TICE-Healthy
A Dynamic Extensible Personal Health Record

Pedro Catré[2], Alcides Marques[1], João Quintas[1] and Jorge Dias[2]
[1]*Instituto Pedro Nunes, Coimbra, Portugal*
[2]*University of Coimbra, Coimbra, Portugal*

Keywords: Personal Health Record, Informal Healthcare, e-Health, Databases.

Abstract: This paper presents a system that enables the configuration of a Personal Health Record supporting informal care services in a way that is extremely flexible and simple and allows the development of dynamic extensions even in a production environment. With this system powering the Personal Health Record it is possible to modify the repository's data structure, clinical data viewer and business logic, adapting the system to new requirements in a fast and easy manner without the need for redeploy.

1 INTRODUCTION

TICE-Healthy is a research and development project supported by a consortium of 24 companies and R&D institutions that aims to design and create innovative e-health products. The first line of action of this initiative is to create an information and interaction channel for selling products and health services, which will provide a Personal Health Record (PHR) – a repository of clinical information of an individual whose maintenance and updating can be performed by himself or by his caregivers.

This PHR will have to satisfy the needs of several service providers with evolving requirements. Even though typical clinical data is very standardized, this repository will also store data for informal care services which are not uniform. Moreover, there is a great overhead associated with the change in requirements which usually force not only revisions to the data access and business layers, but also to the interface. The code produced in these types of scenarios is repetitive; its development is slow and potentially subject to many mistakes and changes would not be directly applied in a production environment as would be desirable.

To satisfy the requirements of the platform's healthcare management system we implemented a module called TICE-GenericEntity that administers entities and generates components to visualize and manipulate entity records. Additionally, it allows the creation from scratch and dynamic modification of entire web applications for data visualization and manipulation, with menus containing web components, and it generates a REpresentational State Transfer (REST) Create, Read, Update and Delete (CRUD) Application Programming Interface (API) for the clinical entities that are defined.

This paper presents a system that enables the configuration of a PHR supporting informal care services in a way that is flexible and simple and allows the development of extensions even in a production environment.

2 ARCHITECTURE

Any system seeking the extensibility described needs mechanisms to save the settings of the clinical entities that are created, which can include: the mapping of entity attributes to database columns; the properties of the attributes; the different web views for inserting, updating, showing and listing entity records; and the events and business rules to validate submitted records and perform operations when specific conditions are met. The decision was to store all the entity configurations in a field of a database table using the JavaScript Object Notation format. This way it is possible to apply automatic serialization, it is easy to manage versions of the configurations as they evolve and allows good performance given that a single database access is required to acquire all the metadata pertaining to an entity. Custom server-side rules are also stored in a field of the database and compiled at runtime using

Catré P., Marques A., Quintas J. and Dias J..
TICE-Healthy - A Dynamic Extensible Personal Health Record.
DOI: 10.5220/0004244903480351
In *Proceedings of the International Conference on Health Informatics* (HEALTHINF-2013), pages 348-351
ISBN: 978-989-8565-37-2
Copyright © 2013 SCITEPRESS (Science and Technology Publications, Lda.)

JBoss Drools business rules engine. The result of the compilation is stored and it is only recompiled when changes to the rules occur.

Figure 1 presents the logical organization of the system.

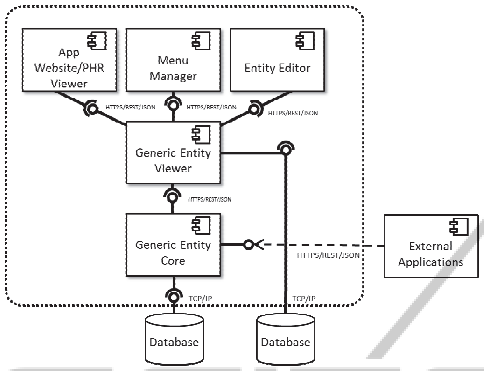


Figure 1: Overview of the system's main components.

The figure identifies 5 major components:

- Core Module – is responsible for the storage and provision of the entities' configuration and records to the viewer and external applications;
- Viewer Module – interprets the configuration and creates the views;
- The Menu and Entity Manager – is a back-office where system administrators can manage entities and define which menus make up the PHR's website, what components (for example: forms, list of records, charts of the data) appear in each menu, how are the components related (for example editing a record can cause an automatic update in a list of records or a chart, or even trigger a health alert message);
- Entity Editor – allows the creation of new entities and editing of existing entities in a visual manner;
- App Website (in this case the PHR Viewer) – is a rendering of the configurations that were previously described that uses client-side routing and templating together with Asynchronous JavaScript and XML (AJAX) requests to provide a comfortable and rich user experience.

Note that the Menu Manager, Entity Editor and PHR Viewer are client-side applications created using the JavaScript framework Backbone. The Generic Entity Core and Viewer are server side applications created using the Java web framework Play. The databases are implemented in PostgreSQL. Together, the 5 components presented allow a database schema to be mapped to a user interface.

3 DATA STORAGE METHODS

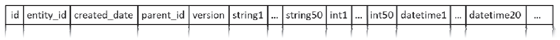
In this section we will analyse four methods for storing data in the repository and present the strategy implemented by our system. The aim was to have a flexible repository that handles new data types and attributes without the need to change the physical database schema, while still supporting efficient and easy to construct ad-hoc queries to the data.

A classic approach is to use a text format like Extensible Markup Language (XML) or JSON to store the keys (entity's attribute) and corresponding values (Xie et al., 2010). This method would require the database management system to support Structured Query Language (SQL) over XML fields in order to apply ad-hoc queries, which is an unusual and costly operation.

Another approach frequently employed in clinical scenarios and other cases where the number of attributes, properties or parameters that can be used to identify an entity is potentially unlimited, is to use an Entity-Attribute-Value (EAV) model (Gilchrist et al., 2011). An EAV design usually involves a central table with three columns that contain data referring to: the entity, the attribute and the value of that attribute. While this approach allows ad-hoc queries, they are not straightforward and would typically require a resource consuming transform/pivot operation.

Alternatively the system can use a flexible schema strategy involving the dynamic creation of a table with specific fields for each new entity that needs to be defined. In this case the majority of changes to entities would lead to Alter Table operations which are usually blocking and can be slow if the table is storing a lot of data.

Another solution we call Generic Table involves the use of a sufficiently generic table with pre-created columns of various types. For example a table may contain 80 text fields, 20 integer fields and so on, and there will be a mapping for each entity specifying which columns it uses for its attributes. An illustration of this approach is presented in Figure 2.



id	entity_id	created_date	parent_id	version	string1	...	string50	int1	...	int50	datetime1	...	datetime20	...

Figure 2: Example of a generic table.

Using this technique all records corresponding to each entity may be stored in the same table and we avoid the runtime alteration of the data structure. Microsoft SharePoint platform uses a similar approach (Krause et al., 2010). The main disadvantages of this solution is that it will result in sparsely populated columns with many null values, something that is not elegant and usually leads to a waste of space, and ad-hoc queries become less intuitive since the names of the database columns will not correspond to the names of the entity attributes.

TICE-GenericEntity supports the last two strategies. By default it uses a Generic Table for storage but the developer can also choose to create and configure custom tables. Therefore, it is possible to use our solution on top of a regular database schema by adding a few specific configuration columns to each table used by an entity.

4 PERFORMANCE TESTS

Since the solution presented in this paper is highly configurable and flexible it was expected that it would display a significantly lower performance than static solutions. Therefore, it was important to analyze whether the overhead of this solution would have a manageable impact on the overall context of a web application.

The goal of these tests was to compare the performance of the solution developed against a typical Object Relational Mapping (ORM) implementation using the same web framework. For that purpose we created two applications with the Java web framework Play: one that used this framework's ORM mechanisms and another that used our solution (TICE-GenericEntity).

The tool used for the tests was Apache JMeter and the tests were executed with the following configurations:

- With and without the use of cache mechanisms;
- With an empty database and a database populated with 100.000 records created at random, but equal for both applications being compared;
- In TICE-GenericEntity we used the system's generic table with 138 pre-created columns (30 strings, 14 booleans, 12 texts, 13 dates, 13 floats, 13 longs, 14 integers, 13 numbers, 12 reals and 4 times);
- With entity A composed of 5 attributes (3 strings, 1 integer and 1 text) and entity B with 15 attributes (8 strings, 2 integers, 1 text, 1 date, 1 float, 1 long and 1 time);
- Each test result is the average of 500 test runs.

In these tests we chose not to use concurrent requests (multiple users) since the first tests showed that this caused considerable deviations between identical runs and it did not help in the comparison of approaches.

4.1 Record Insertion and Retrieval

Figure 3 presents the average time needed for the insertion of 50 records in the two applications being compared, for both entities A and B. Since the test is executed 500 times and each test inserts 50 records, the database becomes increasingly populated. These tests are executed in an originally empty database and on a database that is initially populated with 100.000 records. We also performed the tests caching the requests to the database for entity configurations.

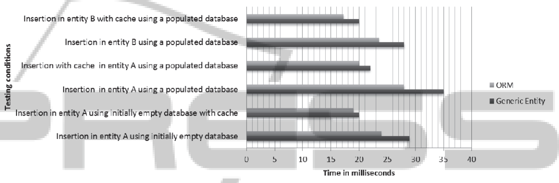


Figure 3: Comparison of the average time needed to insert 50 records.

The results obtained were surprising and better than anticipated. It was expected that our approach would be significantly slower since it had to iteratively build the object to insert in the database verifying each parameter received in the Hypertext Transfer Protocol request against the attributes defined in the configuration and also create the insertion statement using a dynamic configuration. According to the results these extra operations exhibited very low overhead.

The tests using the extended entity B showed a negligible impact. Therefore, the solution does not appear to show limitations related to the number of attributes defined in the entity.

Figure 4 presents the average time needed along with the standard deviations for the retrieval of 50 records from the database.

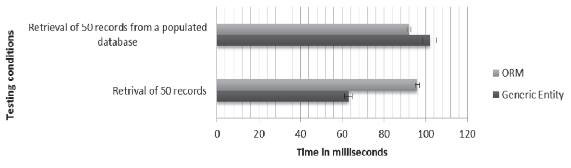


Figure 4: Comparison of the average time needed to obtain 50 records.

Both applications show a low variance in their response times. The results also indicate that TICE-GenericEntity is slower running the tests on a populated database. The difference is less than 10 milliseconds and it is due to the fact that the ORM

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

based application does not have to retrieve as many attributes (for example it does not have the timestamp of creation or last update of the object) and it does not order the results before returning them, while by default TICE-GenericEntity orders the results by timestamp of creation since it always returns them paginated because of salability requirements.

4.2 Form Generation

The purpose of these tests is to compare the average time needed by TICE-GenericEntity to generate and return an insertion form against the time needed for a normal application to return the same form already created. In other words, our solution will interpret the configuration of an insertion view in order to generate it while the ORM application will simply return the same form (pre-created) as static content. Figure 5 shows the results of the tests performed.

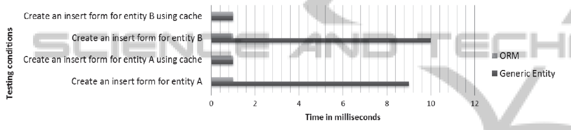


Figure 5: Comparison of the average time it takes to return an insertion form.

In these tests the creation of the form is roughly 10 times slower than presenting a static form. Once again this result was surprisingly positive given the nature of the comparison. In fact, it was clear that the solution that had to dynamically generate the form would be slower and it is perfectly acceptable for it to be 10 times slower since there are mechanisms to mitigate this difference. Using cache in both applications the results were, naturally, identical.

5 DISCUSSION & CONCLUSIONS

As shown, despite its dynamism and flexibility our solution does not display a great overhead when compared with a traditional approach. Naturally, the generation of views is slower than the simple return of static content, but it is a relatively insignificant overhead when taking into account factors such as network latency. It is also import to state that the applications comprising our solution are stateless so it is simple to apply a load balancing pattern to scale the system. Moreover, the tests revealed that using cache mechanisms to mitigate the extra resource usage is extremely effective.

In summary this paper presents a module that enables the configuration of a PHR supporting informal care services in a way that presents major benefits for the entire TICE-Healthy initiative. With this component powering the PHR it is possible to modify the repository's data structure without redeploy and it is easy to make changes to the data viewer and business logic with minimal coding.

In the future we will explore non-relational alternatives for storing data that may prove to be more flexible and perform better. Specifically, migrating the system to such a data store might free us from the limitations of having pre-created columns. However, several concerns will have to be addressed first, such as the security mechanisms provided, the ability to execute ad-hoc queries and the support for transactions.

ACKNOWLEDGEMENTS

The TICE.Healthy project is co-financed by the European Community Fund through COMPETE - Programa Operacional Factores de Competitividade.

REFERENCES

Gilchrist, J., Frize, M., Ennett, C. M. & Bariciak, E., 2011. "Performance Evaluation of Various Storage Formats for Clinical Data Repositories," *IEEE Transactions on Instrumentation and Measurement*, vol. 60, no. 10, pp. 3244–3252, viewed 5 March, 2012, <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5738684>.

Krause, J., Langhirt, C., Sterff, A., Pehlke, B. & Doring, M., 2010. *SharePoint 2010 as a Development Platform*, Apress; 1 edition, p. 800, viewed 24 November, 2011, <http://www.amazon.com/SharePoint-Development-Platform-Experts-Sharepoint/dp/1430227060/ref=cm_cr_pr_product_top>.

Xie, L., Yu, C., Liu, L. & Yao, Z., 2010. "XML-based Personal Health Record system," *2010 3rd International Conference on Biomedical Engineering and Informatics*, IEEE, pp. 2536–2540, viewed 24 October, 2011, <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5639706>.