Computer Science Journal of Moldova, vol.**18**, no.1(52), 2010

A flexible navigation mechanism for complex data models

Oleg Burlaca

Abstract

The paper presents a way to build flexible navigation tools over a big dataset of well structured data models. The mechanism is underpinned by a search engine that is used to slice and dice the database. By applying a series of consecutive groupings, the result of a search query can be organized in a hierarchical structure and browsed using traditional user interface controls.

1 Introduction

Building successful user interfaces (UI) requires a good understanding of the end user needs. As the data model of an application evolves with the addition of new objects and relationships, it's hard to create a sustainable UI because of the user's ever changing requirements. To alleviate the issue we should devote more time to the initial development phase: the software architecture. Nevertheless, it's not always possible to encompass all aspects at the beginning. Moreover, the requirements may change at a later stage when the software is already delivered and launched.

The approach presented in [1] works well for websites due to simple data models that were used. As the number of entities in a model increase, there is a need to quickly seek a desired object basing on different criteria. It may happen that the user knows the chain of interrelated objects that are connected to the object he is looking for, and would like to easily spot it by "jumping" from one object to another.

Any information system that deals with a vast amount of data should feature a search engine. Our idea is to leverage the search engine and build the navigation system on the fly. In the next section we introduce the data model we used for our application. Then we discuss the search engine implementation, and finally we present the idea of a ”navigation widget” that is powered by the search engine.

2 The Data Model

The Events Data Model elaborated by HURIDOCS [2] is depicted in Figure 1.

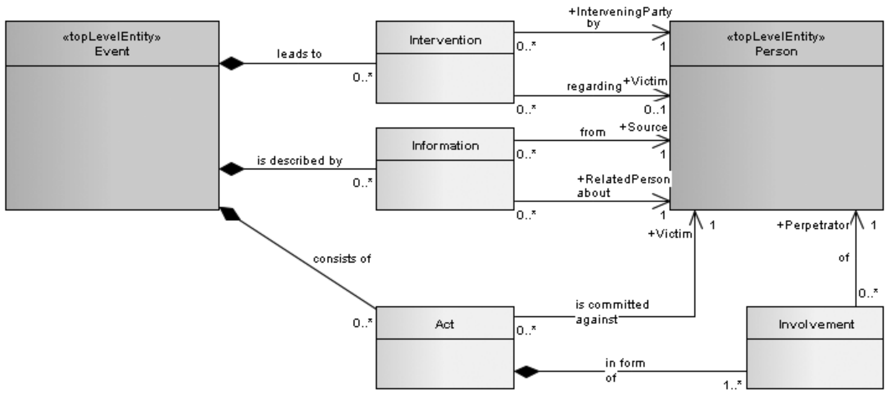
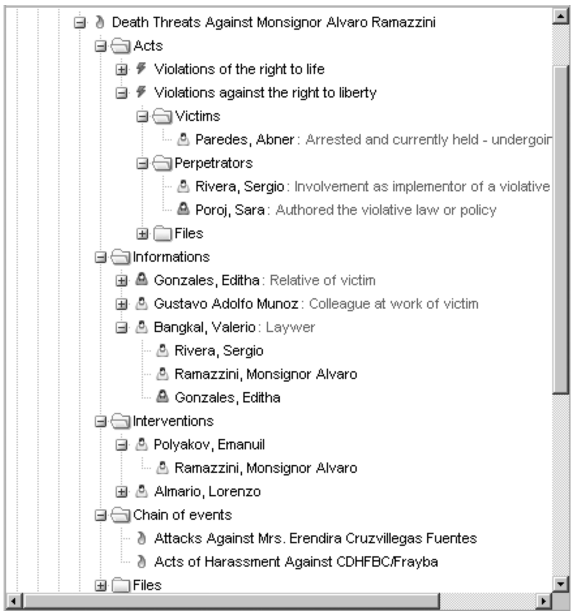


Figure 1. Events Data Model.

An Event is a complex thing, it is comprised of several entities: intervention, information and act. Acts are further comprised of victims, perpetrators. Trying to build an interface for editing an Event seems to be a tedious task and it might happen that due to the diverse number of sub-entities (interventions, acts) and complementary tasks (adding a new person or uploading a file), it might get cumbersome and a hard nut to crack for simple users. The proposed solution is to work with only one entity at a time and provide a suitable ”contextual skeleton” for the user. In other words, we try to split a complex problem into smaller, but simpler ones, thus applying the divide and conquer design paradigm. (See [3] for an in depth discussion about the UI part).

By looking at the model in Figure 1, one can notice that if to exclude the person entity, the remaining structure has only 1 to Many relationships, i.e.: An event has many intervention, information and act entities. An act has many involvement entities. *It allows us to display an event as a tree.* In Figure 2 one can see the "Death Threats Against Monsignor Alvaro Ramazzini" event with all its acts, victims, perpetrators, information and other entities. We can think about this tree as a "contextual skeleton": the user will be able to edit a single entity from the tree, but he will always know to which parent entities it belongs (is it a victim or a perpetrator and to which act and event it belongs).



```
Death Threats Against Monsignor Alvaro Ramazzini
  Acts
    Violations of the right to life
    Violations against the right to liberty
      Victims
        Paredes, Abner : Arrested and currently held - undergoir
      Perpetrators
        Rivera, Sergio : Involvement as implementor of a violative
        Poroj, Sara : Authored the violative law or policy
      Files
  Informations
    Gonzales, Editha : Relative of victim
    Gustavo Adolfo Munoz : Colleague at work of victim
    Bangkal, Valerio : Laywer
      Rivera, Sergio
      Ramazzini, Monsignor Alvaro
      Gonzales, Editha
  Interventions
    Polyakov, Emanuil
      Ramazzini, Monsignor Alvaro
    Almario, Lorenzo
  Chain of events
    Attacks Against Mrs. Erendira Cruzvillegas Fuentes
    Acts of Harassment Against CDHFBC/Frayba
  Files
```

Figure 2. A hierarchical representation of an Event.

Let's assume that the data entry aspect is solved [3] and we have a quite big database of such events (some real world databases have more than 20,000 events). We need a mechanism to easily browse this dataset, but we can't know beforehand how a user might want to browse the collection. In the next section we describe a search engine that will power our navigation mechanism: imagine that you'll be able to browse events in different ways: by date (an entity attribute), by type of act (a relationship), by method of violence, etc.

3 The Search Engine

There are different types of entities in the system, thus creating a search query will start by selecting what kind of entities to return (Events, Acts, Information). But it doesn't mean that it will be impossible to search for one type of entity and have attributes of another type in search results. For example, it will be possible to search for acts and display the Event.Title. Because of the 1 to Many nature of the event data model (an event has many acts, an act has many involvements, but an act CAN'T belong to many events), one can uniquely identify the parent entity. It means that if there are Involvements in search results, one can also display Act.TypeOfAct and Event.Title. Conversely, one can't search for Events and display Act.TypeOfAct in search resutls, because an event has Many acts.

In conclusion, *you need to search for the deepest type of entity that appears in search conditions if you need to display its attributes in search results.* It may sound a bit abstract for now, practical examples provided in the next section will clarify things.

3.1 Simple search

A simple search query consists of:

a) Type of entity to fetch results from (Events, Acts, Interventions etc.). We'll refer to it as *ResultEntity*.

b) A set of search conditions coupled by logical operators: AND, OR, AND NOT, OR NOT. A search condition may refer to a type of entity (let's call it EntityX) that is a parent or child for ResultEntity. This rule is *transitive.* i.e. the relationship is not required to be direct, it's ok if EntityX is a parent of a parent of ResultEntity or vice versa: EntityX is a child of a child of ResultEntity.

c) A set of attributes from ResultEntity and all its *parent entities* to display in search results.

The search query illustrated in Figure 3 has 2 conditions, the first one refers to the Event entity and the second refers to the Act entity.

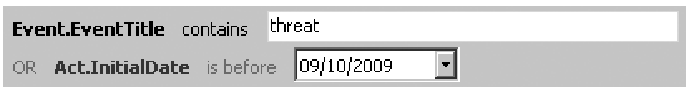

Event.EventTitle contains threat
OR **Act.InitialDate** is before 09/10/2009

Figure 3. A simple search query.

Following the rule "you need to search for the *deepest type of entity* that appears in search conditions", the user needs to select the Act entity as ResultEntity if he needs to display the Act.TypeOfAct attribute in search results. If he selects Events as ResultEntity, the search will work, but one will not be able to display Act attributes.

You should be aware that search results depend on the ResultEntity, and sometimes even the correctness (validity) of a query changes. Let's see this in practice. Take the example from Figure 3. Suppose we have two events in the database and we need to display only the title of the event in search results:

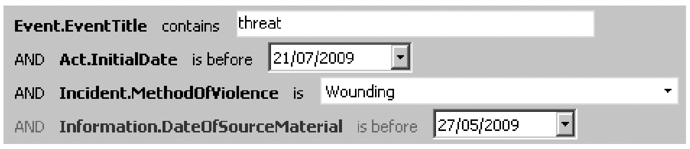
```
Event1
     Title: "threat against somebody"
     Acts: no acts

Event2
     Title: "a threat to the party leadership"
     Act1
```

```
TypeOfAct: Violations of the right to life
InitialDate: 01/05/2008
```

If ResultEntity = Event, the results will be fetched from the event collection and one will get both events. If ResultEntity = Act, one will get results from the act collection. Because Event1 doesn't have any acts, it will not be listed at all in the EventAct relationship table, thus only Event2 will be in search results. It should be stressed once again: a search condition is evaluated/performed against the ResultEntity database table. It means that a search condition's entity MUST be linked to the ResultEntity.

Let's examine a situation when the validity of a search query depends on ResultEntity, Figure 4.



	Event.EventTitle	contains	threat
AND	**Act.InitialDate**	is before	21/07/2009
AND	**Incident.MethodOfViolence**	is	Wounding
AND	**Information.DateOfSourceMaterial**	is before	27/05/2009

Figure 4. A simple search with incompatible search conditions.

If ResultEntity = Event, everything is ok.

If ResultEntity = Act or Incident, then we get into trouble.

Let's assume ResultEntity = Act. Because an Act is not linked by a parent/child relationship to an Information entity (see the rule in point *(b)* above), we can't uniquely identify an Information entity for an Act and vice versa. Technically speaking, the results we get from the fourth search condition do not contain an Act column, so it is not possible to join these results with results from other search conditions (we don't have a common entity by which to link two results sets).

3.2 Complex search

Imagine that we encapsulate a simple search into a block, we'll call it a *search block*. By applying logical operators (AND, OR, AND NOT, OR NOT) we can create a complex search query that consists of several blocks, see Figure 5.

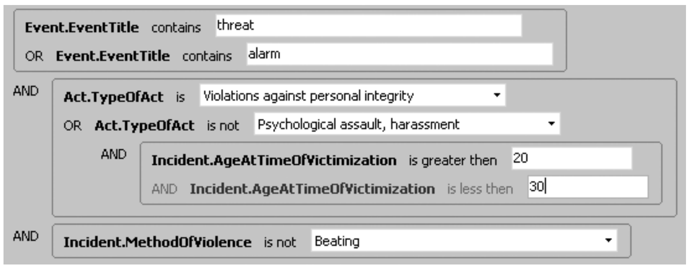

Event.EventTitle contains threat
OR Event.EventTitle contains alarm
AND Act.TypeOfAct is Violations against personal integrity
OR Act.TypeOfAct is not Psychological assault, harassment
AND Incident.AgeAtTimeOfVictimization is greater then 20
AND Incident.AgeAtTimeOfVictimization is less then 30
AND Incident.MethodOfViolence is not Beating

Figure 5. A complex search query.

Because each block has its own ResultEntity, we need to use the least common denominator Entity before joining the result sets. The system can automatically infer it, but as one can see in Figure 6, sometimes the user may need to set it manually.

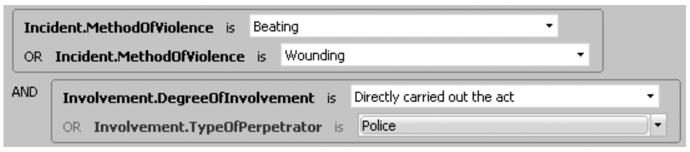

Incident.MethodOfViolence is Beating
OR Incident.MethodOfViolence is Wounding
AND Involvement.DegreeOfInvolvement is Directly carried out the act
OR Involvement.TypeOfPerpetrator is Police

Figure 6. Normalization of search blocks.

In Figure 6 the ResultEntity is Incident and Involvement correspondingly. The least common denominator is the Act entity, and the system will determine it automatically. But what if we need to display only the title of found events? Since the overall ResultEntity of these two blocks is Act, the search results will contain act entities. Surely, we can display the event's title, but the same event will appear many times because there are many acts per event. Let's run the search query on a database that contains just one event:

```
Event1
```

```
Act1
   Incident1
      MethodOfViolence = Beating
      Victim = PersonX
   Involvement1
      DegreeOfInvolvement = Planned the act
      TypeOfPerpetrator = Police
      Perpetrator = PersonY

Act2
   Incident2
      MethodOfViolence = Wounding
      Victim = PersonZ
   Involvement2
      DegreeOfInvolvement = Directly carried out the act
      TypeOfPerpetrator = Paramilitary forces
```

The first block returns `Incident1` and `Incident2`, the second block returns `Involvement1` and `Involvement2`. Joining two results sets will give us `Act1` and `Act2`. So, if we need to display only event attributes in search results, we'll get `Event1` listed twice. In order to remove duplicates, the user is required to manually select the `NormalizationEntity`. In our case, it will be the Event entity.

Remark: since NormalizationEntity applies to a block, it means it applies to a simple search. As a result, the definition of a simple search query will be expanded to include the (d) point: NormalizationEntity.

A query that doesn't have a NormalizationEntity is invalid. In Figure 7 the first search condition returns `Person` entities while the second one returns `Events`. These are top level entities that miss the parent/child relationship.

3.3 Person roles in search conditions

The following example illustrates how to search for acts that have victims from one of the specified countries.

The same applies to other entities:

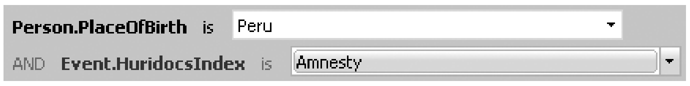

Person.PlaceOfBirth is Peru
AND Event.HuridocsIndex is Amnesty

Figure 7. Invalid search query.

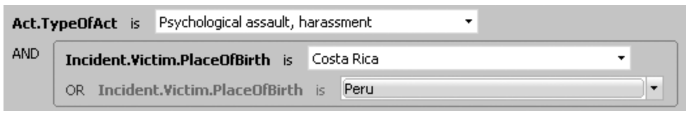

Act.TypeOfAct is Psychological assault, harassment
AND Incident.Victim.PlaceOfBirth is Costa Rica
OR Incident.Victim.PlaceOfBirth is Peru

Figure 8. Using person roles in search conditions.

```
- Intervention.InterveningPary.[PersonAttribute]
- Intervention.Victim.[PersonAttribute]
- Involvement.Perpetrator.[PersonAttribute]
- etc
```

3.4 Query Infeasibility

The proposed search mechanism is not able to perform all kinds of searches you might have in mind, but we've tried to find a trade off between ease of use and power. As a last resort, one can write a bunch of SQL queries and PHP scripts to perform a complex and exotic search query. Here is an example of an unfeasible query: *Show me those events that have more than 3 acts and where one of the victims is also a perpetrator.*

4 The Navigation Mechanism

Having a powerful search engine capable of querying different types of entities at the same time and returning fields of different entities in the same row, we are able to introduce the concept of Navigation Widget.

What if we start grouping the flat list comprised of found rows by several columns? The output will be a hierarchy, where each level is a ”folder” that is actually an entity attribute. Once a complex search query is created, the user can specify a list of entity attributes used to group the results and save the query and attribute list as a *hierarchical result container*, such container we call *Navigation Widget*. The user can create an unlimited number of widgets, and the system will automatically display them as top level folders. Figure 9 illustrates three widgets: ”By date”, ”By type of act”, ”[Event Widget]”. The second folder ”By type of act” is a two level navigation widget. Starting from the third level the system displays the event entities.

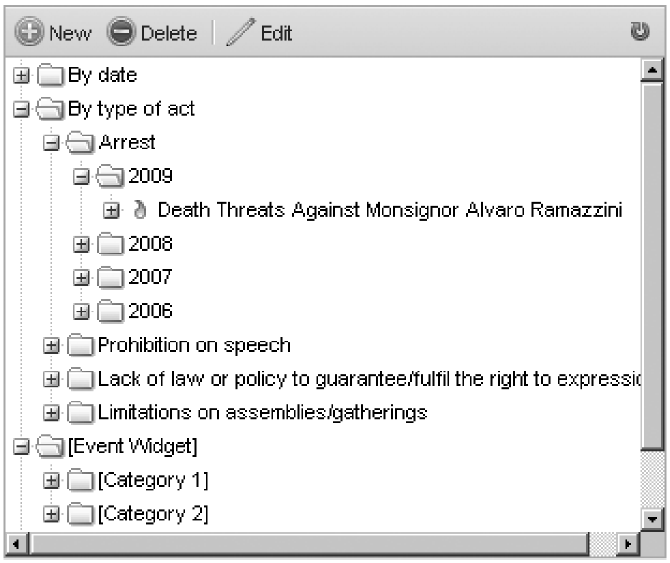
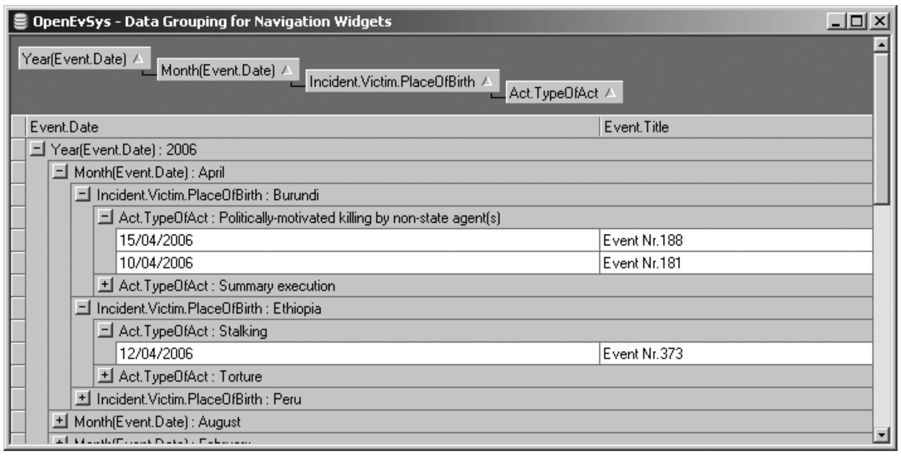

New
Delete
Edit
By date
By type of act
Arrest
2009
Death Threats Against Monsignor Alvaro Ramazzini
2008
2007
2006
Prohibition on speech
Lack of law or policy to guarantee/fulfil the right to expressi
Limitations on assemblies/gatherings
[Event Widget]
[Category 1]
[Category 2]

Figure 9. Navigation folders in event’s tree.

A small Win32 application was developed that displays a collection of events along with attributes from child entities (see Figure 10). The application is available for download from http://openevsys.burlaca.com/. The user can drag&drop columns on the header and group search results, and he can group by as many columns as he wishes. The described approach depicts the way navigation folders are created. The system may have two types of widgets:

- **global**: the administrator creates a widget that is automatically visible by all other users and they will not be able to modify/remove it;

- **local**: a user creates his own widgets that are not visible by others.


OpenEvSys - Data Grouping for Navigation Widgets
Year(Event.Date)
Month(Event.Date)
Incident.Victim.PlaceOfBirth
Act.TypeOfAct

Event.Date	Event.Title
Year(Event.Date) : 2006	
Month(Event.Date) : April	
Incident.Victim.PlaceOfBirth : Burundi	
Act.TypeOfAct : Politically-motivated killing by non-state agent(s)	
15/04/2006	Event Nr.188
10/04/2006	Event Nr.181
Act.TypeOfAct : Summary execution	
Incident.Victim.PlaceOfBirth : Ethiopia	
Act.TypeOfAct : Stalking	
12/04/2006	Event Nr.373
Act.TypeOfAct : Torture	
Incident.Victim.PlaceOfBirth : Peru	
Month(Event.Date) : August	

Figure 10. Grouping search results to create navigation folders.

5 Conclusions

Rapid Application Development tools became very flexible and powerful nowadays. Nevertheless, a developer has to rethink the User Interface when the underlying data model changes. The approach described in this paper implies considerable development efforts to implement the search engine, but the flexibility in managing an ever changing data model greatly reduces maintenance costs.

The approach is not universal in a sense that it can't be applied to an arbitrary data model, i.e. the search engine implementation may change.

For future work there are two directions: a) expand search engine capabilities to process different types of data models b) apply other paradigms in navigation mechanisms, e.g., the pivot tables concept as a complement to the hierarchical grouping.

References

[1] O. Burlaca. *Generic Interfaces for Managing Web Data.* Computer Science Journal of Moldova, vol. 13, nr. 1(37), 2005. pp. 70–83

[2] HURIDOCS - Human Rights Information and Documentation Systems, International. http://www.huridocs.org/

[3] OpenEvSys2 prototype. http://openevsys.burlaca.com/

Oleg Burlaca, Received June 16, 2010

O. Burlaca
Institute of Mathematics and Computer Science,
Academy of Sciences of Moldova
Academiei 5, Chişinău MD-2028 Moldova
E–mail: *oburlaca@neonet.md*