Advancing Computational Architectures for Analyzing and Simulation of Systems of nanoporous particles filtration Software

[1]Mykhailo Petryk[a], Vitaly Brevus[a], Dmytro Mykhalyk[a] and Oleh Kyshkevych[a]

[a] *Ternopil Ivan Puluj National Technical University, 56 Ruska str., Ternopil 46001, Ukraine*

Abstract

This article offers a comprehensive review of an ageing nanoporous particle filtration simulation software program. We examine its outdated architecture, highlighting its strengths and weaknesses. Strengths include decoupling while weaknesses encompass visualisation lack and performance issues. We propose innovative updates, integrating modern computational architectures and enhancing user experience, aiming to revitalize this pivotal tool for contemporary research.

Keywords

Software architecture, Legacy systems, Graphical user interface, Modeling software, Nanoporous particle filtration, Architecture enhancements

1. Introduction

Solid-liquid expression of biological materials represents a fundamental unit operation with a profound impact on industries such as food processing. This process plays a crucial role in the extraction of fruit juices, vegetable oils, and the removal of moisture from fibrous materials and wastewater sludge. In the industrial setting, a porous layer, known as the press-cake, forms as a result of compressing whole fruits, such as grapes and berries, or fragmented materials like sugar beet slices, apple mash, and ground oilseeds. The application of mechanical pressure during expression can occur under various conditions, including constant or variable parameters like pressure and deformation rate. The raw biological materials involved in these processes comprise cells filled with liquid, hydrated cell walls, microchannels between cells (plasmodesmata), and intercellular spaces that contain air. These materials are essentially porous media characterized by diverse types of pores and channels [1]. Additionally, raw materials can undergo pretreatment, which may involve mechanical methods like slicing and grinding, thermal treatments such as heating and freezing-thawing, or other techniques like chemical treatment, enzymatic processes, or pulsed electric field treatment [2]. This pretreatment serves to disrupt or denature the cellular structure and facilitate the subsequent expression of liquids from the materials.

The mechanisms underlying solid-liquid expression differ when working with fresh raw materials that have initially intact cells versus materials with initially disrupted cells. The process of breaking cells in fresh tissue is influenced not only by applied mechanical pressure and deformation rate but also by the structural characteristics of the tissue itself, such as cell size, shape, thickness of cell walls, middle lamella, and osmotic pressure inside the cells [1, 3]. In fresh tissues, intact cells can break individually or in groups, with the weakest cells breaking first, followed by the stronger ones. This intricacy results in slow liquid release from fresh tissue, as intact cells remain impermeable and hinder overall tissue permeability. The kinetics of expression also depend on the size of the particles, as coarse particles (slices) have a lower proportion of disrupted cells compared to finely fragmented materials

[1]Proceedings ITTAP'2023: 3rd International Workshop on Information Technologies: Theoretical and Applied Problems, November 22–24, 2023, Ternopil, Ukraine, Opole, Poland

EMAIL: petrykmr@gmail.com (A. 1); dmykhalyk@gmail.com (A. 2); v_brevus@tntu.edu.ua (A. 3); oleh.kyshkevych29@gmail.com (A. 4)
ORCID: 0000-0001-6612-7213 (A. 1); 0000-0001-9032-695X (A. 2); 0000-0002-7055-9905 (A. 3); 0009-0003-8878-3205 (A. 4)

(mash). Furthermore, the effectiveness of cell pretreatment significantly impacts tissue permeability, and the permeability of extra particle pathways (channels) also plays a pivotal role [4]. These interparticle channels can be compressible and may partially or completely close during expression, especially with plate-like or very fine particles. The presence of air within tissue structures and surrounding fragmentized particles contributes to the compressibility of the press-cake and further influences expression kinetics, making the mechanism of solid-liquid expression from biological materials a complex and not entirely understood phenomenon.

For many years, the study of solid-liquid expression in biological materials has been informed by previous research, as referenced in [1] and [2], where researchers employed software to model and analyze these intricate processes. However, it's essential to acknowledge that significant developments in technology and methodologies have transpired since the active development of this software in the 2010s. Now, over a decade has passed, during which the landscape of computational architectures and software engineering approaches has seen substantial evolution. Consequently, there is a pressing need to revisit and update this software to align it with contemporary standards and harness the full potential of modern computational architectures. This article embarks on precisely this endeavor, aiming to enhance the software's capabilities and provide a renewed tool that meets the demands of cutting-edge research in nanoporous particle filtration and related fields.

The historical context of the ageing software adds depth to its significance. Initially developed during the time when Ant and Java Swing were popular tools, but now they are considered legacy technology. While the research was ongoing, a major transition in the Java ecosystem happened, moving from Ant to Maven and now embracing Gradle as the modern standard for project management build processes [5, 6]. In this era, JavaFX has emerged as a sophisticated library for developing desktop applications, offering a significant leap beyond the traditional Swing framework [7]. Importantly, the software's core architecture, including its class structure, has stood the test of time, providing a solid foundation for research. As we venture into this article, we aim to rejuvenate this software, respecting its architectural strengths while incorporating modern innovations to enhance its user interface and overall functionality. This revitalization aligns the software with contemporary software engineering practices and computational architectures, ensuring it’s a valuable asset for cutting-edge research in nanoporous particle filtration and related domains.

2. Analysis of liquid extraction from biological materials software solution

The existing software solution under examination, known as “Pressing Biological Materials” (PBM) stands as a desktop application that has been crafted using Java and relies on the Swing framework for its user interface. Notably, the application's development process leveraged the NetBeans Integrated Development Environment (IDE) and employed Ant as its primary build tool [5, 8]. PBM is structured into four distinct modules, with each module serving a specific purpose. Three of these modules, “PBMFlow”, “PBMLinear” and “PBMLinearCoefCmp” are dedicated to various computational algorithms, each tailored to address specific aspects of nanoporous particle filtration. The fourth module, aptly named “PBM” is primarily responsible for constructing the user interface and facilitating user interaction, serving as the gateway between the user and the underlying computational components of the software.

The current software solution (PBM) exhibits a notable strength in its well-implemented decoupling, which enhances its modularity and maintainability. Decoupling, by reducing interdependencies between components, allows for easier modification and extension of the software. This quality is nicely exemplified in PBM's class hierarchy, as it skillfully employs object-oriented programming (OOP) techniques [9] such as inheritance, implementation and interfaces to achieve decoupling between different software modules as shown in Figure 1. Module PBM.Direct is used for math modelling of pressing liquids from biological materials by utilizing numerical and analytical algorithms. PBMLinear and PBMNumerical interfaces can be extended to provide flexibility for modification [9].

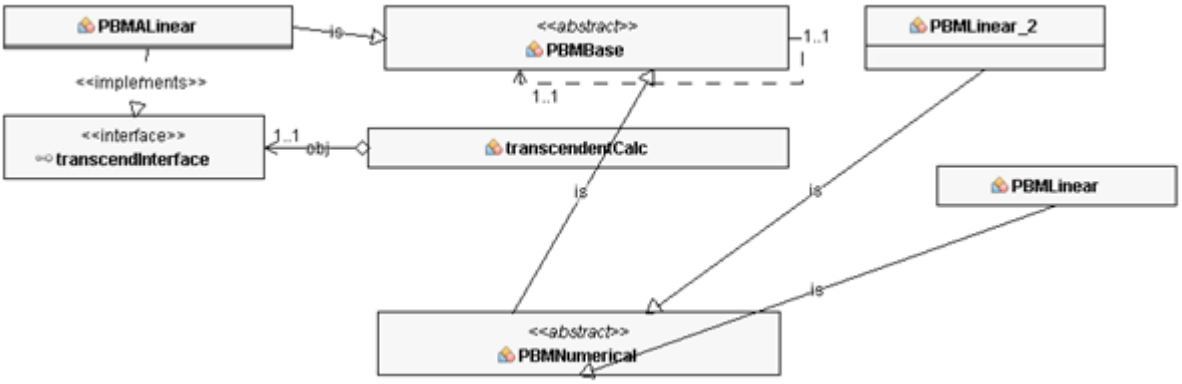

Figure 1: Module PBM.Direct

Figure 2 illustrates a PBM.GUI.Components class diagram, which showcases the harmonious use of various classes in constructing the user interface. This organized class hierarchy, combined with the modular architecture, not only enhances the software's scalability but also simplifies the debugging and maintenance process, thus contributing to its overall robustness.

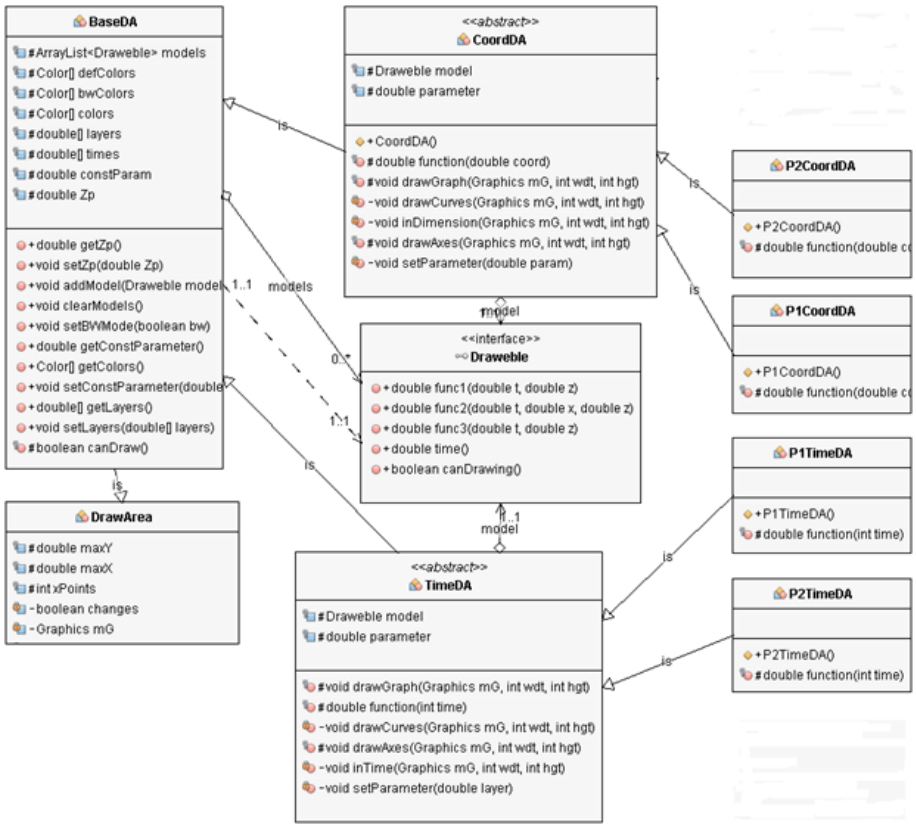

Figure 2: Module PBM.GUI.Components

After highlighting the strengths of the current modelling software (PBM), several weaknesses come to light. One notable limitation pertains to the software's visualization capabilities, which are relatively rudimentary. While the program excels in computational aspects, the visualization of data and results remains underdeveloped, which can hamper the user's ability to interpret and analyze the output effectively [10]. Furthermore, performance issues have been observed, especially when handling larger datasets, leading to potential delays and inefficiencies in processing. Additionally, one substantial constraint pertains to the choice of an outdated build tool, Ant, which impedes the software's capacity

to harness more contemporary build technologies. The software's tight dependency on the NetBeans Integrated Development Environment (IDE) restricts developers from exploring alternative IDEs, constraining the development environment. An additional issue arises when considering the software's intended cross-platform functionality. Despite being designed as a Java application, it falls short of working seamlessly on UNIX file systems due to compatibility issues stemming from differences in directory styles. These limitations, particularly in visualization, performance, build tool, and cross-platform compatibility, underscore the urgency and importance of the software's renewal and modernization.

3. Modern computational architectures

In the realm of modern computational architectures, several pivotal trends and innovations are reshaping the software industry. A fundamental aspect to consider is the choice of build tools, and here, the ageing modelling software PBM, presents an opportunity for adaptation. The software currently employs Ant as its build tool, a choice indicative of its historical context. However, it is worth noting that Ant, which relies on XML configurations, has become somewhat outdated [5]. XML configurations, though effective, lack the readability and flexibility that newer technologies offer. This inadequacy is evident when compared to Groovy, which has gained prominence for its concise, expressive syntax. In the context of modern Java applications, Gradle has emerged as a powerful and adaptable build tool, offering a more efficient and flexible approach to project management and build processes.

Visualizations represent another critical facet of modern software architecture. "PBM" currently relies on the Swing framework for constructing user interfaces. While Swing has served its purpose over the years, modern Java applications have shifted towards the more advanced and feature-rich JavaFX framework [11, 12]. JavaFX offers several advantages, including superior graphics capabilities, a wider range of UI components, and improved performance. Its modern, declarative approach to UI development simplifies the creation of visually appealing and responsive user interfaces, aligning perfectly with the expectations of today's software users [12].

Beyond these considerations, a noteworthy trend in modern computational architectures involves a transition to cloud-based solutions. Many contemporary applications have embraced cloud infrastructure, providing scalable resources that can be dynamically adjusted to meet computational demands. This flexibility ensures that software can handle increased workloads and resource-intensive computations efficiently. The ability to scale resources in real-time is particularly beneficial for software solutions like "PBM," which involve complex calculations and data processing, thereby contributing to improved performance and enhanced user experiences.

These transformations in software architecture, spanning build tools, visualization frameworks, and the shift towards cloud-based solutions, collectively signal the need for the revitalization of "PBM" to align it with the expectations and capabilities of the modern software industry.

4. Proposed updates and improvements

In the pursuit of modernizing PBM modelling software the first pivotal update entails a transition from the outdated Ant build tool to the contemporary Gradle build system [5]. To facilitate this transition, we have diligently removed the "nbproject" directory, which housed build files tightly dependent on the NetBeans IDE. This adjustment marks a crucial step in disentangling the software from NetBeans' constraints, thereby affording greater flexibility in selecting development environments.

As part of the modernization effort, we have also undertaken a reconfiguration of the project's directory structure to align with Gradle's recommended standards. This updated structure adheres to

modern software project organization principles, fostering clarity and consistency in development workflows.

Moreover, to implement a specific module structure in Gradle that accommodates the interdependencies among three subproject modules while utilizing a common base module, "PBM" the following Gradle code is presented on Figure 3.

```
// build.gradle for subproject modules (PBMFlow, PBMLinear, PBMLinearCoefCmp)
dependencies {
    implementation project(":PBM")
}
```

Figure 3: Module structure of PBM project implemented in Gradle

This Gradle script efficiently orchestrates the modular structure, allowing seamless interaction between the subprojects and the core "PBM" module, while granting the flexibility to specify dependencies unique to each subproject. This update sets the stage for a more streamlined and organized software architecture, conducive to effective development, extension, and maintenance.

One of the crucial issues identified during the update of PBM modeling software pertains to its limited cross-platform support, particularly in areas where the software interacts with the file system. Currently, the software is primarily designed to work with Windows folder path styles, which creates a significant hindrance for users operating on non-Windows systems. To address this challenge and enhance the software's cross-platform compatibility, we delved into the code responsible for file system operations.

In our effort to mitigate this issue, we have revisited the pertinent code segments and made strategic updates. Specifically, we have transitioned from using hard-coded path separators to utilizing the "File.pathSeparator" constant [13]. This adjustment allows the Java Virtual Machine (JVM) to dynamically build file paths in accordance with the operating system on which the software is executed. Consequently, whether the software is run on Windows, macOS, or Linux, the file path construction becomes platform-agnostic, ensuring consistent and reliable functionality across diverse operating systems. This modification is instrumental in bolstering the software's cross-platform compatibility, ultimately making it more accessible and user-friendly for researchers and developers across various computing environments.

The transition from the Swing framework to JavaFX within the PBM modelling software has been a significant undertaking. This migration was prompted by the need to modernize the software's user interface to meet contemporary standards and expectations. However, this transformation has not been without its challenges.

One of the primary difficulties we encountered in the process was the inherent differences in the architectural paradigms of Swing and JavaFX. Adapting the existing codebase to the new JavaFX framework required a meticulous examination of UI components, layouts, and event handling mechanisms. Furthermore, ensuring a seamless integration of the existing logic with the JavaFX-based interface presented a substantial challenge, given the divergent approaches to user interface development between the two frameworks.

While this migration is ongoing, it's worth noting that the software currently operates with a blend of Swing and JavaFX components. This transitional phase allows the software to retain compatibility with the legacy interface while gradually incorporating new JavaFX elements. This approach grants us the flexibility to develop and enhance the user interface progressively, without the necessity of a complete overhaul. It's a strategic compromise that bridges the gap between legacy and modern UI development, facilitating the software's continued utility and adaptability as we embark on its journey toward full JavaFX integration.

In our efforts to modernize PBM, we've not only addressed architectural and user interface changes but also introduced a standardized code formatting style. This new code formatting style encompasses conventions for code layout, indentation, naming, and documentation, ensuring uniformity and clarity in the software's source code. The introduction of this code formatting style enhances the codebase's readability and maintainability, streamlining collaboration among developers and facilitating the debugging and review processes. Adhering to consistent code formatting conventions contributes to a more organized and efficient development environment, aligning with contemporary software engineering best practices. This initiative fosters a shared understanding of coding standards, further enhancing the software's quality and longevity.

5. Conclusion and future work

In conclusion, this study has shed light on the paramount importance of modernizing the software architecture of PBM solution. The aging software, rooted in the past, required significant updates to align with contemporary software engineering standards and practices. Through a meticulous analysis of the existing software and a thorough examination of modern computational architectures, we've identified critical strengths and weaknesses, revealing a clear path forward for enhancement.

The key findings of this study underscore the pivotal role that decoupling, JavaFX, and Gradle can play in rejuvenating the software. The deliberate removal of the "nbproject" folder, the adoption of Gradle, and the transition to JavaFX for the user interface mark significant advancements that empower "PBM" with the flexibility, readability, and scalability it lacked previously.

Furthermore, this modernization effort is not a mere update; it is a transformative journey towards ensuring the continued relevance of "PBM" in the face of evolving software requirements. The significance of this modernization extends beyond this single software, offering a case study for the scientific software development community. It underscores the importance of adapting to contemporary software engineering principles, empowering researchers and scientists with more efficient, user-friendly, and adaptable tools.

Looking ahead, several exciting avenues beckon for future research and development. The most prominent of these is the upgrade to a client-server architecture, a transformation that would enable "PBM" to harness the full potential of cloud-based computing. A client-server model can facilitate resource scaling, parallel computing, and collaboration among users, offering unprecedented computational power and flexibility. Embracing cloud-based solutions will allow "PBM" to handle increasingly complex simulations, enhancing its utility and impact in the field of nanoporous particle filtration and related research.

As we embark on this next phase of development, we recognize that further research is needed to optimize the client-server architecture and explore the potential of cloud-based computing. This includes developing secure data transfer mechanisms, designing efficient parallel processing algorithms, and ensuring seamless user access to cloud resources. These efforts will not only propel "PBM" into the forefront of scientific research tools but also contribute to the broader landscape of scientific software development. The journey has just begun, and the future holds the promise of pioneering advancements in computational research and beyond.

6. References

[1] M. Petryk and E. Vorobiev, Numerical and analytical modeling of solid–liquid expression from soft plant materials, AIChE J. 59, 4762 (2013).

[2] Petryk M., Khimitch A., Petryk M.M., Fraissard J. Experimental and computer simulation studies of dehydration on microporous adsorbent of natural gas used as motor fuel. Fuel. Vol. 239, 1324–1330 (2019)

[3] S. S. Haramkar, G. N. Thombre, S. V. Jadhav, and B. N. Thorat, The influence of particle (s) size, shape and distribution on cake filtration mechanics-a short review, Comptes Rendus. Chimie 24, 255 (2021).
[4] Lebovka N., Petyk M., Tatochenko M. and Vygornitskii N. Two-stage random sequential adsorption of discorectangles and disks on a two-dimensional surface. Physical Review E. Vol.108, 024109 (2023) DOI:https://doi.org/10.1103/PhysRevE.108.024109
[5] Hassan, F., Mostafa, S., Lam, E.S.L., Wang, X. Automatic Building of Java Projects in Software Repositories: A Study on Feasibility and Challenges (2017) International Symposium on Empirical Software Engineering and Measurement, 2017-November, pp. 38-47.
[6] Gradle inc., Gradle Build Tool, 2023. URL: https://gradle.org/
[7] OpenJDK, JavaFX, 2023. URL: https://openjfx.io/
[8] NetBeans, Apache NetBeans 19, 2023. URL: https://netbeans.apache.org/
[9] R. C. Martin, Clean Architecture: A Craftsman's Guide to Software Structure and Design, Pearson; 1st edition (September 10, 2017)
[10] O. Vasyliev, Програмування мовою Java, 2020.
[11] B. Verhaeghe, N. Anquetil, A. Etien, S. Ducasse, A. Seriai, M. Derras. GUI visual aspect migration: a framework agnostic solution (2021) Automated Software Engineering, 28 (2), art. no. 6, .
[12] Robillard, M.P. , Kutschera, K. Lessons Learned while Migrating from Swing to JavaFX (2020) IEEE Software, 37 (3), art. no. 8725586, pp. 78-85.
[13] Tarnum Java SRL, Java File Separator vs File Path Separator, 2022. URL: https://www.baeldung.com/java-file-vs-file-path-separator