Development of Dynamic Subject Domain Based on Distributed Expert Knowledge

Anna Karkanitca
Yanka Kupala State University of Grodno, Belarus, Grodno, 230023, 22 Ozheshko str.,
karkanica@gmail.com

***Abstract**: The paper describes the technology of development a dynamic subject domain based on distributed expert knowledge. The implementation of algorithms and technological components of this technology is discussed.*

Keywords: dynamic subject domain, domain model, distributed resources, knowledge acquisition.

1. INTRODUCTION

In a modern society there is a considerable quantity of complicated tasks which demand the prompt and qualitative solutions.

Quality of solution can be provided at the expense of use advanced innovative knowledge. Experts are the sources of this knowledge. They are able to provide an actuality, accuracy and completeness of knowledge. Experts are dispersed worldwide. They are distributed territorially, removed geographically from each other. Actually their knowledge is distributed resource [1]. Thus, we face a problem of acquisition of the distributed expert knowledge. It is not only aspect of a problem of the solution of complicated tasks.

Under conditions of the prompt rates of development of technical progress, information technology, communication facilities, knowledge quickly becomes obsolete and loses the topicality. Hence, it is necessary to reduce the time expended on knowledge acquisition and their processing.

Such tasks type has a subject domain which is defined as the sum of the innovative knowledge received from distributed experts. Properties of knowledge are specific (knowledge are distributed, dynamical and have a short life cycle). So we are faced with the problem of constructing a subject domain which is, first, is formed from a variety of distributed resources, and secondly, is dynamic. Tasks of a performance of large-scale projects by the team of territorially remote executors concern the tasks of this type. Also it can be manage task of the distributed organizational structure united in some administrative hierarchy, software development by the distributed teams of developers and others. Characteristic feature of these tasks is a need for allocation of work between the executors (task decomposition), followed by the union of the results. Necessity to get a quick solution requires automation of the process of forming the subject domain. For this purpose it is necessary to develop a technology which would contain means and the mechanisms allowing us to form a structurally-information model of the subject domain, to produce a modification of the model and to acquire knowledge from distributed sources.

In terms of computer science, the important components of this technology are:

- information component (set of formalized knowledge necessary and sufficient for the task solution);
- algorithmic component (set of methods and algorithms providing obtaining of task solution);
- technological component (technical tools and software for automation of processes of creation domain model and acquisition of expert knowledge).

In this paper the implementation of algorithmic and technological component is proposed.

2. PROBLEM DEFINITION

Problem definition we will formulate as follows: let **T** be the task demanding the operative solution. We will define **T** as a set of following components:

$$T = (S, Group, S^1, \ldots, S^n, Req^1, \ldots, Req^k, Solution^1, \ldots, Solution^n, t^1, \ldots, t^n), \quad (1)$$

where S – statement of the general problem;
Group = (Center, $Expert^1$, $Expert^2$, …, $Expert^n$) – group of executors including center (Center), initiating the task and the distributed executors (Expert) which implement the subtasks solution;
S^1, …, S^n – set of subtasks received as a result of decomposition S;
Req^1, …, Req^k – requirements to solutions;
$Solution^1$, …, $Solution^n$ – formalized information received in the course of subtasks solution;
t^1, …, t^n – restrictions of temporal expenses for subtasks solution.

It is required to develop the program technology providing:

1. Construction of model domain of task T.
2. Decomposition of the model on fragments corresponding to subtasks (S^1, …,S^n).
3. Modification of the model in case of subtasks topology changing and connecting of new executors.
4. Acquisition of knowledge ($Solution^1$, …,$Solution^n$) from remote sources ($Expert^1$, $Expert^2$, …, $Expert^n$).

It is proposed to implement the solution of the problem in several stages:

1. To define a concept of dynamic domain model as an information structure which represents hierarchy of subtasks and has internal mechanisms of dynamic adaptation (structural and informational) throughout all life cycle of the model.
2. To select a method of representation of model in the form of the tree-like graph, each vertex of the graph is marked by set of attributes identifying a subtask.
3. To develop algorithms for constructing and modifying the model of the dynamic subject domain based on algorithmic graph theory.
4. To develop a method for describing the model in the form of formal specification.

5. To develop architecture of program system for automation of process of creation, visualization and modification of dynamic domain model.

Further we will consider implementation of each of stages.

3. ALGORITHMS FOR CREATION AND MODIFICATION OF MODEL DOMAIN

We will represent a domain model of task T using a graph theory. Let define a concept of the dynamic graph.

Graph **G** we will name dynamic graph if it is possible to passage from state G^1 (V^1, E^1) to state G^2 (V^2, E^2), and sets (V^1, V^2) and (E^1, E^2) accordingly, don't coincide.

We say that a modification of the graph G is a process of passage of G from state G^1 at the moment of time t^1 to state G^2 at the moment of time t^2 which can be caused because of some sequence of operation (adding vertex, removing vertex, graph partition, merging graphs). According to (1) we will associate each vertex of the graph with the following set of attributes:

$$v = \langle id, task, name, addr, status, inf \rangle, \quad (2)$$

where id – the unique identifier of vertex (task), task – task description (requirements to solution), status – vertex state (0 - the task is initiated, but it is not solved; 1 – the task is in the course of the solution, 2 – the task is solved), name – the unique identifier of the expert, addr – the address of the expert, inf – information component (actually the solution of the task presented in one of admissible formats).

Further we will name the tree-like hierarchical graph as a tree.

Let on set of vertex of tree G the set of the admissible operations designated above is defined. We need to develop a data structure for representation of the dynamic tree under a sequence of two kinds of operations: a link operation that combines two trees into one, and a cut operation that divides one tree into two.

To solve this problem we propose a modification of the dynamic structures and algorithms developed by R. Tarjan and D. Sleator (A Data Structure for Dynamic Trees [2]). The choice is justified by the fact that this data structure has a time bound of O(1ogn) per operation when the time is amortized over a worst-case sequence of operations.

We want to represent the trees by a data structure that allows us easily to extract certain information about the trees and easily to update the structure to reflect changes in the trees caused by two kinds of operations:

- link(v, w): If v is a tree root of G and w is a vertex in another tree, link the trees containing v and w by adding the edge(v, w), making w the parent of v.
- cut(v): If node v is not a tree root, divide the tree containing v into two trees by deleting the edge from v to its parent.

We will describe these operations under a sequence of simple operations, which can be intermixed in any order and that are allowed on the set of vertices and the set of edges of the dynamic tree:

parent(vertex v): Return the parent of v. If v has no parent (it is a tree root), return a special value null.
root(vertex v): Return a root of the tree containing v.
path(vertex v): Return the path containing v.
head(path p): Return the head (first vertex) of p.
tail(path p): Return the tail (last vertex) of p.
after(vertex v): Return the vertex after v on path(v). If v is the tail of the path, return null.
concatenate(path p,q): Combine p and q by adding the edge (tail(p),head(q)) . Return the combined path.
split(vertex v): Divide path(v) by deleting the edges incident to v. Return a list [p, q], where p is the subpath consisting of all vertices from head(path(v)) to before(v), q is the subpath consisting of all vertices from after(v) to tail(path(v)).

The set of valid operations allows us to formulate algorithms for creation and modification of a dynamic tree in terms of these operations. We describe these algorithms using a version of Dijkstra’s command language:

1. Operation *parent*

```
function parent(vertex v);
  if v = tail(path(v)) return dparent(v);
  if v< > tail(path(v)) return after(v);
end parent;
```

2. Operation *root*

```
function root(vertex v);
 return tail(expose(v))
end root;
```

3. Operation *link*

```
procedure link(vertex v, w, real x);
concatenate(path(v), expose(w), x)
end link;
```

4. Operation *cut*

```
function cut(vertex v);
path p, q; real x, Y;
expose(v);
[p,q,x,y] := split(v);
dparent(v) := null;
return y
end cut;
```

4. FORMAL SPECIFICATION OF DYNAMIC TREE MODEL

There are some conventional standards for the description of graphs and graph models. The most of them don't give us the flexible mechanism of the extension of language by adding new properties or new structural elements of the graph. That doesn't allow us to use these formats for representation of dynamic domain model.

DGML is deprived this lack. DGML is a directed graph markup language which is based on simple XML. Significantly is the fact that DGML can be expanded to include structured elements corresponding to the specific subject domain.

In this section, we describe how the basic graph-topology (nodes and edges) are represented in DGML. We propose to extend the format DGML by defining additional elements that will represent the dynamic graph model in accordance with the model (1) and given a set of attributes (2).

The nodes of a graph are represented by a list of <Node> elements. Each node must have an id attribute. The edge set is represented by a list of <Link> elements. Edges and nodes may be ordered arbitrarily and it is not required that all nodes are listed before all edges. Clearly, the space requirement for storing a graph with n nodes and m edges in DGML is in O(n + m).

Let an element <Node/> be a node of the graph corresponding to a subtask, an element <Link/> - a link connecting an initial node with a target node and setting a graph edge. An element <Node/> should contain a set of the obligatory attributes allowing to describe a subtask. An element <Link/> should contain a set of the obligatory attributes defining hierarchy of subtasks. With the help of the extension DGML-attributes one can specify additional information of simple type for the elements of the graph. Simple type means that the information is restricted to scalar values, e. g. numerical values and strings. So we can define a list of <Property/> elements with additional attributes:

```
<Properties>
  <Property Id=”TaskID” Label=”task identifier”
DataType=”string” />
  <Property Id=”Status” Label=”task status”
DataType=”int” />
   <Property Id=”NameID” Label=”expert identifier”
DataType=”string” />
  <Property Id=”Addr” Label=”expert address”
DataType=”string” />
</Properties>
```

For determination of hierarchy of subtasks the element <Link/> should contain at least two attributes: Source – unique identifier of initial node of an edge, Target – unique identifier of target node. The set of edges of the graph can be presented as follows:

```
<Links>
        <Link Source=”…” Target=”…”/>
        <Link Source=”…” Target=”…”/>
        <Link Source=”…” Target=”…”/>
        ...
</Links>
```

As a result we have the formal specification for representation of dynamic domain model of the task (1) which is implemented at the expense of usage of the DGML extension mechanism.

5. ARCHITECTURE OF PROGRAM SYSTEM

It is necessary to develop software to automate the process of constructing the domain model. The architecture of a software system should be focused on what sources of information are distributed.

We propose a multi-component architecture of software. Each component implements an appropriate process of solving the problem (1): construction and modification of the domain model, knowledge acquisition, synthesis of the final solution.

The architecture of a software system is represented by a set of the following components:

- Builder: Solves the problem of constructing and modifying the domain model.
- Analyzer: Analyzes the subject domain in terms of its completeness.
- Miner: organizes the interaction between distributed experts to acquire knowledge.
- Combiner: performs the synthesis of the final solution.

All components use a common unit of data access.

The component *Builder* is a software component designed to allow for the construction, storage, modification and visualization of the domain model. The implementation of components is executed on the Java platform using open source frameworks and tools for Java-environment (Hibernate, Spring Framework, Spring Security).

6. CONCLUSION

In this paper we have considered a class of problems whose solution is formed as a result of the joint activity of distributed experts. We have identified a subject domain of this class of problems as a dynamic subject domain. Domain model, algorithms for creation and modification of model domain, formal specification of dynamic tree model are proposed. As a result, we have described the automation technology for creation dynamic subject domain. Implementation of several stages of this technology is presented.

7. REFERENCES

[1] V. Krasnoproshin, O. Konovalov, A. Valvachev. Technology of building knowledge bases on distributed cognition resources, *Herald of the National Technical University "KhPI". Subject issue: Information Science and Modelling. – Kharkov: NTU "KhPI*". – 2010. – №. – P

[2] D.D. Sleator, R.E. Tarjan. A Data Structure for Dynamic Trees, *Journal of Computer and System Sciences*, 26 (3) 1983 p. 362-391