Article

CINCO de Bio: A Low-Code Platform for Domain-Specific Workflows for Biomedical Imaging Research

Colm Brandon 1,2,*, Steve Boßelmann 3, Amandeep Singh 1,2, Stephen Ryan 1,4, Alexander Schieweck 1,4, Eanna Fennell 5,6,7, Bernhard Steffen 3, and Tiziana Margaria 1,2,4,5,*

1 Department of Computer Science and Information Systems, University of Limerick, V94 T9PX Limerick, Ireland; colm.brandon@ul.ie (C.B.); amandeep.singh@ul.ie (A.S.); stephen.e.ryan@ul.ie (S.R.); alexander.schieweck@ul.ie (A.S.); eanna.fennell@ul.ie (E.F.)
2 Centre for Research Training in Artificial Intelligence (CRT AI), T12 XF62 Cork, Ireland
3 Lehrstuhl Für Programiersysteme, TU Dortmund University, Otto-Hahn-Str. 14, 44221 Dortmund, Germany; steve.bosseleman@cs.tu-dortmund.de (S.B.); steffen@cs.tu-dortmund.de (B.S.)
4 Lero-Science Foundation Ireland Research Centre for Software, University of Limerick, V94 DNY3 Limerick, Ireland
5 The Health Research Institute (HRI), University of Limerick, V94 T9PX Limerick, Ireland
6 The Bernal Institute, University of Limerick, V94 T9PX Limerick, Ireland
7 School of Medicine, University of Limerick, V94 T9PX Limerick, Ireland
* Correspondence: tiziana.margaria@ul.ie

Abstract: Background: In biomedical imaging research, experimental biologists generate vast amounts of data that require advanced computational analysis. Breakthroughs in experimental techniques, such as multiplex immunofluorescence tissue imaging, enable detailed proteomic analysis, but most biomedical researchers lack the programming and Artificial Intelligence (AI) expertise to leverage these innovations effectively. Methods: CINCO de Bio (CdB) is a web-based, collaborative low-code/no-code modelling and execution platform designed to address this challenge. It is designed along Model-Driven Development (MDD) and Service-Oriented Architecture (SOA) to enable modularity and scalability, and it is underpinned by formal methods to ensure correctness. The pre-processing of immunofluorescence images illustrates the ease of use and ease of modelling with CdB in comparison with the current, mostly manual, approaches. Results: CdB simplifies the deployment of data processing services that may use heterogeneous technologies. User-designed models support both a collaborative and user-centred design for biologists. Domain-Specific Languages for the Application domain (A-DSLs) are supported through data and process ontologies/taxonomies. They allow biologists to effectively model workflows in the terminology of their field. Conclusions: Comparative analysis of similar platforms in the literature illustrates the superiority of CdB along a number of comparison dimensions. We are expanding the platform’s capabilities and applying it to other domains of biomedical research.

Keywords: web-based collaboration; low-code/no-code application development; collaborative modelling; model driven development; workflows; domain specific languages; biomedical image processing; Artificial Intelligence; computational biology

1. Introduction

In biomedical imaging research, experimental biologists produce data at an ever-increasing scale, especially concerning rich and large images, with the subsequent need to analyse them. Breakthroughs in experimental methods such as multiplex immunofluorescence tissue imaging [1] have opened an avenue for researchers to undertake detailed proteomic analysis of cells within their original spatial context [2]. New experimental methods, computational analysis and artificial intelligence techniques from the open source/open science community advance the support of such innovations. However, realising the full
potential value of such techniques is beyond the reach of the majority of biomedical re-
searchers: besides the knowledge and experience in their research domain, they also need
expertise in programming, AI and software engineering. Today, they need to acquire those
skills in months of bespoke training, with a huge effort overhead even to obtain a mediocre
skill level.

Extracting actionable knowledge from the raw experimental data is a significant
methodological and computational challenge, far beyond what can be expected from
specialists without a computer science or software engineering background. This is the
opposite of simplicity and ease of use. It is not infrequent to hear from PhD students in
the biomedical domain that their first 6–18 months were effectively training in computer
science and that they suffered greatly through.

To address this problem, we introduce CdB, a web-based Low-Code/No-Code (LCNC)
platform for the design and execution of biomedical image processing and analysis work-
flows (The current source code and preliminary documentation are available at: https:
//github.com/colm-brandon-ul/cincodebio (accessed on 1 August 2024). As the project
progresses, they will be updated accordingly). The platform aims at both simplicity (of use)
and openness: it is designed to facilitate the reproducibility of analysis and data provenance
tracking. It can be deployed on a single machine, a cluster, or the cloud and is accessed via
a web browser.

The CdB system consists of several services, each of which falls broadly under these
categories: workflow modelling, workflow execution, data processing and data storage.
Together, they facilitate a user to define a workflow using a graphical DSL and then
subsequently execute that workflow from within the CdB system.

In essence, CdB is designed to enable biologists and bioinformaticians to focus on the
object of their research rather than having to learn how to code, thereby increasing their
productivity and experimental output. In addition, as far as we know, CdB is the only
platform in this domain supporting semantic data flow validation at design time.

As far as we are aware, CdB will also become the only platform supporting collabora-
tive modelling of workflows, akin to a shared Google document being edited by multiple
users. From the point of view of computation for the natural sciences, CdB provides ease of
use and ease of modelling. This way of generating and deploying code produced from user-
designed models supports both a collaborative and user-centred design for the biologists,
and it also enables end-user constructive pro-activity. This is in alignment with our prior
research in “Simplicity in IT”, based on the notion of simplicity as a driving paradigm in
Information and Communications Technology (ICT) development, maintenance and use.

We believe that the philosophy of simplicity is strategically important yet poorly
understood and rarely systematically applied. Instead, design principles attempt to focus
on increased functionality within thinly disguised complexity, often at the expense of life
cycle costs and total cost of ownership issues (e.g., training, system malfunctions and
system upgrades). Often, designers are unaware of the trade-offs and impacts. With the
increased use of ICT in such socially critical areas, such as research for health, society can
no longer afford systems that do not perform as specified. Computational image analysis
is not new, with various tools being developed in the past two decades. New problems,
technologies and the accelerating rate of innovation present challenges to biomedical
domain researchers when trying to incorporate the latest AI-based tools into their research.

The motivation behind the design and development of the CdB system is to alleviate
or solve these hurdles, with a scalable and evolvable architecture that directly supports
the biology experts. While we feel the CdB system will assist biology researchers in a
broad spectrum of use cases, the initial motivation for the development of CdB is to
assist researchers working with highly-plexed immunofluorescence images [1]. The initial
Application Domain-Specific Language (A-DSL) developed for use in CdB is for highly-
plexed tissue image analysis, and the CdB system is discussed in this paper through that
lens. We start by introducing and discussing the motivating use case.
The remainder of the paper is laid out as follows. Section 1.1 introduces the use case, which initially motivated the development of CdB. Section 1.2 discusses the background and related work. Section 2 introduces and outlines our proposed solution and how it works. Section 3 presents our results in the form of a case study and comparison of CdB to similar tools. Finally, Section 4 offers some conclusions to the paper and the steps we will take moving forward.

1.1. Use Case: Highly-Plexed Tissue Image Analysis

High-dimensional multiplex immunofluorescence imaging has the potential to revolutionise our understanding of tissue-based disease. Concerning cancer, these images can reveal interesting interactions between tumour cells and their immune microenvironment. Analysis based on these observations could offer insight into a patient’s response to therapies or their risk of mortality. However, the analysis of these images presents unique computational challenges. For example, an imaging run configured to output an 80-plex image of a tissue slide would result in an output file size in excess of 100 GB. As biologists and other wet-laboratory researchers have little experience in image and bioinformatics analyses, they wish to have bespoke workflows that process and analyse these images within their own domain expertise and level of IT proficiency.

Figure 1 shows an overview of highly-plexed tissue processing. The Image Capture Workflow (top) shows what happens in the wet laboratory, and the Image Processing Workflow (bottom) shows the computer-based processing of the resulting images. A third and final step in the analysis is the actual spatial/phenotypic analysis and, for brevity, it will not be discussed here.

1.1.1. Image Capture

To capture an image, the experimental biologist places a tissue sample or samples on a glass slide. They use an antibody stain on the slide before placing it into the imaging machine. The imaging machine then performs a number of image capture cycles, which produce a multi-page image of the slide. Each page is the image of a specific protein...
channel that is captured by the machine. With the image, a Channel Marker File is output as well: a text file containing the protein marker label for each page in the image. The image capture process is performed either in a Whole Slide-Tissue Imaging (WSI) configuration or in Tissue Micro Array (TMA) configuration. Whole Slide Image (WSI) is appropriate when the glass slide contains a single large tissue sample. TMA is adequate when there are multiple small tissue samples (0.6 to 2.0 mm in diameter), called cores, on a single glass slide. The imaging procedure is essentially the same, irrespective of the configuration.

1.1.2. The Traditional Approach to Computational Analysis of Highly-Plexed Tissue Images

We consulted with many biologists and bioinformaticians to ascertain how they undertake this analysis. Several researchers are currently simply not able to effectively execute a computational workflow of this nature due to the skills shortage within their labs in one or more of the tasks. From those who are able, we discovered that they follow a common approach. Here, is therefore, an overview of how a bioinformatician would typically perform the image processing workflow seen in Figure 1, i.e., before CdB.

Handling Raw Data:
The raw data are uploaded from the experimental machine to the cloud storage system provided by their institution. They then download the data from the cloud storage to their own workstation, where the analysis typically takes place.

Executing the workflow:

- **Initial Data Preparation**: In a typical computing environment, these Tag Image File Format (TIFF) files are too large to be loaded into the memory, and most image processing tools are not designed to handle images in the multi-page TIFF format. Therefore, the first step in the workflow is to split the multi-page TIFF into a set of single grey-scale TIFFs, where each grey-scale TIFF is the image of a single protein channel acquired from the experiment. This is typically done in MATLAB or Python, using a TIFF Reader to load a single page from the multi-page TIFF into memory, and then writing that single page to disk as a TIFF file (with the name of the protein marker the page denotes).

- **De-Arraying**: When handling a TMA, the de-arraying processing step crops each individual tissue sample on the slide. This step can be done manually using a tool to draw the regions of interest, or it can be automated using AI, with a semantic segmentation model drawing bounding boxes for the tissue samples. To manually perform the de-arraying of the TMA, the grey-scale image, which contains the nuclear stain (such as 4′,6-diamidino-2-phenylindole (DAPI)), is loaded into MATLAB or Python. The image is resized to fit on the screen, then it is opened in a window where a human manually draws an Region of Interest (ROI). The coordinates of the ROI are then translated back to the full-size image. Every grey-scale image is then cropped using those coordinates, and the cropped images are saved to disk. This process is repeated for every core in the TMA. Automated de-arraying is done using AI tools available as libraries in Python. The steps are the same, but the AI model replaces the need for the windows to manually draw the bounding box.

- **Cell Segmentation**: As a given tissue sample could comprise tens of thousands to millions of cells, manual cell segmentation is not feasible, and therefore, it is done almost exclusively with AI-based tools. These tools are almost exclusively available as Python libraries. The grey-scale TIFFs containing the protein channels with the strongest nucleus and membrane expressions (in some cases, nucleus only and the membrane region is inferred from the nucleus) are loaded into Python and passed to one of the Python-based AI tools which perform the segmentation. The AI tool outputs a nucleus and a membrane mask in the form of arrays or tensors, which are converted to an image and saved to disk. For a
WSI this process is only performed once. However, for a TMA it needs to be performed for each tissue core.

- **Extraction**: The nucleus mask image, the membrane mask image and each of the protein channel images are loaded into either Python or MATLAB. Then, an image processing library extracts from the masks the indexes for each pixel, which makes up the nucleus and membrane of a cell. The morphological features are extracted solely from the masks (using an image processing library to get the perimeter, etc.), the proteomic features are extracted by using the pixel indexes from the masks as a binary gate on the protein channel images before getting the mean value of the non-zero pixels or similar.

From these descriptions, it is evident that the researchers must become proficient in several IT respects in order to undertake the analysis workflow. Programming languages such as Python, R or MATLAB are used for image manipulation such as cropping and splitting. In some cases, desktop applications such as QUPath [3], CellProfiler [4] or ImageJ [5] are used to visually inspect the full-resolution multi-page TIFFs to identify adjustment parameters when needed. The majority of the AI tools are Python-based; however, they often have complex dependencies and require their own execution environments to run, e.g., python virtual environments, conda environments, or containerisation via Docker. This complicates their installation and deployment. Altogether, this illustrates why many researchers are unable to execute such a workflow.

Even when researchers get all the various components to work on their machine and are sufficiently proficient in programming to write the code to perform the computational processing, analysis workflows conducted in this setting are difficult to reproduce and data provenance is difficult to track. However, both are essential for reproducible science. Additional traps are the inefficient code (in space and time complexity) written by those who are not experts in code optimization, an advanced skill that is typically not taught outside computer science courses. This situation leads to longer running times and memory issues, wasting resources and, in the worst case, the inability to complete the computations. Well-optimized code can drastically speed up the processing time and make more efficient use of resources.

### 1.2. Background and Related Work

We consider the perspectives of tools for workflow management and execution, model-driven development and, in particular, low-code/no-code approaches.

#### 1.2.1. Workflow Management and Execution Tools

Computational analysis of biomedical imaging data is typically driven by open-source tools developed in research labs around the world. These tools are typically designed to perform a single analysis task (such as cell segmentation, de-arraying, etc.). Chaining such tools together results in the creation of sophisticated analysis workflows, enabling biomedical researchers to glean new insights from their experimental data. In the past, custom scripts or Make Files were used to chain together computational tasks into pipelines [6]. However, this approach typically operates within a single computing environment, and the aforementioned open-source tools tend to have dependency conflicts, making it impossible for them to run from the same environment. Even in the cases where there are no conflicts, this approach does not lend itself to reproducible science due to variability in operating systems, computational resources, ambiguities with versions and poor documentation [6,7].

In order to address many of these issues, a variety of systems have been developed. These workflow management tools can be grouped into two types: graphical workflow managers, which require little to no programming knowledge, and textual workflow managers.

Examples of graphical workflow managers are Galaxy [8], Yabi [9] and KNIME [10]. Galaxy is a web-based application with several thousand tools available, that has a drag-and-drop interface for users to design and subsequently execute their workflow using the compute resources of one of three servers provided by the Galaxy Project. Yabi is a
web-based application which abstracts the complexity of workflow deployment on legacy HPC resources and data stores. KNIME is a desktop application for graphically building general machine learning and data science workflows. It is not specifically designed for biomedical imaging analysis but it contains generic tools which could be used in that context. An early example of this LCNC technology put into practice for executing analysis workflows in the health informatics domain is Bio-jETI, a modelling environment that enabled domain experts to graphically combine bio-informatics services to create arbitrarily complex executable workflows [11] without worrying about details of their interfaces, data type inconsistencies induced by service composition and, most importantly, without having to write any code. Further prior work successfully used low-code/no-code approaches to address workflows in bioinformatics [12], computational science and in education [13], paired with computational thinking. Those approaches share similar abstraction, encapsulation and coordination mechanisms to ours; however, their underlying tools were desktop or server-oriented, the system used Java for the low-code part and modelling was single-user.

Examples of textual workflow managers are Nextflow [14], Snakemake [15] among others [16–18]. Nextflow is a very expressive DSL and is designed for users who are experienced programmers. It breaks down each step of a workflow into modular components and connects these through channels that determine pipeline execution. Each component is the process code and any dependencies it may have. Nextflow supports a variety of dependency managers (Conda, Spack, Docker, etc.) and can be deployed locally, in the cloud or in an High Performance Computing (HPC) environment. Snakemake is interoperable with any installed tool or available web service with well-defined input and output file formats. Similarly to Nextflow, it can be deployed locally or on a cluster.

There are also several programming language-specific workflow managers, namely SciPipe [19] and SciLuigi [20]. SciPipe is a library in the Go programming language based on Flow-based Programming [21] principles. It enables users to define a workflow that contains a set of processes (with each process, in essence, being a shell command that has a set of inputs and outputs), and then the user defines the data dependencies between processes and executes. SciLuigi is a Python library that, in essence, is a wrapper around the Luigi workflow management tool but tailored for use in biomedical imaging applications. The library is based on object-oriented programming [22] and enables the user to define a “Workflow Task” object that contains several “Task” objects in Python, connect the data dependencies between them and execute the workflow.

Such frameworks, in theory, enable users to chain together services to create a workflow. They provide a single language where the user can orchestrate several different services with data passed from one to the next. However, none of them meet the requirements necessary to solve the problem which we are addressing with this work.

1.2.2. Model-Driven LCNC

In software engineering, disparate mindsets of participating stakeholders have been identified as a primary cause for the so-called semantic gap [23]. This term refers to the discrepancy that exists between varying interpretations and understandings related to different representations and conceptions [24,25], mostly between technical and non-technical stakeholders. To bridge this semantic gap, Language-Driven Engineering (LDE) [26] has been conceived as a solution that addresses and accommodates the different mindsets of the participants in application and system development. Based on the general concepts of MDD, the LDE approach specifically addresses those non-technical stakeholders. The goal is to provide tailored modelling languages that match their particular needs and capabilities [27] and, by doing so, enhance their personal modelling experience [28]. LDE is an evolution of Extreme Model-Driven Design (XMDD) and conceptually aligns with the One Thing Approach (OTA) that advocates for a global, consistent model at the heart of a development endeavour.

The model design steps are supported by providing the users with visual interfaces tailored towards their mindsets, in particular using (a) terminology from the application
domain and (b) a graphical, simple and intuitive approach to designing the processes and workflows. This combination enables such users to actively participate in the co-design process by process logic orchestration activities in a no-code manner. This way, users can focus on what the application should be doing rather than how they are going to implement it in code. Using graphical abstractions, the developer models the system and, subsequently, automatic model-to-code generators are used to transform the models to executable code [29].

Following these principles, the Java Application Building Center (jABC) framework [30] based the design and development of applications on Lightweight Process Coordination (LPC) and formal models. jABC accelerated the development cycle of applications involving subject matter experts to directly contribute by modelling the essential workflows, leveraging the concept of reusable components that represent native executable services, called Service-Independent Building Blocks (SIBs), orchestrated into analysable control structures that define the business logic, called Service Logic Graphs (SLGs). The DyWA system [31] combined online aspects, like data structure definition and application execution, with offline aspects in jABC (automated SIB generation for the DyWA data structures, process logic definition and code generation to the DyWA execution platform). This was an intermediate approach before developing DIME [32], the most ambitious modelling environment developed with CINCO, our meta tooling suite for developing domain-specific, graphical languages.

CINCO [33] is an Eclipse-based workbench for graphical modelling that follows the LDE paradigm. It allows users to easily create their own graphical languages and associated modelling environments. While CINCO is an Eclipse-based application, CINCO Cloud [34] is fully browser-based. Like CINCO, it follows the LDE mindset and allows language developers to easily create graphical languages and associated Integrated Modeling Environment (IME)s. In addition, CINCO Cloud runs as a distributed web application, creating new workspaces on potentially new computing resources for each concurrent user. To accomplish this, CINCO Cloud was developed using Kubernetes [35] to enable state-of-the-art cloud computing capabilities.

CdB is based on CINCO Cloud [34]. It is a distributed web application that functions as a fully functional, collaborative IME. Correspondingly, CdB is a domain-specific modelling environment that allows biologists to collaboratively model their workflows in their mindset, upload their data and execute their workflows from the web-browser. In comparison with the solution proposed in this paper, earlier technologies up to the jABC did not adequately support the data modelling, even DyWA was not really integrated, and up to DIME (DIME) inclusive, they did not run in the cloud. Effectively, every user needed to install Eclipse and Java, and collaborative work needed complex project versioning.

2. Material and Methods: CINCO de Bio

The CdB system is a new CINCO-based platform developed to support the modelling and execution of AI-Intensive Analysis (AIIA) workflows in the biomedical imaging domain. CdB is designed following the SOA and MDD approach. The life-cycle of an analysis workflow, seen as a software application, has the traditional design phase and run-time phase. Accordingly, the system architecture shown in Figure 2 includes services for the four core functionalities: modelling workflows (design time), executing workflows (run time), and data management: data processing (at run time), and data storage. These services are deployed as microservices that are loosely coupled and communicate over web protocols.

In the following, Section 2.1 explains how CdB supports users in modelling their analysis workflow, and Section 2.3 explains how CdB handles the runtime phase of a workflow. Both descriptions refer to the specific lens of the application domain-specific language (A-DSL) that was created for the motivating use case.
2.1. Designing a Workflow in CdB

A user modelling a workflow using the CdB IME is carrying out the activity A in Figure 2. The CdB is an IME designed as a CINCO product and developed and deployed via CINCO Cloud.

The Application Domain DSL of the Case Study in the CdB

The A-DSL developed for our motivating case study is summarized in Figure 3 in the form of an exemplary palette of SIBs. We see here an organization of the collection of SIBs designed for the application of the case study.

Figure 2. High-level overview of the CINCO de Bio System architecture. Letters on edges indicate processes that involve the front-end, numbers indicate back-end only processes.

Figure 3. Exemplary SIB palette for the A-DSL for processing highly-plexed immunofluorescence images. The SIB palette hierarchically stores SIBs by SIB type (here, Automated) and the TMA type (TissueMicroArray), and then hierarchically the Domain-Specific Language (DSL) they belong to. As seen in the expanded DSL, the 4 SIBs (CellPose-TMA, CellSeg-TMA, Deepcell-TMA and FeatureNet-TMA) belong to the DSL for automated cell segmentation operating on TMA files.

As we see in Figure 3, the A-DSL is a collection of facet or topic-specific DSLs: here, Feature Extraction, Cell Segmentation and Dearraying are the domain-specific aspects of
relevance. This modelling is consistent with the fact that these are the three main phases of the analysis workflow and they are a natural organization lens for the vocabulary and the operations in the workflow. Additionally, we see that there are other top-level facets, in this case Automated and Interactive, that define additional lenses through which to organize the operations populating this A-DSL.

Each of these DSLs is a collection of SIBs, i.e., fully implemented reusable functionalities tailored to a specific aspect of the overall application domain. Together, they deliver the required functionality for the target application domain. An SIB represents, in our case, a single data processing service. Its description in the Service Taxonomy is the collection of properties on any path from the Service root to it. This enables a semantic, property-oriented description, filtering, and addressing of sets of SIBs. When modelling their workflow, the domain users select the relevant SIBs from the SIB palette. The domain users place their selected SIBs from the SIB palette onto the CdB IME’s canvas in a drag-n-drop manner. This can be seen in Figure 4, showing a simple workflow for de-arraying a TMA.

![Figure 4. An annotated example of a simple workflow modelled in the CdB IME illustrating how both control and data flow are modelled. It is accompanied by concretised examples of data as it progresses through the workflow and the front-end components of Interactive SIBs.](image)

There, the two SIB types are visually distinguishable: the Interactive SIBs has a human silhouette icon, while the Automated SIBs has a gear symbol. The users then connect the outgoing branches of the SIBs, defining the control flow, which is represented by thick dark arrows. The light grey edges denote the data flow so that it is clear where outputs go and inputs come from.

While the user is modelling their workflow, syntactic and static semantics checks are proactively applied by the IME (supported by the language server). These checks ensure
that the current version of the workflow model is valid. If there are violations, the IME displays an error message to the user.

### 2.2. Interactive vs. Automated SIBs

The CdB Programming Language Domain-Specific Language (PL-DSL) supports two SIB types: **Automated SIBs** and **Interactive SIBs**:

- **An automated SIB** pertains to a data processing service which takes some input, follows some set of predefined steps and produces the output *without requiring human intervention at any point*.

- **An interactive SIB** instead pertains to a service that takes some input and follows some set of predefined steps. However, at a certain point in the processing journey, the service will *require a person to provide some input*, blocking the execution. Once that input is available, the service resumes and completes the remaining processing steps, resulting in the output.

While deterministic algorithms can be automated, certain services may need human expertise to refine outputs or validate solutions, particularly in AI algorithms due to their stochastic nature. In workflows, computations should only proceed when the intermediate quality is acceptable to an expert. A concrete example is illustrated in Figure 4: SegArray is an automated SIB for a Machine Learning (ML)-based algorithm, which performs de-arraying. As the underlying ML model’s performance varies across different types of tissue, it is possible that the predicted ROIs are not correct. The Validate Predicted ROIs SIB mitigates the negative outcome by generating a window that displays the nuclear stain channel superimposed with the predicted ROIs. The user can then visually inspect the predicted ROIs and decide whether they are valid or not.

Besides deciding the control flow, interactive SIB serve to provide additional user input. **Edit ROIs** in Figure 4, is an example of an interactive service opening a front-end where the user can manually edit a set of ROIs. We believe that there are many more cases where interactive services occur; therefore, we have included adequate support mechanisms in the IME.

A critical design decision of the A-DSL is the data model. As can be seen in Figure 4, the input and output types of the SIBs are not generic or primitive types. Instead, they are **semantic types**, which capture concepts that are familiar to domain users. This familiarity eases the learning curve and facilitates the semantic validation of models. Taking as an example the motivational use-case (highly-plexed tissue imaging), the output from an experiment run by a biologist is either a WSI or a TMA. Ignoring domain knowledge and looking at this data from a purely syntactic perspective, those outputs are identical: both WSI and TMA are stored as multi-page TIFF files (The actual file extensions vary depending on whether it is an Open Microscopy Environment (OME) format or a proprietary format from the imaging equipment provider, but they are all derivatives of the TIFF format); however, the types of processing that it makes sense to apply to WSIs and TMAs are different, and data structures as a result of processing steps differ.

Figure 5 shows an excerpt of the *taxonomy for the semantic data model* for the A-DSL for the motivational use-case in the form of a Directed Acyclic Graph (DAG). In Figure 5,

- The dotted lines show the definition of WSI and TMA data types outside of CdB: these are both multi-page TIFF files.

- The solid lines denote the type definition of the data within the CdB environment. A TMA is a hash map of **TMA Protein Channels**, where **TMA Protein Channel** is a file-pointer to a single page TIFF, whereas a WSI is a hash map **WSI Protein Channel** and whereby the **WSI Protein Channel** is also a file-pointer to a single page TIFF.

Therefore, the two are distinct and distinguishable in the type system and this information is used in determining the appropriate processing. For example, in highly-plexed tissue image processing, it never makes sense to apply a de-arraying algorithm to a WSI. Similarly, it never makes sense to apply cell segmentation to a TMA before applying a
dearraying algorithm. The use of semantic data typing facilitates the tactic and static semantics checkers to identify such errors at the design stage. Thus mitigating error, which results in a waste of energy, potentially of space and computation fees, and of human time.

Once the user has modelled a valid workflow and wishes to execute it, the workflow model is passed to execution Application Programming Interface (API). The PL-DSL describes both the control flow and data flow, so the workflow model is already in the correct format to be automatically translated to the imperative programming language used to execute the workflow. Upon successful request, the runtime phase of the workflow is scheduled, and the execution API returns to the user the URL of the execution front-end. The execution front end for a given workflow keeps the user updated on the progress of the workflow during execution and notifies them of any interactive tasks they must perform.

![Data Model Taxonomy](image)

**Figure 5.** An excerpt of the data model taxonomy for the highly-plexed tissue image analysis A-DSL. It classifies the application domain-specific data types in terms of their syntactic data types (as in computer science data structures and formats) as well as the semantic data types, expressing their meaning in the application domain. We also distinguish atomic and non-atomic data types. An atomic type is a single irreducible piece of data (in the context of the given application domain). For example, a Tissue Core Protein Channel equates to a single-page tiff (greyscale image) file. A non-atomic type is a data structure that acts as a collection containing atomic or non-atomic types. For further illustration, we have overlayed a concrete example of the data structure corresponding to a De-arrayed Tissue Micro Array (DTMA).

### 2.3. Workflow Runtime in CdB

The run-time phase of a workflow is supported in the CdB system by several components: the Execution API, the Model-to-Code Transformer, the Execution Runtime Environment, the Data Processing API, the Job Management System (JMS), the Data Processing Services and the Data Storage. The steps of which are denoted in Figure 2. However, in essence, the components that make up the core CINCO de Bio (CDB) architecture work in tandem to do the following:

- Transform the workflow model to a program in an imperative programming language which can be executed and orchestrate the workflow.
The workflow orchestration program submits a data processing job (i.e., one step of a workflow) to be scheduled by the job management system.

When resources are available to do so, the data processing job is then running, retrieving the data to be processed from the data storage API and subsequently writing the intermediate results to the data storage API for retrieval by subsequent data processing jobs.

The job management system via the executing API notifies the workflow orchestration program that a data processing job is complete so that it can retrieve the results and proceed to the next step. The execution API simultaneously updates the execution front-end to keep the user informed of the workflow status (or retrieving the final results upon completion of the workflow).

In the case of interactive services, the data processing job first renders a front-end, which is made available via the data processing API, the URL of which is then presented to the user via execution front-end where they are re-directed to the service front-end to undertake the required task. The results of which are then submitted via the service API, and from that point, the system handles it as if it were a regular automated data processing job.

3. Results and Discussion

We present the results of our work along a case study that illustrates how a domain expert would use CdB to design and execute a pre-processing workflow for the proteomic and spatial analysis of a highly-plexed tissue image, using the A-DSL developed for that use case in Section 3.1. We also present a comparison of the CdB tool to similar tools we encountered in the literature in Section 3.2.

3.1. Executing the Analysis Workflow with CdB

Considering the same analysis workflow described in Section 1.1, we execute it now in CdB deployed on a server on the researcher’s local network.

1. **Handling Raw Data**: The raw data are uploaded from the experimental machine to the CdB data storage via the CdB upload portal.

2. **Modeling**: Using CdB’s IME, the researcher models the workflow using the graphical DSL, as seen in the simple sequential workflow in Figure 6. The *Initiate TissueMicroArray* SIB handles the initial data preparation phase of the workflow. It is an interactive service: the user selects the data on which the workflow will execute and inputs some other parameters. The *SegArray*, *Edit ROIs* and *Crop Tissue Cores* SIBs handle the de-arraying phase. The *Deepcell (TMA)* SIB conducts the cell segmentation, and finally, the *Extraction Spatial and Proteomic Features (TMA)* SIB handles the feature extraction.

3. **Executing**: Once the modelled workflow is finished and checked as valid, i.e., it passes all the syntactic and static semantic checks of the CdB IME, the user simply hits the execute workflow button. The user is then redirected to the execution front-end to monitor the status of the workflow execution.

   - The execution runtime handles the orchestration of the various data processing services which comprise the workflow. Passing the appropriate data to each service to undertake their processing. Updating the status on the execution front-end for the user to monitor the status of their workflow.

   - In the case of interactive services such as “*Initiate TissueMicroArray*” and “*Edit ROIs*”, the user is presented a redirect URL via the execution front-end, which launches the interaction front-end for the respective services where they can enter the required additional input.

   - Upon workflow completion, the execution front end presents the user with the URL(s) to retrieve the output from the workflow, with the options to also retrieve the results from the intermediate processing steps.
Figure 6. TMA pre-processing and feature extraction workflow modelled in CdB.
3.2. Comparison to Similar Tools

3.2.1. Criteria

Our active engagement with various stakeholders throughout the requirements engineering process resulted in the following set of criteria that a tool needs to satisfy in order to meet the functionality and usability requirements of these users.

- **Learning Curve:** Considering users with no prior knowledge of programming, how much IT knowledge would they need to acquire in order to use the tool proficiently to design and execute their own workflow? A *Low* learning curve indicates a system that is very intuitive to the user without specialized IT knowledge, e.g., just being able to use a browser and a mouse; a *Medium* learning curve indicates that the system is relatively intuitive to use but the user will have to learn some IT or programming concepts to use the tool effectively; a *High* learning curve denotes that the user will essentially have to learn to write code in order to use the tool.

- **DSL Type:** Does the user model the workflows within the system via a graphical DSL or a textual DSL? This is synonymous with whether the system reduces the amount of code, which needs to be written (low-code) or eliminates coding altogether (no-code).

- **Control/Data Flow:** This criterion addresses the level of expressiveness of the PL-DSL for modelling within the system. Can the user model both the control flow and the data flow of the workflow, or just the data flow?

- **Semantic Typing:** This criterion concerns the data modelling facilities offered by the tool: is the data model composed of solely primitive and generic data types, or are domain-specific concepts captured in the data model? The rationale behind this is that a semantically typed modelling language presents concepts directly familiar to the user (i.e., lowers the learning curve) and secondly, it facilitates semantic data-flow analysis for compatibility checks of types that would be syntactically indistinguishable.

- **Design Validation:** This criterion captures the level of checks that the system applies to the user-defined workflows before submitting them to be executed. *Extensive* refers to both syntactic and static semantic checks, *limited* refers to simple checks to ensure that there are no cycles in the workflow, etc.

- **Interactive Services:** This criterion pertains to how well (if at all) a system caters for services with user interaction at run-time.

- **Deployment:** This criterion covers how and where the system can be deployed. For example, is it a desktop application or a web application that can be deployed on the cloud? The deployment options have knock-on effects for the system scalability, support of interaction and collaboration and ease of set-up.

3.2.2. Comparison

Table 1 outlines how the various tools in the literature measure with respect to the criteria described in Section 3.2.1. It is followed by a brief description of each tool with comments.

- **Galaxy:** [https://usegalaxy.eu/](https://usegalaxy.eu/) [36] (accessed on 1 August 2024) is a browser-based workbench for scientific computing that supports the execution of biomedical analysis workflows through a graphical LCNC manner. Galaxy has a mature ecosystem that supports an extensive number of use cases. It has a unified front end with high flexibility over the execution back end. However, the Galaxy modelling language is data-flow only; as AI use increases in this context, modelling control flow will be essential when deciding whether AI can be used to automate decisions (i.e., which direction a workflow should take) or for use cases similar to the one we described in Section 2.1. Galaxy provides only minimal support for interactive tools, and, as described in their own documentation, integrating new interactive tools into Galaxy is a non-trivial task. Finally, Galaxy uses generic typing (json, list, TIFF, etc.), so it does not capture data concepts familiar to a user’s domain, and it permits invalid workflows to execute. For example, a list of images and a list of text are treated the
same in both cases: the type is a list, design time. Galaxy does not prevent a user from inputting a list of textual data into a service that expects a list of images. There is accordingly a significant burden of knowledge and checks on the users, who need training and carefulness at every step.

- **KNIME**: [https://hub.knime.com/](https://hub.knime.com/) [37] (accessed on 1 August 2024) is a desktop application for the execution of data analytics workflows. It supports users to model workflows graphically with both control and data flow, where the control flow is modelled using control nodes for if-else branching, etc. Third-party services, which support Representational State Transfer (REST) communication, can be integrated into applications by users. The tools make use of a generic data model and do not natively capture any concepts specific to the biomedical domain. Owing to the fact that it is a desktop application it lacks scalability. It has an extension to work with Apache Spark; however, that would require the user to deploy a spark cluster separately before linking it to KNIME. The KNIME Server version of the tool has a free community version, but the vast majority of the functionality is reserved for the paid version and, therefore, is not discussed here.

- **QuPath**: [https://qupath.github.io/](https://qupath.github.io/) (accessed on 1 August 2024) is an open-source desktop application designed for digital pathology, specifically for the analysis of TMAs and WSIs. It is primarily an image viewer but offers limited support for data processing and executing workflows. A user can use a variety of built-in tools to process an image, these then show up in a command history widget, and that history can be exported as a script to re-execute it as an automated workflow. Even though many tools support interaction (i.e., manual annotation, etc.) it is not recommended by the developers to use such tools in a workflow with automated tools. Given that QuPath is a desktop application, it has limited scalability. Also, due to it being designed from the ground up, specifically for digital pathology, the system cannot be easily repurposed for processing/analysis of other biomedical images. It also does not support the integration of many of the state-of-the-art open-source tools.

- **CellProfiler**: [https://cellprofiler.org/](https://cellprofiler.org/) [38] (accessed on 1 August 2024) is a desktop application for processing Biomedical Images. The interaction is mainly conducted through a Graphical User Interface (GUI). To define a workflow (or pipeline, as it is called in the application), users drag-n-drop modules such as *Crop Image*, then input parameters for each module using forms displayed in the application. The workflow is data-flow only and limited checks are applied to the workflow before it can be executed. The data model does not include semantic types as it is tailored for general biomedical image processing. It does not support any interactive services as a part of a workflow. Being a desktop application, it is limited in terms of scalability. It also does not support the integration of many of the state-of-the-art open-source tools.

- **NextFlow**: [https://www.nextflow.io/](https://www.nextflow.io/) (accessed on 1 August 2024) [14] is a Groovy-based textual DSL and workflow management system for the creation and execution of data analysis workflows. It is highly expressive and supports a variety of runtime environments for services, like Docker, Shifter [39], Singularity [40], Podman and Conda. Whilst Nextflow eases the deployment of workflows, the technical expertise required to use it is far beyond what could be expected from a typical domain researcher.

- **SnakeMake**: [https://snakemake.github.io/](https://snakemake.github.io/) (accessed on 1 August 2024) [15] is a Python-based textual DSL and workflow management system for the creation and execution of data analysis workflows. It is highly expressive and supports multiple runtime environments for service execution (Singularity and Conda). Similar to Nextflow, while SnakeMake eases the deployment of workflows, the technical expertise required to use it is far beyond what could be expected from a typical domain researcher.

- **CdB**: In comparison, **CdB** is superior in many comparison dimensions and equivalent in others: it offers a powerful modelling environment that covers both dataflow and control flow with associated syntactic correctness checks and it supports semantic types that are intuitive for the domain experts, with associated semantic compatibility.
checks. With the purely graphical modelling environment, it has a low learning curve, especially if connected with a cloud-based hosting, which eliminates the need for installation. The graphical models are stored in a JSON format, so they are actually also available in a textual form. The support for interactive services is a core requirement of CdB, so, in fact, we designed the execution environment in such a way as to maximally simplify this capability. Finally, the entire environment is designed for future growth, with ease of integration and extension in mind.

**Table 1.** This table shows a comparison between CdB and similar tools found in the literature. The criteria are learning curve (low, medium, high), visual interface (textual, graphical), expressiveness (control flow, data flow), semantic typing (yes or no), design validation, interactive services (no support, limited support, extensive support), deployment (desktop application, local or cloud).

<table>
<thead>
<tr>
<th>Platform</th>
<th>Learning Curve</th>
<th>DSL Type</th>
<th>Expressiveness</th>
<th>Semantic Typing</th>
<th>Design Validation</th>
<th>Interactive Services</th>
<th>Deployment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cinco de Bio</td>
<td>Low</td>
<td>Graphical</td>
<td>Control and Data Flow</td>
<td>Yes</td>
<td>Extensive</td>
<td>Extensive</td>
<td>Local/Cloud</td>
</tr>
<tr>
<td>Galaxy</td>
<td>Medium</td>
<td>Graphical</td>
<td>Data Flow</td>
<td>No</td>
<td>Limited</td>
<td>Limited</td>
<td>Local/Cloud</td>
</tr>
<tr>
<td>Knime</td>
<td>Medium</td>
<td>Graphical</td>
<td>Control and Data Flow</td>
<td>No</td>
<td>Limited</td>
<td>Limited</td>
<td>Desktop</td>
</tr>
<tr>
<td>Qupath</td>
<td>Medium</td>
<td>Graphical and Textual</td>
<td>Data Flow</td>
<td>Yes</td>
<td>Limited</td>
<td>Limited</td>
<td>Desktop</td>
</tr>
<tr>
<td>CellProfiler</td>
<td>Medium</td>
<td>Graphical</td>
<td>Data Flow</td>
<td>No</td>
<td>Limited</td>
<td>No</td>
<td>Desktop</td>
</tr>
<tr>
<td>Nextflow</td>
<td>High</td>
<td>Textual</td>
<td>Data and Control Flow</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Local/Cloud</td>
</tr>
<tr>
<td>Snakemake</td>
<td>High</td>
<td>Textual</td>
<td>Data and Control Flow</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Local/Cloud</td>
</tr>
</tbody>
</table>

4. Conclusions and Future Work

We have presented CdB, a web-based, collaborative LCNC platform for the design and execution of AI-driven biomedical image processing and analysis workflows, which is designed for ease—a necessary requirement for wide adoption:

- Easy access: installation-free, ubiquitous, access via a Web Browser;
- Easy use: domain-specific LCNC modelling support that requires no technical knowledge.

The platform is built following model-driven engineering principles and aims to facilitate biology domain researchers to leverage the value that AI and other computational processing methods provide in deriving knowledge from experimental data. We have outlined the system architecture, the PL-DSL and the A-DSL, and illustrated its use in our motivating use case. The tool and the example were co-designed with the respective domain experts in order to maximise usability and extensibility.

The CdB modelling services and the services that deliver the runtime execution of workflows play a crucial role in streamlining biomedical image processing workflows: they reduce both the time and the expertise required. Hence, CdB paves the way to more accessible, efficient and reproducible biomedical imaging research. The primary role of the modelling environment is to allow users to define and modify process models using a visual modelling language. This functionality enables researchers to tailor image-processing workflows to fit their own specific requirements without needing advanced programming knowledge. This is particularly important when designing new workflows or or modifying existing ones for extension or re-purposing; in such cases, the semantic support is particularly useful, as as turn-around time for modifications is short when it is most needed when exploring new solutions. An intuitive, user-friendly interface allows users to visually compose their process models, drag-and-drop components and configure
parameters, thereby reducing the complexity and learning curve, as associated with process model creation in a no-code approach.

We have presented a detailed case study where we canvassed domain researchers who currently apply analysis workflows to highly-plexed immunofluorescence tissue images to discover the tools and methodology currently in use. We then compared their current activity to modelling and executing an equivalent workflow in CdB. Overall, CdB represents a significant step forward in enabling biologists to leverage computational analysis tools effectively, democratizing the use of advanced image processing techniques, and potentially increasing the productivity of researchers working in this domain. We will continue to develop prototypes and conduct usability tests to obtain feedback on whether the modelling language and services we are providing are suiting the users’ needs. Such an iterative approach allows us to refine the platform on the basis of real user feedback and to continuously improve the solution in an agile manner.

A promising avenue for future exploration involves evaluating the applicability of workflow synthesis in CdB. Building on the concept of workflow design with loose specifications, workflow synthesis was implemented in PROPHETS [41], a framework for the synthesis of processes from a collection of services. PROPHETS has shown potential advantages in terms of automating the creation of scientific workflows based on logical specifications in the context of the jABC and Bio-jETI platforms [42]. Such a synthesis technique could be particularly useful in biomedical research, where workflow design may become more complex and intricate with an increasing number of available services.

Author Contributions: Conceptualization, C.B., E.F. and T.M.; methodology, C.B. and S.B.; software, C.B., S.B., S.R. and A.S. (Amandeep Singh); validation, C.B., S.B. and A.S. (Amandeep Singh); data curation, C.B., E.F. and S.B.; writing—original draft preparation, C.B. and S.B.; writing—review and editing, T.M., A.S. (Alexander Schieweck) and B.S.; visualization, C.B. and S.B.; supervision, T.M. and B.S.; project administration, T.M. and B.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work was conducted with the financial support of Science Foundation Ireland (SFI) under grants number 21/SPP/9979 (R@ISE), 13/RC/2094-1 (Lero, the Software Research Centre) and 18/CRT/6223 574 (SFI Centre of Research Training in AI) as well as University of Limerick Health Research Institute ULCan grant Pillar 4. For the purpose of Open Access, the authors have applied a CC BY public copyright license to any Author Accepted Manuscript version arising from this submission.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Conflicts of Interest: The authors declare no conflicts of interest.

References


22. Rentsch, T. Object oriented programming. ACM Sigplan Not. 1982, 17, 51–57. [CrossRef]


24. Dorai, C.; Venkatesh, S. Bridging the semantic gap with computational media aesthetics. IEEE MultiMedia 2003, 10, 15–17. [CrossRef]


Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.