Abstract. Modelling is considered as a universal approach to define and simplify real-world applications through appropriate abstraction. Model-driven system engineering identifies and integrates appropriate concepts, techniques, and tools which provide important artefacts for interdisciplinary activities. In this paper, we show how we used a model-driven approach to design and improve a Digital Humanities dynamic web application within an interdisciplinary project that enables history students and volunteers of history associations to transcribe a large corpus of image-based data from the General Register Office (GRO) records. Our model-driven approach generates the software application from data, workflow and GUI abstract models, ready for deployment.

Keywords: Software and System Engineering; Model-Driven Development; Web Application; Historical Civil Record; Digital Humanities; XMDD; DIME

For citation: Khan R., Schieweck A., Breathnach C., Margaria T. Historical Civil Registration Record Transcription Using an eXtreme Model Driven Approach. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 3, 2021, pp. 123-142. DOI: 10.15514/ISPRAS-2021-33(3)-10

Acknowledgments. We are grateful for the full cooperation of the Registrar General of Ireland for permission to use these data for research purposes. This research is funded by the Irish Research Council Laureate Award 2017/32 and by Science Foundation Ireland through the grants 13/RC/2094 to Lero - the Irish Software Research Centre (www.lero.ie).

"Death and Burial Data: Ireland 1864-1922" is a project funded by Irish Research Council Laureate Award IRCla/2017/32 to Dr. Ciara Breathnach (Department of History – DH), in cooperation with Prof. Tiziana Margaria (Software Systems, Dept of Computer Science and Information Systems – CSIS) at the University of Limerick.
the Industry 4.0 context, it often leverages technologies such as the Internet of Things (IoT), Artificial Intelligence (AI), Cyber-Physical Systems (CPS) and Big Data for digitization. By definition, digital twins refer to a ‘live’ model that continuously updates and changes as its physical counterpart changes [3]. In the Humanities, the DT concept unfold a massive potential to transform the landscape of how DH methods can assist in the representation, analysis and understanding of our past, which in turn can provide useful learnings for the present and future. It promises a tremendous innovation potential, and most of the current research on digital twins is focusing on specific implementations for concrete use cases and the generalization towards reusable abstract models [4]. Developing a mirror of a traditional Digital Humanities record system through the digital twin lens is time-consuming, complicated and requires deep interdisciplinary knowledge in the humanities domain and model creation and software development. Too often, this induces a knowledge gap, giving rise to fundamental research questions on how to connect the two disciplines in such a way that a ‘lingua franca’ bridge the concepts and the means of expression and analysis of both disciplines.

We use a specific kind of Model-Driven Design, called XMDD (eXtreme Model-Driven Design) [5] to bridge this gap. Model-Driven Development (MDD) specifically focuses on supporting the collaborative (software) development process by using abstract representations of data and processes. Using these models, we combine computing knowledge with the formal descriptions of the historian’s knowledge, and this way succeed in reducing complexity and improve productivity, as described by [6].

To reduce the discipline-specific knowledge gap between humanities and technology, the project DBDIrl [6] - Death and Burial Data: Ireland 1864-1922 [7] adopts a data-driven public-history and digital-humanities research methodology which uses advanced MDD for application development. DBDIrl is an interdisciplinary project that combines historians’ understanding of Big old data with computer analysts’ tools and methodologies. Its objective is to build an extensible and reusable Big Data interoperability and analysis framework that supports flexible Big Data integration between different historical data sources and provides a web-based platform for the analysis of its underlying corpus. The corpora stem from various sources of national records, like the civil registration records of the General Register Office, the individual level census returns of 1901 and 1911, and various coroner’s court records within the period 1864 to 1922, i.e., from the introduction of civil registration records in 1864 to 1922, when the Irish Free State was established. This Digital Humanities platform needs to be robust and easily executable, able to integrate different data and interpreted terms, able to manage and analyze various data representations and enrichments, all in a transparent and FAIR (i.e., Findability, Accessibility, Interoperability, and Reuse of digital assets) [8] data context. This paper focuses on developing an efficient and flexible data access mechanism to make the heterogeneous sources of historical data available to a wider range of researchers through adequate user interfaces.

DBDIrl applies the eXtreme Model-Driven Approach for complete design, development and execution of a Big Data interoperability framework. The first component of that framework is a Web application that supports efficient and correct data entry. We refer to it as the Historian DIME app or Historian app in short, and it is completely developed following a model-driven approach. Key contributions of this work are:

- A model-driven Web application for input and storage of Irish Civil Registration data, specifically death registration data, from 1864 to 1922, introducing a database for subsequent digital data analysis.
- Producing a systematic and clean data source for (relevant subsets of) the death records. Massive information regarding the death records was previously collected as images of the original registers stored as TIFF files, plus an excel index summary. The page-by-page images of the handwritten records were digital, but it was impossible to analyze them. The database of systematic and clean data can now be processed for further research, concerning the discovery of information, its evolution, trends over time, and finding insightful patterns about individuals and families.

- Illustrating the impact of the MDD approach on the adaptation and evolution of the Historian App, from its first version to the current one, including the re-usability of components and the refinement of its organization, to support increasing levels of error prevention and embedded error checking. It is essential if we want to gradually build a platform, where such applications and data analytics applications can be quickly and correctly assembled from a service-oriented Domain Specific Language that covers the functionalities and the data occurring in a history research context.
- Showcasing the use of DIME [9], a specific low-code application design framework, where stakeholders can develop their specific application without any coding knowledge.

To our knowledge, this is the first attempt to work with Historians as customers using a model-driven approach.

The paper is organized as follows: Section 2 presents the project background and motivation. Section 3 describes related work in the fields of MDD and Big data analysis. Section 4 discusses the co-development methodology and its life-cycle along with an explanation of the abstract architecture and workflow of the proposed XMDD based application. Section 5 illustrates the model types of the XMDD technology and the concrete design of the Historian App. Section 6 describes some major challenges with corresponding proposed solutions. Finally, Section 7 concludes the paper and highlights some future work.

2. DBDIrl Project Background

The General Register Office (GRO) is responsible for recording Irish civil information of birth, marriage and death. In 2016 it placed historical data online for free on irishgenealogy.ie. To initiate a search at the site, some basic personal information is required, but it has limited functionalities. This site holds civil data sets regarding individuals, but for the fundamental objectives of DBDIrl, a centralized data storage containing complete and correct data is needed for future research and exploration.
As the primary data, DBDIrl uses the Death Registration Data (DRD) from 1864 to 1922 directly shared by from the GRO. We received approximately 4.3 million individual Civil Register records of a manual, but highly assisted and supported data enrichment through a web based application. In this sense, the first and second task now align much more closely: We have now an XMDD based web application for the historians' data entry, where application developers and historians work side by side in application design and development within a model-driven, low-code environment. This application development approach helps the historians to further develop and maintain their own application at the model level, without the need of any programming knowledge.

3. Related Work

Since the emergence of UML and its predecessors, several MDD approaches have been proposed in the literature to address the generation of code from models representing various aspects of the system [10], including for telecommunications [11], web and client applications [12-15]. MDD techniques are mainly used for decreasing the effort needed for application development and maintenance and increase the portability to new platforms. The eXtreme Model-Driven Development (XMDD) [5] approach is a low-code approach that combines several software designs and programming paradigms such as agility, model-driven development, service orientation, domain-specific languages, data management, data flow and control flow design, Formal models and methods, generative programming, eXtreme programming, aspect orientation and full code generation [5], [16]. According to [17], «Models allow sharing a common vision and knowledge among technical and non-technical stakeholders, facilitating and promoting the communication among them.» In terms of specific MDD approaches and applications, [18] proposed automated extraction, analysis, and visualization of data and metrics on model-driven artifacts. In cyber-physical systems, [19], and [20] demonstrate the use of MDD in robotics. [21] proposed a DSL for service customization for telecommunications systems. [22] proposed a Domain-Specific Modelling Language for smart home applications with two transformation templates that generate code from instances of SmartHomeML for SmartThings and Alexa. They designed the transformation using an MDD approach in a platform-specific model-to-code implementation artefact.

In e-learning, [23] propose a course management system that stores a course model as machine-readable components that generates a final course in different platform-specific target models. In web applications, modern Single-Page Applications (SPA) use MDE to connect between client and server of a web application, and [24] present a model-driven approach for the consumption of RESTful Web services in SPA.

Ref [25] defines a Machine Learning based MDE approach that analyzes Big Data for probabilistic modelling by defining a domain-specific modelling language. In Big Data, [26] introduced SkyViz, a model-driven approach for automating the translation of user objectives to visualize the Big Data Analytics’ results into a set of most suitable and concrete visualizations. [27] proposed a design method to specify, deploy, and monitor Big Data Analytics solutions using MDD. While all this shows that MDD is applied in a variety of relevant areas for the DBDIrl project, as per our study there are no MDD based context-aware web applications that work with real-world big data archiving, management and analysis.

4. The Historian App as a MDD Application

The Historian App we developed and evolved in a number of iterations is the DBDIrl solution to data entry, storage and management for the historical civil registration (i.e., death) data of Ireland from 1864 to 1922. We adopt the eXtreme Model-Driven Development (XMDD) [28], which provides a fast turnaround of easily modifiable prototypes understandable to the non-IT experts. In
this way, a more collaborative approach between domain experts (here the historians as central stakeholders) and developers establishes itself along the entire project life cycle. The agile model-based approach helps repeat the feedback and co-design cycles with the historians in a continuous refinement process. In addition, using models also helped the developer team when reflecting, presenting and explaining the work progress to the historians and the historians when understanding and monitoring the development.

We chose the DIME Integrated Modelling Environment [9, 29], based on Domain-Specific Libraries (DSLs), as the XMDD framework for our project. DIME provides reusable features, and functionalities [20, 30] where developers can develop web applications within a low-code environment without having any programming knowledge. DIME supports model types for processes, services, data, and the UI that are integrated and kept consistent to a reasonable extent by the platform. Many domain-specific libraries (DSLs) are already available, for example, for the GUI design of the web applications. New services as well as entire new DSLs can be introduced in an easy way. These characteristics help the IT specialists and the domain specialists to better understand and monitor the development throughout the project life cycle on the basis of the domain knowledge.

4.1 The IME-based co-development lifecycle

The application development life cycle of the Historian App is illustrated in fig. 2.

The project development life cycle involves in each phase both the computer scientists and historians, in different roles. The historians become successively more skilled in dealing with the models and application design. At project completion the historians may be able to modify and evolve, or even design and implement, their own web applications on the basis of the existing DSLs, without any coding knowledge.

4.1.1 Phase 1 – Application Modelling

As illustrated in fig. 2, the project life cycle starts by collecting and collaboratively analyzing the historians' requirements. They are materialized as abstract workflow models with the corresponding (unique and coherent) data model. Data and processes go hand in hand in DIME, so they are typically co-developed and co-evolved in an XMDD approach. In this phase, the historians used their expert knowledge about handling historical data and the correctness of the data and the records. The historians collected the data and analysed their characteristics. While historians were finalizing the data properties that they need for their further analysis, the computer scientists started designing the data models including entities, attributes and relations. Then the historians specified how they want the data to be stored, explaining what is already there and usable, what else needs to be added, and how. Next, the CS team designed the corresponding data and workflow models, expressing the high-level application logic and the elementary operations required for application development. Gathering this expert knowledge in terms of workflows and properties or conditions (on the individual data item, the record, the workflows) corresponds to gathering the historian data entry application's static and behavioral requirements. At this phase, the historians also validated the models and helped in finalizing them.

4.1.2 Phase 2 – Model Completion and Compilation

The second phase includes all the XMDD: model refinement, followed by DSL extension and implementation of new functionalities. Here the historians participated as stakeholders for detailed questions, the CS team as fine granular designers and developers. The CS team extended the DSLs where functionalities were missing, implemented them in a reusable, service-oriented way and modelled the Web application GUI. A growing hierarchy of nested workflows structure the...
The application logic in behavioural features. For the business logic they acted as application configurators on the basis of these models and services. We reuse existing DIME process models such as RetrieveEnumLiteralSIB that gets a field status (illustrated in fig. 3) but also designed new processes for further required operations such as GetPrePopulated to load in the application a predefined set of data from a file. This phase also includes the models-to-code generation phase from the collection of validated models, and the deployment on a standard web stack. It produces a deployed, running application, that is further examined, updated, recompiled and redeployed.

4.1.3 Phase 3 – Application Execution and Testing

In this phase, the Historians and other end users (like history students and volunteers in the transcribaths for the data entry) test and use the application, as shown in fig. 4. Small adjustments and optimizations may be carried out as a consequence of live testing. This is the validation and use phase of the current version of the application. It includes live debugging, error handling and fixing, as well as the definition of new features and changes for the next development phase.

![Fig. 4: Model-driven abstract architecture of DBDIrl’s data entry web application – Feature level](image)

The whole cycle follows an agile software development procedure.

4.2 Modeling the Historian Web Application: The Full Workflow

We describe now the application workflow along with the explanation of the main processes and GUI models developed for the application. The Model-Driven Development (MDD) of DBDIrl starts with listing and developing process models, the data model and identifying user roles. Fig. 4 illustrates the feature-level abstract architecture of DBDIrl’s data entry web application in terms of Processes, GUIs, Actions and Event Handlers (as indicated by the respective stereotypes <<Process>>, <<Action>> etc.) along with the connections among them.

The application homepage is a GUI model where users can login. Its action Login calls the process IsSupervisorGuard, that checks the login credentials and establishes the user role: Supervisor or Student.

In the Student role, a successful login directly links to the EntryTable page, a GUI model where the student sees all the entries recorded by him/her. Action AddNewEntry leads the user to the EntryForm page, a GUI model which calls the CreateEntry process shown in Fig. 3. On the EntryForm page, users enter the data of all the records from the .TIFF file of the death record register page, by filling up field by field the record's properties in the corresponding fields on the web page.

CreateEntry is a big process: it receives the data entered by the user and to do so in an error-free way it calls other processes that provide support functions. For example, it uses the GetPrePopulation process for reading pre-populated data from a file, GetSuggestions to provide pre-populated options in the Web form as drop-down menu for certain data attributes, CreateAddress to create a new (complex) address object with the individual attributes city, county, district and street.

Similarly, CreateTimeDuration creates a duration object from various time properties. The GetPrePopulation process receives the group id of a death record provided by the user, and it reads name, age, superintendent's district and .TIFF file path from the .csv files we received from the GRO. It also auto-fills the corresponding fields of the EntryForm.

The GetSuggestions process reads large lists of pre-populated, validated values for a number of properties. It displays those options as a drop-down menu in the form, to ease the input of attributes like cause of death, registrar name, assistant name, rank profession and street names of Ireland from 1864 to 1922. These data collections are pre-validated, as the Historians collected them from 18's Ireland records.

CreateEntry performs all the individual operations needed to successfully save an entry with all its values entered by the user (either by hand or by selecting pre-populated fields), property by property. Event listeners on CreateEntry process help check data validity and show alert messages in case of a wrong entry (incorrect value or format).

![Fig. 5: The Entry Table of the Historian App: Web page (Left) and its corresponding GUI model (Right)](image)
Finally, the action SaveEntry from the EntryForm successfully saves an entry and sends the user back to the EntryTable, to process the next record.

Fig. 5 shows the entry table as it is displayed on the Historian App web page, with the corresponding GUI model in DIME. We see here that the structure and look and feel are very recognizable. The data flow is explicitly modelled, and we recognize buttons (like the CreateEntry button) and other elements like fields filled from the database and status indicators that are color coded (orange, green and blue).

From the EntryTable, selecting an entry leads the users to the EntryDetails GUI: there they can (re)view the entered entry details and choose to edit the entry (this brings them back to the EntryForm, filled with all the previously entered data), or submit the entry for review. The Entry Delete option is only available to the Supervisor role, who can delete an entry from the database.

In the Supervisor role, a successful login directly links to the EntryTable GUI. The supervisor is displayed the user’s entries and also has other options, like seeing all submitted and approved entries individually. The Supervisors have a validation and approval function: they can see the details of all the entries stored by Student users, and have actions to perform edits, approve, as well as remove each entry.

Fig. 6: Entry table management process of Historian app, with flows for the Approve, Submit for Review, Edit, Remove, Close entry operations

Fig. 6 shows the ShowEntry process, with flows for the Approve, Submit for Review, Edit, Remove, Close entry operations. Selecting ManageUser leads the supervisor to a UserTable page that includes the AllUser GUI (displaying all the users) and the AddNewUser action. AddNewUser calls the UserForm GUI, containing all the fields required to creating an user.

Similar to CreateEntry, the CreateUser process performs all the operations necessary to create a user. It also includes an event listener that alerts the supervisor in case wrong or ill-formatted information is provided. After saving a user, the action EditUser calls the UserForm again with the previously provided data to edit, while DeleteUser calls the RemoveUser process to delete the user.

Supervisor can also import and export data to and from the application. An event listener is used to check issues regarding import/export operations, together with the processes ExportEntry and ImportData that export and import data, respectively.

Altogether, Fig. 4 presents an overview of how the Historian App is organized, and shows the interplay among all the Processes, GUIs, (GUI)Actions and Event Handlers.

5. Model Types and Concrete Models

We describe now the main model types, model elements and models of the Historian App.
5.1 Data Model
Fig. 7 shows the data model of the application, representing both the concrete and abstract data. It contains both unidirectional and bidirectional relations such as association and inheritance. In our finer granular representation of a record, every Entry has 27 individual properties. Some properties like Sex, Address, Age include sub properties, which are at the elementary granularity needed for data analysis. The decision of moving to these 27 properties from the original 11 properties of fig. 1 is an example of the design choices for the Historian App stemming from the co-design practice. As shown in fig. 7, every concrete user can have base user who as act as creator (only students) or approver of an entry. The entry itself is a concrete type data at DIME (green data objects at fig. 7). Most of the attributes of entry are stored as text or number i.e. primitive attributes of DIME (small yellow components at fig. 7). Some are Enum type attributes e.g. Civil Status with some optional values (brown data objects at fig. 7). Some properties may not be present in the original record: the corresponding cases are captured by the FieldStatus. Some attributes e.g. Registrar Name are created as concrete type object so that they can receive list of data options and presented as drop-down menu to the application and user can choose the correct information from options. Duration and Address are also concrete type objects with required values.

5.2 Graphical User Interface Models
In DIME, the GUI model type represents the structure (layout and contents) of the Historian app's individual web pages. A collection of GUI models defined, therefore the abstract and concrete «looks» of the presentation layer of a DIME application. We see in fig. 4 that the created GUI models connect the GUI and Process Models. Every GUI models of Historian DIME app is created using components from DIME palette. The GUI models call process models to execute an operation. These GUI models are also reusable, for example we use EntryForm at fig. 4 for both create and update operation of each entry.

5.3 Native DSLs
In DIME, the actions and services are collected in domain specific palettes that are basically a service or component oriented DSL. The DSL elements correspond to (calls to) individual functionalities that are either directly implemented or provided by an external service provider, like e.g. the database. The individual functionalities are modelled as special native types called SIBs, for service-independent building blocks, where service-independent means that they are widely reusable across applications. These Native SIBs enable interoperability on a structural level. Within the Historian app, besides the pre-existing DIME SIBs we create Native SIBs for different operations such as data pre-population, CSV file import and export and to get field suggestions etc.

5.4 Data-flow
In DIME data flow is explicitly modelled within the process models. The input/output ports of SIBs can either be connected directly with each other or used to read and write from/to variables placed in a dedicated container representing the data context [9]. Fig. 4 shows the data flow connections of the proposed application using arrows. All other figures of the Historian DIME app shows data flow connection.

5.4 Process models
Process models express the business logic in a fashion roughly similar to Activity Diagrams, but with a clean formal semantics. There are several process types: basic, interactive and interaction processes. Each process type follows certain rules regarding which kind of SIBs they contain and the kind of tasks they express. The graphical syntax and general handling are the same for all the types of SIBs and processes. Fig. 4 shows the process models that are created for the Historian application. The collection of process models together with the connected GUI, actions and data flow expresses the behaviour of the application in terms of its operation. Fig. 3 shows the actual process model CreateEntry of Historian app (also presented in fig. 4) that performs all necessary operations and successfully stores an entry to the database.

Process models can also be of different type such as
a) Basic Processes: Basic Processes consist of native SIBs and built-in SIBs, and express the smallest processes of the application's business logic. In the Historian app, basic processes models are the CRUD (i.e., create, read, update, and delete) operations and data operations. In fig. 4 CreateUser and RemoveEntry are two examples of basic processes.
b) Interactive Processes: Interactive Processes work as interfaces between the front-end layer and the backend of the application. They are similar to Basic processes, but are restricted to non-native type. The StartUp process is the only interactive process in the Historian app. This process includes operation of successful login of user with different role.
c) Interaction Processes: Interaction Processes are used to define the immediate interaction between user and application, accordingly they can be seen as a sitemap [9]. Where interactive SIBs communicate with the backend, interaction SIBs establish a new hierarchy level with the frontend. As the Historian app is essentially a sophisticated DaTa entry app, most of its processes are developed as interaction process, like GetPrePopulation, GetSuggestions, CreateAddress etc.
d) Security Processes: Security Processes realize the (role based) access control with a predefined interface. The IsSupervisorGuard and ExportFileGuard processes are two examples of security processes in the Historian application. In IsSupervisorGuard the start node must include the currently signed-in user (i.e., the Supervisor) as an input, and all following nodes are restricted to be labelled with «granted».

6. Challenges and Solutions
6.1 Defining the major context parameters
In a context-dependent application, a DSL should enable modelling the different context situations that may occur during user interface usage. This DSL will eventually help developers to separately specify context-specific services to monitor various parameters and react accordingly. For example, the application's abstract GUI rules cover various adaptation dimensions: layout, navigation, reusability. Accordingly, modelling, adaptation, transformation and execution of processes and GUIs take into consideration the context management and the corresponding adaptation. In particular, the processes and functionalities are associated with responsive GUIs.

6.2 User interface adaptation at runtime
To achieve a responsive web application, the integrated execution environment must be equipped to generate adaptation services in dependence of the context. For this, the generated adaptation processes need to be coupled with generated code that enables an automatic dynamic reaction of the runtime UI to the context-of-use. In the Historian App, the data entered by the user is the predominant part of the dynamic context to which the approach reacts. The reaction manifests itself in a validation of the entry or an error message if problems are detected. To address various run-time errors, we introduced 'Alert' models with the event handler. To this aim, we introduced native SIBs for several condition checks, detecting e.g., whether an unintentional special character is entered, or a date entered in an incorrect format, a required field is left unattended, the ID not unique etc. Process events are connected with those detections, and respective event-listeners are introduced at the corresponding GUI models, enabling this was a run time error handling. Fig. 8(a) shows an event and corresponding event listener model connected with respective alert that warns the user that
‘Name can’t have special character’ (fig. 8(b)). We also proposed a rule-based classifier [31] for overall data monitoring and error detection. The integration of the classifier with the DIME application is currently ongoing.

6.2 Data Entry with minimum error

In 2020 we conducted a pilot Transcribathon using the Historian application version 1, which was not a responsive application with built-in data checks. Examining the resulting data entry, it emerged that most of the wrong entries occurred at the fields Cause of death, Address, Age/Duration and at the Registrar names. The date format also posed problems.

As a solution, we worked together to create drop-down lists of cause of death, registrar name and street name which can be used to provide a predefined list of suggestions, thereby eliminating the free text entry, and reducing error rate. For registrar name prediction, it is possible to create a registrar names list for the period and location of interest. Using the method employed by 18 we ordered the geographical data like street, city and county names of Ireland, which occur in various address fields. All these lists are added to the application, by augmenting it with native SHBs and adequate GUI and process elements. Fig. 9 shows an example. The Web page screenshot of fig. 9b shows the Registrar data entry page, and in particular the pre-populated data field for the registrar name of a death record. This drop-down menu or combo-box lists all the registrar names collected from the early 1900s Dublin Street Directory. Once the historians found and verified the names of the registrars who registered the death records relevant to this specific time and place, the IT specialists created the GetSuggestion process shown in fig. 9(a). This process shows the list as a drop-down menu in the right location of the application page. Thus, instead of inputting a free string that needs to be validated, in version 2 for this field users can select the correct option from this menu, avoiding errors instead of repairing them.

6. Conclusions

In this paper, we presented the first MDD based application developed in the DBDirl project to support the correct and reliable data entry of Civil registration records. As this project deals with a large dataset, we need a reliable application that prevents as much as possible errors. To this aim, we co-developed with the Historians a model-driven application using XMDD as an agile version
of MDD and the DIME integrated modelling environment. We briefly introduced the various model types and showed how they are used in conjunction to create a coherent data, process, GUI and role-based access model. The main advantage of these choices is the ability to quickly react to the findings, exemplified here by the evolution from the V1 to V2 of the App. In particular, the V2 greatly improves the achieved data quality by making the Application reactive to context-specific events, and equipping most of the data entry fields with pre-populated lists of plausible options, as for addressing, courses of disease, and registrar names, and with context-specific rule checks, as for date and age formats. The main lesson learned is thus that an application is necessarily long lived, due to the sheer enormous amount of data to be digitized over time, by many groups of volunteers, and never really «finished». In such a context of continuous improvement, the ability to collaborate with the Historians on the basis of models rather than code is an essential asset, producing successive versions of the app that improve or customise specific aspects of the functionality and the presentation.

The work currently in progress concerns on the one side the inclusion of rule-based classifiers in the application, and on the other side the development in the same paradigm of a data analytics application. The App needs to be as flexible and customizable as this one, because it will serve the certainly diverse and specialized analysis needs of a growing community of researchers working on big data archival systems in the digital humanities.

References

[1] J. Schnapp, L. Peter, and T. Presner. Digital humanities manifesto, 2008. Available at https://tcp.hypotheses.org/category/manifeste.

[2] M. Grieses and J. Vickers. Digital twin: Mitigating unpredictable, undesirable emergent behavior. In Transdisciplinary perspectives on complex systems, Springer, 2017, pp. 85-113.

[3] Y. Lu, C. Liu et al. Digital twin driven smart manufacturing: Concept, reference model, applications and research issues. Robotics and Computer-Integrated Manufacturing, vol. 61, 2020, article 101837.

[4] M. Dalibor, J. Michael et al. Towards a model-driven architecture for interactive digital twin cockpits. Lecture Notes in Computer Science, vol. 12400, 2020, pp. 377-387.

[5] T. Margaria and B. Steffen. Extreme model-driven development (xmdd) technologies as a hands-on approach to software development without coding. Encyclopedia of Education and Information Technologies, 2020, pp. 732-750.

[6] R. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In Proc. of the Conference on Future of Software Engineering (FOSE’07), 2007, pp. 37-54.

[7] C. Breathnach, N. M. Ibrahim et al. Towards model checking product lines in the digital humanities: An application to historical data. Lecture Notes in Computer Science, vol. 11865, 2019, pp. 338-364.

[8] M.D. Wilkinson, M. Dumontier et al. Addendum: The fair guiding principles for scientific data management and stewardship. Scientific data, vol. 6, no. 1, 2019, pp. 1-2.

[9] S. Bollellmann, M. Frohme et al. DIME: A Programming-Less Modeling Environment for Web Applications. Lecture Notes in Computer Science, vol. 9935, 2016, pp. 809-832.

[10] B. Steffen, T. Margaria et al. The metaframe’95 environment. Lecture Notes in Computer Science, vol. 1102, 1996, pp. 450-453.

[11] B. Steffen, T. Margaria et al. An environment for the creation of intelligent network services. In INAIN Technologies, Operations, Services, and Applications A Comprehensive Report, International Engineering Consortium, 1996, pp. 287-300.

[12] F. Pratennai, S. Comai et al. Engineering rich internet applications with a model-driven approach. ACM Transactions on the Web (TWEB), vol. 4, no. 2, 2010, pp. 1-47.

[13] T. Margaria. Web services-based tool-integration in the eti platform. Software & Systems Modeling, vol. 4, no. 2, 2005, pp. 141-156.

[14] T. Margaria, C. Kubczak et al. Model-based design of distributed collaborative bioinformatics processes in the jabc. In Proc. of the 11th IEEE International Conference on Engineering of Complex Computer Systems (ICECSS’06), 2006, 8 pp.

[15] A-L. Lamprecht, T. Margaria et al. Genefisher-p: variations of genefisher as processes in bio-jeti. BMC bioinformatics, vol. 9, no. 4, 2008, pp. 1-15.

[16] D. Withers, E. Kawas et al. Semantically-guided workflow construction in Taverna: the SADI and BioMoby plug-ins. Lecture Notes in Computer Science, vol. 6416, 2010, pp. 301-312.

[17] U. M. da Silva. Model-driven engineering: A survey supported by the unified conceptual model. Computer Languages, Systems & Structures, vol. 43, 2015, pp. 139-155.

[18] L. J. Mengerink, A. Serebrinik et al. Automated analyses of model-driven artifacts: obtaining insights into industrial application of mde. 27th International Workshop on Software Process and Product Measurement, 2017, pp. 116–121.

[19] S. Jerges, C. Kubczak et al. Model Driven Design of Reliable Robot Control Programs Using the JABC. In Proc. of the Fourth IEEE International Workshop on Engineering of Autonomic and Autonomous Systems (EAS’07, 2007), pp. 137-148.

[20] T. Margaria and A. Schieweck. The digital thread in industry 4.0. Lecture Notes in Computer Science, vol. 11918, 2019, pp. 3-24.

[21] B. Steffen, T. Margaria et al. A constraint-oriented service creation environment. Lecture Notes in Computer Science, vol. 1055, 1996, pp. 418-421.

[22] P. Mikulecky. Formal models for ambient intelligence. In Proc. of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement, 2017, pp. 116–121.

[23] D. Breuker. Towards model-driven engineering for big data analytics – an exploratory analysis of domain-specific languages for machine learning. In Proc. of the 47th Hawaii International Conference on System Sciences, 2014, pp. 758-767.

[24] M. Gofarelli and S. Rizzi. A model-driven approach to automate data visualization in big data analytics. Information Visualization, vol. 19, no. 1, 2020, pp. 24-47.

[25] C. Castellanos, B. Pérez et al. A model-driven architectural design method for big data analytics applications. In Proc. of the 2020 IEEE International Conference on Software Architecture Companion (ICSA-C), 2020, pp. 89-94.

[26] T. Margaria and B. Steffen. Agile IT: Thinking in User-Centric Models. Communications in Computer and Information Science, vol. 17, 2009, pp. 490–502.

[27] S. Bollmann, D. Kühn, and T. Margaria. A fully model-based approach to the design of the secure community web app. In Proc. of the 12th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS), 2017, pp. 1–7.

[28] S. Jörges, A.-L. Lamprecht et al. A Constraint-based Variability Modeling Framework. International Journal on Software Tools for Technology Transfer (STTT), vol. 14, no. 5, 2012, pp. 511-530.

[29] E. O’Shea, R. Khan et al. Towards automatic data cleansing and classification of valid historical data an incremental approach based on mdd. In 2020 IEEE International Conference on Big Data (Big Data), 2020, pp. 1914-1923.
Alexander SCHIEWECK, Master in Computer Science form the TU Dortmund University, Germany, currently Research PhD Student at Computer Science and Information System Department. Research Interests: Low-Code and Formal Methods for Software Verification as part of the Software Engineering process.

Alexander ШИВЕК, магистр компьютерных наук Технического университета Дортмунда, Германия, в настоящее время аспирант-исследователь факультета компьютерных наук и информационных систем. Научные интересы: малокодовые и формальные методы верификации программного обеспечения как часть процесса разработки программного обеспечения.

Ciara BREATHNACH, PhD in History and Digital Humanities from University College Cork (UCC), Associate Professor in History. Research Interest: modern Ireland, social history, gender, medicalization, death, migration, health-history; social determinants of health in nineteenth and early twentieth Irish history; infant and maternal mortality, social history of medicalisation; the social function of modern medicine in acculturating Irish immigrants in New York and Boston, 1860-1912.

Сиара БРЕТНАХ, кандидат исторических наук и цифровых гуманитарных наук из Университетского колледжа Корка (UCC), доцент кафедры истории. Область исследования: современная Ирландия, социальная история, пол, медикализация, смерть, миграция, история здоровья; социальные детерминанты здоровья в Ирландии девятнадцатого и начала двадцатого века; младенческая и материнская смертность; социальный анамнез медикализации; социальная функция современной медицины в воспитании ирландских иммигрантов в Нью-Йорке и Бостоне, 1860-1912 гг.

Tiziana MARGARIA, PhD in Computer and Systems Engineering, Politecnico di Torino, Italy, Professor at Computer Science and Information System Department. Research Interest: eXtreme Model Driven Design, lightweight formal methods, automatic program synthesis, system correctness, in particular compliance and security, future education in SE and IT.

Тициана МАРГАРИА, кандидат компьютерных наук и системной инженерии, Туринский политехнический университет, Италия, профессор кафедры компьютерных наук и информационных систем. Область научных интересов: проектирование на основе экстремальных моделей, упрощенные формальные методы, автоматический синтез программ, правильность системы, в частности соответствие и безопасность, будущее образование в области программной инженерии и информационной технологии.