1

Advanced Total Lab Automation System (ATLAS)

Christoph Wagner, Andreas Genner, Georg Ramer and Bernhard Lendl
Vienna University of Technology
Austria

1. Introduction

In modern analytical chemistry not only do new measurement principles have to be investigated but newly found techniques also need to be developed into integrated and more and more automated analytical solutions. Exploring new techniques in research and development (R&D) environments mostly means designing different setups incorporating a huge variety of different equipment. Whether this hardware is a pump, a valve or more complex elements such as oscilloscopes, lasers or detectors, in one way or another they can be controlled by a computer or the acquired data is digitized and evaluated afterwards. Those setups, however, change frequently in R&D labs to evaluate new ideas or to improve a current setup. Inevitably whenever the setup chances, the software controlling it will also need adaptation. An ideal software platform, therefore, would reduce the time needed for making those changes to a minimum, giving the researcher the possibility to focus on the evaluation of new approaches and the characterization of new techniques. Especially for small startup companies it might also be of great interest that the needed software packages can be maintained and adopted by the scientist himself instead of the need to outsource it to a software company. This would not only reduce the cycle time from a new idea to the first prototype but also has the potential to decrease the associated costs.

One programming language ideally suited for such a software environment is LabVIEW from National Instruments. Due to its graphical programming language all data flow is visualized directly in the source code. This fact makes it easier to read and understand a given program, especially for beginners without programming experience in classical programming languages, such as C++ or Visual Basic. Another point where LabVIEW outperforms other programming languages is it's rich library of Graphical User Interface (GUI) elements, for example XY graphs are predefined as well as 3D graphs. Another advantage of LabVIEW is the very simple way to create parallel running tasks. In combination with LabVIEW's queues, which represent a "first in - first out" buffer system, separating the readout of data and its evaluation can be separated easily. Using National Instruments hardware for the data readout brings easy integration for the software developer. Secondly interfaces for RS-232, GPIB, USB etc. are also provided for communicating with third party hardware.

All these advantages make LabVIEW a good choice for developing software in R&D laboratories. Generally speaking, laboratories focus on the development of software for their

current experimental setup ending up with a LabVIEW program built for this specialized task. Dominguez et al. designed a LabVIEW program to control a Sequential Injection Analysis (SIA) system (Dominguez et al., 2010) which consisted of a valve and a peristaltic pump. Jitmanee et al. used a more complex liquid handling system consisting of three valves and three pumps to automatically prepare samples for Inductive Coupled Plasma Mass Spectrometry (ICP-MS) measurements (Jitmanee et al., 2007). Barzin et al. proposed a spectrophotometric method for pH measurements in miniaturized systems where LabVIEW handles data acquisition from a flowmeter and a spectrometer and is linked to a Matlab feedback loop to calculate the appropriate flow rate which is sent to a syringe pump afterwards via LabVIEW (Barzin et al., 2010). In all these examples the program is limited to control the hardware it was designed for. In the first two examples automation tasks are carried out whereas in the latter one a complex control feedback loop is realized. When we started working with LabVIEW we wanted to create a software platform, which gives the operator the freedom to combine different hardware elements in any imaginable way without the need to make changes to the program code. Only adding new hardware to the software platform should require software changes, which would be easier using LabVIEW than any other programming language.

First we created a program that allowed the combination of valves, pumps and an high voltage supply for the sample preparation with an UV detector and infrared spectrometers from Bruker Optics to build different setups including SIA (Růzicka, 1992) and flow injection analysis (FIA) (Růzicka, 1981) systems as well as capillary electrophoreses systems (Wagner et al., 2010). To address all necessary components in a simple way we decided to use a scripting language containing commands representing tasks including switching a valve, applying voltages or starting measurements. The script was written manually and afterwards started sending commands to the connected hardware consecutively. This system allowed the easy combination of the integrated hardware into different analytical systems. However, after adding support for additional hardware and the need for more complex commands the GUI became too crowded with elements and writing the scripts became more and more difficult due to the number of available commands.

Our first program allowed the scientist to combine all integrated components with ease but adding new hardware components to the program became more and more sophisticated due to the growing source code. Therefore, we decided to create a new platform based on a client server approach, where every hardware component is controlled by a dedicated client which can be controlled remotely by a server application. A schematic of this software platform can be seen in figure 1. Each of the clients can be run in three different modes. The *local mode* directly interacts with the connected hardware upon user input, the *programming mode* sends commands to the server application and in the *server mode* the clients receive commands from the server and execute them on their hardware. In that manner adding new hardware is done by adding client specific program code to a provided client template, containing all necessary code to handle the communication with the server. Additionally the user doesn't need to remember client specific commands because the programming of a sequence can be done with the GUI of each client.

In the following we will describe our *Advanced Total Laboratory Automation System* (ATLAS), including a list of existing clients as well as two examples of experimental setups controlled by ATLAS.

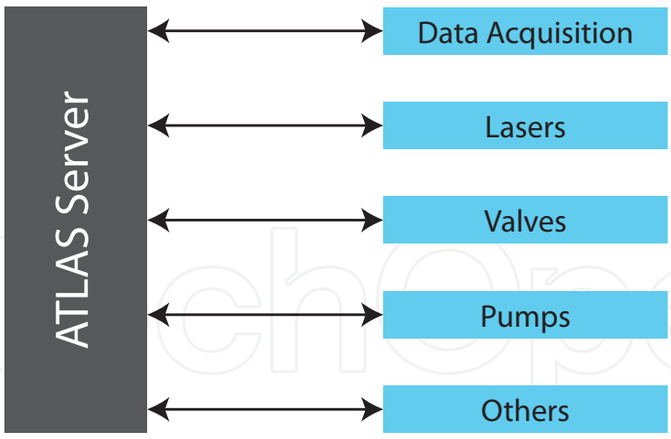


Fig. 1. Schematic of our *Advanced Total Laboratory Automation System* (ATLAS). Each hardware component is controlled by a dedicated client program which can communicate with the ATLAS server to be remote controlled.

2. The ATLAS platform

The ATLAS platform consists of one *server* application and numerous *client* applications. Each client controls connected hardware such as a pump, valves or a spectrometer. The *server* is used to remote control connected *clients* by sending script commands over a TCP/IP connection to the *clients*. Each *client* can be used in a *local* mode, as well, to control connected hardware directly. The *server* application was designed so that adding new *clients* can be done without modifying the *server* application.

2.1 Client server communication

2.1.1 Simple TCP/IP Messaging (STM)

The ATLAS software is based on the *Simple Messaging Reference Library (STM)*[1] maintained by National Instruments, which contains a set of VIs for developing server/client-based applications. Beside the *TCP Connection Manager VI*, which handles connection informations, the library contains also VIs for transporting data over TCP/IP connections.

First of all, a server, handling several clients, requires a *New Connection Monitor*[2], which creates a listener on a certain port (e.g. 8080) and waits until a client tries to establish a TCP network connection. In the next step the server defines meta data, which basically define data channels for data transportation. All information concerning a certain client connection is stored using the *TCP Connection Manager VI*, which represents a Functional Global Variable (FGV). All further communication between the server and the connected clients is handled in a seperate loop, called the server loop, by accessing this FGV.

The *TCP Connection Manager VI* offers also the functionality for listing all established connections. The resulting output is an array of connection-IDs which has to be indexed for sending or receiving data to a specific client. If the communication to a client breaks down due to hardware- (e.g. network cable is unplugged), or software problems (e.g. client program crashes) it is necessary to remove the connection information of this client from the

[1] http://zone.ni.com/devzone/cda/tut/p/id/4095

[2] http://zone.ni.com/devzone/cda/tut/p/id/3055

FGV. Otherwise the server may try to send data to an unreachable client which would lead to a timeout. All data between the server and the clients is transported by using the *STM Write Message* and *STM Read Message VIs*. Beside the string that has to be sent, a cluster, containing timeout settings, can be attached to the *STM Write Message VI*. At first, the meta data index, containing the data channel names is obtained, flattened to a string and concatenated with the data to send. In the next step the byte length of this data string is calculated and attached to the beginning of the string. Finally, the complete string is written to the TCP connection.
The receiving program utilizes the *STM Read Message VI* to read the first four bytes of the transported string. These four bytes are converted to an integer representing the length of the remaining data string. In a second step this value is taken to read the residual data. Finally, the transported string can be accessed for further processing.

2.1.2 Communication principle

The communication between the server and the connected clients is controlled by the server only. This means that the server sends commands to the clients, which reply to the server afterwards (for a typical communication sequence see figure 2). Every 50 ms the server queries the status of the connected clients by sending the command *status*. Receiving the status request for the first time after establishing the connection to the server a client replies with *cmd:ID* identifying itself with its *client identifier*, where ID consists of two device depending alphanumerical characters. Upon the reception of a *cmd:ID* string the server stores the new client and the according STM connection ID. This data is accessed by the server to send client specific commands only to the according client in any further communication.

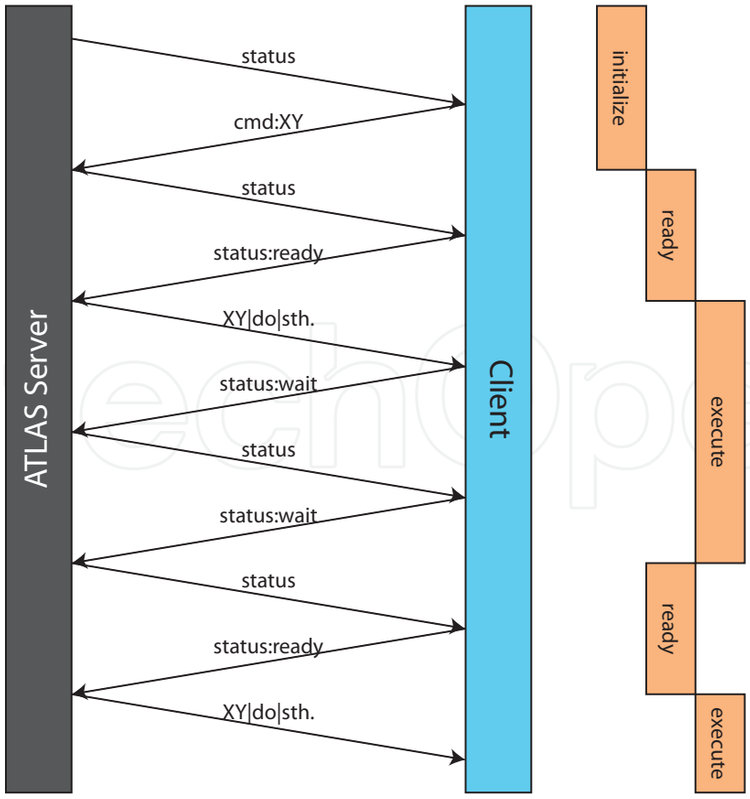


Fig. 2. Simplified communication scheme between the ATLAS server and one client.

Receiving further status queries from the server clients respond with either *status:ready*, *status:wait* or *status:error*. *status:ready* means that the client is ready to accept instructions from the server and that both, software and hardware, are working correctly. *status:wait* will be sent if the client is executing a command, such as measuring a voltage or recording a spectrum. In the case that a hardware failure occurred or an invalid command was sent to the client, it will respond with *status:error*.
After the user starts the execution of the sequence by clicking the corresponding button, the server checks whether all required clients are connected. If so, it will run the command interpreter, described in section 2.2.2. Afterwards the server begins distributing commands to the correct clients. After a command was sent the server goes on with polling the status of the connected clients. The addressed client starts executing the received command and answers *status:wait* to the server. The server will continue the execution of the script as soon as its status requests are answered with *status:ready* instead of *status:wait*.
In case a device failure occurs (e.g.: a tubing is plugged, a device becomes disconnected) or an invalid command was sent, the client controlling the according device will answer the status request from the server with *status:error*. As a result the server terminates the running sequence and executes the alias-command *AL|error*. As explained later in section 2.2.2 the server notifies all connected clients of the error event. The clients will execute a special protocol that contains commands for such an error case. Usually it is sufficient to abort the experiment, but some devices have to be shut down in a special way to avoid further problems.

2.2 Server application

2.2.1 Description

The central part of ATLAS server is a timed loop structure, further referred to as *server loop,* and can be seen in the middle of the plotted code in figure 3. As already mentioned in section 2.1.1 the *TCP Connection Manager VI* is used to list all connections established to clients. The resulting array of connection IDs of connected clients is treated in a for-loop structure. In this for-loop the server requests status information from the currently addressed client. It can respond with one of the following strings which is evaluated by the server in a case structure:

- *cmd:ID*: The client-ID is added to the list of *available client commands*.
- *programming*: The received string is added to the *commands_list*.
- *status:ready*: In that case the corresponding LED in the *available client commands*-list turns green.
- *status:wait*: A variable is set to prevent sending the next command. In addition, the LED next to the client-ID turns red, indicating that the client is currently executing a task.
- *status:error*: The command queue is flushed and therefore the command `AL|error` is interpreted. As the software should also display an error message, the client-ID is inserted into a queue. Outside of the communication loop is a while loop which is only used for displaying error messages. This makes it possible, to go on with sending the error information immediately to all clients instead of waiting for an user interaction.

If a time-out occurs while the server awaits the answer from the client because the connection was interrupted, the *TCP Check Connection VI* removes the connection from the *Connection Manager VI* and the client-ID is removed from the list of *available clients*. This client list is displayed on the GUI to visualize the status of the connected clients. Finally, the received string from the client is logged together with a timestamp for debugging purposes.

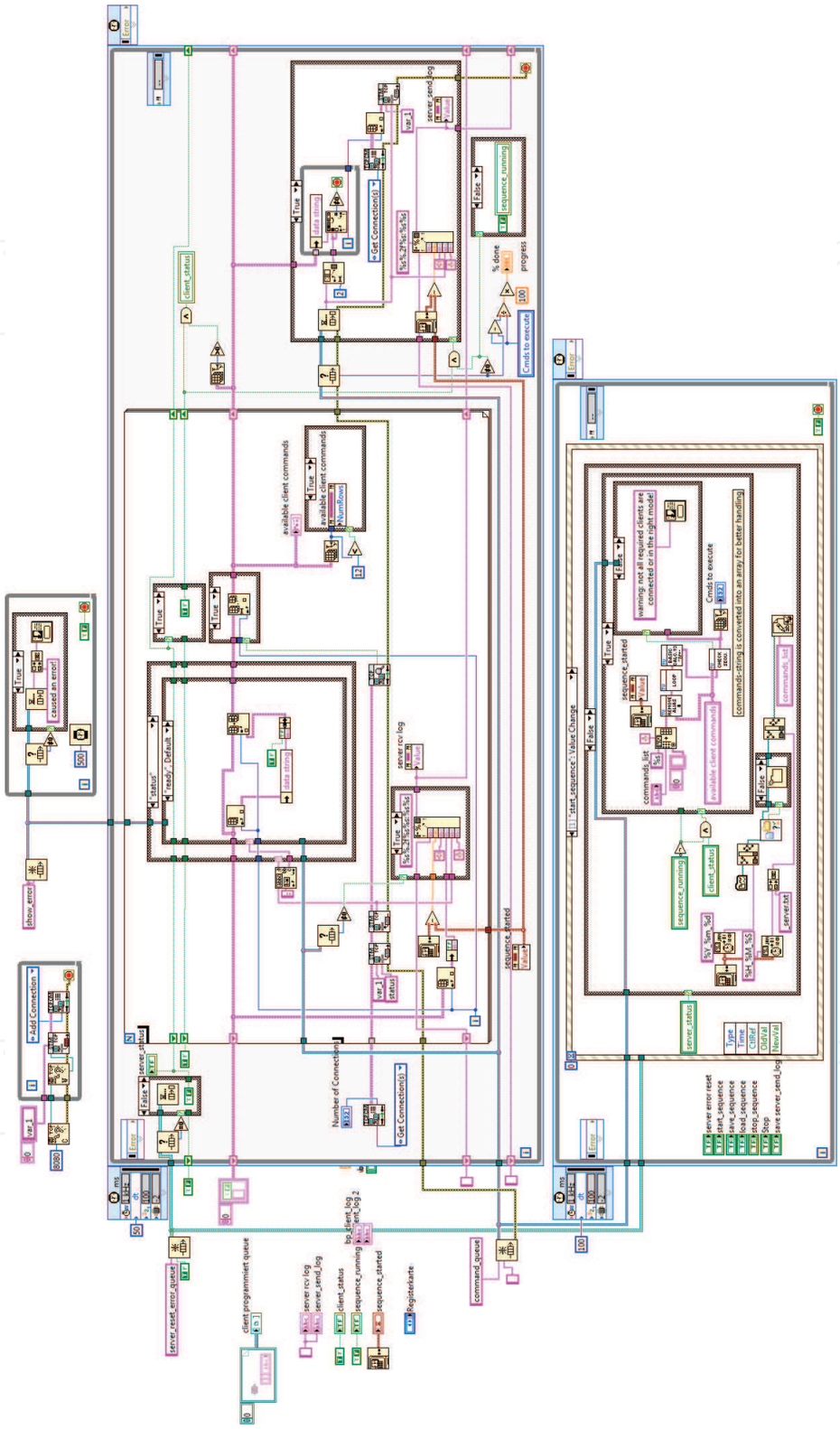


Fig. 3. Code of the ATLAS server application.

This procedure is repeated in the for-loop for each client and after each client responds *status:ready* the server takes the next available command from the script. The corresponding client is identified by its client ID and the command is sent over the corresponding TCP connection. In addition the timestamp and the sent string are logged and the progress of the script execution is visualized.
User input such as loading, saving, starting and canceling of sequences, as well as saving the server-send-log, is treated by an additional event handler structure. Whenever a script is executed the server automatically saves the script sequence to be able to connect recorded measurement data unambiguously with the used sequence.

2.2.2 Additional server features

Before the ATLAS server starts distributing the commands to the corresponding clients, the sequence has to pass an interpreter. It enables three useful features which are a for-loop, a mathematical formula solver and the use of aliases. Moreover, the software supports the usage of comments, which have to start with a hash key. These features can be used to program complex script sequences and are summarized in table 1.

feature	code	result
comment	`#this is a comment`	
loop	`loop($i=5\|9\|1)` `WA\|$i|sec` `loop_end`	`WA|5|sec` `WA|6|sec` `WA|7|sec` `WA|8|sec` `WA|9|sec`
mathematics	`loop($i=0\|2\|1)` `WA\|(5^$i+2^2-exp(0.5)+sin(1.5))` `loop_end`	`WA|4.349|sec` `WA|8.349|sec` `WA|28.349|sec`
alias	`AL|name|sample_001`	`AA|name|sample_001` `AB|name|sample_001` `AC|name|sample_001` `AD|name|sample_001`

Table 1. Examples for the script-interpreter features. It is able to handle loops, formulas and aliases.

2.2.2.1 Loop constructions in the script sequence

Often a part of a sequence has to be executed many times. For example, a scientist wants to measure a large number of liquid samples by using valves and pumps. One way to solve this problem is to enter every line of the sequence manually, which means that every change of the valve-position and every operation of the pump has to be written down in the sequence. As this way is inefficient and hard to adopt to further experiments, the ATLAS server implements a function that is similar to a for-loop in conventional programming. In C++, for example, a for-loop begins with `for(int i=5; i<13; i++){` and ends with } whereas the repeated code is located between the braces. Scientists, who want to use such a function in their ATLAS sequence, have to write the repeated section between the lines `loop($i=5|13|1)` and `loop_end`. The opening bracket is followed by `$`, a variablename, `=` and the starting point for the variable. The next number is the last variable value and the third number represents the increment. Nested loops are also allowed whereas different variable names have to be used.

2.2.2.2 Solving mathematic formulas

It is important to mention that the loop-feature only allows for a constant increment. To overcome this limitation the ATLAS server can solve formulas based on *NI_Gmath.lvlib:Eval Formula String.vi*[3]. The mathematical expressions are defined by writing them between brackets and the result will have a precision of 3 digits after the decimal separator. As the loop-function is interpreted before any formula is solved, it is also possible to use loop-variables in these mathematical expressions.

2.2.2.3 Aliases

Another feature in the ATLAS scripting language are aliases. These are abbreviations for longer program sequences which are used frequently. For example, a cleaning procedure in a liquid handling system, consisting of many single instructions, can be saved in the file `<ATLAS directory>\Server\alias\cleaning.txt`. After that the cleaning procedure is inserted in the script sequence by using the command `AL|cleaning.txt`, which shortens the script sequence and makes it easier to comprehend the whole sequence.
Aside from custom built aliases, the ATLAS server can access the aliases `AL|name|measurement_name` and `AL|error`. Their special characteristics are that the server replaces the alias-indicator `AL` by all available client-identifiers. Using `AL|name|measurement_name` will distribute a measurement filename to all connected clients which will save their measurement data by this name. `AL|error` is used to distribute the information, that an error has occurred, to all connected clients immediately.

2.3 Client applications

Each client can be operated in 3 different modes: The *server* mode, the *programming* mode and the *local* mode. If a client is switched to the *server* mode, it tries to connect to the ATLAS server over TCP/IP and after the connection has been established the device can be remote controlled by the server. The *programming* mode offers a comfortable way to create a script sequence on the server since every user event will be sent to the ATLAS server as the according script command instead of executing it. In *local* mode the connected device is controlled directly and user input is executed immediately.
The code structure is based on a state-machine[4] and a simplified scheme of the state-machine used is shown in figure 4. It consists of three states called *init*, *state changed* and *running*. The *init* state is used for initializing the user interface, setting constants, configuring a RS-232 connection etc. The *state changed* state is called if the user changes the client mode. Depending on the selected mode the network connection to the ATLAS server is established. If the connection initialization fails, the client automatically switches back to the *local* mode. All other functionality of the client is located in the *running* case of the state-machine.
The *running* state contains code elements for communicating with the ATLAS server, handling user inputs and controlling the connected device. An overview of the code can be found in figure 5. To simplify the block diagram all client specific hardware functionalities have been removed. The code inside the *running* state is based on a *producer consumer architecture*[5] consisting of three loops in our case. The first loop contains an event handler for user interactions whereas a second loop receives script commands from the ATLAS server

[3] http://zone.ni.com/reference/en-XX/help/371361G-01/gmath/advanced_formula_vis/
[4] http://zone.ni.com/devzone/cda/tut/p/id/2926
[5] http://zone.ni.com/devzone/cda/tut/p/id/3023

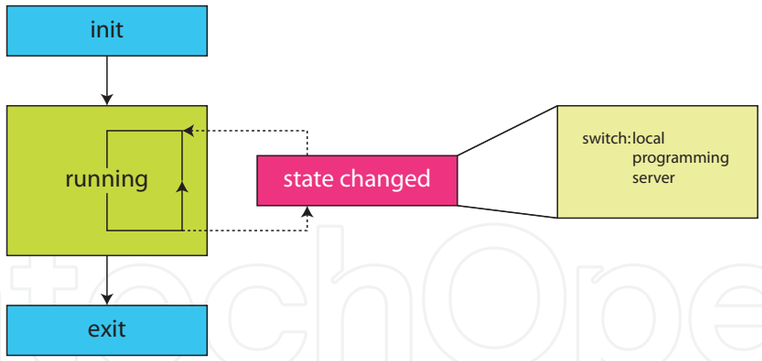


Fig. 4. Scheme of the client's architecture, which consists of a state-machine. It enables the software to switch between the different operating modes "client", "server" and local".

in *server* mode and passes them on to the third loop functioning as the consumer loop. In *programming* mode, however, the second loop is used to send script commands to the ATLAS server upon user interaction. The consumer loop interprets the script commands and translates them into hardware interactions. The exchange of commands between these three loops is realized by a queue (referred to as command queue) code structure. A second queue (referred to as status queue), which can also be seen in figure 5 is used to transport the status of the client from the consumer loop to the server loop and on to the ATLAS server if the client is in *server* mode. The differences in program execution for the three different states are explained in further detail below.

- **local** mode: The communication between the server and the client is deactivated. User interaction on the front panel triggers the event handler which assembles a script command, representing the user input from the GUI, and adds this script command to the command queue triggering the consumer loop execution.
- **programming** mode: The server loop is used to send script commands received via the command queue to the ATLAS server after receiving a status request from the server. The consumer loop, however, does not pass on any command to the connected hardware.
- **server** mode: The server loop is used to receive script commands from the ATLAS server and adds them to the command queue. The event handler structure, however, can still add commands to the command queue on user input to enable the user to interrupt automated script execution. The server loop is also used to reply to status requests from the ATLAS server according to the current state of the client, which is transported over the status queue as described above.

2.3.1 Available clients

At the moment 16 different clients are available and listed below. The list contains the client identifiers and a short description of the client functionality.

AC: Self-developed 230V AC power socket.
The AC client can switch a power socket and is connected to the PC via a RS-232-port.

BC: Self-developed Boxcar-Integrator.
This client reads the data, whereas the gating- and trigger-delay have to be set manually at the device.

BP: Beep client.
If the ATLAS server has to alert the user, the BP client can be used. It produces an audible tone with a certain frequency for 100 ms and is already included in the ATLAS server.

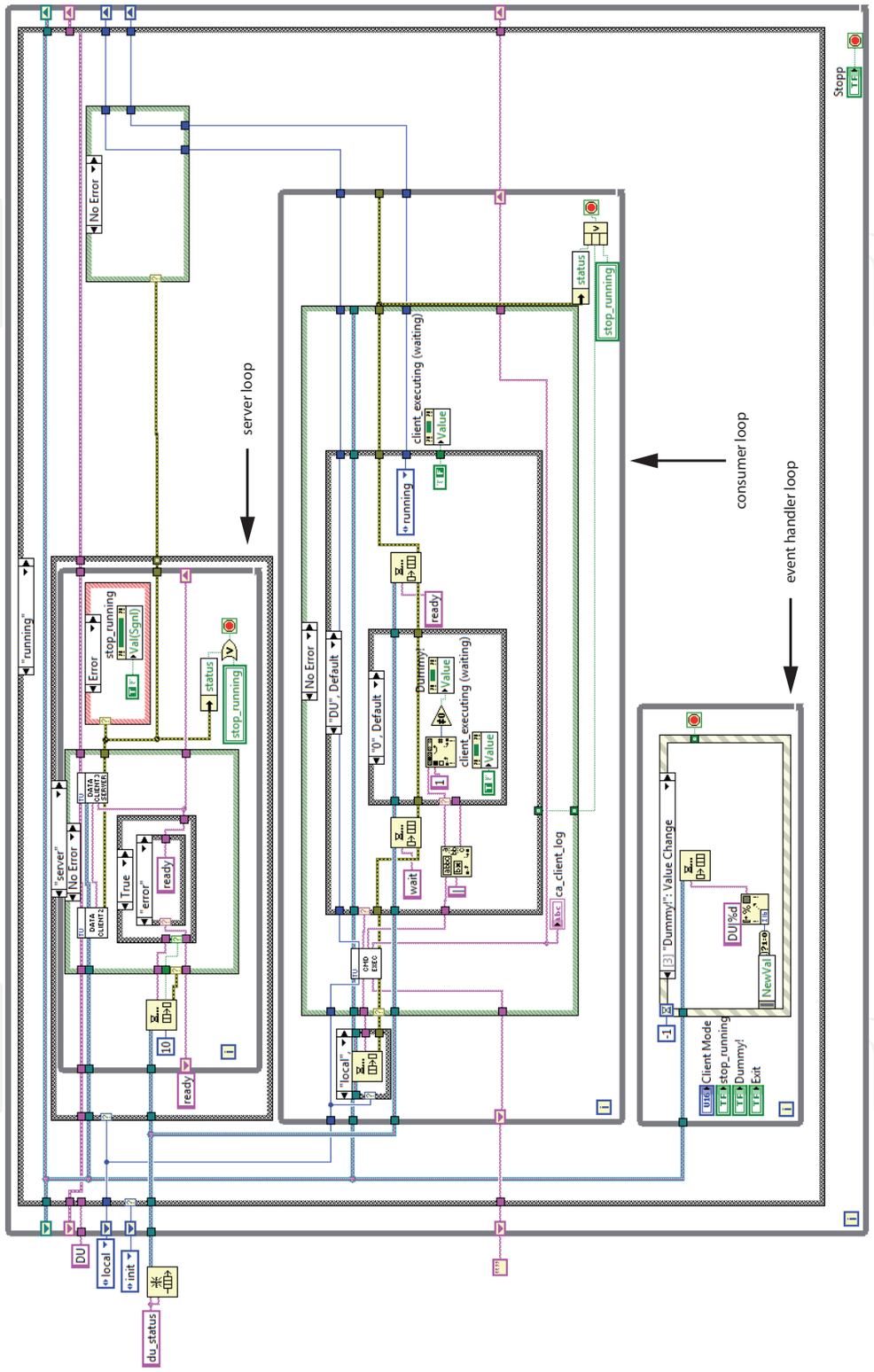


Fig. 5. Block diagram of a simplified ATLAS client, showing the *running* state. See the text for further information.

CA: Syringe pumps by Cavro Scientific Instruments (Sunnyvale, CA, USA).
Liquids can be aspired and dispensed from two liquid connections at different speeds. The ramping up and down to the programmed speed can also be programed.
DB: A digital board from TVE Elektronische Systeme GmbH (Vienna, Austria).
This board is a combination of a 16-bit-AD converter for a thermoelectrical cooled infrared detector. The integrated CPU supports features such as trigger delay, internal averaging and base line correction and is connected to the PC via USB.
DI: Micro dispenser from Picology AB (Sweden).
The piezo-element of the dispenser is controlled by burst signals from an Agilent 33120A function generator. The size of the dispensed drops and the number of drops per second can be adjusted.
DL: External Cavity Quantum Cascade Laser from Daylight Solutions Inc. (CA, USA).
The wavenumber and the intensity of the emitted light can be set among other laser parameters. The DL client also supports a scanning mode, where the emission wavenumber of the laser is tuned automatically.
GI: Miniplus 3 peristaltic pump from Gilson Inc. (Middleton, WI, USA).
The speed as well as the pumping direction can be controlled.
HV: CZE 1000R high voltage supply from Spellman (NY, USA).
It is supported via an DAC module from National Instruments. The output voltage of the module is transformed into high voltage output (10 V equal 30 kV).
LC: Waverunner 64-Xi oscilloscope from LeCroy Corp. (NY, USA).
Only the most important features such as setting V/div, timebase and trigger-parameters are implemented. Single pulses can be read and stored.
OP: FTIR spectrometers from Bruker Optik GmbH (Ettlingen, Germany).
This client connects to OPUS over a DDE connection and can trigger measurements utilizing predefined experiment files. Single and repeated measurements are supported.
TE: TDS 220 oscilloscope by Tektronix (OR, USA).
Only the most important features such as setting V/div, timebase and trigger-parameters are implemented. Single pulses can be read and stored.
UV: Capillary UV-Detector from Dionex (Sunnyvale, CA, USA).
The client allows to set four wavelengths for absorption measurements. It can autozero the absorbance values and record the absorbance over time.
VI: Valves from Vici AG (Switzerland).
Injection and selection valves are supported by this client.
WA: Wait client
The wait client allows a delay of the execution of the next command by a defined time period. It is also included in the ATLAS server as the BP client.
XY: Custom-built XY stage.
The XY stage is operated by a Trinamic TMCM-610 stepper motor control, which is controlled by the XY client. This client allows the user to move the stage for a certain distance by a predefined speed.

3. Sample applications

3.1 Example I: Quantification of lactate in aqueous solution using an External Cavity Quantum Cascade Laser

In clinical diagnostics numerous parameters, such as glucose, phosphate and lactate in blood have to be measured with high precision and accuracy in short time intervals. One

measurement principle suitable for such requirements is infrared spectroscopy. Using classical Fourier transform infrared spectrometry (Griffiths & Haseth, 2007) two approaches can be chosen. Measuring the solution in transmission requires low path lengths typically below 25 μm in the spectral region of 1000 – 1250 cm^{-1} due to the high absorbance of water in the region of interest. The measurement cell can suffer from bio-fouling on the window materials when biological samples are measured. Additionally, particles in the sample solution can cause cell-clogging in such small path lengths. The second suitable measurement method overcoming the clogging problem is attenuated total reflection (ATR). Using the ATR technique the light doesn't transmit through the sample solution but is coupled into an IR transparent crystal (e.g. Germanium). At the surface between the crystal and the sample total reflection of the light occurs and the evanescent field of the propagating light is reaching into the sample. This evanescent field interacts with the sample and is absorbed by the target molecules. Applying the ATR technique allows the use of flow cells on top of the crystal with bigger dimensions since the light doesn't need to penetrate the whole sample thickness. This advantage comes with the tradeoff that the penetration depth of the light into the sample is small compared to the transmission measurement. Therefore, one must expect higher limits of detection limits of detection and less sensitivity according to Lambert Beer's law (1). Additionally bio fouling on the crystal surface is an even bigger issue due to the low penetration depth.

$$A(\mathrm{cm}^{-1}) = -\log\left(\frac{I(\mathrm{cm}^{-1})}{I_0(\mathrm{cm}^{-1})}\right) = \varepsilon \cdot c \cdot l \tag{1}$$

In this formula I_0 denotes the measured intensity of the light reaching the detector without sample whereas I gives the intensity after the light was absorbed by the sample. ε[L/cm] is the molar absorption coefficient at a given wavenumber. c[mol/L] denotes the concentration of the analyte and l[cm] gives the path length of the measurement cell.
Increasing the optical path length of the measurement cell would solve both drawbacks, described above, at the same time. Larger cell dimensions would make the cleaning of the cell easier, accompanied by better sensitivity because of the increased interaction length. Replacing the IR spectrometer with a Quantum Cascade Laser (QCL) (Faist et al., 1994) enables the penetration of higher path lengths at one wavenumber due to the higher energy output of the laser compared to the light source of an IR spectrometer. Applying an External Cavity Quantum Cascade Laser (EC-QCL), which can shift its emission wavenumber over a range of 200 cm^{-1} gives a spectrum instead of a single absorbance value at a given wavenumber.
Using an EC-QCL enabled us to use an optical path length of 130 μm for the measurement of physiological relevant compounds in Ringer solution. Using ATLAS we realized an experimental setup for the measurement of liquid samples and tested it on lactate samples in Ringer solution.

3.1.1 Experiment

For our studies we combined an EC-QCL (tuning range: 1030 – 1230 cm^{-1}) from Daylight Solutions Inc. with a thermoelectrically cooled Mercury Cadmium Telluride (MCT) detector and a Sequential Injection Analysis (SIA) system for sample preparation (Brandstetter et al., 2010). The detector signal was digitized by a specially designed analogue to digital

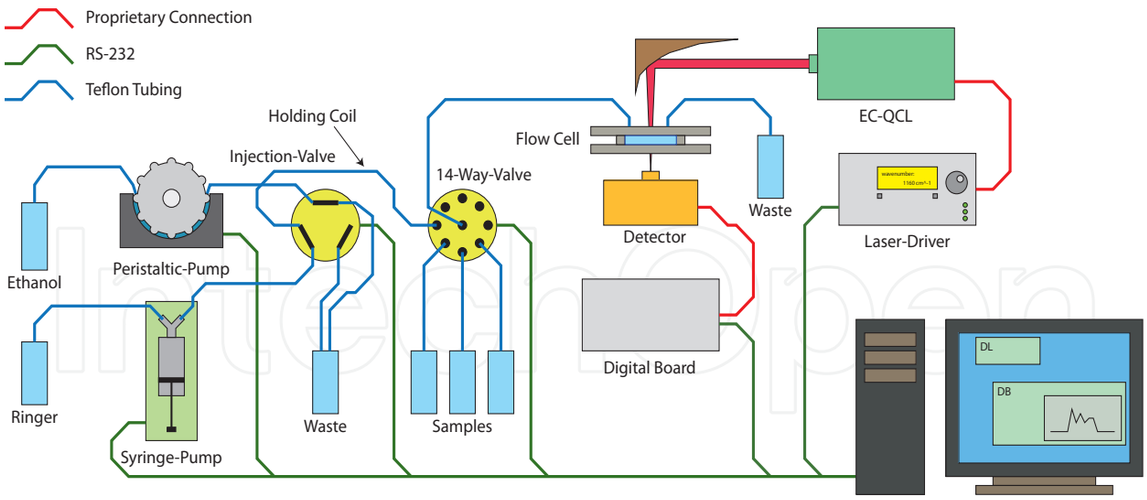


Fig. 6. Experimental setup for the measurement of lactate in Ringer solution using an External Cavity Quantum Cascade Laser.

converter (ADC) board which was read out by the *digital board* ATLAS client. Using a SIA system improves on manual sample injection by means of reproducibility and repeatability. Applying the SIA system also helps to reduce sample contaminations since the system can be cleaned automatically before a new sample is injected for the next measurement.

For this setup, shown in figure 6, the ATLAS clients for the digital board, the EC-QCL from Daylight Solutions Inc., the syringe pump from Cavro Inc., valves by Vici Inc. and the client for the Gilson Miniplus 3 peristaltic pump were connected to the ATLAS server. All hardware parts were connected to a notebook by RS-232 connections over USB-to-RS-232 converters. The peristaltic pump in the setup was only used if air bubbles were found in the tubings. In that case the system was purged with ethanol first to drive the bubbles out of the system and afterwards the ethanol flask was exchanged with Ringer solution manually and the tubings were filled with Ringer solution again.

In a first experiment lactate in Ringer matrix was pumped through the measurement spot repeatedly to examine the achieved flow profiles. Therefore, the whole system was filled with Ringer solution first. The laser parameters were sent by the client to the laser control unit and the emission wavenumber was tuned to 1130 cm^{-1}. After initiating the emission of light the data acquisition was started and the intensity of the detected signal was streamed onto the hard drive continuously. In the meantime the syringe pump picked up the lactate sample through the 14-way-valve into the holding coil. After switching the 14-way-valve back to connect the holding coil with the measurement cell the syringe pump dispensed the lactate sample through the measurement cell. The pickup and dispense procedure was repeated while the data acquisition continued. In figure 7 the obtained flow profile for two repetitions is plotted against time. Whenever the sample solution passes the measurement cell the measured intensity on the detector decreases due to absorption taking place in the cell. The obtained flow profiles are in good agreement with theory. Optimization of the pumped lactate volume and the used flow rates allowed adjustments of the system in such a way that, at the maximum of the absorption, undiluted sample was measured. Knowing the exact time when this maximum reached the flow cell was crucial for the following experiment.

To obtain a calibration curve for lactate in Ringer solution a second command sequence was programmed. Firstly Ringer solution was pumped through the measurement cell utilizing a

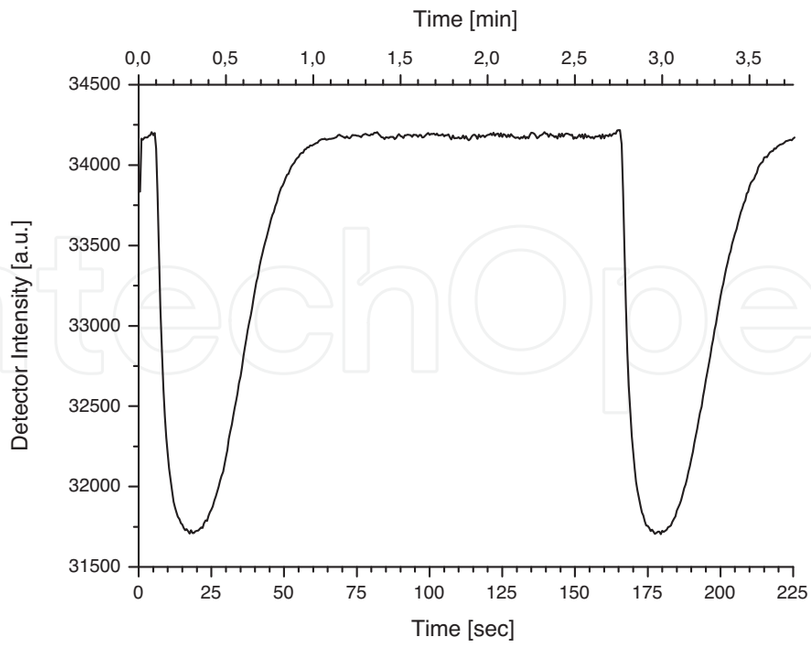
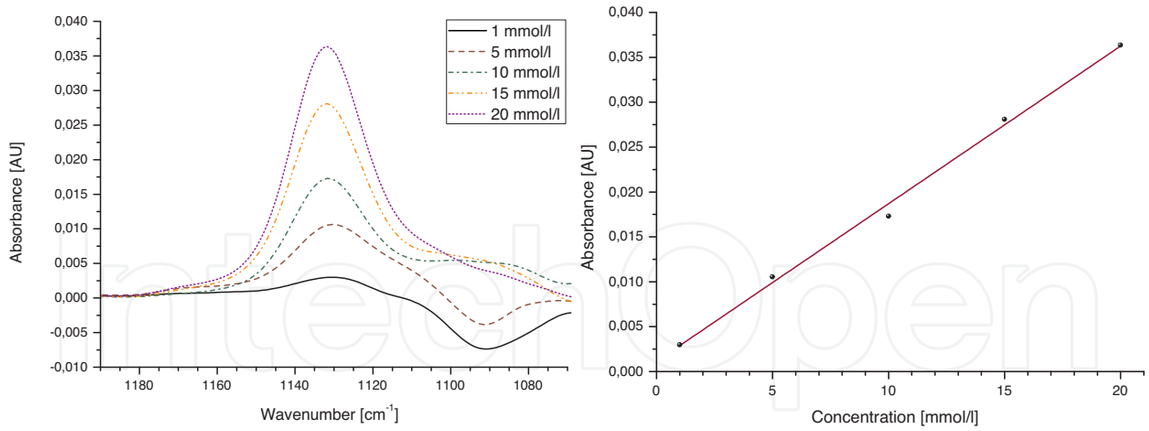


Fig. 7. Lactate solution was pumped through the measurement cell twice while the EC-QCL emitted light at 1130 cm^{-1} and the digital board read out the detector values.

Cavro syringe pump. After the cell was purged the flow was stopped and the ATLAS server sent a signal to the digital board to start recording. Afterwards the EC-QCL was started by the server to scan over a wavenumber region ranging from 1030 to 1230 cm^{-1}. This measurement served as $I_0(cm^{-1})$ in equation (1). Then the valves switched positions allowing the syringe pump to pick up a sample of lactate into the holding coil. After the 14-way-valve was switched to connect the holding coil with the flow cell the sample was pumped into the measurement spot. Subsequently, the flow was stopped at the absorption maximum determined earlier. Now the EC-QCL received a trigger signal from the server to start another measurement, which gave $I(cm^{-1})$ in equation (1). Then the valve of the syringe pump was switched to the opposite direction and picked up Ringer solution from the storage flask. After switching the valve back the whole system was purged again. This step was repeated twice to flush any possible remaining lactate out of the measurement spot.
The sequence described above was automatically repeated for all five sample concentrations by changing the 14-way-valve to the appropriate position in each of the measurement runs. Afterwards the absorbance spectra were calculated manually and the data shown in figure 8 were plotted in Origin. Calculating a linear calibration function for the measured absorbances at the band maxima gave a R^2 value of 0.996.
In conclusion we can report that setting up a SIA system using ATLAS can by achieved easily. The integration of the EC-QCL and the data acquisition electronics into the SIA sequences was simple too after the ATLAS clients for the laser and the ADC board had been developed.

3.2 Example II: Using IR spectroscopy as a detector for liquid chromatography

Liquid chromatography (LC) is a widely used technique in analytical chemistry to separate mixtures of substances before detecting and quantifying them. The sample is injected onto a



(a) Recorded spectra (b) Calibration line

Fig. 8. 5 different concentrations of lactate in Ringer solution were measured and the absorbance values at 1131. 7 cm^{-1} are plotted as a calibration curve giving a R^2 value of 0.996.

separation column where different molecules in the analyte mixture are retained for a varying time dependent on the nature of the column and the analyte itself. The analytes are driven through the column by a single solvent mixture, called isocratic mode, or the composition of the solvent mixture is changed during the elution, which is called gradient mode. Classical detectors used for detection in LC are mass spectrometry (MS) and UV-VIS spectroscopy. IR spectroscopy was not commercialized until today as a detection principle for LC because the solvents used cause high absorption in the infrared spectral region and overlap with the very small absorptions of the analyte molecules. However IR spectroscopy has some advantages as detection principle. First it is a non destructive detection principle, compared to MS. Second, the IR spectrum can be seen as a fingerprint of a target molecule which simplifies the identification of target molecules compared to UV-VIS spectroscopy.

Using IR spectroscopy as the detection principle in LC can be done in two ways. In the online method the exit of the separation column is connected to a flow cell and IR spectra are recorded during the elution of the analytes. Since the time resolution of the chromatogram depends mainly on the time needed to record a spectrum a tradeoff has to be made here. Increasing the spectral averaging will give better signal to noise levels and therefore better sensitivity. However, the time resolution of the chromatogram will decrease and two peaks, which are eluted shortly after each other, might be not detected separately. The strong absorption of the solvent can be easily subtracted in an isocratic eluation whereas background correction algorithms have to be used in gradient elution (Quintás et al., 2009a;b).

The second possible method is based on physically eliminating the solvent from the sample and the IR spectra are measured afterwards as traces on an IR transparent substrate. This method enables averaging of as many spectra as needed without decreasing the time resolution of the chromatogram because the time resolution is mainly determined by the deposition method of the eluents on the measurement substrate.

Using ATLAS we have coupled a Dionex Ultimate 3000 capillary LC system to a micro dispenser (Laurell et al., 1999), which was used for eluate deposition on CaF_2 windows followed by subsequent IR detection. The setup described below was already used by manually controlling all components before ATLAS became available and proved capable of measuring pesticides down to concentrations of 2 μg/mL (Armenta & Lendl, 2010).

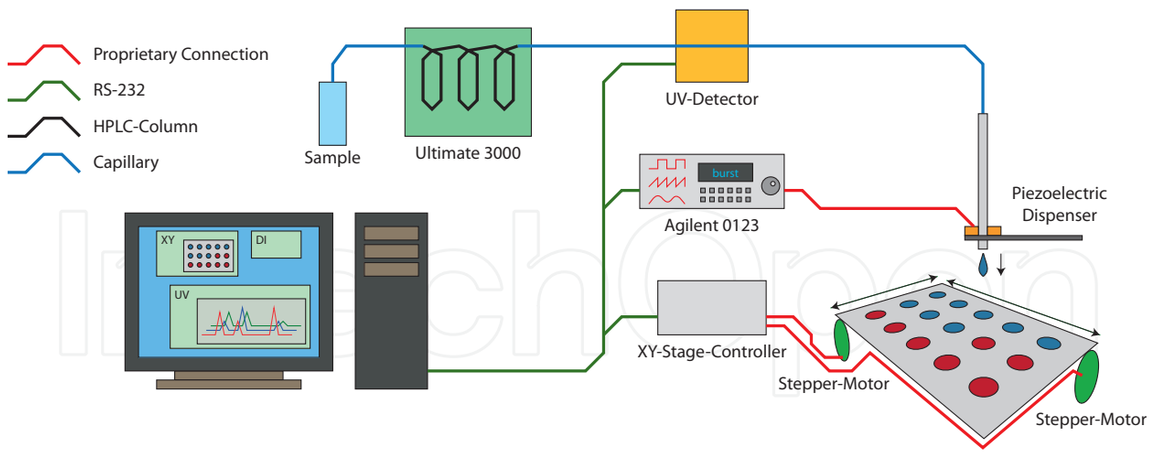


Fig. 9. Experimental setup for the coupling of a liquid chromatography system with offline IR detection utilizing a micro dispenser for sample deposition.

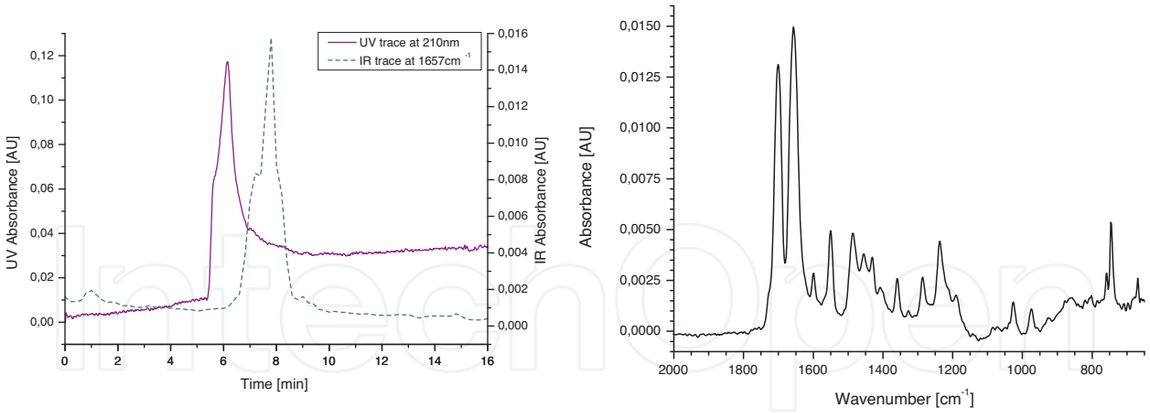
3.2.1 Experiment

Our system, as shown in figure 9, was built combining an Ultimate 3000 capillary LC system (Dionex, Sunnyvale, CA, USA), which is not implemented in ATLAS yet, a Dionex UV detector, a home built XY stage and a micro dispenser from Picology AB (Bjaerred, Sweden). The UV detector was configured and read out via a RS-232 connection by an ATLAS client. The stepping motors of the XY stage were connected to a controller board (Trinamic TMCM-610) which was controlled by another ATLAS client. The piezo crystal of the micro dispenser was actuated by a specially designed electronics board which amplified a burst signal generated by an Agilent 33120A function generator. The burst period, giving the number of droplets dispensed each second, and the burst amplitude, giving the volume of each dispensed drop, were set by a dedicated dispenser ATLAS client. The droplets of the eluent were dispensed onto a heated plate of IR transparent CaF_2 mounted on the XY stage moving the plate constantly during the deposition.

A typical experiment consisted of the following steps. First the injection of the sample onto the LC column was initiated manually since the LC system is not implemented into the ATLAS system yet. Immediately afterwards a preprogrammed ATLAS control sequence was started containing the following steps:

- The desired wavelengths for the UV measurement are set
- The absorbance signal of the UV detector is set to zero
- The dispenser parameters are set
- The movement of the XY stage is initiated at a certain speed and a programmed distance to travel
- The UV measurement is started
- The dispensing of the droplets is triggered

After the elution of the injected substances was completed the CaF_2 plate was removed from the XY stage and was analyzed on a Hyperion 3000 IR microscope from Bruker Optics (Ettlingen, Germany). The plate was measured in transmission mode applying a line scan along the deposited material.



(a) Traces of the eluted caffeine sample

(b) IR spectrum of caffeine at the maximum of the eluted peak

Fig. 10. Measurement results from the coupling of a liquid chromatography system with offline IR detection.

First results testing the ATLAS system were obtained by injecting 1 μL of a 60 ppm solution of caffeine onto a C18 separation column. A mixture of water and acetonitrile (50:50 % v/v) with 0.1% acetic acid served as the mobile phase flowing at a speed of 2 μL/min. The UV detector was set to measure at 200, 210, 250 and 350 nm wavelengths every 500 ms for 15 min and the XY stage was moved with 25 μm/s for a distance of 3 cm. The dispense rate was set to 25 drops/s.
The obtained chromatogram of caffeine is plotted in figure 10. The time delay between the UV and IR measurement can be explained by the distance between the UV measurement cell and the micro dispenser. The IR trace was obtained by integration of the caffeine band located at 1657 cm^{-1}. The plotted IR spectrum of caffeine represents the maximum of the IR trace.
Concluding we can state that using ATLAS enabled us to setup a measurement system coupling a LC system with online UV detection and offline IR detection. Inclusion of the Dionex Ultimate 3000 chromatograph would allow for fully automated measurements with this system eliminating the manual injection of the sample.

4. Conclusion and Outlook

In this chapter we introduced ATLAS, our LabVIEW programmed lab automation system. While LabVIEW is routinely used in many labs to control laboratory equipment, usually these programs are ad-hoc solutions which are used for one type of experiment only and are never published. What sets our system apart from previously reported works are its expandability and flexibility. Through its Server-Client layout the ATLAS system can easily be expanded to be able to control a wide range of lab equipment – any instrument, that can be controlled by LabVIEW, can be controlled by our program – and experiment parameters can be easily changed.
To give the reader an idea of possible applications of ATLAS we described two ATLAS controlled experiments, which would have been very inconvenient to do manually, but were easily implemented with the help of the presented lab automation system.
In the future we plan to extend our program by the ability to send measurement data from clients to the server and have the server make decisions based on this data. Furthermore, the

final version and the source code of ATLAS will be made available on our website (http://www.cta.tuwien.ac.at/cavs), to allow other users to use and improve it.

5. Acknowledgment

The authors want to thank Eva Aguilera Herrador, Julia Kuligowski and Markus Brandstetter for supplying us with their measurement data which are presented in section 3.

6. References

Armenta, S. & Lendl, B. (2010). Capillary liquid chromatography with off-line mid-ir and raman micro-spectroscopic detection: Analysis of chlorinated pesticides at ppb levels, *Anal Bioanal Chem* 397(1): 297–308.

Barzin, R., Shukor, S. & Ahmad, A. (2010). New spectrophotometric measurement method for process control of miniaturized intensified systems, *Sensors and Actuators, B: Chemical* 146(1): 403–409.

Brandstetter, M., Genner, A., Anic, K., Lendl, B. (2010). Tunable external cavity quantum cascade laser for the simultaneous determination of glucose and lactate in aqueous phase, Analyst, available online, doi: 10.1039/c0an00532k.

Dominguez, R., Muñoz, R. & Araiza, H. (2010). Automated analytical system based on the sia technique, *Pan American Health Care Exchanges, PAHCE 2010* pp. 117–119.

Faist, J., Capasso, F., Sivco, D., Sirtori, C., Hutchinson, A. & Cho, A. (1994). Quantum cascade laser, *Science* 264(5158): 553–556.

Griffiths, P. R. & Haseth, J. A. D. (2007). *Fourier Transform Infrared Spectrometry*, Wiley-Interscience.

Jitmanee, K., Teshima, N., Sakai, T. & Grudpan, K. (2007). Drc™ icp-ms coupled with automated flow injection system with anion exchange minicolumns for determination of selenium compounds in water samples, *Talanta* 73(2): 352–357.

Laurell, T., Wallman, L. & Nilsson, J. (1999). Design and development of a silicon microfabricated flow-through dispenser for on-line picolitre sample handling, *J. Micromech. Microeng* 9(4): 369–376.

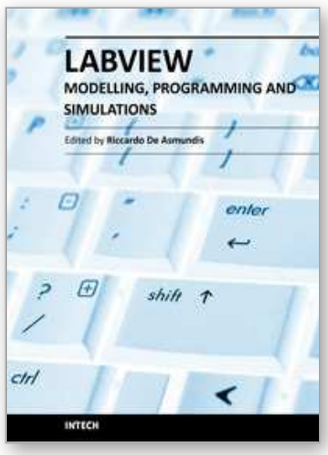
Quintás, G., Kuligowski, J. & Lendl, B. (2009a). On-line fourier transform infrared spectrometric detection in gradient capillary liquid chromatography using nanoliter-flow cells, *Analytical Chemistry* 81(10): 3746–3753.

Quintás, G., Kuligowski, J. & Lendl, B. (2009b). Procedure for automated background correction in flow systems with infrared spectroscopic detection and changing liquid-phase composition, *Appl Spec* 63(12): 1363–1369.

Růzicka, J. (1981). *Flow Injection Analysis*, Vol. 62 of *Chemical Analysis*, Wiley, John & Sons, Incorporated.

Růzicka, J. (1992). The second coming of flow-injection analysis, *Analytica Chimica Acta* 261(1- 2): 3–10.

Wagner, C., Armenta, S. & Lendl, B. (2010). Developing automated analytical methods for scientific environments using labview, *Talanta* 80(3): 1081–1087.



Modeling, Programming and Simulations Using LabVIEW™ Software

Edited by Dr Riccardo De Asmundis

ISBN 978-953-307-521-1
Hard cover, 306 pages
Publisher InTech
Published online 21, January, 2011
Published in print edition January, 2011

Born originally as a software for instrumentation control, LabVIEW became quickly a very powerful programming language, having some characteristics which made it unique: simplicity in creating very effective User Interfaces and the G programming mode. While the former allows for the design of very professional control panels and whole applications, complete with features for distributing and installing them, the latter represents an innovative way of programming: the graphical representation of the code. The surprising aspect is that such a way of conceiving algorithms is extremely similar to the SADT method (Structured Analysis and Design Technique) introduced by Douglas T. Ross and SofTech, Inc. (USA) in 1969 from an original idea by MIT, and extensively used by the US Air Force for their projects. LabVIEW enables programming by implementing directly the equivalent of an SADT "actigram". Apart from this academic aspect, LabVIEW can be used in a variety of forms, creating projects that can spread over an enormous field of applications: from control and monitoring software to data treatment and archiving; from modeling to instrument control; from real time programming to advanced analysis tools with very powerful mathematical algorithms ready to use; from full integration with native hardware (by National Instruments) to an easy implementation of drivers for third party hardware. In this book a collection of applications covering a wide range of possibilities is presented. We go from simple or distributed control software to modeling done in LabVIEW; from very specific applications to usage in the educational environment.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Christoph Wagner, Andreas Genner, Georg Ramer and Bernhard Lendl (2011). Avanced Total Lab Automation System (ATLAS), Modeling, Programming and Simulations Using LabVIEW™ Software, Dr Riccardo De Asmundis (Ed.), ISBN: 978-953-307-521-1, InTech, Available from: http://www.intechopen.com/books/modeling-programming-and-simulations-using-labview-software/avanced-total-lab-automation-system-atlas-

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元

Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen

IntechOpen