# Python4Delphi as application server

**Dmitrii V. Konnov**
ORCID 0009-0009-0935-4626; konnov72@knights.ucf.edu

University of Central Florida, 4000 Central Florida Blvd. Orlando, Florida, 32816

## ABSTRACT

The article describes an experimental project in which a cloud service was created that combines software technologies and languages of heterogeneous nature, such as Delphi, Python, and JavaScript. The goal of the experiment was to close the gap between Python and Delphi and provide Delphi developers the ability to deliver feature-rich multi-user Windows applications to a web browser using all the power of Python. The article presents the implementation of a software platform that combines the advantages of rapid development of applications for the Web in the Delphi environment, and the possibility of linking the Application Server with the Python interface on the Windows platform. Python, as a language originating from the Linux environment, provides powerful access to cloud-based scientific computing and artificial intelligence. To ensure the platform independence of the user from devices and the operating system, as well as the absence of installation requirements, it was proposed to implement the client part of the web application based on uniGUI Delphi components. Thus, Delphi developers can create systems that provide the full range of cloud computing capabilities, not limited to VCL capabilities. The article also provides an overview of the significance of Parallel Computing and the rationale for building a system using them. As a means of isolating user processes on the server side, the implementation of the interaction between Delphi and Python through IPC is proposed. As an example of using the Python cloud service, the HITRAN Molecular Spectroscopy database was chosen. The result of the research is a demonstration software in the form of web server that delivers high quality Spectra charts into Client's browser. A comparative description of the graphical user interface in a web browser is given in comparison with the matplotlib provided by Python. Proposal for integration with NVIDIA CUDA technology based on the researched architecture is made. The novelty of the research lies in the heterogeneity of the combined technologies: Delphi (P4D, uniGUI, IPC), Python (HAPI), and JavaScript (Bokeh).

**Keywords:** Parallel computing, cloud computing, application server, HITRAN, Delphi, Python, Bokeh, IPC, Python for Delphi, uniGUI, CUDA.

***For citation*: © Konnov D.V 2023**

## INTRODUCTION

According to the "IEEE Spectrum", the 2022 Official Journal of the Institute of Electrical and Electronics Engineers (IEEE) [1], the Python programming language [2] has been and remains the world's most popular programming language today.

It is no exaggeration to say that the development of scientific software and artificial intelligence today is mainly carried out in this language. The Delphi, Object Pascal language, once especially popular in the scientific community, occupies only 0.69% of the market, although it has been and remains the unsurpassed leader in Rapid Application Development [3] for desktop applications and databases, and is especially popular in Europe. The use of scientific applications and services written in Python for obvious reasons, such as historically strong dependence on the Linux OS, insufficiently advanced support for the graphical user interface, and environment specificity, unfortunately, is not available to a wide range of users. The purpose of this study was to fill the gap between Python and Delphi programming languages and the availability to deliver state of the art applications to the end user. To enable users, regardless of their platform, to access data and services developed in Python, to be able to scale the developed systems. The requirements for system performance are growing every year, so the architecture of the project being developed was initially decided to be developed based on the Parallel Computing paradigm.

## THE PURPOSE AND THE OBJECTIVES OF THE STUDY

The goal of the project is to provide the Delphi developer with a solution that allows them to implement scalable multi-user systems using the full power of Python. It's worth noting the pros and cons of this approach, as many may notice that everything can be easily implemented in Python inside and out [4], from working with memory to outputting HTML to the browser. In this sense, Python has proven itself as a universal language and its application is especially focused on the scientific field and in the field of artificial intelligence. Delphi is a high-level language that covers mainly business and desktop applications for databases.

That is why the author of the project set out to fill the gap between Delphi and Python.
Let's note some of the shortcomings of Python, which, in our opinion, are significant.

1) for larger projects, labor costs will increase in an area where Python cannot surpass Delphi. Namely - fast and high-quality GUI development.

2) a developed infrastructure on the Windows platform, with which Delphi has a much deeper integration than Python. For example, MS-SQL, Oracle, support for a wide range of DBMS.

3) In the manufacturing sector, the vast majority of top-level SCADA, MES, ERP systems are based on the Windows platform and historically have a closer connection with Delphi since Microsoft has been successfully promoting COM technology for a long time.

COM technology is being actively replaced by .NET these days, but this does not detract from the merits of Delphi as a more convenient tool for development in the field of industrial automation in relation to any other languages.

## FORMULATION OF THE PROBLEM

Thus, to fill the gap between Delphi and Python, it is necessary to implement a binding library that will allow you to call Python services in a Win32/64 environment. Such components are available in the Delphi environment and have long proven themselves - the library is called Python4Delphi, an open-source project. Python for Delphi (P4D) [12] is a set of free components that wrap up the Python DLL into Delphi and Lazarus (FPC). They let you easily execute Python scripts, create new Python modules and new Python types. You can create Python extensions as DLLs and much more. P4D provides different levels of functionality:

- Low-level access to the python API
- High-level bi-directional interaction with Python
- Access to Python objects using Delphi custom variants (VarPyth.pas)
- Wrapping of Delphi objects for use in python scripts using RTTI (WrapDelphi.pas)
- Creating python extension modules with Delphi classes and functions

At the time of publication of this article, the author is not aware of any other alternative components of interacting with Python from Delphi.

However, the main problem that stops developers from developing multi-user systems when using Python4Delphi is that this library does not support multi-session explicitly. By design, Python4Delphi is a singleton, a wrapper over pythonXXX.dll. Only one copy of a DLL can be loaded into process memory, and that DLL serves only one Python session. This is the interface provided by Python developers on the Windows platform. Hence the following possible ways to implement the above problem, the following are possible use cases:

- **Python C API**: Python provides a C API that allows you to embed Python into your C/C++ applications. Since Delphi can interface with C/C++ code through its Foreign Function Interface (FFI), you could potentially use the Python C API directly from Delphi. This approach requires a solid understanding of both the Python C API and Delphi's FFI capabilities.
- **COM Automation**: Python can expose its objects through COM (Component Object Model) interfaces, allowing you to access Python objects and functions from Delphi as COM objects. This method involves creating Python COM components that you can then access from your Delphi application using COM interfaces.
- **Execute Python as External Process**: You can run Python scripts as external processes from within Delphi using the **ShellExecute** or **CreateProcess** functions. While this approach doesn't directly integrate Python into your Delphi application, it allows you to interact with Python scripts by passing input and reading output.
- **Inter-process Communication**: You can establish inter-process communication between a Python process and a Delphi process. This could involve using methods like sockets, named pipes, or other communication mechanisms to exchange data and commands between the two processes.

Due to the above limitations imposed by pythonXXX.dll to build a multi-user system, the author conducted a series of experiments and decided to use the IPC (inter-process communication) mechanism. Thus, separating each user connected to the system into a separate Windows process. This approach is a classic, such as the long-proven CGI interface in Linux systems when developing for the Web.

## APPROACHES AND IMPLEMENTATION METHODS

An *Application Server* is a software tool specifically designed to efficiently execute procedures, such as programs, mechanisms, or scripts, forming the foundation for the development of scalable and parallel systems. It comprises a set of components that software developers can access through a platform-defined API (Application Programming Interface) [10,11,6]. In this case, the chosen server platform is Windows.

The system's architecture, depicted in Fig. 1, demonstrates the use of an application server. The advantages of utilizing Windows servers include their relative ease of administration, and access to a wealth of information, manuals, and software resources. Additionally, if the company's software ecosystem relies on Microsoft libraries and components, the Windows server becomes an essential and indispensable component. Delphi as a language development tool is no doubt to be native Windows was chosen as the primary tool [11].

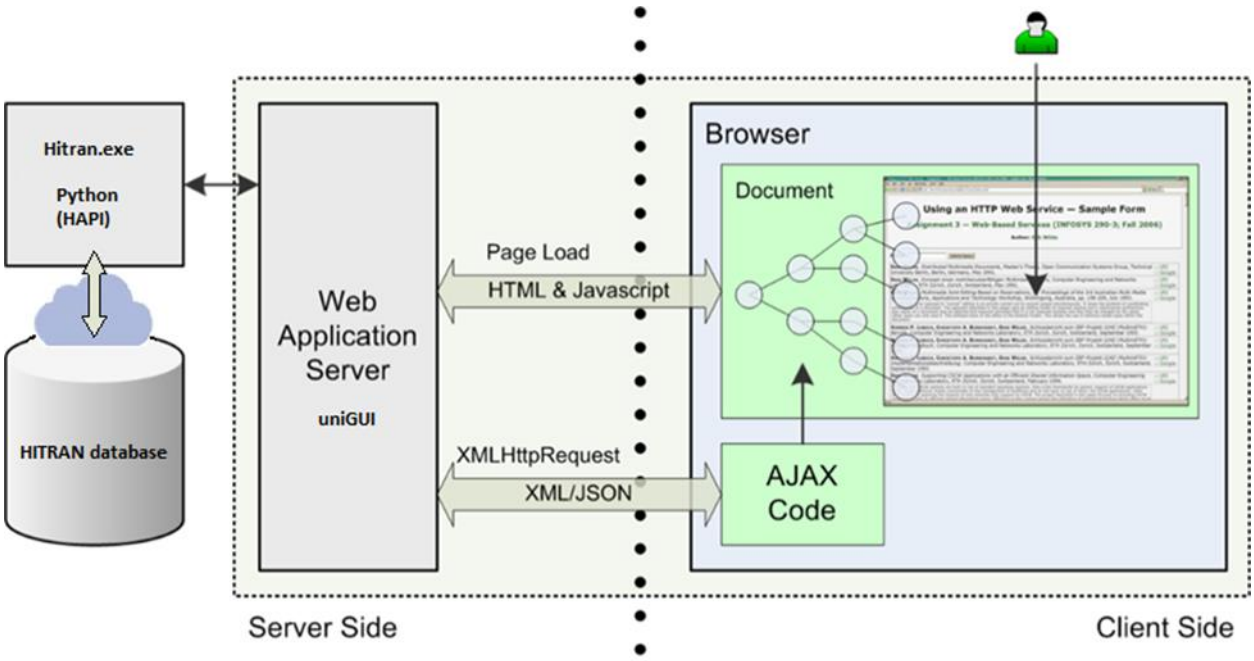


*Fig 1.* **System architecture.**

## INTERACTION WITH PYTHON SESSION PROCESS USING IPC

Every POSIX-compliant system provides an IPC inter-process communication mechanism [13]. Windows supports the following IPC mechanisms: Clipboard, COM, Data Copy, DDE, File Mapping, Mailslots, Pipes, RPC, and Windows Sockets. In this project, the Mailslots mechanism was used to implement IPC. The IPC server (Hitran.exe) is a VCL application utilizing the abilities of P4D components.

Let's consider the scenario of interaction with the IPC server from the client side.

The source code of this project, written by the author of this article, is available under the MIT license and can be downloaded from:

https://github.com/dima72/Python-UniGUI-HITRAN-API [14]

First, to pass data across process border, the client needs to create an IPC server process with which it is supposed to interact, using the CreateProcess WinAPI method.

```
CmdLine := FileName + ' ' + IPCServName + ' ' + ClientName;

CreateProcess(nil, @CmdLine[1], nil,  nil,  False,   CREATE_NEW_CONSOLE or
NORMAL_PRIORITY_CLASS,  nil,   nil,  StartupInfo, FProcessInfo)
```

IPCServName – unique identifier of the IPC server, ClientName – unique client identifier

IPCServName and ClientName parameters are passed on the command line so that the server executable understands that it is being launched for an IPC dialog communication and that it internally creates the corresponding MailSlot with a unique name, for example, IPCServName + ClientName. From this point on, the IPC server will switch to the mode of waiting for commands from the client side and will listen to this MailSlot.

To send a message to an IPC server, a client needs to create a MailSlot with the same name (IPCServName + ClientName) and get the Handle of the file.

```
ServerHandle := CreateFileW(PWideChar('\\.\mailslot\' + IPCServName + ClientName),

GENERIC_WRITE,          FILE_SHARE_READ, nil, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, 0);
```

Using ServerHandle, sending IPC data to the server from the DataBuffer is carried out by the WriteFile method.

```
WriteFile(ServerHandle, DataBuffer ^, MemoryStream.Size, BytesWritten, nil)
```

The client sends a command to the IPC server and immediately switches to the response mode:

```
if GetMailslotInfo(ServerHandle, nil,
BufferSize, @MessageCount, nil)  then
begin
  if (BufferSize <> MAILSLOT_NO_MESSAGE) and (MessageCount > 0)  then
  begin  …
    if ReadFile(ServerHandle, DataBuffer^, BufferSize, NumberOfBytesWritten, nil) then
….
```

If ReadFile call is successful data from the server comes to the DataBuffer

## IMPLEMENTATION OF APPLICATION SERVER USING uniGUI

The application server is implemented based on uniGUI components from FMSoft [15]. The Web Application Framework takes the development experience to the next level by allowing Delphi developers to create, design and debug web applications in an IDE using a unique set of visual components. Each component of the framework provides the same functionality as the Delphi VCL visual components, which creates a comfortable development environment that is very close to developing VCL applications with a small learning curve. Web applications can be run and debugged directly in the RAD Studio Delphi IDE, making the development process straightforward. uniGUI provides new opportunities for web application development where performance is the focus. It allows developers to focus on the application's business logic instead of spending time on web application development details such as working with HTML, JavaScript, XML templates, and other web technologies. This makes uniGUI a great tool for small development teams with limited resources, as well as large teams building enterprise web applications on a tight schedule. It uses the powerful Sencha Ext JS library for client-side rendering. These libraries are among the industry leaders in building Single Page Applications (SPAs). It combines the power of Ext JS with the power of Delphi's RAD to provide the fastest way to build SPA applications in Delphi. uniGUI exposes Ext JS classes as a custom set of Delphi controls, allowing developers to create powerful web applications.

Deployment is an important step in the web application development process. Developers have the option to choose from the available deployment options such as Windows Service, Standalone Server, or ISAPI module. ISAPI modules can be deployed using Microsoft IIS, Apache Web Server for Windows, or any other compatible web server that supports ISAPI. Consider the simplest and most practical way - Standalone Server. In this mode, the application server operates as a desktop application. It is also used for debugging the application. After launching the application executable file, it minimizes to an icon on the taskbar (Fig. 2) and will continue to run until the user manually terminates it.

**uniGUI Standalone Server**

*Fig. 2.* **uniGUI Standalone Server, system tray Icon.**

Standalone Server can be accessed from a browser by simply typing: http://localhost:8077
When deploying a server in the Cloud, it is necessary to correctly configure the forward ports of the local router and organize the redirection of incoming HTTP requests to the address of the physical or virtual machine on which the Application Server is running, as well as properly configure the DNS records of the Internet hosting provider so that incoming requests by domain name are automatically redirected to the

Application Server. As an example of integration with Cloud Computing in this project, the subject of Molecular Spectroscopy was chosen using the HITRAN database as an example. An example of a graphical user interface is shown in Fig. 3.
HITRAN is an acronym for High-Resolution Molecular Absorption Transmission Database. The above database is a set of spectroscopic parameters that are used by various computer systems to predict and model the transmission and emission of light in the atmosphere. The database is a long-term project started by the Cambridge Air Force Research Laboratories (AFCRL) in the late 1960s in response to a need for detailed knowledge of the infrared properties of the atmosphere. [16]
The HITRAN Application Programming Interface (HAPI) [17] is a set of Python subroutines whose purpose is to provide remote access to the functions and data provided by the *HITRANonline* service. Currently, the API can download, filter, and process line-by-line spectral transition data from a huge variety of different gas molecules. The main purpose of this API is to extend the functionality of the main engine to calculate spectra using several types of line shapes, including the flexible HT profile (Hartmann-Tran) and optionally taking into account instrumental functions.

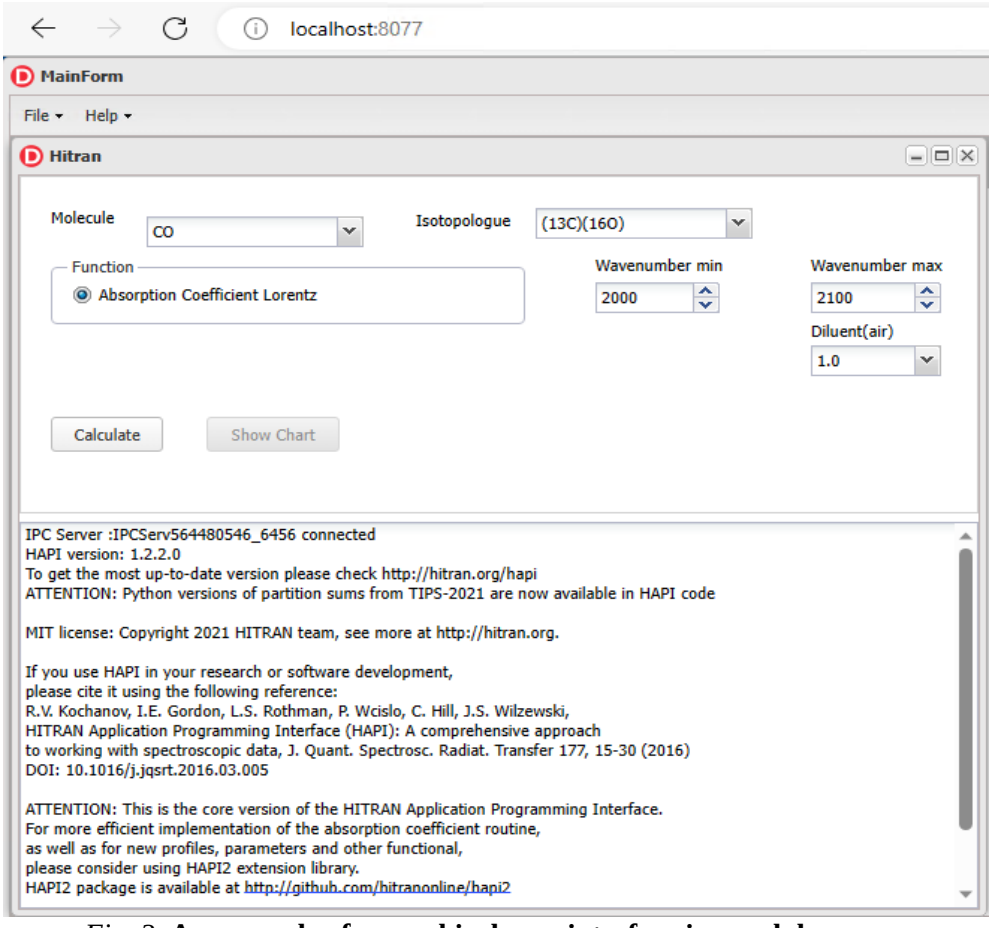


*Fig. 3.* **An example of a graphical user interface in a web browser.**

Each API function is represented by a Python function that takes a set of arguments that describe the parameters that define the task. HAPI is installed directly into the Python environment using:

*pip install hitran-api*

**Interaction with cloud computing on the example of the HITRAN database**

The HITRAN data server is physically hosted by Harvard University and HAPI accesses it via REST requests, however, given the high bandwidth of the Internet these days, this creates absolutely no problems with the delay in issuing HITRAN data by the Application Server to the user interface. As noted earlier, on the Application Server the IPC server (Hitran.exe) is a VCL application utilizing the abilities of P4D components. When the **Hitran.exe** is initializing, the initial Python script is executed by the PythonEngine

component, which is the result of Python session creation and HAPI libraries initialization. The result of HAPI initialization is displayed in the executable's Memo, see Fig. 3. Since then, the IPC server is listening for commands from the IPC client.
The Python script command from User enters Hitran.exe as a string, parameters are formed and executed by the PythonEngine.ExecString(Command) method;
See the example of the call *absorptionCoefficient_LorentzCalc* [18] of HAPI (Fig. 4.)

```
procedure TMainForm.ServerRecieveIpcData(Sender: TObject; var ClientName:
  WideString; var ClientWaitingForResponse: Boolean; var Data: Pointer);
var
  ResponseData: TResponseData;
  ScriptSL: TStringList;
  Command: string;
begin
  try
    ScriptSL := TStringList.Create;
    try
      ScriptSL.Text := TData(Data^).Text;
      if ScriptSL.Count > 0 then
      begin
        if ScriptSL.Strings[0] = 'absorptionCoefficient_LorentzCalc' then
        begin
          Command := 'absorptionCoefficient_LorentzCalc("' +
            ScriptSL.Strings[1] + '","' + ScriptSL.Strings[2] + '",' +
            ScriptSL.Strings[3] + ',' + ScriptSL.Strings[4] + ',"'+
            ScriptSL.Strings[5]+ '",' + ScriptSL.Strings[6] + ',' +
            ScriptSL.Strings[7]+ ',' + ScriptSL.Strings[8] + ')';
          PythonEngine.ExecString(Command);
        end;
      end;
    finally
      ScriptSL.Free;
    end;
```

*Fig. 4.* **Execution of Python script by PythonEngine(P4D) component.**

Work scenario:

- The user requests to generate the corresponding spectral line in the browser by selecting the appropriate options.
- The application server, having received a request, creates an instance of Hitran.exe, establishes an IPC session with it, and sends the user's request.
- Hitran.exe creates a Python session and calls the API function, the result of which is a JSON data set saved to a local file.
- The name of the resulting file is already known, it consists of the session number, client ID, molecule name, and other parameters that the client has already passed by clicking the "Calculate" button, for example, 1775160906_9776_H2O_1_2_air_1_0_2000_2100_Chart.txt.
- Thus, when the client presses the “Show Chart” button in the browser, the Application Server returns to it a page generated with JavaScript in mind, there is already a generated file name substituted in the Bokeh.AjaxDataSource. The client browser, having received the HTML, activates JavaScript on the page, which eventually loads data from the web server using Bokeh.AjaxDataSource.
- When a client session ends, the corresponding Hitran.exe process is unloaded from memory.

## DISCUSSION OF THE RESULTS

The system has been piloted and has shown that multiple users can work with the same server without interfering with each other at the same time. Thus, the main goal of the project as a multi-user system has been achieved.

A reference project in the field of HITRAN that works on the Web is a project of Tomsk University [19] or a project [20] that works only on a desktop only in single-user mode. Both projects are closed sources. The GUI Python is also known as an open-source project [21], but it only works on the desktop and single-user.
Thanks to the openness of the source code and the implementation of the multiplayer mode, Author believes that the project described in this article has many development prospects.

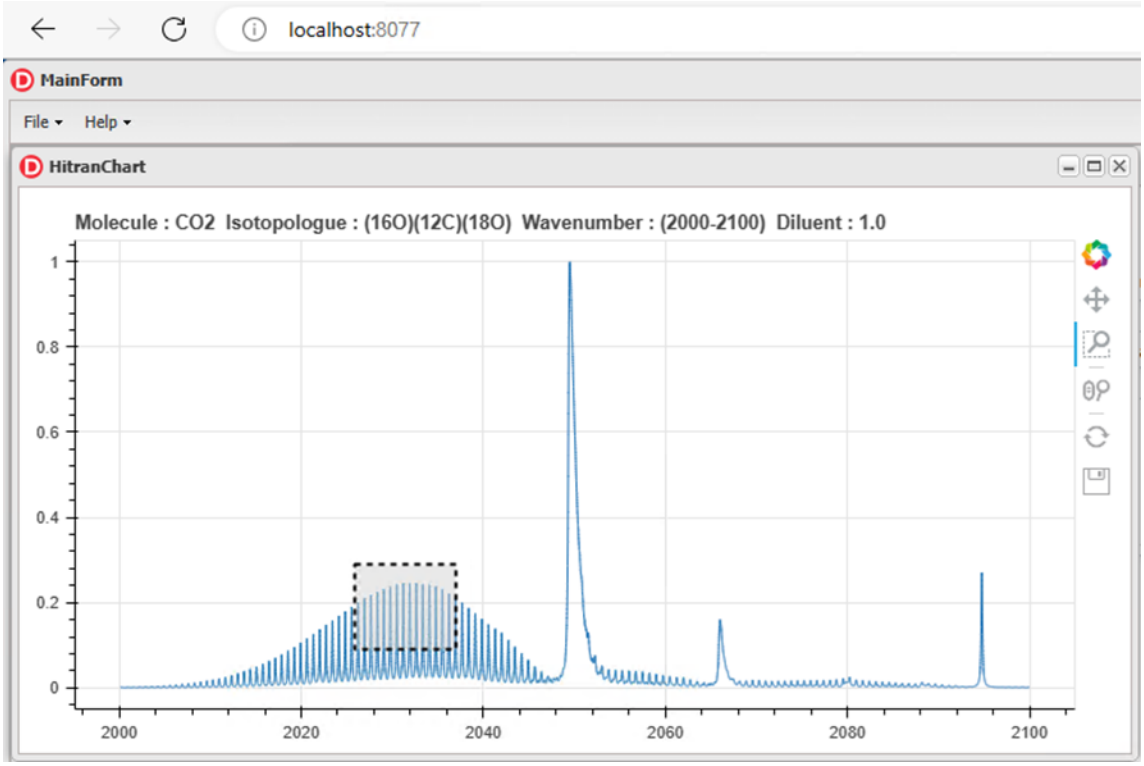


*Fig. 5.* **Example of a $CO_2$ isotope spectrum graph in a web browser.**

One of the other goals of this project was to provide the user with a convenient and friendly graphical interface for interacting with the system. To completely decouple the user from the requirements of various installations, it was decided to use a standard web browser and JavaScript and AJAX [22] technology as a visualization library. Although the HTTP protocol for communicating with the Application Server does not restrict us in any way from using a standard desktop VCL application or any other HTTP-compatible client. Bokeh is a Python-integrated JavaScript library for creating interactive visualizations that can be displayed in modern web browsers [23]. It allows you to create quality charts,

including simple graphs and complex dashboards with streaming data. With Bokeh, you can create JavaScript-based interactive visualizations without having to write your own JavaScript code. In the described project, the integration of HAPI results and the output of spectral lines to a web browser is implemented through a local file. The result of working out the cloud service can be displayed in the user's web browser (Fig. 5).

## FUTURE WORK

The described Application Server can be extended and used in turn to access various Cloud services, and you can also install NVIDIA hardware and the CUDA[24] vector computing library or the PyTorch machine learning framework directly on the server machine. The practical significance of the study lies in increasing the coverage of the technology stack for Delphi developers by integrating with the scientific development stack in Python, which in turn opens broad prospects for developing applications with a high-quality graphical interface and access to artificial intelligence services. The novelty of the research lies in the heterogeneity of the combined technologies: Delphi (P4D, uniGUI, IPC), Python (HAPI), and JavaScript (Bokeh).

## REFERENCES


1. Stephen Cass “Top Programming Languages 2022” IEEE Spectrum

   Available from : https://spectrum.ieee.org/top-programming-languages-2022 [Accessed: Jule, 2023]

2. Eric Matthes “Python Crash Course”, 3rd Edition: A Hands-On, Project-Based Introduction to Programming *No Starch Press*, US, 2023

3. Martin, James “Rapid Application Development”. *Macmillan* 1991, pp. 81–90

4. Bukunov S. V. “Development of applications with a graphical user interface in the Python language” *Publishing house* "Lan", 2023

5. By Steve Teixeira, Xavier Pacheco “Borland Delphi 6 Developer's Guide” 2002, pp. 382

6. Bannikov N.A. “Universal Database” Available from : http://www.stikriz.narod.ru [Accessed: Jule, 2023]

7. Amdahl, Gene M. “Achieving Large Computing Capabilities Through the Single-Processor Approach. Managing Requirements Knowledge”, *International Workshop on. Vol. 1. IEEE Computer Society*, 1967.

8. McCool, Michael D., Robison, Arch D., Reinders, James "2.5 Performance Theory". Structured Parallel Programming: Patterns for Efficient Computation. *Elsevier*. 2012, pp. 61–62

9. Dan C. Marinescu “Cloud Computing: Theory and Practice” *Elsevier Science* 2022

10. Douglas J. Reilly, “Inside Server-based Applications” Microsoft Press 2000 page 14

11. David Cornelius “Fearless Cross-Platform Development with Delphi” *Packt Publishing* 2021

12. Kiriakos Vlahos “Python for Delphi (P4D)”

    Available from : https://github.com/pyscripter/python4delphi [Accessed: Jule, 2023]

13. By Thomas J. Watson IBM Research Center, Dinesh C. Verma, Paul A. Stirpe “Procedure Based Inter-process Communication in a Shared Memory System” *IBM Thomas J. Watson Research Division* 1993

14. D.V Konnov “Python-UniGUI-HITRAN-API”

    Available from : https://github.com/dima72/Python-UniGUI-HITRAN-API [Accessed: Jule, 2023]

15. Farshad Mohajeri “uniGUI Web Application Framework for Delphi”

    Available from : http://unigui.com [Accessed: Jule, 2023]

16. Edward A. Cohen, Jean Demaison, Kamil Sarka “Spectroscopy from Space” *Springer Netherlands* 2012, page 242

17. R.V. Kochanov, I.E. Gordon, L.S. Rothman, P. Wcislo, C. Hill, J.S. Wilzewski, “HITRAN Application Programming Interface (HAPI): A comprehensive approach to working with spectroscopic data”, J. Quant. *Spectrosc. Radiat. Transfer* 177, 15-30 (2016)

18. K. N. Liou “An Introduction to Atmospheric Radiation” *Elsevier Science*, 2002, pp. 21

19. “HITRAN on the Web (iao.ru)” Available from : https://hitran.iao.ru/ [Accessed: Jule, 2023]

20. “Hitran (ontar.com)” Available from : https://ontar.com/hitran [Accessed: Jule, 2023]

21. “A GUI frontend which allows manipulation and processing of data from the HITRAN database.” Available from : https://github.com/hitranonline/hapiest [Accessed: Jule, 2023]

22. Ryan Asleson, Nathaniel Schutta “Foundations of Ajax” *Apress*, 2006

23. “Bokeh JavaScript library” Available from : https://bokeh.org/ [Accessed: Jule, 2023]

24. Shane Cook “A Developer's Guide to Parallel Computing with GP

## ABOUT THE AUTHOR

**Dmitry V. Konnov** - graduated from the North-Western State Polytechnic University of St. Petersburg in 2008, majoring in "Computers, complexes, systems and networks". He has more than 25 years of experience in developing software for industrial automation, calibration, AIM (asset instrument management), MES, LIMS, SCADA, FIDS, CMMS. Recently, Dmitry Konnov has been working for American companies under contracts, supporting Delphi projects.