A DSL for Fluorescence Microscopy*

Birthe van den Berg[1][0000−0002−0088−9546],
Peter Dedecker[2][0000−0002−1882−2075], and
Tom Schrijvers[3][0000−0001−8771−5559]

[1] *Student*, Department of Computer Science, KU Leuven, Belgium
birthe.vandenberg@kuleuven.be
[2] Department of Chemistry, KU Leuven, Belgium
[3] Department of Computer Science, KU Leuven, Belgium

Abstract. Experiment-driven observation of the natural world is a key driver of scientific discovery and innovation. Yet, the growing technical complexity of experimental equipment has become a pressing problem that limits the number and diversity of experiments that can be performed, as well as the institutes and enterprises that can afford them. This project identifies an opportunity to increase the rate of discovery and innovation by lowering the threshold for experimentation, exemplified in the applications of fluorescence microscopy, a key approach in the life sciences. Our solution is an operational, deeply embedded domain-specific language (DSL) in Haskell. We aim to tackle this problem further by advancing the state-of-the-art in programming languages and knowledge representation to automate the equipment control and allow users to state *what* experiments they want, rather than *how* to perform them.

Keywords: DSL · Haskell · Fluorescence Microscopy · Instrument Control · Data Acquisition

1 Introduction

Since at least the 17th century, the scientific method, based on observations of the natural world, has been the driving force behind most of our knowledge discovery and innovation. In fact, in today's knowledge economy, experimentation-driven research is the basic modus operandi by which new knowledge and applications are generated. A prominent example is fluorescence microscopy, a true workhorse technique in the life sciences. It is essential for unravelling the inner workings (architecture, dynamics, interactions) of cells and tissues and driving the technological development in many fields of industry and research, including the (bio-)medical, pharmaceutical, agricultural, food industry and so on. Fluorescence microscopy beats other microscopy methods in terms of its superior spatiotemporal resolution, its non-invasiveness and the simple sample preparation. This paper describes the recently initiated project of applying functional programming, and more specifically a deeply embedded DSL in Haskell, to the

* Project Article

field of fluorescence microscopy. Moreover, the goals, objectives and approaches on how to extend this project are discussed in detail.

1.1 Fluorescence Microscopy

Fluorescence is the emission of a photon by a molecule that is in the electronically excited state. A Jablonski diagram [11] (Figure 1), named after the Polish physicist Aleksander Jabłoński, shows how a substance at the ground state absorbs light in the form of a photon and transitions to a higher energy level. Non-radiative transitions, such as vibrational relaxation or internal conversions, cause the molecule to relax to the lowest vibrational level of the excited state. The emission of a photon when returning to the ground state is known as fluorescence.

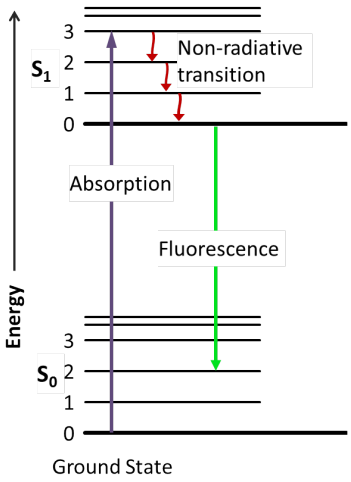

Fig. 1. A Jablonski diagram.

The absorbed and emitted light typically have different wavelengths, which makes it possible to detect only the emission light using special filtering. In order to use fluorescence for doing research on cells and tissues, fluorescent proteins or labels are required. A wide variety of fluorophores is available, including fluorescent proteins, organic dyes, and semiconductor particles. A good example is the Green Fluorescent Protein (GFP) [1,16,17], discovered and developed by M. Chalfie, O. Shimomura and R. Tsien, who won the 2008 Nobel Prize in Chemistry.

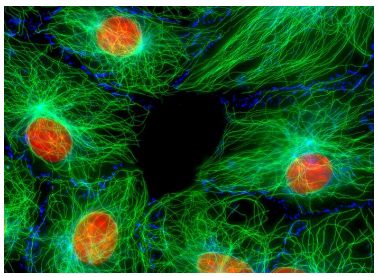

Fig. 2. Fluorescence on Mouse Embryonic Fibroblasts, Olympus Bioscapes 2007, Dr. Jan Schoranzer.

Fluorescence Microscopy is a popular and fast-moving field in the domain of biochemistry. It is so popular due to its selectiveness and non-invasiveness: experiments can be performed in situ and in vivo. Sample preparation is simple and fluorescent probes are widely available. These and other factors (optical transparency, spatiotemporal resolution, ...) make fluorescence microscopy a widely used imaging technique for doing research on cells and tissues. Figure 2 shows an example of an image taken by a fluorescence microscope, more specifically on mouse embryonic fibroblasts, from the bioscapes 2007 gallery of Dr. Jan Schmoranzer[4].

[4] `https://www.olympus-lifescience.com/en/bioscapes/authors/jan-schmoranzer/`

The general setup of doing fluorescence microscopy consists of multiple hardware elements. First of all, a *light source* (e.g., high-power LEDs or lasers) illuminates a specimen with light of a specific wavelength. This light is absorbed by the fluorophores, which then emit light of lower energy and thus a longer wavelength and another colour. The emitted light is caught by *detector*, often a camera. *Filters* filter out specific wavelength from the light of the light source, allowing only fluorescence to pass to the detector. *Dichroic mirrors* pass the light of a specific wavelength and reflect the light of other wavelengths.

1.2 Problem Statement

The field of fluorescence microscopy evolves very fast and innovations in fluorescence imaging continue to be made at a rapid pace, evidenced by multiple Nobel Prizes, e.g., the 2014 Nobel Prize in Chemistry for the development of super-resolved fluorescence microscopy. The boundaries of knowledge are pushed further and further requiring functionality and capabilities to evolve at the same speed. Instrument manufacturers such as Nikon, Olympus, Zeiss, Leica and more, sell equipment that costs between a few hundred thousand to over a million euro. Consequently, further growth in experimentation is impeded by the costly manual labor and growing technical complexity involved in setting up new experiments and observing them. Dedicated staff members are required to operate advanced instruments, both to utilize these systems to their full potential and to guard against erroneous measurements or interpretations. Overall, it easily takes months to years of practice before an operator becomes fully proficient with the imaging systems, limiting the type of organizations that can provide the necessary support to fully leverage the abilities of a particular instrument.

A number of different vendors and open-source initiatives provide software for the control of imaging systems, including fluorescence microscopes. Unfortunately, while these software solutions are useful for taking over basic low-level manual tasks, they come with a risk of stifling rather than enabling new forms of experiments. They do not adapt themselves dynamically to changing circumstances (e.g., burst events), the specific nature of the sample (e.g., motile cells), and relevant events (e.g., cell death). At the same time, the control is fairly low-level and still requires substantial technical expertise from their users, rather than offering a high-level declarative interface. Moreover, the software is typically tied to a particular hardware setup and exclusively compatible with hardware from a single vendor. As a result, advanced experimentation relies on highly trained and multi-purpose operators who are difficult to retain in many of the organizations performing innovative research, including small to medium enterprises and academic institutions.

Hence, there is an opportunity to increase the rate of discovery and innovations by lowering the effort and cost, and thus the threshold for implementation.

2 Towards a Haskell-based Solution

Since industrial setups are expensive, not modular and only take over low-level tasks, they are insufficient to meet the needs of the users. As a starting effort, we designed a software solution that allows the biochemistry researchers to construct arbitrarily complex experiment programs with any of the hardware configurations available in the lab. A high-level overview of the software architecture is shown in Figure 3. The core of the software is a measurement controller, implemented in Haskell. This controller has three important functions.

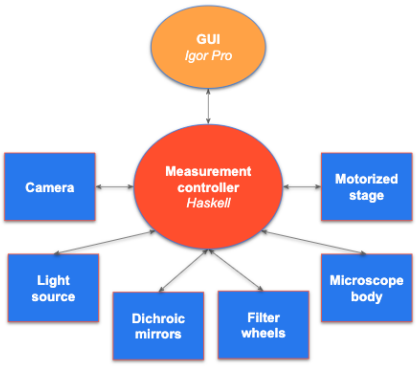

Fig. 3. High-level overview of our solution.

First of all, the controller represents and reasons over **domain-specific knowledge**, more specifically on how a user can construct a program. This domain-specific knowledge is represented by a deeply embedded domain-specific language in Haskell. A domain-specific language (DSL) [2,6,3] is a custom-designed specification or programming language that allows us to concisely formulate complex imaging tasks. A DSL is especially appropriate here as it considerably lowers the threshold to use it for life scientists, who are not familiar with programming but already highly familiar with the domain-specific concepts featured in the DSL. The embedding approach [9] lowers the development cost compared to a stand-alone implementation from scratch, as we leverage Haskells active research on this topic [7,8] and get inspired by many existing examples for other problem domains, such as music [10], financial engineering [13] or monadic constraint programming [15]. We choose a deep embedding in order to allow for extending the DSL with inspectors that take care of the safety of the experiments. For instance, the total light intensity that a sample is exposed to can be inspected and should remain limited in order to prevent the sample from photobleaching, which means permanently fading. Similarly, the time needed to perform an experiment can be inspected. This kind of inspections are examples of what we later refer to as 'safety and sanity checks'.

The following represents a simplified version of the syntax and semantics of the DSL.

```
data MeasurementElement = MEDetect
                        | MEWait Double
                        | MEIrradiate Double ( String , Double )
                        -- duration ( light source , power )
                        | MEDoTimes Int Prog
                        | MEStageLoop [ StagePosition ] Prog
```

```
type Prog = [ MeasurementElement ]
data StagePosition = StagePosition { x :: Double
                                   , y :: Double
                                   , z :: Double }
executeProg :: Prog -> IO ()
executeProg prog = foldMap executeME prog

executeME :: MeasurementElement -> IO ()
executeME  MEDetect =
        executeDetection
        >> putStrLn ("detecting...")
executeME (MEWait dur) =
        threadDelay (round $ dur * 1e6)
        >> putStrLn ("waiting...")
executeME (MEIrradiate dur params) =
        executeIrradiation dur params
        >> putStrLn ("irradiating...")
executeME (MEDoTimes n pr) =
        mapM_ (\prs -> executeProg prs) (take n . repeat $ pr)
        >> putStrLn ("times...")
executeME (MEStageLoop poss pr) =
        mapM_ (\pos -> setStagePosition pos >> executeProg pr) poss
        >> putStrLn ("stage looping...")
```

In contrast with already existing software, users can define arbitrarily complex programs using this syntax. Moreover, stage loops are provided, in which an operator can look at a sample at different user-defined positions in a user-defined order. Essential for this stage loop is a motorized stage that can move in different directions.

Secondly, the measurement controller is responsible for communicating with a **graphical user interface** (GUI). This interface is written in Igor Pro[5], scientific software developed by WaveMetrics Inc. that is especially suited for image processing and data acquisition. The GUI and measurement controller communicate via a socket sending serialized JSON objects. The GUI polls to see if data is available at the controller. In Igor Pro, the user can construct a measurement program using buttons provided in the interface. This way, operators can only construct syntactically well-formed programs.

Finally, where possible, the measurement controller uses serial communication via COM ports to control the **hardware**. Hardware consists of cameras, light sources, dichroic mirrors, filter wheels, microscope bodies, motorized stages and more. Often, manufacturers provide a DLL for the hardware, written in C. The measurement controller is able to operate with different hardware configurations, which makes the software solution modular to different microscope setups.

[5] https://www.wavemetrics.com/

As an example of how the entire software solution works, assume an operator uses the GUI in Igor Pro to construct the following program:

```
do 5 time(s) in total
        irradiate 2 s using MarcelLumencor:violet@15;
        wait 3 s
        acquire image(s)
wait 10 s
acquire image(s)
```

`"MarcelLumencor:violet"` is the name of the light source together with the colour of light that is emitted (in this case UV light), whereas @15 refers to 15%, the used laser power. This translates to the following program in the DSL:

```
prog :: Prog
prog = [ MEDoTimes 5 [ MEIrradiate 2 ("MarcelLumencor:violet" , 15)
                     , MEWait 3
                     , MEDetect ]
       , MEWait 10
       , MEDetect ]
```

When executing these instructions with the `executeProg`-function, the measurement controller operates the hardware by communicating with the COM port connected to, in this case, the specified light source and the filter wheel filtering the violet light.

3 Goal and Objectives

The goal of this project is to be on the vanguard in automation of experiments, to reduce their duration, cost and required technical expertise, while utilizing the hardware to its full potential. We will focus on fluorescence microscopy as a showcase of our work. Taking routine tasks out of the microscopist's hands and performing those tasks more efficiently and more consistently than currently done leads to increased productivity, especially compared to non-expert operators.

We aim to advance automation on two levels. On the one hand, we want to achieve automation to **optimize behaviour** by learning from previous experiments. On the other hand, there is a need to extend the class of automatable routine tasks with **adaptive behaviour**. These routine tasks currently only support performing a fixed set of low-level instructions. Adaptive behaviour supports the usage of observations made during the experiment to reason about which instructions are best executed next, resulting in better quality and relevance, with minimal manual oversight or intervention.

Clearly, automatic operation of the instruments is faster, and thus experiments can be conducted more quickly. Moreover, actions can be parallelized, improving experiment **performance**. Automated design can find more optimal ways of scheduling and executing steps in the experiments. This way, it enables

experiments that were previously impossible due to tight timing constraints. However, our research on the building blocks and adaptive reasoning techniques needed is more general in nature, and a wide range of opportunities are envisioned. Specifically, our objectives are to meet the following requirements that arise naturally from the problem domain.

- When automating experiments and leaving most decisions up to software or non-expert users, there is a danger that the hardware will be used in inappropriate ways. Hence, we will build **safety checks** into the backbone of the automation. These serve to ensure that the hardware is not pushed beyond its operational limits to avoid ruining expensive or time-intensive samples and damaging or wearing out the equipment. At a higher level, when the user instructs an experiment in terms of timing or other operational requirements, we will automatically carry out satisfiability checking to verify whether the requirements can be met by the available hardware given physical, biological or chemical constraints, and that the generated plans will never violate the safety conditions.
- The system should be **usable** for a wide range of users: non-experts who are aware of the general capabilities of the devices but not of how the specific microscopes (and other devices) are controlled, experts who want to finetune the software where needed, as well as technical users who want to control the microscope to set up experiments. Setting parameters and configurations is limited to a minimum since the software will take care of this. Procedures are abstracted over and users are only required to declare *what* they want to measure.
- The reasoning behind automated procedures and settings should be clear and **transparent** to users. Even when the system optimizes behaviour, using for instance smart planning of actions, the scientists remain responsible for the experiment. So they should get insights into *why* decisions were made by the system, whether alternatives were possible, and, if not, which (safety) constraints prevented alternatives.
- Laboratory technicians often require a specific hardware setup for a particular experiment. Switching between setups should require low effort, time and expertise in order to increase the throughput of experiments. It should also be possible to provide support for new devices (with new capabilities and operational limits) and their integration into an existing setup. To enable this, the software should allow **modular** extension with support for new hardware, without replacing or otherwise changing the existing software. This keeps the software development and maintenance costs at a minimum.

We will convincingly demonstrate the impact of our solution. We expect substantial new contributions in the area of fluorescence microscopy that critically rely on the advances in computer science made in this project.

4 Three-layer approach

The project consists of three essential and interactive layers, that contain tasks on different levels of abstraction. The software is designed in collaboration with the Lab for Nanobiology at the KU Leuven, which provides a direct feedback loop to test the computer scientific advances made during the project.

4.1 Fluorescence Microscopy Applications

The first layer encompasses selected applications of imaging and generates specific automation needs. The following use cases are representative of current challenges in the imaging community:

- *Tracking motile cells over time:* cells may move over time on their own accord within the petri dish, causing them to disappear from view. The system should be able to dynamically adjust to these motions.
  The software will need to intelligently scan the sample, interpret the image and change the execution.
- *Dynamic changes and rare events:* many biological processes are characterized by the occurrence of burst events (e.g., viral entry), alternated with long less-active periods. Many images should be captured during these burst events, but not during idle moments to avoid problematic phototoxicity.
  The software will need to intelligently scan the sample and decide when and where images must be acquired.
- *Examining cell death:* Cell death is a clear example of cellular heterogeneity: cells can die in different ways, at different times and in different conditions. Such heterogeneity is critical when treating specific cancers or autoimmune conditions since the cellular response can be vastly heterogeneous to different drugs.
  The software needs to intelligently scan the sample, decide when and where images must be acquired, and identify classes of cells based on their response in space and time, making sure that all classes are covered in the experimental acquisition.

4.2 DSLs

The second layer focuses on the development of domain-specific languages, in our case for describing fluorescence microscopy experiments and their design. Inherently, all research towards DSL design and the methodology developed as part of that are applicable beyond the scope of fluorescence microscopy as well.

On the one hand, we have developed an operational domain-specific programming language, as described in Section 2, able to control the hardware and software needed to perform the experiments, giving expert users very precise control over how experiments should be performed.

On the other hand, we develop a domain-specific knowledge representation language allowing users to specify at a higher level *what* needs to be done in a given experiment, without mentioning the details of *how* this should be achieved.

We also develop knowledge bases of background knowledge about operation and restrictions of the components involved in these experiments. By exploiting these knowledge bases, we give the system many options to optimize the behaviour of the experimental set-up, to reason about safety conditions, and even to suggest new experimental set-ups that might be better suited for the task at hand. We connect both by developing a methodology for translating plans obtained from reasoning based on knowledge, expressed in the knowledge representation domain-specific language, into executable code of the operational domain-specific programming language.

4.3 Intelligent System

The third layer is focused on bringing intelligence into the system, by reasoning about observed information as well as the knowledge base, and by developing interactive reasoning methods that can improve over time.

At the level of the operational DSL, our framework will reason over safety and satisfiability to guarantee that all experiments, both operator driven and machine driven, can be run and are run safely. For this, we can benefit from Haskell's type system but we will also use recently discovered and new techniques in type inference (beyond graded monads [4,5,12]) and light-weight program analysis (with algebraic effect handlers [14,18]) with properties to support the modular nature of the DSL and the extensible problem domain. At the level of the knowledge representation language, we build on top of constraint programming technology and new methods for automatically finding optimal execution plans, based on a declarative specification of the goals of the experiments in the developed DSL.

The system will be adaptive by having low-level machine learning building blocks, e.g., for object detection and tracking in the images, as part of the reasoning process. Furthermore, it can learn from previous experiments and from interaction with the user through preference learning, that is, learning to estimate how good execution plans are, and proposing new plans accordingly in an interactive problem solving setup. To support the interactive solving, our approaches will provide explanations of the chosen execution that enable expert-level feedback.

5 Conclusion

In summary, we want to leverage and advance the state-of-the-art in programming languages and knowledge representation to automate equipment control, applied to the field of fluorescence microscopy. This automation can be extended to a larger part of the scientific process: automatically refine high-level research questions into lower-level ones, choosing and designing experiments, and turning low-level research results into high-level conclusions that are reported back. The aim is to design:

A system that can learn from scientists and operators, and vice versa.

References

1. Chalfie, M., Kain, S.R.: Green Fluorescent Protein: Properties, Applications, and Protocols, vol. 47. The name of the publisher, 2 edn. (10 2005). https://doi.org/10.1002/0471739499
2. van Deursen, A., Klint, P., Visser, J.: Domain-specific languages: An annotated bibliography. SIGPLAN Not. **35**(6), 26–36 (Jun 2000). https://doi.org/10.1145/352029.352035
3. Fowler, M.: Domain Specific Languages. Addison-Wesley Professional, 1st edn. (2010)
4. Fujii, S., Katsumata, S.y., Melliès, P.A.: Towards a formal theory of graded monads. In: Jacobs, B., Löding, C. (eds.) Foundations of Software Science and Computation Structures. pp. 513–530. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
5. Gaboardi, M., Katsumata, S.y., Orchard, D., Breuvart, F., Uustalu, T.: Combining effects and coeffects via grading. SIGPLAN Not. **51**(9), 476–489 (Sep 2016). https://doi.org/10.1145/3022670.2951939
6. Ghosh, D.: Dsl for the uninitiated. Commun. ACM **54**(7), 44–50 (Jul 2011). https://doi.org/10.1145/1965724.1965740
7. Gibbons, J.: Free delivery (functional pearl). In: Proceedings of the 9th International Symposium on Haskell. pp. 45–50. Haskell 2016, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2976002.2976005
8. Gibbons, J., Wu, N.: Folding domain-specific languages: Deep and shallow embeddings (functional pearl). In: Proceedings of the 19th ACM SIGPLAN International Conference on Functional Programming. pp. 339–347. ICFP ?14, Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2628136.2628138
9. Hudak, P.: Building domain-specific embedded languages. ACM Comput. Surv. **28**(4es), 196–es (Dec 1996). https://doi.org/10.1145/242224.242477
10. Hudak, P., Makucevich, T., Gadde, S., Whong, B.: Haskore music notation - an algebra of music. J. Funct. Program. **6**, 465–483 (1996)
11. Jablonski, A.: Efficiency of anti-stokes fluorescence in dyes. Nature **131**(3319), 839–840 (Jan 1933). https://doi.org/10.1038/131839b0
12. Milius, S., Pattinson, D., Schröder, L.: Generic trace semantics and graded monads. In: CALCO (2015)
13. Peyton Jones, S., Eber, J.M., Seward, J.: Composing contracts: An adventure in financial engineering (functional pearl). SIGPLAN Not. **35**(9), 280–292 (Sep 2000). https://doi.org/10.1145/357766.351267
14. Pieters, R.P., Schrijvers, T., Rivas, E.: Handlers for non-monadic computations. In: Proceedings of the 29th Symposium on the Implementation and Application of Functional Programming Languages. IFL 2017, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3205368.3205372
15. Schrijvers, T., Stuckey, P., Wadler, P.: Monadic constraint programming. Journal of Functional Programming **19**(6), 663–697 (2009). https://doi.org/10.1017/S0956796809990086
16. SHIMOMURA, O.: The discovery of aequorin and green fluorescent protein. Journal of Microscopy **217**(1), 3–15 (2005). https://doi.org/10.1111/j.0022-2720.2005.01441.x
17. Tsien, R.Y.: The green fluorescent protein. Annual Review of Biochemistry **67**(1), 509–544 (1998). https://doi.org/10.1146/annurev.biochem.67.1.509
18. Wu, N., Schrijvers, T., Hinze, R.: Effect handlers in scope. ACM SIGPLAN Notices **49**(12), 1–12 (2014)