1
Introduction

1.1 A Brief Story of Laboratory Automation

Through human history, there has always been a quest for automation. For instance, the water mill, which has been used by different ancient cultures, can be considered an example of mechanical automation in the general sense of replacing human labor with a more reliable and powerful alternative. By replacing human labor with a water mill, our ancestors achieved the following aims of automation: reduced production costs, increased production efficiency, and improved safety in the production process. With the industrial revolution, when powerful engines became available, automation in this general sense maximized production efficiency as never before in human history. Further development of electronics and computing brought automation to its present and familiar stage, in which finely controlled motors execute precise tasks emulating (and surpassing) human efforts, movement sensors open doors for people to pass through, and computer programs save works without users needing to remember doing that.

Scientific laboratories have adopted automation as its technology developed. The first documented solutions in laboratory automation were devised by scientists to improve their own work. Devices such as automated filters and siphons have been built by ingenious means since the late nineteenth century. With the advent of electronics, a wide range of new devices, such as conductivity meters, gas analyzers, pH meters, and automated titrators, became available. Soon after the Second World War, automation tended to become predominantly provided by specialized companies making the devices, due to the increased complexity in manufacturing. The exponential progress in computing then enabled the opening of the first fully automated laboratory by Dr Masahide Sasaki at the Kochi Medical School in Japan in the early 1980s. In the following decade, similar laboratories were opened in Japan, the United States, and Europe. The approach followed in such fully automated laboratories started becoming known as *total laboratory automation* (TLA).

TLA was and still is very expensive. Because of that, only a few laboratories, normally those involved in fields that can produce high financial returns, such as drug discovery, can afford to implement it. In order to make TLA more accessible for medium- and small-scale laboratories, the concept of modular automation was introduced in the late 1990s. Through this concept, smaller laboratories could purchase one or a few automated instruments and progressively upgrade them when money became available. A very recent development in modular automation that has the potential to reduce even further costs in laboratory automation is the adoption of open-source hardware, which has its blueprints freely available. Enabled by open-source microcontrollers, and open-source building devices such as three-dimensional (3D) printers, this technology enables users to build their own devices at a surprisingly low cost.

Modular automation, including open-source hardware, however, does not work in all situations because of the difficulty in integrating instruments built by different manufacturers. The negative effect of this lack of integration cannot be overemphasized. It has been recognized as a problem for more than 25 years, and attempts have been made to resolve it by means of standardization of the communication between instruments. Unfortunately, such standardization has never become widespread, and to date, it has remained difficult to integrate instruments built by different manufacturers. This book aims to present a way to overcome this limitation, and thus enable laboratories to implement automation at a much lower cost than by traditional means. The approach presented in this book is very simple and accessible for most professionals in laboratories even if they do not have a background in electronics or computing. It is also a very powerful approach, which overcomes virtually any lack of compatibility between instruments.

1.2 Approaches for Instrument Integration

As explained above, a fundamental aspect of laboratory automation is instrument integration, that is, the ability to make instruments work together. The following are the two ways to enable instrument integration.

1.2.1 The Usual Approach for Instrument Integration

Communication between laboratory instruments is usually implemented by a computer controlling one instrument, which in turn controls others (Figure 1.1). There is data interchange between the computer and the first instrument, and between instrument 1 and the other instruments. However, there is no direct communication between the computer and the instruments being controlled by instrument 1. In other words, only instrument 1 can be directly controlled by the computer and, consequently, by the user. An advantage of this approach is simplicity: the user only needs to operate a single program that controls the whole set of instruments.

However, this approach also considerably limits options for users. For example, let us assume that instrument 2 breaks down and needs to be replaced. The user then finds an alternative to instrument 2, which performs better and costs less than the usual instrument supplied by the manufacturer of instrument 1. Ideally, the user should be able to connect this alternative instrument to instrument 1 and continue the work. In practice, however, this is not possible in most cases, because instrument 1 was built to communicate exclusively with instrument 2 and vice versa. Therefore, the user has no choice but to buy a second instrument 2.

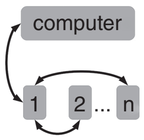


Figure 1.1 Common approach to make instruments work together in a laboratory. Arrows show the paths for data exchange.

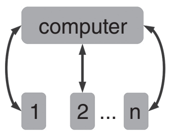


Figure 1.2 Instruments working together in a framework enabled by scripting. As in Figure 1.1, arrows indicate exchange of data.

1.2.2 Instrument Integration with Scripting

The limitation of the traditional approach of instrument integration (Figure 1.1) can be eliminated by scripting. As explained above, with scripting, the user coordinates the programs that control different instruments. This way, if the different instruments set up to work together have each a software interface, they can be integrated using scripting (Figure 1.2).

In Figure 1.2, there is no direct communication between instruments; instead, the computer communicates with all instruments. It is important to note that this is conceptually different from the scenario in Figure 1.1, in which the computer controlled only instrument 1, and instrument 1 controlled the others.

In the example outlined earlier, the user needed to replace instrument 2. If the user finds a replacement for instrument 2, which has a program controlling it, he/she can readily make it work together with instrument 1 using the arrangement in Figure 1.2. To do so, the user does not need any knowledge of electronics or even advanced computing; scripting is all that is necessary. Thus, in our hypothetical history, the user could both save money and obtain a better instrument.

The example is hypothetical, but resembles the routine activities of a laboratory technician. It is common that when replacing a broken instrument the normal alternative is either too expensive or takes a long time to become available. In such cases, the damaged instruments could be replaced by cheaper and ready-to-use alternatives, by means of scripting. Two examples can be found among the reading suggestions at the end of this chapter. In one, the autosampler of a machine was coupled to a water analyzer, because the analyzer did not have an autosampler, and was not even designed to work with one. In the other case, a low-cost robotic arm (less than US$ 500) was used as autosampler for an automated titrator, rather than using the autosampler originally designed for that instrument, which would cost more than US$ 50 000.

Another aspect that gives an advantage to scripting is that it is not always possible to use one instrument to control several others (Figure 1.1). In most cases, instruments are designed to communicate with only another one. With scripting, there is no limit on the number of instruments that can be synchronized.

1.3 Scripting versus Standardization in Laboratory Automation

Scripting is not the first proposed solution for the problem of lack of compatibility between instruments. However, as will be explained below, it is the only one with real odds of solving this problem.

The usually proposed solution for the compatibility problem in laboratory automation is the adoption of standards. Such standards would mean that all instruments would communicate using the same protocols. This would make the integration of instruments very simple, and the problem of replacing instrument 2 (Figure 1.1) illustrated above would be very easily solved.

Although such a solution looks desirable, it is very difficult to be put in practice. A recent effort in this direction that stands out by its large magnitude is SiLA (Standards in Laboratory Automation). According to their website, http://www.sila-standard.org, SiLA is a consortium of several system manufacturers, software suppliers, system integrators, and pharmaceutical corporations, among others. There are several working groups composed of high-skilled experts dealing with device control and interfaces, command dictionary specification, data standard, process management system, and so on. If, on the one hand, it is impressive to see such a combined work with the aim of improving laboratory automation, then, on the other hand, the necessity of such endeavor warns that the problem at hand is complex.

In addition to the complexity of the problem itself, there is resistance of manufacturers in adopting a potential standard that may or may not become widespread. As the proposers of SiLA themselves admit (see the relevant publication in the reference list at the end of this chapter), newcomers need to spend time, resources, and money to implement SiLA. In addition, SiLA will be successful only if all players in the industry adopt it. It is a very ambitious goal, which is still far from being reached.

Now let us compare this situation with scripting. The first aspect is that the technology for enabling scripting is ready; there is no need for further development (although of course this development continues in the form of bug fixing and improvements in the scripting language). In other words, integration of laboratory instruments using scripting can be implemented now, while uniform standards are not yet available for most laboratory equipment.

The second aspect is that it is much easier for manufacturers to adapt to scripting than to adopt standards. In fact, manufacturers do not need to do anything, as scripting was designed to deal with software "as is." As will be explained in the last chapter of this book, however, there are measures that manufacturers can adopt if they want their software to be "script-friendly." These are minor modifications that are much easier to adopt than standards.

The third aspect is that standardization necessarily creates restrictions to further development. If a new and more efficient way of implementing a software solution is devised, but it is outside the scope of the standard, either the solution must not be implemented or the standard needs to be modified. With scripting, there is no risk of such problem unless a very radical change in the way computers work comes to place, like, for example, the abolition of mouse and keyboard use (which is extremely unlikely).

The fourth aspect is how users need to adapt to changes brought by scripting or standards. In order for them to work, standards need a universal interface, called *integrator*, that controls all programs. Scripting also demands that users learn how to control several different programs using a common framework. This book provides information on this aspect, which, as will be seen, is quite accessible. An important point is that the programs used for scripting are free of cost. By contrast, integrators are not always free.

The fifth aspect is that adoption of standards indicates that in many cases perfectly working devices would become obsolete for just not following the standard. This could be translated into huge costs of modifying existing equipment, or obtaining new ones. Nothing of this is necessary if scripting is used, since it demands no modification in the equipment. Another consequence of this aspect is that old equipment, many times

Table 1.1 Summary of differences between scripting and standards for laboratory automation.

	Scripting	**Standards**
Technological availability	Ready	In development
Backward compatibility	Yes	No
Universal acceptance	Not necessary	Necessary
Difficulty for manufacturers	None	High
Easily adaptable for new technologies	Yes	No
Difficulty for users	Little	Little to high
Cost for manufacturers	Zero or small	High
Cost for users	Zero or small	High in most cases

compatible with only older computers, can be "resurrected" and made to work with newer counterparts (up to a point; Windows versions before XP are not fully supported, see more details in Chapter 2).

The sixth aspect is that by enforcing standards, laboratories miss the opportunity of using the myriad of new devices and technologies that appear at a frantic pass every day. These devices do not necessarily follow the standards that are proposed to be used by laboratory instruments. Therefore, any creative use of traditional instruments and revolutionary new devices becomes hampered with the adoption of standards. The same is not true with scripting: any new device can be used in conjunction with traditional laboratory instruments by means of this technology.

In summary, scripting is by definition a better solution than the adoption of standards for laboratory automation. It is easier, cheaper, and can be implemented now. Standards cost money, time, and are not ready yet (and may never be). Table 1.1 summarizes the differences between the two approaches.

1.4 Topics Covered in this Book

Before knowing what this book is about, it may be important to clarify what it is not about, because this book differs in many aspects from previously published books on laboratory automation.

The previously published books on laboratory automation often covered electronics, and even some aspects of mechanics. They also often presented deep discussions about communication protocols, or some advanced concepts in programming. In some sense, these books demanded (or aimed to provide) an encyclopedic knowledge to users who are already necessarily specialized in another profession. This book is different from these previous ones in that it does not cover any of those diverse subjects, except superficially for some of them. This is because of the novel approach presented here: instead of building instruments or creating software interfaces for them, using scripting we can simply coordinate the software interfaces, which are already easily available for existing instruments and combine them. Therefore, this book is mainly focused on how to use scripting to coordinate software interfaces, which is much more accessible than the subject of previously published books for the ordinary laboratory technician or scientist without a deep background in computer science or robotics, and without time to devote to learning a complex new subject.

It is important that the book be read in sequence, especially for those without knowledge on programming. It is suggested for those who do not have a background in programming to read this book from Chapters 1 to 5, at least, and test all the scripts presented therein. The next chapters can be understood much more easily, provided the initial ones are studied first. For readers with experience in programming, it can still be useful to have a look into the initial chapters to know how AutoIt works. It will be seen that the codes used throughout this book are often very simple, and the readers will probably have no problems understanding them. A reference for the language is not presented in this book, rather only the aspects of the language that are useful in specific contexts are presented. Therefore, chapters about loops or conditionals will not be found; however, these subjects are presented as parts of chapters dealing with their uses, as they become necessary. A final comment for those with a background in programming: throughout the book, the "best practices" for writing scripts in AutoIt are deliberately ignored, but are available at http://https://www.autoitscript.com/wiki/Best_coding_practices. It has been done to simplify the learning for people without a background in programming, as these best practices may be confusing for uninitiated readers.

Chapter 2 introduces AutoIt. As it is explained there in detail, AutoIt stands out as a very powerful yet simple scripting language that is perfect to be used for laboratory automation.

Chapters 3–5 are arguably the most important in this book. They describe how AutoIt can be used to synchronize instruments built by different manufacturers and controlled by a single computer. All examples in these chapters are for the teaching software written for this book.

Chapter 6 shows how AutoIt can be used to implement alarms to deal with problems during automated measurements. In this case, not only the teaching programs are used, but also third-party software is introduced, demonstrating that AutoIt is versatile and can combine communication software very easily with laboratory instruments.

Chapter 7 shows how AutoIt can be used to integrate low-cost automation devices, such as CNC routers, 3D printers, and robotic arms, with laboratory equipment. Again, third-party software is also used in the examples.

Chapter 8 was written for readers without any background in programming, as it covers arrays and strings. However, readers with some programming background can also find this chapter beneficial by reading it to check the details of the language on these topics.

Chapters 9 and 10 show how AutoIt can automate the export of data generated by instrument software, and how the processing of such data can be made easier and faster.

Chapters 11–16 present several techniques that enable the synchronized control of instruments by more than one computer. In most cases, this demands the use of computer networks, including the Internet. Most are very simple, but Chapter 14 demands deeper study.

Chapters 17–22 cover how AutoIt can be used as a programming language and not a scripting language. In other words, AutoIt is used to directly control an instrument, by means of building user interfaces and using communication protocols. These chapters are more advanced than the rest of the book, but still accessible for readers with a background in programming or those who follow the book chapters in order from the beginning.

Chapter 23, the last one, presents some suggestions and guidelines to users and manufacturers of laboratory instruments so that both sides can take advantage of scripting. It is solely a personal view, but hopefully it will help to convince instrument manufacturers to implement a few small changes that can make laboratory automation even more accessible for laboratory users, and orient users to choose the best options when implementing laboratory automation.

After the main text, there are three appendices in this book. The first one shows some useful features of the standard script editor for AutoIt, which are not essential but can help the user in the day-to-day writing of scripts. The second appendix presents optical character recognition (OCR), a technology that is still imperfect, but could be very useful for scripting in some situations. The third one also presents a developing technology, which is the use of Windows User Interface Automation in AutoIt. This technology in particular can be very useful for automating some popular programs used in the laboratory, which are those developed using LabView.

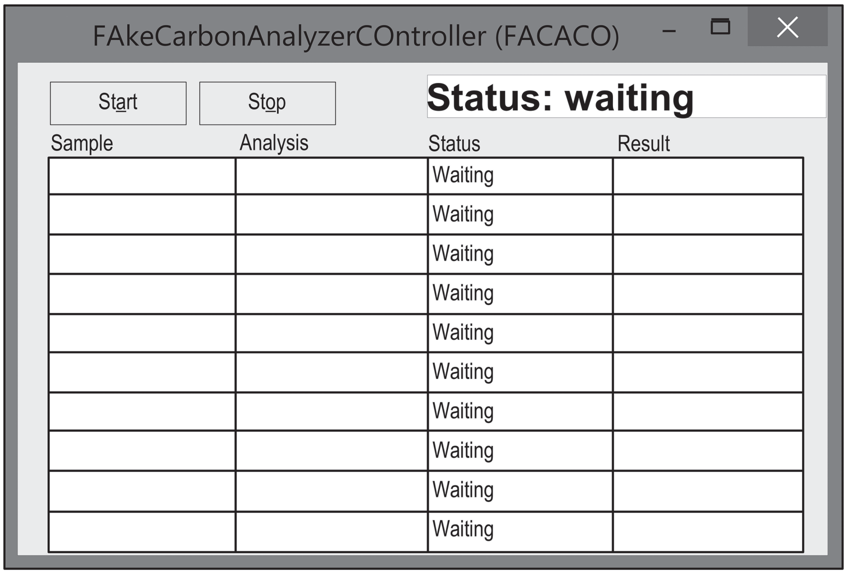
1.5 Learning by Doing: FACACO and FAKAS

An aspect of this book is that it is entirely based on examples, because the author believes that the best way to learn to script is by practicing. Scripting is so simple that in most occasions readers will be able to simply copy the examples, and, with minimal modifications, have a script working for their needs. Still, in order that the examples are fully understandable, it is necessary that readers gain access to the same software than the author. In order to ensure that, two very simple programs that share similarities with software normally used for controlling laboratory equipment were written: FACACO and FAKAS. FACACO stands for FAke Carbon Analyzer Controller, and FAKAS FAKe AutoSampler.

The first step in the learning process is to become familiar with the two programs. They will be described in detail, but it could be seen that a fundamental requisite when scripting is to be very familiar with the programs being automated.

First, download them from http://www.wiley-vch.de/publish/en/books/ISBN978-3-527-34158-0/. Warning: The code for these two programs was written using AutoIt, and some antivirus software treated them as contaminated with a virus. FACACO and FAKAS are safe to use. The programs do not need to be installed. Double click each of the icons to see the interfaces.

FACACO simulates the controller of a carbon analyzer for water samples. Without going deep into this type of analysis, it is useful to understand the fundamentals of it in order that FACACO becomes more familiar. There are two different types of dissolved carbon in water: dissolved inorganic carbon (DIC) and dissolved organic carbon (DOC). FACACO simulates an instrument that pumps a fixed amount of a water sample to a reaction chamber where DIC and DOC are extracted. DIC is extracted by acidification, while DOC is extracted by oxidation. Both extractions consist in converting DIC or DOC to CO_2. The extracted CO_2 is then measured using an appropriate sensor. A relevant detail is that DIC must be extracted beforehand so that DOC can be measured in isolation.



Sample	Analysis	Status	Result
		Waiting	
		Waiting	
		Waiting	
		Waiting	
		Waiting	
		Waiting	
		Waiting	
		Waiting	
		Waiting	
		Waiting	

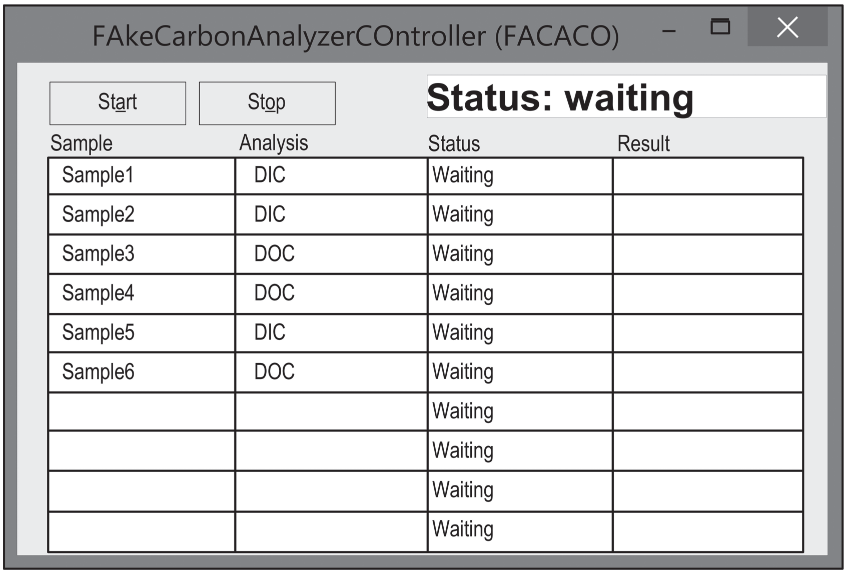
Figure 1.3 FACACO interface.

FACACO interface contains two buttons: Start and Stop (Figure 1.3). There is a spreadsheet-like table below it, with four columns: Sample, Analysis, Status, and Result. Finally, there is the phrase “Status: waiting” in large font inside a white background.

FACACO can be explored by clicking on the buttons and typing into the cells. For example, when you press the Start button, you will see a pop-up message asking if you really want to start the measurements, and you can say yes or no. If your “Sample” cells are all empty, nothing will happen, regardless if you press yes or no. Then, let us fill the “Sample” columns with some names, like “sample1” or “sample2,”. Now try the Start button once more, and respond yes to the pop-up box. You will see another pop-up box complaining that the type of analysis must be DIC or DOC. Then, on the “Analysis” column, fill the analysis type (DIC or DOC) for each sample, like, for example, in Figure 1.4.

Now, press Start again and say yes to the pup-op window. Another pop-up window will appear, this time with a warning that the sample must be connected to the pump (the program assumes that there is no autosampler and that the user will replace the vials connected to the pump through the sampling line). After that, the sample will be measured, and you will see the contents of the “Status” column change. For DIC analysis, the status changes from Waiting to Sampling to Measuring DIC to Calculating DIC, and to Done. Similar changes appear in the large Status field, including changes in the color of its background. When done, the result of the analysis appears in the Result column. For DOC, the sequence of status is Waiting, Sampling, Removing DIC, Adding oxidant, Measuring DOC, Calculating DOC, and Done. The removing DIC step is to ensure that the sample will be acidified before measuring DOC, as explained before for carbon analyses in water. The pop-up window appears for the next sample, and for all subsequent samples until the end of the list.

You can press the Stop button anytime during the measurement. Doing so, the status of the sample currently being measured goes to “Interrupted” and the background



Status: waiting

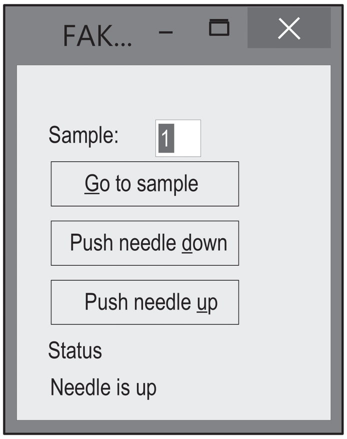
Sample	Analysis	Status	Result
Sample1	DIC	Waiting	
Sample2	DIC	Waiting	
Sample3	DOC	Waiting	
Sample4	DOC	Waiting	
Sample5	DIC	Waiting	
Sample6	DOC	Waiting	
		Waiting	
		Waiting	
		Waiting	
		Waiting	

Figure 1.4 FACACO interface with samples and analyses loaded.

behind the large Status label becomes red. If you press start again, you should be able to continue with the remaining samples. If you want to start over, the easiest way is closing the program and opening it again.

FAKAS (Figure 1.5) simulates the controller of an autosampler for water samples. This imaginary autosampler can handle 10 samples, and be imagined as a carousel, where these samples stand on, and a needle connected to a tube. This tube is connected to the pump of the hypothetical carbon analyzer controlled by FACACO. Therefore, in order to sample water from a given sample, we need to tell FAKAS its position (between 1 and 10), tell it to move to the sample by pressing the button “Go to sample,” and then, when there, tell it to push the needle down (“Push needle down” button) so that it reaches the

Figure 1.5 FAKAS interface.



Sample: 1

Status

Needle is up

water in the sample vial. When the sampling is done, we can tell the autosampler to pull the needle back up ("Pull needle up" button) so that it can move to other samples. The status of the autosampler is given at the bottom of the FAKAS window. You can test the buttons on FAKAS interface and see the error messages that come if you try to move the needle when it is down, or if you input a number smaller than 1 or larger than 10, or even letters in the "Sample" field.

When writing scripts, it is fundamental to be familiar with the interfaces of the programs that control the instruments. Many times, this familiarity comes with the intention of performing the scripting. This will become clear in the following chapters.

1.6 Summary

- First efforts in laboratory automation were carried out by technicians and scientists themselves.
- Specialized companies started providing automated equipment in large scale after the Second World War.
- TLA was first implemented in Japan by Dr Masahide Sasaki in the early 1980s.
- TLA never became widespread due to high costs, which were partly driven by lack of compatibility between instruments from different manufacturers.
- Scripting is the easiest and cheapest way to enable compatibility between instruments.
- AutoIt is the scripting language presented in this book.
- Two programs (FACACO and FAKAS) are used as examples for the scripts in this book.

Suggested Reading

History of laboratory automation:

Boyd, J. (2002) Robotic laboratory automation. *Science*, **295**, 517–518.

Feiglin, M. and Sterling, J.D. (2008) Laboratory automation. *Nat. Rev. Drug Discovery*, **7**, 625.

Felder, R.A. (1998) Modular workcells: modern methods for laboratory automation. *Clin. Chim. Acta*, **278**, 257–267.

Felder, R.A. (2006) Masahide Sasaki, MD, PhD (August 27, 1933 – September 23, 2005). *Clin. Chem.*, **52**, 791–792.

King, R.D., Rowland, J., Oliver, S.G., Young, M., Aubrey, W., Byrne, E., Liakata, M., Markham, M., Pir, P., Soldatova, L.N., Sparkes, A., Whelan, K.E., and Clare, A. (2009) The automation of science. *Science*, **324**, 85–89.

Olsen, K. (2012) The first 110 years of laboratory automation: technologies, applications, and the creative scientist. *J. Lab. Autom.*, **17**, 469–480.

Sasaki, M., Kageoka, T., Ogura, K., Kataoka, H., Ueta, T., and Sugihara, S. (1998) Total laboratory automation in Japan past, present and the future. *Clin. Chim. Acta*, **278**, 217–227.

Standards in laboratory automation:

Bär, H., Hochstrasser, R., and Papenfuß, B. (2012) SiLA: basic standards for rapid integration in laboratory automation. *J. Lab. Autom.*, **17**, 86–95.

Delaney, N.F., Rojas-Echenique, J.I., and Marx, C.J. (2012) Clarity: an open-source manager for laboratory automation. *J. Lab. Autom.*, **18**, 171–177.

Hawker, C.D. and Schlank, M.R. (2000) Development of standards for laboratory automation. *Clin. Chem.*, **46**, 746–750.

Fast technological development:

Butler, D. (2016) Tomorrow's world. *Nature*, **530**, 399–401.

Open-source hardware for laboratories:

Pearce, J.M. (2012) Building research equipment with free, open-source hardware. *Science*, **337**, 1303–1304.

Pearce, J.M. (2014a) Cut costs with open-source hardware. *Nature*, **505**, 618.

Pearce, J.M. (2014b) *Open-Source Lab: How to Build Your Own Hardware and Reduce Research Costs*, Elsevier.

Scripting in laboratory Automation

Carvalho, M.C. (2013) Integration of analytical instruments with computer scripting. *J. Lab. Autom.*, **18**, 328–333.

Carvalho, M.C. and Eyre, B.D. (2013) A low cost, easy to build, portable, and universal autosampler for liquids. *Methods Oceanogr*, **8**, 23–32.