Visualizing Data in Software Cities

Susanna Ardigò, Csaba Nagy, Roberto Minelli, Michele Lanza

REVEAL @ Software Institute — USI, Lugano, Switzerland

Abstract—The city metaphor for visualizing software systems in 3D has been widely explored and has led to many diverse implementations and approaches. However, when looking at software systems in general and when using specifically a city approach, it is evident that something is missing: The data. Indeed, software systems are intrinsically driven by data, which is usually managed using databases or often also simply stored in files coming in a variety of formats, such as CSV, XML, and JSON. While such data files are part of a project’s file system and can thus be easily retrieved, the situation is different for databases: A database is usually not contained in the file system, and its presence can only be inferred from the source code which contains the database accesses.

We present an extension of the CodeCity implementation, M³TRICITY², with two new contributions: First, we consider data files and use simple metrics to integrate them in the visualization seamlessly. Second, we present a novel way to add a database to the visualization by making use of the one remaining space left unused: the sky and the underground. We present our contributions and illustrate them on various software systems.

Index Terms—Software visualization, Database access visualization, Software maintenance, CodeCity

I. INTRODUCTION

The city metaphor has been widely explored to visualize the structure and the evolution of software systems [1]–[9]. When looking at a software city, while we see its structure and implementation, something is missing: The data. But Software systems run on data. It intrinsically drives them, giving them a “reason” to exist. If data is an integral part of software systems, then it should be part of the city visualization.

Classic software visualization approaches consider systems as files located in nested folders, above all because this is the way their source code is stored and versioned. Data on the other hand can originate from various sources. The most straightforward approach is to manage data using data files, which often come in various formats, such as XML and JSON. Such files are usually versioned together with the source code. A more systematic and scalable approach is for a system to make use of databases. Relational databases (e.g., PostgreSQL, MySQL, Oracle) have been widely used for decades, while recently there has been an uptick in the use of NoSQL databases (e.g., Neo4j, MongoDB). Databases are generally not contained in a system’s repository, and as a result they have so far remained largely outside the focus of visualization approaches. Some researchers have developed approaches to visualize databases together with software systems [10]–[13], but no state-of-the-art method considers the system and its various data sources as a whole, presented in one view where they evolve together and interact with each other.

We present M³TRICITY², an extension of M³TRICITY [14], which seamlessly integrates various data sources of a system into the visualization. First, we represent data files in the city and map simple metrics on their meshes (i.e., their shapes rendered in the visualization). Second, we add the database to the visualization using the free space of the sky above or the underground below the city. We infer the schema of an application’s database using SQLInspect [15], a static analyzer to inspect database usage in Java applications. We calculate metrics of the database entities, such as the number of columns of tables or the number of classes accessing them. Finally, we present the system with its data sources using the history-resistant layout of M³TRICITY, i.e., the new entities remain at their reserved place throughout the evolution. The resulting view seamlessly integrates data sources to a software city and enables a more comprehensive understanding of a system in terms of its source code and its data.

M³TRICITY² is a web application available at https://metricity.si.usi.ch/v2.

II. RELATED WORK

Since the seminal works of Reiss [1] and Young & Munro [2], many studied 3D approaches to visualize software systems. The software as cities metaphor has been widely explored and led to diverse implementations, such as the Software World approach by Knight et al. [3], the visualization of communicating architectures by Panas et al. [4], [16], Verso by Langelier et al. [5], CodeCity by Wettel et al. [6], [17], EVO-STREETS by Steinbrückner & Lewerentz [7], CodeMetropolis by Balogh & Beszedes [8], and VR City by Vincur et al. [9].

Only a few approaches considered presenting the databases together with the source code, and interestingly, most use the city metaphor. Meuirce and Cleve presented DAHLIA to visualize database schema evolution [10], which uses the city metaphor where buildings in the city represent database tables. DAHLIA 2.0 presented the city of the source code and the database side-by-side [11]. Zirkelbach and Hasselbring presented RACCOON [12], a visualization approach of database behavior, which uses the 3D city metaphor to show the structure of a database based on the concepts of entity-relationship diagrams. Marinescu presented for enterprise systems a meta-model containing object-oriented entities, relational entities and object-relational interactions [13], stating that “to perform reverse engineering on enterprise software systems we need a specific meta-model which contains [...] entities from the object-oriented part and [...] entities regarding the relational part [...] and the interactions between the two paradigms.”
III. VISUALIZING DATA IN SOFTWARE CITIES

The goal of M3TRICITY2 is to represent a software system and its data sources (i.e., data files and databases), while differentiating the two parts in the visualization. M3TRICITY2 builds on M3TRICITY [14], to which we refer for details on the history-resistant layout.

Figure 1 shows a screenshot of M3TRICITY2 visualized in M3TRICITY2. The software city visualization is in the center with the database as a cloud above the city. Information panels present the name of the repository (top left), the metrics of the system (top right), the actual commit (bottom left), and its commit message (bottom right). The timeline at the bottom depicts the evolution of the project where one can spot significant changes in the metrics. The evolution can be controlled with the buttons below the city.

We illustrate the model of M3TRICITY2 and discuss how to analyze and visualize the data in software cities.

A. Evolution Model with Data Entities

M3TRICITY2 extends the evolution model of M3TRICITY with entities representing data sources. The underlying model is depicted in Figure 2. The purpose of M3TRICITY was to keep the visualization resistant to changes throughout the evolution. Accordingly, the model keeps track of the histories of each version of the entities.

In M3TRICITY a Repository is composed of Packages and Classes. M3TRICITY2 adds Databases and Tables to the model. Additionally, TableAccess association classes represent when Classes access Tables to query or modify data in the database. We also add an entity for DataFiles, i.e., files in the repository to store data (e.g., XML or JSON files). Finally, we add a Binary entity to distinguish other non-source files in the repository (e.g., images). In Figure 2, a database icon highlights the new parts of the model.
B. Analysis of Data in Software Cities

The top part of Figure 3 summarizes the process in place when the user asks for the analysis of a system.

C. Visualization of Data in Software Cities

The bottom of Figure 3 depicts the visualization process. The Front End sends the request to the Back End, which returns the meshes with their locations for the current snapshot. The main components of M3TRICITY2 in this process are the Layout Processor and the City Creator in the Back End, and Canvas Creator and Mesh Renderer in the Front End.

M3TRICITY2 defines new meshes for the new entities (see Figure 4). An orange datafile-cylinder represents a data file. A binary file has a gray binary-hemisphere mesh, and a database table is drawn as a red table-cylinder.
In the City with Clouds visualization the file system is positioned at the ground level, and the database is positioned above the city so that the two are facing each other (see Figure 5). In the City with Underground, the city is lifted, giving space to the databases and tables represented below (see Figure 6). In both layouts, edges connect the accessed tables with the classes. To emphasize the inferred nature of the tables, the database is rendered with light transparency, giving it a ghostly appearance above/below the city.

IV. CASE STUDIES

A. SwissCovid Android App

Figure 7 presents the evolution of the official SwissCovid Android App. Its Java back end does not use a local database but has many resource files. It has around 8k and 4k lines of code written in Java and Kotlin.

The repository was initialized with a commit of license and readme files on April 15, 2020. In the third revision, on May 4, 2020, they committed the ch.admin.bag.dp3t package. The structure remains stable after this commit, but the project is developed with regular changes, as seen in the commits of July 30 and April 27, 2021.

Its growth is easily observable during the evolution. The timeline in Figure 7 shows regular contributions to data files. Indeed, the XML files grow from an initial 10k to 25k lines.

Interesting districts can be spotted in the new layout with various languages. The biggest table, Accounts, has 14 columns and is modified 31 times throughout the evolution. It was initially added to the project with only four columns. The DatabaseHelper class handles the communication with the database.

Above the district of the source files, we can see the neighborhoods of resource files. There are several tiny PNG and SVG files and folders with smaller layout XMLs. An interesting district is a folder with strings.xml files of various languages. The initial version supported only three official Swiss languages (Italian, French and German). As the app evolves, the XMLs grow, and the number of supported languages increases to twelve.

B. GnuCash Android App

GnuCash Android is a companion app of the open-source GnuCash accounting software. It has an SQLite database with a central role in the project to store the transactions. Its codebase is in Java with around 27k LOC. In addition, it has XML files with about 22k lines.

Figure 8 depicts the evolution of GnuCash Android. The project started with an “Initial commit” by “codinguser” on May 13, 2012. The commit already contained source files, but no database yet, which was added two commits later, in the commit “Created database store for accounts and transactions” on May 24, 2012. The visualization confirms the appearance of two tables: Transactions and Accounts.

On April 11, 2015, a refactoring took place in the project and they also refactored the database schema (removed deprecated tables and code). This revision has four tables in the database, and they are accessed only from a few classes. The biggest table, Accounts, has 14 columns and is modified 31 times throughout the evolution. It was initially added to the project with only four columns. The DatabaseHelper class handles the communication with the database.

The other classes accessing the database are adapter classes: AccountsDbAdapter and TransactionsDbAdapter. M3TRICITY2 shows 11 tables throughout the evolution of the project, with some tables introduced only temporarily.
The last commit of the project is from December 2, 2020.\textsuperscript{11} Districts of resources similar to the SwissCovid App can be spotted easily. The larger XML files are again the \texttt{strings.xml} files of the 35 languages of GnuCash Android \textsection. It is also interesting to observe some smaller and wider XML files in one district \textsection. These are Import*.xml files in the \texttt{test} package: They store test data.

V. CONCLUSIONS & FUTURE WORK

\textsc{M3TRICITY2} extends the city metaphor with a novel way to add “data” to the visualization, making it an integral part of the visualization. We implemented two views to separate the database as an inferred element: \textit{City with Clouds} and \textit{City with Underground}, and connected the data tables to the classes accessing them. We added data files and used descriptive shapes and colors to differentiate them from the buildings of source classes. We demonstrated the approach on two systems, one with small data files and one with a relational database.

There are limitations to the approach: It requires the database schema, which is rarely accessible with the source code. In case it is missing, schema inference can help. Moreover, schema-less NoSQL databases that allow unstructured code. In case it is missing, schema inference can help. More dbase schema, which is rarely accessible with the source classes. We demonstrated the approach on two systems, one with small data files and one with a relational database.

There are limitations to the approach: It requires the database schema, which is rarely accessible with the source code. In case it is missing, schema inference can help. Moreover, schema-less NoSQL databases that allow unstructured code. In case it is missing, schema inference can help. More dbase schema, which is rarely accessible with the source classes. We demonstrated the approach on two systems, one with small data files and one with a relational database.

Finally, as we demonstrated the approach on a system with only one database, multiple databases pose additional challenges. While conceptually the model can handle them, questions arise about their positioning. Seeking answers to these questions is part of our future work.

ACKNOWLEDGMENTS

We gratefully acknowledge the financial support of the Swiss National Science Foundation (SNF) and the Fonds de la Recherche Scientifique (F.R.S.-FNRS) for the project “INSTINCT” (SNF Project No. 190113).\textsuperscript{11}

\textsuperscript{11}See https://github.com/codinguser/gnucash-android/commit/2ad44ad

REFERENCES