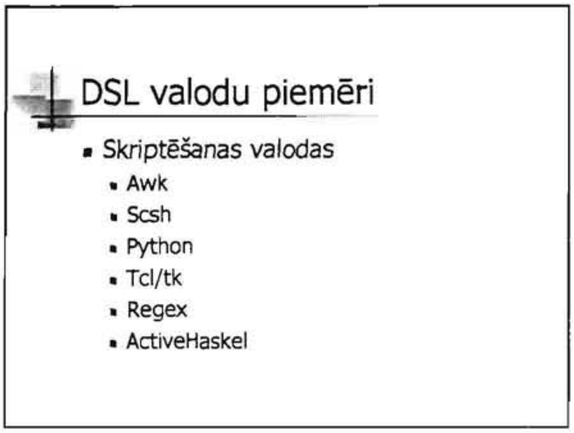
DSL valodu piemēri

- Grafisku objektu un animāciju veidošana
  - Fpic (divdimensionāliem objektiem)
  - Fran (trīsdimensionāliem objektiem)
  - G, OpenGL (vizuālajiem efektiem – grafika, teksts, animācija, skaņa)



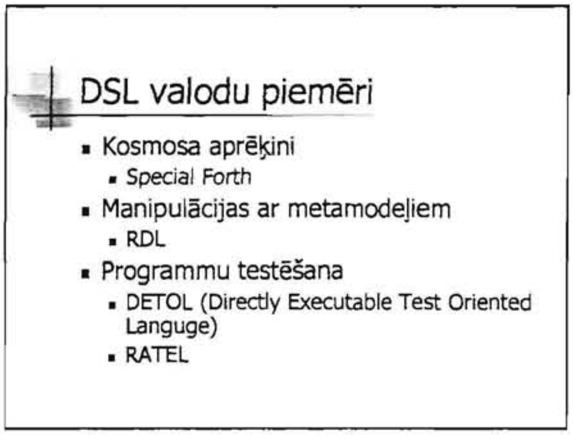
DSL valodu piemēri

- Iekārtu programmēšana
  - IRL (Industrial Robot Language)
  - AML
  - MVCL (Micrion's Vacuum Command Language) – motoru vārstu vadība
  - Devil – elektronisko iekārtu vadība
  - GAL (Graphical Adaptor Language) – videoiekārtām



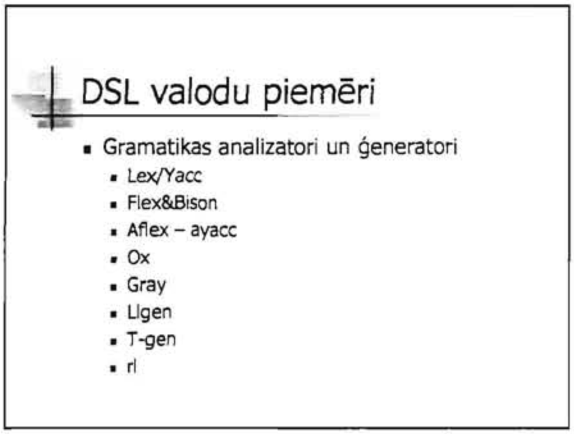
DSL valodu piemēri

- *Skriptēšanas valodas*
  - Awk
  - Scsh
  - Python
  - Tcl/tk
  - Regex
  - ActiveHaskel



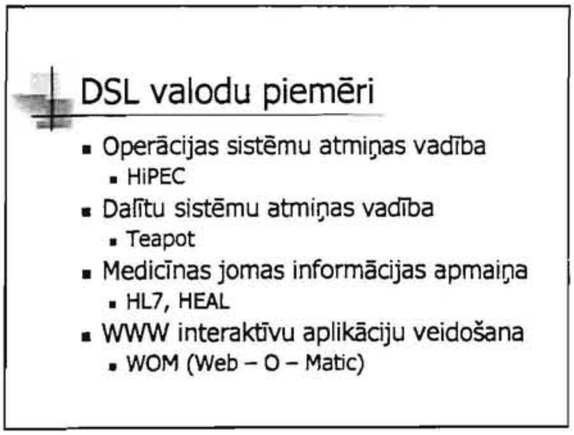
DSL valodu piemēri

- Kosmosa aprēķini
  - *Special Forth*
- Manipulācijas ar metamodeļiem
  - RDL
- Programmu testēšana
  - DETOL (Directly Executable Test Oriented Languge)
  - RATEL



DSL valodu piemēri

- Gramatikas analizatori un ģeneratori
  - Lex/Yacc
  - Flex&Bison
  - Aflex – ayacc
  - Ox
  - Gray
  - Llgen
  - T-gen
  - rl



DSL valodu piemēri

- Operācijas sistēmu atmiņas vadība
  - HiPEC
- Dalītu sistēmu atmiņas vadība
  - Teapot
- Medicīnas jomas informācijas apmaiņa
  - HL7, HEAL
- WWW interaktīvu aplikāciju veidošana
  - WOM (Web – O – Matic)

Information Processing Tools and Environments

Guntis Arnicans

Faculty of Physics and Mathematics
University of Latvia
Raina Blvd. 19, Riga LV-1586, Latvia
garnican@lanet.lv

Abstract

The various ways exist how we can build an information system. We offer to look at a information system like a suit of tools that are integrated into a collaborative environment. The other tools are used to design, implement and test such environment. They increase a convenience, productivity and quality of development process providing the implementation of target tools, its integration and satisfying the development methodology and requirements. These supportive tools make a specific software environment. In the paper we present basic things what the developers have to know following to this principle to build an information system: 1) the concept of a tool, 2) the classical approaches for tools' integration, 3) the principles of tools development, management and control, 4) the questions before tool designing, and 5) the possible conceptual architecture of a tool. The mentioned things are equally referred to both the information processing tools and the development supporting tools.

1 Introduction

Data is a formal representation of facts or ideas with possibility to be communicated or manipulated by some automated process. Information is a meaning that humans assign to data during automated data processing using the definite habits to present it (meaning of data). Disinformation is information with delusive meaning and/or off-grade data was being used to produce information.

It is increasingly difficult to draw a line around an application system and say that you own and control it. Data is distributed over a multitude of heterogeneous, often autonomous information systems, and an exchange of data among them is not easy. Let us look at a relatively simple situation – the data source and services are located in one organization but the information consumer (user) is located into another organization (Figure 1).

Process how the data is transformed to information and presented is long and difficult (going through many applications, operating systems, defense systems, data transmission protocols, etc.). If the system builders make mistakes in some part of this process, then we receive a disinformation but not the desirable information. The problem become more serious when we produce information from many data sources, and when the services must work without interruptions.

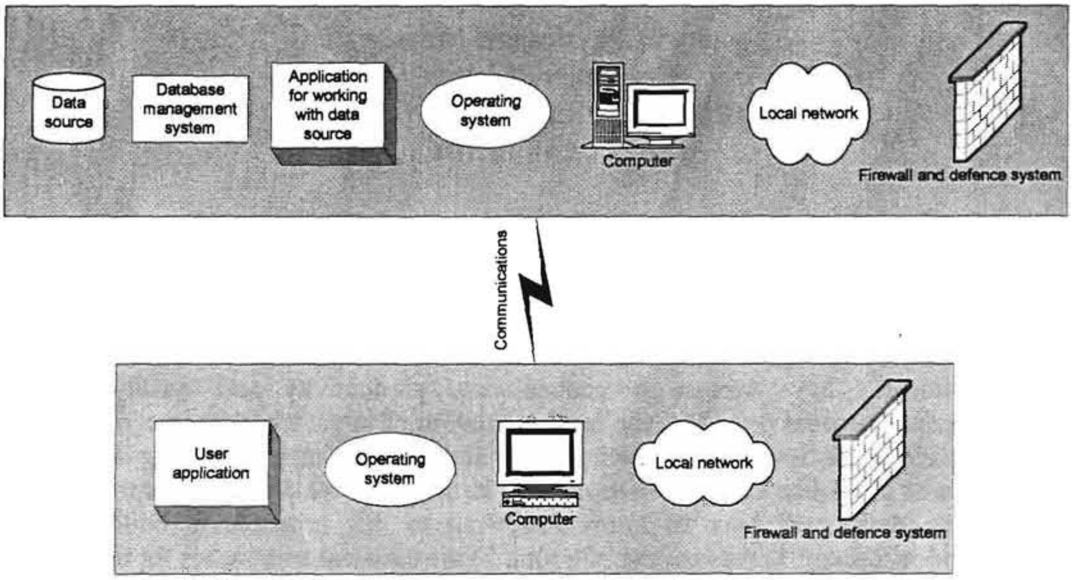

Data source
Database management system
Application for working with data source
Operating system
Computer
Local network
Firewall and defence system
Communications
User application
Operating system
Computer
Local network
Firewall and defence system

Figure 1 The information processing in heterogeneous environment

These problems emphasize the need for tools to mediate between databases, servers and front-end application. And we need tools to create these mediator tools also. We need to maintain descriptions of data structures, content, data properties, available services (metadata of data sources and services). It increases the need for dynamic manipulating of both data and metadata. Besides we have to worry about system quality that leads to need for testing, simulating and monitoring tools.

Over time, the number and variety of tools has grown tremendously. They range from traditional tools like editors, compilers and debuggers, to tools that aid in requirements gathering, design, building GUIs, generating queries, defining messages, architecting systems and connecting components, testing, version control and configuration management, administering databases, reengineering, reverse engineering, analysis, program visualization, and metrics gathering, to full-scale, process-centered software engineering environments that cover the entire lifecycle, or at least significant portions of it [HOT00].

2 The Concepts of Information Processing Tools and Environments

Any system that assists the programmer with some aspect of programming can be considered a *programming tool*. Similarly, a system that assists in some phase of the software development process can be considered a *software tool.* A *programming environment* is a suite of programming tools designed to simplify programming and thereby enhance programmer productivity. A *software engineering environment* extends this to software tools and the whole software development process [Rei96].

The definitions above we can refer to any software. Let us look to the narrower class of software – a software for information processing. Similarly to previous definitions we introduce the concepts of information processing tool and information processing environment.

An *information processing tool* is any system that provides performing some task while information processing. An *information processing environment* is a suit of information processing tools that together makes intended information processing. On the other hand, from the perspective of end-user the information processing environment is simply an *information system.*

An *information system software tool* (simply a *tool* below in the text) is a system that assists in some phase of the information system development process. And finally, an *information system software environment* (simply an *environment* below in the text) is the extension of the information processing environment with the information system software tools.

3 Classification of Information Processing Tools

Tools can be categorized by the phase of information system development and the particular problem that they solve. It is possible categorize them also by development principles, integration principles, runtime behavior, etc.

3.1 Classification by the Functionality

One of the most natural ways to classify the tools is grouping them by its functionality. Many grouping principles exist. We offer to classify them in the following way (only an example how it can be done):

1. Data extracting tools from data sources

    1.1. Relation databases
        1.1.1. Tool for the specific database
        1.1.2. Universal tool for various databases
    1.2. Object-oriented databases
    1.3. Structured files
    1.4. Other data source
2. Communication tools to work with data sources
    2.1. Specific client software for particular data source
    2.2. Internet Browser
    2.3. Email
    2.4. Other communication tool
3. Tools for describing of data sources
    3.1. Repository for data sources descriptions
    3.2. Data source describer
    3.3. Data source services describer
    3.4. Other
4. Inquiry processing tools
    4.1. Inquiry definition tools
    4.2. Inquiry executing tools
    4.3. Other
5. User interface generating tools
    5.1. Static, predefined interface application generator
    5.2. Dynamic, varying interface application generator
    5.3. Other
6. User management tools
    6.1. User registration and general management tools
    6.2. User rights management tools
    6.3. Finances accounting tools for services
    6.4. User profiles management tools
    6.5. Other
7. System auditing tools
    7.1. Audit journals management tools
    7.2. Statistics accounting tools
    7.3. Security control tools
    7.4. Other
8. System quality tools
    8.1. System testing tools
    8.2. Documentation tools
    8.3. Other
9. User temporal data management tools (temporal databases)
10. Other

3.2 Classification by the Runtime Behaviour

The other principle to classify the tools is grouping them by its runtime behavior. Many grouping principles exist. We most of tools divide into two basic groups:

1. Static tool. The tool performs the specific predefined and fixed functionality, and this functionality can be changed only by redesigning and implementing of the

tool. It is possible that functionality can be altered by some simple predefined configuration functions or by configuring these tools before running them. Usually input data is in relative strong predefined format. Basically the source code compilation is used to obtain executable software (tool).

2. Dynamic tool. The tool can vary its own functionality or in the other words – change executing semantics before or during the tool operating time. It is possible to vary functionality in large range. Input data format and meaning can be different. The interpreter is more preferable to implement such tools. The tool works interpreting commands received as a program before or during the operating.

4 Tools Integration

We mentioned above that from the user perspective information processing environment is an information system. If we build the information system as a set of tools, then we need to integrate all tools into a collaborative work. These tools can be combined together in a variety of ways using various integration techniques.

System developers choose integration techniques being guided by practical needs, system complexity, knowledge and skills of developers, etc. It is distinguished three the most popular approaches for an integration that involve ways for the tools to share information and interfaces [Rei]:

1. Data integration. It assumes that tools share information. Usually a database or repository is created. Most of the tools stores and consumes shared information. Via this repository all tools work with the same data and data exchanging also is organized through the repository.

2. Common front end. The user sees and uses all system together. He does not exploit any tool separately and often does not know that the system consists from a set of tools. The user operates with data objects and operations allowed at the specified moment. Usually the common interface is used to integrate tools, and tools do not share information.

3. Control integration. This approach involves message passing between the tools. Tools send messages to other tools whenever they need to share information or whenever a command from one tool is invocated from another.

The message exchanging mostly is organized through a central message server. The tool send a message to the server, and the server send this message to all other tools that are interested in to this message type. The tools can exchange with messages directly, but this approach can arise problems if the amount of tools is large.

A combination of the all integration types is used to develop serious and large environment of integrated tools.

5 The Principles for the Tools' Development, Management and Controlling

Real world changes all the time, and requirements to systems also changes. We have to take into account these changes and to implement them into our information system. In the distributed and heterogeneous computing environment it is a serious problem. Besides the nowadays' services must run without interruptions, and system changes must be done by changing behavior of the tools sending them a new configuration or replacing them dynamically with new tools. The tools have to satisfy the following requirements – convenient configuration and control facilities, a possibility to change behavior semantics, an acceptance of various input and output formats, etc.

To deal with these problems and to provide convenient means for tools integration and running system maintenance we advice exploit common principles of the tools' development, management and controlling. The most important principles are:

1. Common architecture. Most of the tools have a similar architecture. This allows a designing of the tools with common components. The tool development and maintenance becomes less resources consuming, and quality of resulting tool is higher.
2. Common software (modules). If the ideology and architecture of the tools is similar, then an implementation can contain common modules, subsystems, runtime libraries, etc.
3. Common configuration mechanism. It assumes that we can change the behavior of tool (predefined changes without software changes) dynamically

in similar way according to the specific task. These leads to common modules and easier exploiting, integration and maintenance of tools.

4. Common control mechanism. The tools have to have standardized means (interfaces) to control and manage them. Via this interface user or other tools gives commands to the specific tool, and it is a main mechanism for the tools integration.
5. Common monitoring mechanism. Information systems usually have to work in an uninterrupted regime. We need to monitor system operating, find the weakest points and perform some actions to correct the system performance.
6. Common testing principles and means. The system quality is crucial topic for systems. We have to test tools before deployment, and common principles and testing tools can reduce most expensive resource costs (time, money, people). Moreover, we have to continue testing while real system exploiting and check every operation if we use dynamic code generation and immediate just-in-time compilation or interpretation.

6 The Conceptual Architecture of a Tool

There are many various opinions what the conceptual architecture of a tool looks like. Before we present our model let us state the most important questions to understand the essence of a tool:

- What is the main task for desired tool? Why does it necessary us?
- How can we build this tool? What are the possible technologies, data structures and algorithms? Can we use or adapt existing tools or modules?
- What is an input for our tool? Does a tool receive all input data before starting computations or get it by portions?
- What is an output for our tool? Does a tool produce all output data after computations or supply it by portions?
- Is a tool stateless or not? Does a tool remember the history about previous computations?

- What is a way to manage and control the tool? What is a desirable interface for such tool?
- Do we need to change the behavior of a tool dynamically without interrupting all running environment or moreover while operating time? What are the things we need to change (configuration, executing semantics, input/output formats, etc.)?
- Have we got necessity to monitor the state and behavior of a tool before, during or after operating time?
- Does a tool cooperate with other applications or tools excluding desired "official input and output"? Is this cooperation synchronous or asynchronous?
- Can our tool change the state/configuration of other application or tool? Can the other tool change our tool state/configuration? At what time (before or during operating)?

We have created a conceptual model of tool architecture taking into account questions above. The model is shown in Figure 2.

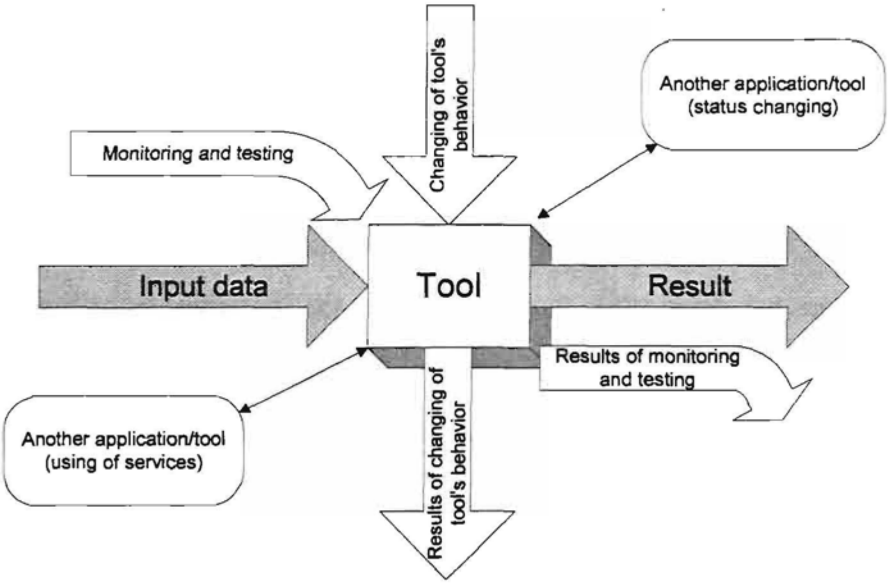

Changing of tool's behavior
Another application/tool (status changing)
Monitoring and testing
Input data
Tool
Result
Results of monitoring and testing
Another application/tool (using of services)
Results of changing of tool's behavior

Figure 2 A conceptual architecture of a tool

7 Conclusions

The creating of tools' collections – environments is actual for many years [How82], and the importance of this topic only grows day by day [HOT00]. In this paper we briefly review some more important concepts and principles to organize (integrate) tools into one collaborating environment.

The main attention is paid to a subset of all possible environments – an information processing environment (information system) and an information system software environment that supports building of information system. There is a challenge for developers to create information system as a suit if tools.

The most important issues in this field are separation of concerns, integration and coordination, "plug and play", and support for multiple views. Traditional software development lifecycle is not acceptable for new emerging technologies, and researchers look for new methodologies. We consider that most of the tools in one environment have to built based on common principles, and that tool has to base on domain specific language (DSL) that describes tool behavior. In that approach the environment is a set of *interpreters* that interprets the tool specification (program in DSL) and each *interpreter* acts like desired tool.

8 References

[HOT00] W. Harrison, H. Ossher, and P. Tarr. Software Engineering Tools and Environments: A Roadmap. *Proceedings of the conference on The future of Software engineering (ICSE '00)*, pp.261-277, 2000.

[How82] W. E. Howden. Contemporary Software Development Environments. *Communications of the ACM*, 25(5):318-329, May 1982.

[Rei96] S. P. Reiss. Software Tools and Environments. *ACM Computing Surveys*, Vol. 28, N0. 1, March 1996.

Description of Semantics and Code Generation Possibilities for a Multi-language Interpreter

Guntis Arnicans

Faculty of Physics and Mathematics
University of Latvia
Raina Blvd. 19, Riga LV-1586, Latvia
garnican@lanet.lv

Abstract

In this paper we describe the definition of semantics for a Multi-language interpreter (MLI), which provides the execution of the given program, receiving and exploiting corresponding language syntax and the desired semantics. We analyze the simplest solution – the MLI receives the language syntax and the semantics descriptions, which have already been compiled to executable objects. Semantics are defined as a composition from several semantic aspects, considering the pragmatics of a language. Semantic aspects are translated to semantic functions by composing descriptions of the aspects. A traversing program's intermediate representation and the calling out of semantic functions similarly to the principle of the Visitor pattern perform the desired semantics. To simplify the semantic descriptions, we use abstract components that are joined by connectors at the meta-level. The implementation of these components and connectors can be very different. Examples of conventional and specific semantics are given for the simple imperative language in this paper.

1 Introduction

The number of new languages that are related to the IT sector has increased rapidly over the last several years (programming languages and data description languages, for example). Problems associated with the implementation and use of these languages have also expanded, of course. Kinnersley [Kin95] has reported that there were 2,000 languages in 1995, which were being put to serious use. Even back then specialists found that the new languages were mostly to be classified as domain-specific languages. Most of them are not easy to implement and maintain [ITSE99, DKV00 (DSL analysis, problems and an annotated bibliography)]. It is also true that we need not just a compiler or an interpreter, but also a number of supportive tools. Questions of programming quality are very important today, and these questions often cannot be answered without specialized and automated ancillary resources.

Computers are being used with increasing dynamism today: systems have been divided up in terms of time and space, the operational environment is heterogeneous, and we have to ensure that implementation of parallel processes while organizing cooperation among components and systems, adapting to changing circumstances without interrupting our work, etc. We are making increasing use of interpreters or of code generation and compilation just in time. The formal resources that are used to describe the semantics of a language, however, cannot fully satisfy our needs in the modern age, and they are starting to lose their positions [Sch97, Lou97, Paa95].

The basic problem that is associated with the formal specifications of programming languages is that these specifications are far too complex. It is not clear how they are administered, we cannot use them to explain all of our practical needs, and in the end we are still faced with a problem – who can prove that these complex specifications are really correct? The literature claims that the best commercial compilers (interpreters or other language-based tools) are written without formalism or are used only in the first phases – scanning and parsing [e.g. Lou97]. Formalisms are mostly elaborated and used for research purposes in educational and scientific institutions at this time.

The development of semantics is gradually moving away from the development of languages and tools. One way to overcome this gap is to take a *tool-oriented approach to semantics*, making the definitions of semantics far more useful and productive in practice and generating as many language-based tools as possible from them [HK00]. We support this approach in principle, but our aim is to propose a different approach toward the definition of semantics, making room for far less formal records.

Those who prepared descriptions of semantics in the past have long since been looking for ways in which semantics can be divided up into reusable components, and it is not yet clear whether the formal or the partly formal methodology is best in this case. We chose a less formal and more free form of description keeping from the theoretical perspective, and our empirical research shows that rank-and-file developers of tools far more easily understand this method.

2 The concept of a Multi-language Interpreter

The concept of a *multi-language interpreter* was introduced in [AAB96]. A Multi-Language Interpreter (MLI) is a program which receives source language syntax, source language semantics and a program written in the source language, then performing the operations on the basis of the program and the relevant semantics. Conceptually, we parse an input token stream, build a parse tree and then traverse the tree as needed so as to evaluate the semantic functions that are associated with the parse tree nodes. Once an explicit parse tree is available, we visit the nodes in some order and call out an appropriate function. This approach is similar to the principle *build a tree, save a parse and traverse it* [Cla99] and to a Visitor pattern [GHJV95], except in terms of the methodology which we apply in obtaining semantic functions and organizing physical implementation. The idea of MLI is expressed in Figure 1.

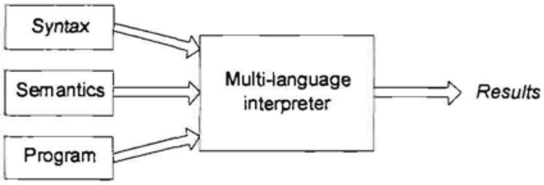


Figure 1 The concept of a Multi-language interpreter

The concept of a MLI presupposes that we can prepare several semantics for one syntax, and we can exploit one semantic for various syntaxes. The descriptions of syntaxes and semantics must be translated to the executable form (before or during the running of the MLI). MLI implementation architectures may vary. The one we use receives and exploits syntax and semantic descriptions that have already been compiled as executable objects (Figure 2). Syntax is represented by the *SyntaxObject*, and semantics by the *TraverserObject*, the *SemanticObject*, the *SymbolTable*, and the necessary volume of the *Component* (the components A, B, C in our figure). The *MLI Kernel*, which provides the initial bonding of all syntax and semantics objects, initializes the execution of the program.

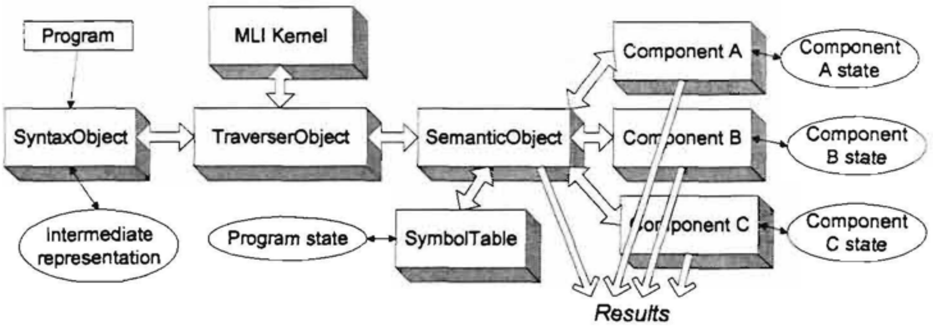


Figure 2 MLI runtime architecture

Each of the components can be implemented in various ways – with a different semantic assignment and physical implementation. Here we have a chance to combine syntaxes and semantics in both ways – in terms of architecture and in terms of implementation. Then, however, we immediately face the question of the compatibility of the syntax and semantics so as to avoid senseless interpretation.

The obtaining of an executable syntax and semantic objects from their descriptions can be done before or during the actual program execution (analogue to a classical compiler and interpreter). Dynamic code generation is more difficult because all generation phases must be done automatically.

3 Language Specifications for MLI

Programming language is an artificial means to communicate with a computer and to fix the algorithms for problem solving. Like a natural language, a programming language's definition consists of three components or aspects: *syntax*, *semantics* and *pragmatics* [Pag81, SK95]. All of these aspects are significant in dealing with our problems. Usually exploited rarely, pragmatics deals with the practical use of a language, and this is an important element in defining semantics.

We can look at syntax and semantics from two perspectives – the definition or description phase and the runtime phase. Our goal is to achieve runtime components, which can freely be exchanged or mixed together in pursuit of the desired collaboration. First we must look at the principles of syntax and semantics descriptions, and then we can view the target code generation steps.

Our basic principle is to divide syntax and semantics into small parts and later, with a simple method, to stick these parts together, thus providing a mechanism to tie together the semantic parts and the syntax elements. Our method is close to some of the structuring paradigms of attribute grammars [Paa95]: The definition phase is similar to the relationship *Semantic aspect* = *Module*, but the runtime phase is similar to *Nonterminal* = *Procedure*. That means that we basically use the language pragmatics and divide the semantics into semantic aspects.

3.1 Syntax

The formalisms for dealing with the syntax aspect of a programming language are well developed. The theory of scanning, parsing and attribute analysis provides not only the means to perform syntactical analysis, but also a way to generate a whole compiler, as well. Such terms, concepts or tools as finite automata, regular expression, context-free grammar, attribute grammar (AG), Backus-Naur form (BNF), extended BNF (EBNF), Lex (also Flex), Yacc (also Bison), and PCCTS are well known and accepted by the computer science community.

We do not need to reinvent the bicycle, and it is reasonable to choose existing formalisms and generators (lexers and parsers). The main task when dealing with syntax description for a given language is code generating which can transform the written program, which uses the syntax, into intermediate representation (IR). Additionally, we need to attach a library with functions, which provide the means to manipulate with the IR and to compile the whole code. The result is the SyntaxObject (Figure 2).

In this paper we concentrate mostly on the class of imperative programming languages, but our method is adaptable for other languages too, such as diagrammatic languages (e.g., Petri nets, E-R diagrams, Statecharts, VPL – visual programming languages, etc.), which exploit other formalisms (e.g., SR Grammars, Reserved Graph Grammar) and processing styles [FNT+97, ZZ97].

3.2 Semantics

The chosen principle for the runtime semantics*parse and traverse* states that the most important things are a traversing strategy and the semantic functions which must be executed when visiting a node (Figure 3). Therefore, the central components of the semantics are TraversalObject and SemanticObject (Figure 2).

The TraversalObject manages the node visiting order, provides semantic functions with information from the IR, and is the main engine of the MLI. The SemanticObject, for its part, contains all of the necessary semantic functions and provides for the execution environment. At the same time, we can also put into the semantic functions certain commands which force the Traverser to search for the needed node and to change the current execution point in the IR (traversing strategy changes and a transition to another node are problems in the Visitor pattern [e.g. Vis01]).

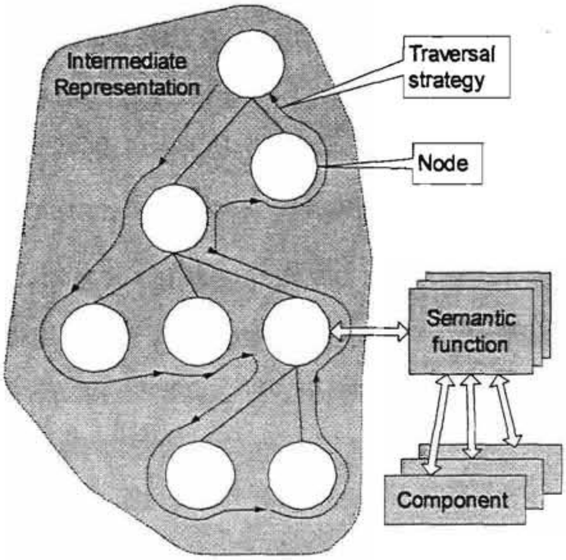


Figure 3 Runtime correspondence between syntax and semantics

Semantic functions have to be as simple and as small as possible. This can be achieved by using a meta-language and by employing high-level expression means, which allow for easy understanding and verification of the description. Following this principle becomes more natural if we use abstract components so that the underlying semantic can be clear without additional explanations (in Figure 3, the abstract components already have a concrete implementation component – A, B and C). This statement may lead to objections from the advocates of formal semantics, because the components are not described with mathematic precision. At the same time, however, formal semantics sometimes use such concepts as *Stack* or *Symbol table*.

Let us introduce a conceptual syntax element, which is a grammar symbol with a name (e.g., a named nonterminal symbol or a named terminal symbol). Considering the various types of syntax elements and the traversing strategy, we separate various visitations and introduce the concept of the traversing aspect. For instance, we can distinguish the arriving into node from the parent node (*PreVisit*) from the arriving into node from the child node (*PostVisit*). Thus we create the semantic functions and name them not only on the basis of the name of the syntax element but also on the basis of the arriving aspect (traversing aspect) into this element (Figure 3).

Runtime semantics or simply semantics for multi-language interpreters are a set of semantic functions. We represent the runtime semantic in Table 1. There is an executable code (□) or nothing (λ) for the syntax element, according to the traversing aspect. *n* depends on the size of syntax (e.g., the count of all nonterminal and terminal symbols), and *m* depends on the complexity of the traversing strategy (usually 1..3). We notice that the matrix mainly consists of empty functions (λ).

Table 1 The matrix of syntax elements and semantic function correspondence

Syntax Element (SE)	Traversing Aspect (TA)				
	TA_1	TA_2	TA_3	...	TA_m
SE_1	□	□	λ	...	λ
SE_2	λ	λ	□	...	λ
SE_3	λ	λ	λ	...	□
...	...	...	...	...	λ
SE_n	□	λ	λ	...	λ

The identification of semantic functions is realized both by the syntax name and by the traversing aspect name. Technical implementation may differ, but it is very advisable that functions identification and calling be performed with constant complexity O(1).

Now we arrive at the most difficult and important problem – how can we obtain semantic functions and ensure correct collaboration between them, and how is it possible to create reusable semantic descriptions? Let us explain our ideas about how to define semantics and how to gain the matrix observed above, i.e., how to generate executable semantics from the semantic description.

4 Semantic Aspects and Abstract Components

4.1 Semantic aspects

In practice, programming languages are frequently presented through the pragmatics of the programming language, i.e., examples are used to show how language constructs are exploited and what their underlying meaning is. Let us call these language constructs and their meaning *semantic aspects*.

We have chosen to define the semantic as a set of mutually connected semantic aspects. Here are some examples for typical groups of semantic aspects for imperative programming languages: execution of commands or statements (e.g., basic operations, variable declaring, assigning of a value to the variable, execution of arithmetic expressions), program control flow management (e.g., loop with a counter, conditional loop, conditional branching), dealing with symbols (e.g., variables, constants), environment management (e.g., the scopes of visibility). Here, too, are examples of nontraditional semantic aspects: pretty printing of the program, dynamic accounting of statistics, symbolic execution, specific program instrumentation, etc.

To describe the semantic aspect, we have chosen an operational approach – we define the computations, which a computer has to do to perform the semantic action.

4.2 Abstract data types and abstract components

The next significant principle to define the semantic aspect is using *abstract data types* (ADT) as much as possible. ADT is a collection of data type and value definitions and operations on those definitions, which behave as a primitive data type. This software design approach decomposes the problem into components by identifying the public interface and the private implementation.

In our case, typical examples of ADT are *Stack*, *Queue*, *Dictionary*, and *Symbol table* (in compiler construction theory [ASU86, FL88], in formal semantics [SK95]). In this way we hide most of the implementation details and concentrate mainly on the logic of the semantic aspect. Later we can choose the best implementation of ADT for the given task. Seeing that some exploited components can be complicated (*E-mail*, *Graph visualization*, *Distributed communication*, *Transaction manager*, etc.) and have no standards, we use another term – *abstract component*. Sometimes we want to utilize an already existing component, and the term *abstract component* seems more appropriate to us.

It is advisable to describe the semantic aspect through meta-language, even if you do not have a translator for this. Then you can translate or simply rewrite it by hand to the target programming language, select appropriate implementation for the abstract components, and use the needed interface, collaborating protocol and execution environment. For instance, Stack can be implemented in a contiguous memory or in a linked memory, Symbol table – as a list or as a dictionary with the hashing technique. Furthermore, instances of abstract components can be viewed as distributed objects in a heterogeneous computing network.

4.3 Examples of abstract components

Some abstract components and their operations are very popular, e.g., Stack (createStack, push, pop, top, etc.), Queue (createQueue, enqueue, dequeue, first, etc.), while some are guessed, e.g., E-mail (prepare, send, receive, open). Among the many specific components we would like to emphasize one that is useful for most of semantics - Symbol table (SymbolTable in Figure 2) or its analogue to provide the execution environment.

While building prototypes of the MLI, we have created an implementation of Symbol table – MOMS (Memory Object Management System) - that is appropriate for implementing the imperative programming languages. It is possible to define basic and user defined data types, to define base operations and functions, to operate with variables and their values, to manage the scope of visibility of all objects, etc. The most important data types, concepts, and operations of MOMS are listed in the appendix to this paper so as to give the reader a better idea about MOMS.

A second important component is Traverser (TraverserObject in Figure 2). Its main task is realizing the traversing strategy, to change the current execution point and to organize cooperation with the syntax object.

There is a depth-first left-to-right traversing strategy, which is used in the following examples (Table 2). This strategy has three visiting aspects: *Visit* (for tree leaves - terminals), and *PreVisit* and *PostVisit* (for the other tree nodes - nonterminals). To define semantic functions for examples, we have used the following operations: *NodeValue()* returns a value for the current terminal or nonterminal symbol (value from the current IR node), and both *goSiblForw(aName)* and *goSiblBackw(aName)* provide for a changing of the current node, searching the node with the name *aName* between siblings going forward or backward.

Table 2 A depth-first left-to-right traversing strategy

```
Traverse(node P)
   if IsLeaf(P)
      Visit(P)
   PreVisit(P)
   for each child Q of P,
   in order, do
      Traverse(Q)
   PostVisit(P)
```

It is possible to describe interfaces for SyntaxObject, TraverserObject and SemanticObject with domain-specific language. Then interfaces for obtaining the IR, manipulating with it and working with the symbol table can be compiled together, and it is possible to engage in high-level optimization and verification [Eng99].

The Traversing strategy can also be described with domain-specific language. This is important if the strategy is not trivial and depends on syntax elements and the program state [OW99 (traversing problems and solutions for Visitor pattern)]. The traversal strategy should be independent from syntax as much as possible and organized (combined) by patterns [Vis01]. In addition to common traversing strategies there are also less traditional ones, e.g., the strategy for reverse execution of the program [BM99]

4.4 Defining the semantic aspect

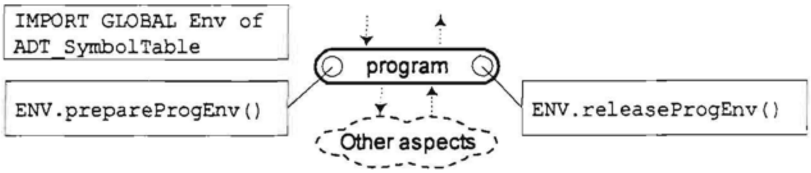
It is more convenient to define the semantic aspect by using diagrams (as in Figure 7). We can write a meta-program or a program in the target language in textual form, too. Diagrams contain syntax elements that are important for the semantic aspect and are visualized with graphic symbols. We can use different graphic notations. If the visiting order of syntax elements is important, then we mark the order with arrows.

Let us call the operations that are performed during the aspect node visiting *semantic action.* Semantic action is similar to semantic function, but it is written at the meta-level and relate only to a given semantic aspect. Semantic action is shown as a box with the meta-code connected to the syntax element and takes into account the traversing aspect.

There are all kinds of abstract data types that are needed for the semantic aspect into the box with the key words *IMPORT GLOBAL.* For better perceptibility of the semantic aspect, it is permissible to use additional graphic symbols that are not needed in real execution. For instance, we use *Other aspects* to signal that we expect there to be a composition with the other semantic aspects.

4.5 Examples of semantic aspects

Let us look at some examples of semantic aspects (Figure 4 - Figure 8) that are applicable for the simple imperative programming language Pam [Pag81]. Terminal symbols are denoted by a rectangle, while nonterminal symbols are indicated by rounded rectangles. The left circle in the nonterminals corresponds to the *PreVisit* semantic action, the right one – to the *PostVisit* semantic action, while for the terminals, the *Visit* semantic action is assigned.

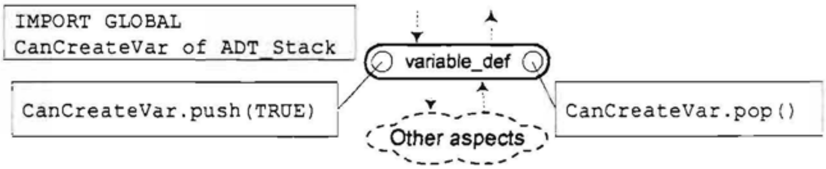


```
IMPORT GLOBAL Env of
ADT_SymbolTable
```

```
ENV.prepareProgEnv()
```

```
ENV.releaseProgEnv()
```

Figure 4 The semantic aspect PROGRAM. It prepares the program environment to manage variables, constants, etc. and operations involving them. The environment is destroyed at the end

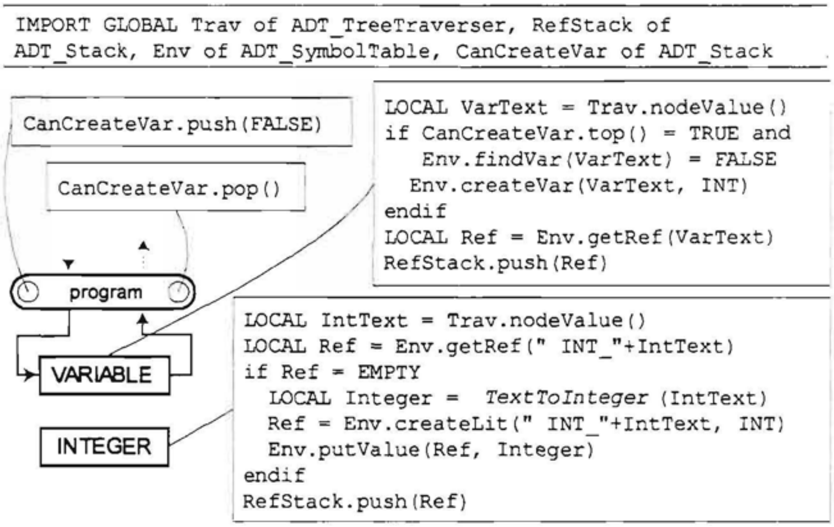


```
IMPORT GLOBAL
CanCreateVar of ADT_Stack
```

```
CanCreateVar.push(TRUE)
```

```
CanCreateVar.pop()
```

Figure 5 The semantic aspect VARIABLE DEFINITION. It allows for variable creation



```
IMPORT GLOBAL Trav of ADT_TreeTraverser, RefStack of
ADT_Stack, Env of ADT_SymbolTable, CanCreateVar of ADT_Stack
```

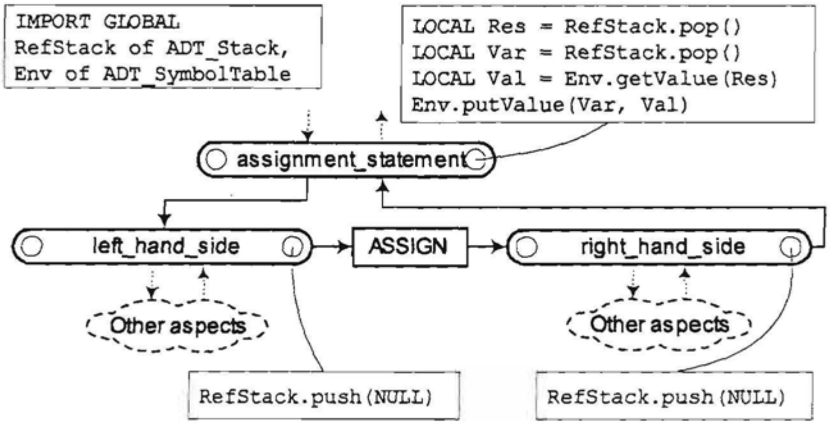
```
CanCreateVar.push(FALSE)
```

```
CanCreateVar.pop()
```

```
LOCAL VarText = Trav.nodeValue()
if CanCreateVar.top() = TRUE and
   Env.findVar(VarText) = FALSE
  Env.createVar(VarText, INT)
endif
LOCAL Ref = Env.getRef(VarText)
RefStack.push(Ref)
```

```
LOCAL IntText = Trav.nodeValue()
LOCAL Ref = Env.getRef(" INT_"+IntText)
if Ref = EMPTY
  LOCAL Integer =  TextToInteger (IntText)
  Ref = Env.createLit(" INT_"+IntText, INT)
  Env.putValue(Ref, Integer)
endif
RefStack.push(Ref)
```

Figure 6 The semantic aspect ELEMENT. It provides for the pushing into the stack all references to each variable encountered while traversing. Variable creation is forbidden by default. The Trav provides for getting the values of the current terminal node in IR.



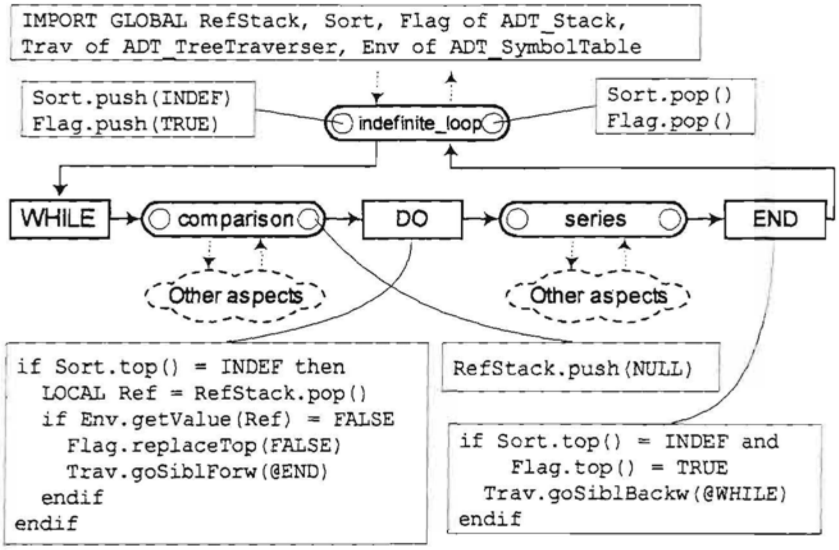
```
IMPORT GLOBAL
RefStack of ADT_Stack,
Env of ADT_SymbolTable
```

```
LOCAL Res = RefStack.pop()
LOCAL Var = RefStack.pop()
LOCAL Val = Env.getValue(Res)
Env.putValue(Var, Val)
```

```
RefStack.push(NULL)
```

```
RefStack.push(NULL)
```

Figure 7 The semantic aspect ASSIGNMENT. It takes reference to the variable and reference to the value from the stack and assigns a value to the variable. Pushing of references is simulated



```
IMPORT GLOBAL RefStack, Sort, Flag of ADT_Stack,
Trav of ADT_TreeTraverser, Env of ADT_SymbolTable
```

```
Sort.push(INDEF)
Flag.push(TRUE)
```

```
Sort.pop()
Flag.pop()
```

```
if Sort.top() = INDEF then
  LOCAL Ref = RefStack.pop()
  if Env.getValue(Ref) = FALSE
    Flag.replaceTop(FALSE)
    Trav.goSiblForw(@END)
  endif
endif
```

```
RefStack.push(NULL)
```

```
if Sort.top() = INDEF and
    Flag.top() = TRUE
  Trav.goSiblBackw(@WHILE)
endif
```

Figure 8 The semantic aspect INDEFINITE LOOP. It "goes through" the series and back to WHILE until the comparison sets a NULL reference or a reference with the value FALSE

5 Meta-semantics

Meta-semantics is a term, which relates to a meta-program that describes the counterparts of which semantics consist and the way in which these counterparts are connected together. The conceptual scheme of meta-semantics is shown in Figure 9.

If we look at semantics from the definition side (the logical view) then semantics are formed by traversing strategy and by a set of semantic aspects that consist of semantic actions realized while visiting the appropriate node of the intermediate representation. If we look at semantics from the runtime side (the physical view) then while visiting one node it is possible that several semantic actions have to be realized, and we have to ensure correct collaboration between all involved instances of abstract components into the desired environment. How can we put all of this together correctly?

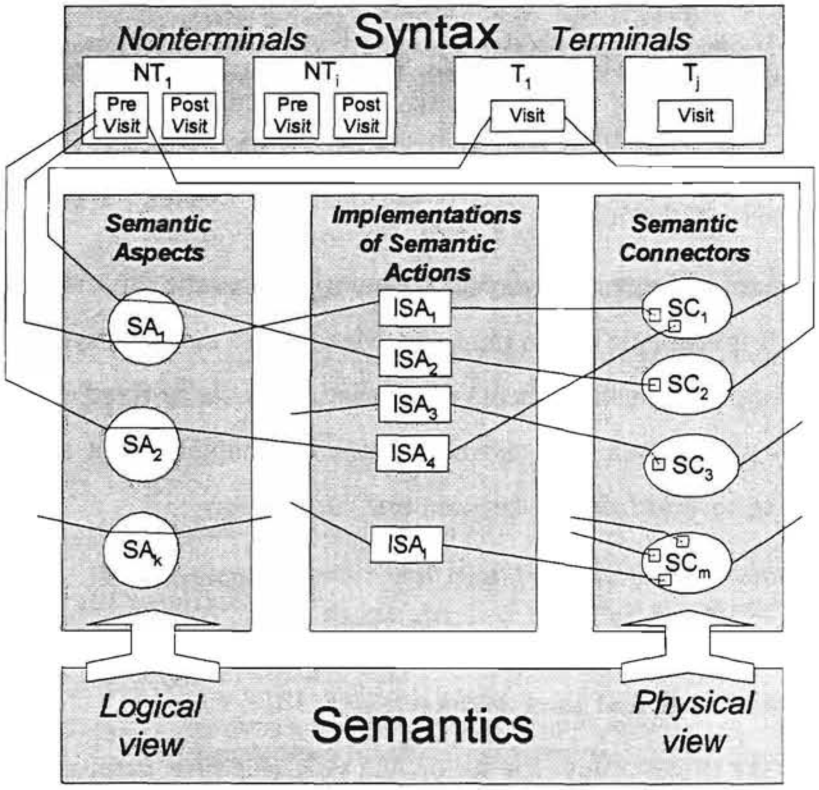


Figure 9 The conceptual scheme of meta-semantics. For instance, SA_1 involves nonterminal NT_1 and terminal T_1, and SC_1 is performed while*previsiting*. SC_1 is a composition of ISA_1 and ISA_4

To solve this problem we introduce the concept of the *semantic connector*. This concept has been adopted from the concept *Grey-box connector*, which solves a similar problem: how to connect the pre-built components in a distributed and heterogeneous environment for collaborating work [AGB00]. The Grey-box connector is a meta-program that introduces a concrete communications connection into a set of components, i.e., it generates the adaptation and communications glue code for a specific connection.

6 From semantic aspects to meta-semantics and executable semantics

The obtaining of semantics for the fixed syntax is achieved in several steps: 1) select predefined semantic aspects or define new ones for desired semantics, 2) rename syntax elements and traversing aspect in the selected semantic aspects with names from fixed syntax and traversing strategy, 3) rename instances of abstract components to organize collaboration between semantic aspects, 4) make composition from semantic aspects, 5) specify the runtime environment and translate the meta-code to the code of the target programming language, and 6) compile the semantics.

- Selection of semantic aspects (step 1)

If we have a library with previously created semantic aspects, then we can search for appropriate ones, i.e. reuse some parts of the semantics. The traversing strategy also has to be considered.

- Syntax element and traversing aspect renaming in semantic aspects (step 2)

Actually it is semantic action mapping. Matching to the fixed syntax elements is achieved by mapping syntax elements of semantic aspects to fixed syntax elements (*rename with* – simple mapping and *duplicate to* – mapping of a semantic aspect syntax element to several fixed syntax elements):

rename <name> <traversing aspect> *with* <target name> <target traversing aspect>
duplicate <name> <traversing aspect > *to*
    <target name> <target visiting aspect> [,<target name> <target visiting aspect> ...]

For instance, *rename* left_hand_side PostVisit *with* VARIABLE Visit ;

duplicate COUNTABLENODE Visit *to* VARIABLE Visit, assignment_statement PostVisit

- Renaming of instances of abstract components (step 3)

Matching of components is necessary because there is no direct data exchange between semantic functions, and we need collaborative work. The program state is fixed by using the runtime states of components. At first we decide what instances they have in common and what names they have to get, and then we rename instances in the semantic aspects:

replace <name> *with* <target name>

For instance, *replace* RefStack *with* DataStack ; *replace* Sort *with* LoopSortStack

- Composition of semantic aspects (step 4)

The goal of a semantic aspect composition is to bring together several semantic aspects into one more complicated aspect that nearly describes entire semantics. While composing, we stick together the meta-code of semantic actions that have the same name. The sticking principles can vary, for instance, *sequential* (one code is appended to the other one), *parallel* (codes can be executed simultaneously or sequentially in any order), *free* (user can modify the code union as he likes). To achieve better results, we ignore some semantic actions or apply the sticking principle to the semantic aspects in the reverse order. At this time we have to be aware of conflicts between local variable names.

compose aspect <<new SA>> (<refined SA>) [[*append* |*parallel*,*free*] (<refined SA>) ...] , where <refined SA> = <<old SA>> [*ignore* <name> <trav aspect> [,<name> <trav aspect>]] | [*reverse*]

The result is meta-semantics. An example of a meta-code fragment for meta-semantics is given in Table 3.

Table 3 An example of meta-semantics

```
compatible with
  ir_type ParseTree
  traverser_type ParseTreeTraverser

syntax elements (program, expression, VARIABLE, ...)
semantic actions (<PROGRAM> program PreVisit { ENV.prepareProgEnv()} ,
                  <PROGRAM> program PostVisit { ...} , ...)

global Trav of ADT_TreeTraverser
global Env of ADT_SymbolTable
create DataStack, OperatorStack, CanCreateVar, LoopSortStack,
    LoopCounterStack, LoopFlagStack, IfFlagStack of ADT_Stack
create InputFile, OutputFile of ADT_FILE

compose aspect <A1>     // composes semantic aspects from aspects given above
  (<PROGRAM>)           // semantic aspects PROGRAM remains the same
append (<ELEMENT>
  replace RefStack with DataStack // replaces stack for collaborating work
  rename INTEGER Visit with CONSTANT Visit)    // renames nonterminal according to PAM syntax
append (<ASSIGNMENT>
  replace RefStack with DataStack
  rename left_hand_side PostVisit with VARIABLE Visit,
         right_hand_side PostVisit with expression PostVisit
  ignore left_hand_side PostVisit)       // ignore pushing of NULL reference
append (<INDEFINITE LOOP>
  replace RefStack with DataStack,
    Sort with LoopSortStack, Flag with LoopFlagStack)
append (<VARIABLE DEFINITION>
  rename variable_def PreVisit with assign_statement PreVisit,
    variable_def PostVisit with ASSIGN Visit)
end compose aspect

compose aspect <A2>
(<A1>
  ignore expression PostVisit, comparision PostVisit)
  append ( ...
  /* Others aspect are appended such as <TYPE AND OPERATOR>, <INPUT>,
<OUTPUT>, <BASE BYNARY OPERATION>, <DEFINITE LOOP>, <CONDITIONAL STATEMENT> */
  ...)
end compose aspect
```

- Meta-code translating (step 5)

Meta-code is translated to the target programming language, taking into account the target language (e.g. C++), the implementation of abstract components (e.g. Stack in linked memory), the operating system (e.g. Unix), the communications between components (e.g. CORBA), MLI components type (e.g. DLL), etc. The translation may be done by hand or automatically (desirable in common cases).

- Obtaining semantic objects (step 6)

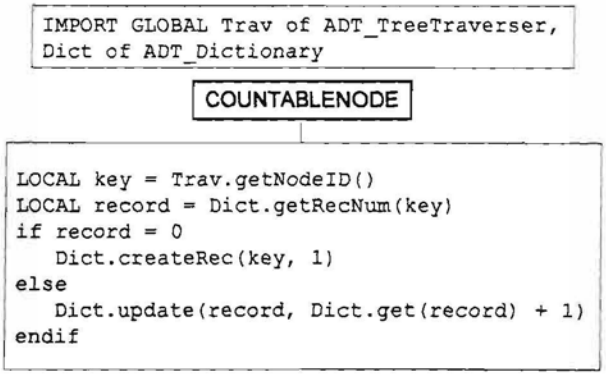
By compiling the code we get executable objects that provide semantic performance, i.e. they contain the semantic functions that are called while traversing the program intermediate representation. The instances of the concrete implementation of abstract components are created, or the existing ones are dynamically linked via selected communications protocols.

7 Examples of alternative semantic aspects

In this section we provide a short insight on how we can build nontraditional semantics. By adding new features to existing semantics we can create a specific tool that works with a given programming language. We would like briefly to survey two examples: 1) statistics accounting of program point visiting, and 2) storing of symbolic values for variables. Both aspects are added to the conventional semantics, and this is program instrumentation if we speak in terms of software testing.

7.1 Accounting of program point visiting

Our goal is to account for any visiting of a desirable program point. That means that we need to set counters at these points. At first we write the semantic aspect NODE COUNTER (Figure 10). We use the abstract component *Dictionary* where we can store, read and update records in form <*key, value*>.



```
IMPORT GLOBAL Trav of ADT_TreeTraverser,
Dict of ADT_Dictionary
```

```
LOCAL key = Trav.getNodeID()
LOCAL record = Dict.getRecNum(key)
if record = 0
   Dict.createRec(key, 1)
else
   Dict.update(record, Dict.get(record) + 1)
endif
```

Figure 10 The semantic aspect NODE COUNTER

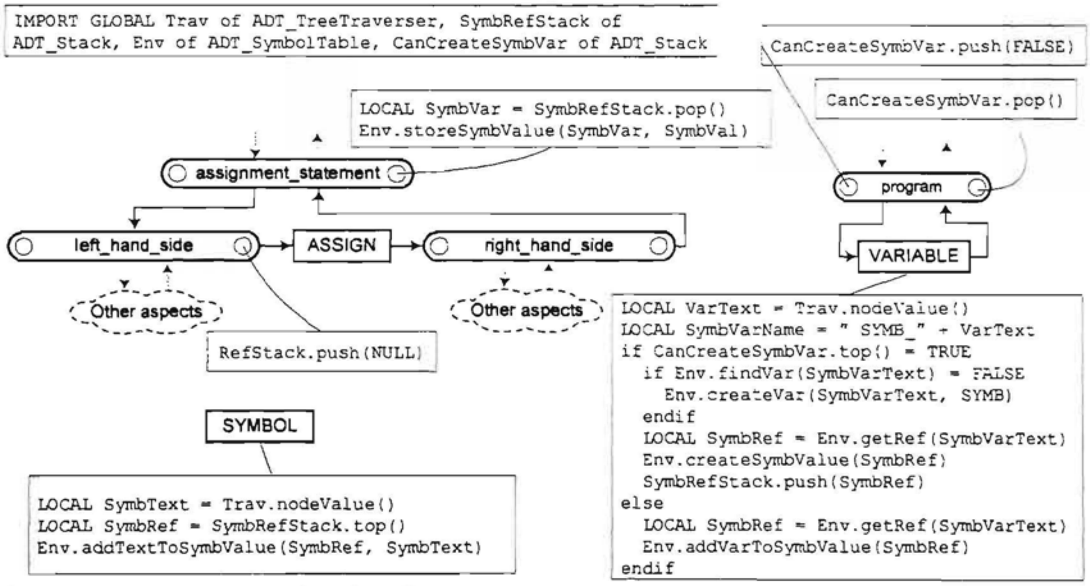
Defining meta-semantics, we add this semantic aspect to the others. For instance, we define accounting for the use of any variable and assignment operation:

dublicate countable_node Visit *to* VARIABLE Visit, assignment_statement PostVisit

Another semantic aspect can be built which accounts for every concrete variable using statistics into an additional dictionary (the variable name serves as the key). We can improve this aspect further by accounting for an aspect of variable use - defined, modified, referenced, released, etc. In the program analysis and instrumentation area, our approach is similar to the Wyong system (based on the Eli compiler generation system and the ATOM program instrumentation system), because specific operations are attached to syntax elements, and in this way we obtain a specific tool with additional semantics [Slo97].

7.2 Storing of symbolic values for variables

The second example provides for the fixing of symbolic values for variables (Figure 11). To do this task in an effective way, we have additional operations in our MOMS (symbol table). The operation *createSymbValue* creates an entry for symbolic value, and with the operations *addTextToSymbValue* and *addVarToSymbValue*, we form the value while traversing all nodes in the desired subtree (we store all needed program symbols and symbolic values of variables). At the end we store accumulated value with the operation *storeSymbValue.*



```
IMPORT GLOBAL Trav of ADT_TreeTraverser, SymbRefStack of
ADT_Stack, Env of ADT_SymbolTable, CanCreateSymbVar of ADT_Stack
```

```
CanCreateSymbVar.push(FALSE)
```

```
CanCreateSymbVar.pop()
```

```
LOCAL SymbVar = SymbRefStack.pop()
Env.storeSymbValue(SymbVar, SymbVal)
```

```
RefStack.push(NULL)
```

```
LOCAL SymbText = Trav.nodeValue()
LOCAL SymbRef = SymbRefStack.top()
Env.addTextToSymbValue(SymbRef, SymbText)
```

```
LOCAL VarText = Trav.nodeValue()
LOCAL SymbVarName = " SYMB_" + VarText
if CanCreateSymbVar.top() = TRUE
  if Env.findVar(SymbVarText) = FALSE
    Env.createVar(SymbVarText, SYMB)
  endif
  LOCAL SymbRef = Env.getRef(SymbVarText)
  Env.createSymbValue(SymbRef)
  SymbRefStack.push(SymbRef)
else
  LOCAL SymbRef = Env.getRef(SymbVarText)
  Env.addVarToSymbValue(SymbRef)
endif
```

Figure 11 The semantic aspect SYMBOLIC VALUES

8 Conclusions

This method was developed with the goal of reducing the gap between practitioners (tool developers) and theoreticians (developers of formal specifications for semantics). Our experience shows that the remarkable acquisition is achieved if abstract components or abstract data types realize the greater part of semantics, because that way it is easier to perceive the full implementation of semantics.

The second acquisition of our method involves significant disjoining of syntax and semantics from each other. It allows us to combine various syntaxes and semantics and to find out the most desirable semantics for the given syntax. So, if we have written syntax for a new language, we can match several semantics to it in a comparatively short time. As a result, we can develop a wide spectrum of tools in support of our new language.

Our approach allows us to change semantics dynamically while the interpreter is running, i.e. replace semantics or execute various ones simultaneously. It is possible to reduce a derived parse tree (by deleting nodes with empty connectors) or to optimize it (tree restructuring statically and dynamically, considering performance statistics).

At this moment the environment for tool construction or semantics generation is not completely developed. Our experience shows that tools can be developed without significant investments, for example, by using Lex/YACC as a generator to create a syntactic object, which produces program intermediate representation. It is not too hard to develop a Traverser and a simple SymbolTable. And as the last job, we have to work up semantic aspects on the basis of our method and compose them, thus obtaining connectors, which can be written in some common programming language (skipping meta-language use and its translation). The use of abstract components depends on target semantics.

We believe that the development of the serious tools demands a more universal implementation of the Symbol table. Our Symbol table implementation - MOMS - is not applicable only for imperative language implementations. It was also the basic object-oriented database for the commercial application Mosaik (Sietec consulting GmbH Co. OHG, graphical CASE tool for business modeling).

The weakness of our method lies in the semantic aspects composition stage. At this moment we have not analyzed all risks in terms of obtaining senseless or erroneous semantics. The problems are not trivial, and they are similar to problems in the proper collaboration of objects or components in object-oriented programming, too. [e.g. ML98]. Most name conflicts can be precluded automatically, but it is considerably harder to organize collaboration among the common components in semantic aspects (it is easier if the semantic aspects are mutually independent).

Another problem is that the language grammar is frequently not context free (this is true of our example above, too). In this case we have to introduce additional flags to memorize the context of syntax elements. It is advisable to rewrite the syntax and to use context-free grammars.

9 References

[AAB96] V. Arnicane, G. Arnicans, and J. Bicevskis. Multilanguage interpreter. In H.-M. Haav and B. Thalheim, editors, *Proceedings of the Second International Baltic Workshop on Databases and Information Systems (DB&IS '96)*, Volume 2: Technology Track, pages 173-174. Tampere University of Technology Press, 1996.

[AGB00] U. Aßmann, T. Genßler, and H. Bär. Meta-programming Grey-box Connectors. *Proceedings of the Technology of Object-Oriented Languages and Systems (TOOLS 33)*, pp.300-311, 2000.

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Technigues, and Tools.* Addison-Wesley, 1986.

[BM99] B. Biswas and R. Mall. Reverse Execution of Programs. *ACM SIGPLAN Notices*, 34(4):61-69, April 2000.

[Cla99] C. Clark. Build a Tree – Save a Parse. *ACM SIGPLAN Notices*, 34(4):19-24, April 2000.

[DKV00] A. Deursen, P. Klint, and J. Visser. Domain-Specific Languages: An Annotated Bibliography. *ACM SIGPLAN Notices*, 35(6):26-36, June 2000.

[Eng99] Dawson R. Engler. Interface Compilation: Steps toward Compiling Program Interfaces as Languages. In DSL-99 [ITSE99], pp.387-400.

[FL88] Charles N. Fisher, and Richard J. LeBlanc, Jr. *Crafting A Compiler.* Benjamin-Cummings, 1988.

[FNT+97]F. Ferrucci, F. Napolitano, G. Tortora, M. Tucci, and G. Vitiello. An Interpreter for Diagrammatic Languages Based on SR Grammars. *Proceedings of the 1997 IEEE Symposium on Visual Languages (VL '97)*, pages 292-299, 1997.

[GHJV95]E. Gamma, R. Helm, R. Johnson, and J. Vlisides. *Design Patterns: Elements of Reusable Software*, pages 331-334. Addison-Wesley, 1995.

[HK00] J. Heering and P. Klint. Semantics of Programming Languages: A Tool-Oriented Approach. *ACM SIGPLAN Notices*, 35(3):39-48, March 2000.

[ITSE99] Special issue on domain-specific languages. *IEEE Transactions on Software Engineering*, 25(3), May/June1999.

[Kin95] W.Kinnersley, ed., The Language List. 1995. http://wuarchive.wustl.edu/doc/misc/lang-list.txt

[Lou97] Kenneth C. Louden. Compilers and Interpreters. In Tucker [Tuc97], pp.2120-2147.

[ML98] M. Mezini and K. Lieberherr. Adaptive Plug-and-Play Components for Evolutionary Software Development. SIGPLAN Notices, 33(10):97-116, 1998. *Proceedings of the 1998 ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages & Applications (OOPSLA '98).*

[OW99] J. Ovlinger and M. Wand. A Language for Specifying Recursive Traversals of Object Structures. SIGPLAN Notices, 34(10):70-81, 1999. *Proceedings of the 1999 ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages & Applications (OOPSLA '99).*

[Paa95] J. Paakki. Attribute Grammar Paradigms – A High-Level Methodology in Language Implementation. *ACM Computing Surveys*, 27(2):196-255, June 1995.

[Pag81] Frank G. Pagan. *Formal Specification of Programming Languages: A Panoramic Primer*. Prentice-Hall, 1981.

[Sch97] David A. Schmidt. Programming Language Semantics. In Tucker [Tuc97], pp.2237-2254.

[SK95] K. Slonneger and B. L. Kurtz. *Formal Syntax and semantics of Programming Languages: A Laboratory Based Approach*. Addison-Wesly, 1995.

[Slo97] A. M. Sloane. Generating Dynamic Program Analysis Tools. *Proceedings of the Autralian Software Endineering Conference (ASWEC'97)*, pp.166-173, 1997.

[Tuc97] Allen B. Tucker, editor. *The computer science and engineering handbook*. CRC Press, 1997.

[Vis01] J. Visser. Visitor Combination and Traversal Control. SIGPLAN Notices, 36(11):270-282, 2001. *Proceedings of the 1998 ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages & Applications (OOPSLA '01).*

[ZZ97] D.-Q.Zhang and K.Zhang. Reserved Graph Grammar: A Specification Tool For Diagrammatic VPLs. *Proceedings of the 1997 IEEE Symposium on Visual Languages (VL '97)*, pages 292-299, 1997.

10 Appendix

Table 4 MOMS types

Type	Description
Constructor	Handle to an object type description
Value	Handle to a byte stream that contains an object value
Reference	Handle to a memory object
MemoryMap	Main central object that organizes other objects (other MemoryMap also)
IdentifDict	Dictionary of all identifiers (variables, constants, etc.)
TypesDict	Dictionary of all types (basic types and user defined)
FunctDict	Dictionary of all functions (basic operators, basic functions, user defined functions)
NamesTable	Compact storing of all strings
ConstrTable	Compact storing of all constructor descriptions
ValuesTable	Storing and managing of all object values
MemoryBlock	Organizing scope visibility in all dictionaries and providing memory management
...Others	Stack, queue, collection, etc.

Table 5 System initialization and global operations

Operation	Description
MemoryMap **initialize**(Uint memoryCount)	Creates MOMS with internal parallel but related memories (MOMS)
Uint **switchMemoryTo**(Uint memNum)	We can exploit only the specific internal memory
Uint **getCurrentMemory**(void)	Returns the number of the actual memory
defineCountOfBaseTypes(Uint countOfBaseTypes)	Defines a count of the basic types of MOMS

defineBaseType(char* typeName, Uint type, Uint typeSize)	Defines base types. This interface is in C and depends on previously defined types. Some examples:
	defineBaseType("long_", LONG_, sizeof(long_));
	defineBaseType("boolean_", BOOLEAN_, sizeof(boolean_));
	defineBaseType("date_", DATE_, sizeof(date_))

defineBaseFunction(char* langFunctName, char* internalName, Uint returnType, int paramCount, ...)	Defines the base operations and functions. This interface is in C and depends on previously defined types. Some examples:
	defineBaseFunction("+", "PLUS", LONG_, 2, LONG_, LONG_);
	defineBaseFunction("day", "day", LONG_, 1, DATE_)

prepareProgramEnv(Uchar scope)	Prepares a new MemoryBlock, defines the scope (visibility) of previously defined variables, types, functions
releaseProgramEnv(void)	Releases a current MemoryBlock and all related memory in other objects (dictionaries, tables) and restores a previously defined MemoryBlock
defineAutomaticMemSwitching(Uint firstMemNum, Uint lastMemNum)	Provides for automatic switching in various functions. For instance, we look up the variable in a local memory and then in a global memory (if the variable is not founded yet).
... Others	

Table 6 Defining of user defined data types

Operation	Description
ConstrPtr **createConstrArray**(Uint minIndex, Uint maxIndex, ConstrPtr ptrToElemConstr)	Defines an array type with the given dimensions and element types. Here and in other functions we can use any previously defined (or partly defined) data type.
ConstrPtr **createConstrFunct**(ConstrPtr ptrToReturnConstr, ConstrPtr ptrToParamConstr)	Defines a function type with the given parameters and return type.
ConstrPtr **createConstrName**(char* aName, ConstrPtr trToSubConstr)	Assigns a user-defined name for the given type.
ConstrPtr **createConstrPointer**(ConstrPtr ptrToSubConstr)	Defines a pointer type to the given type.
ConstrPtr **createConstrProduct**(ConstrPtr ptrToSubConstr1, ConstrPtr ptrToSubConstr2)	Creates a production of two types (establishes some relation between them). It is useful to construct a serious data structure.
ConstrPtr **createConstrRecord**(ConstrPtr ptrToSubConstr)	Defines a record data type (a set of pairs {name, type}).
... Other constructors	For instance, base data type constructors
constrArraySetMinIndex(ConstrPtr ptrToConstr, Uint minIndex)	Modifies the type description (attributes).
Uint **constrArrayGetMinIndex**(ConstrPtr ptrToConstr)	Provides details about data type attributes.
... Others	

Table 7 Operations with variables and similar objects

Operation	Description
createVar(char* aName, ConstrPtr ptrToConstr)	Creates a variable with the given name and type.
createVar(char* aName, char* typeName)	Creates a variable with the given name and type name.
createLiteral(char* aName, ConstrPtr ptrToConstr)	Creates a literal (constant) with the given name and type.
Ref **createRef**(ConstrPtr ptrToConstr)	Creates an object without a name with the given type, for instance, internal loop counter, return value of function.
ConstrPtr **getConstrRef**(char* typeName)	Returns pointer to type with the given type name.
createSynonym(char* aName, Ref aRef)	Creates another reference by name to the existing object.
... Other	

Table 8 Operations with value

Operation	Description
putValue(Ref aRef, char* aValue)	Sets a new value for the object.
char* **getValue**(Ref aRef)	Returns a value for the object.
char* **createDynamicValue**(ConstrPtr ptrToConstr)	Provides dynamic memory allocation for the object given by type.
deleteDynamicValue(char* aValue)	Releases dynamically allocated memory.
setValueProtectionOn(char* aName)	Protects a value of the given object against modification, for instance, protects constants.
setValueProtectionOff(char* aName)	Takes off a value protection.
gotoArrayElementConstr(Ref& aRef, Sint index)	Sets a virtual mark to element constructor and to a given array element value. aRef is modified, it refers to the array element.
gotoNameConstr(Ref& aRef)	Moves the virtual mark to the name constructor.
gotoPointerConstr(Ref& aRef)	Moves the virtual mark to the pointer subconstructor and to the start of value.
gotoProductionLeftConstr(Ref& aRef)	Moves the virtual mark to the left subconstructor and to the start of the corresponding value.
gotoProductionRightConstr(Ref& aRef)	Moves the virtual mark to the right subconstructor and to the start of the corresponding value.
gotoRecordConstr(Ref& aRef)	Moves the virtual mark to the start of record.
gotoNameInList(Ref& aRef, char* aName)	Moves to the appropriate type and value (list of named types linked by productions) E.g., search a field in the user-defined structure.
... Other	