

Control Network Programming Development Environments

KOSTADIN KRATCHANOV¹, TZANKO GOLEMANOV²⁺, BUKET YÜKSEL³, and EMILIA GOLEMANOVA^{2*}

¹Department of Software Engineering
Yaşar University
Universite Cad. No. 35-37, Bornova/Izmir 35100
TURKEY
kostadin.kratchanov@yasar.edu.tr

²Department of Computing
University of Ruse
Ul. Studentska No. 8, Ruse 7013
BULGARIA
⁺TGolemanov@ecs.uni-ruse.bg, ^{*}EGolemanova@ecs.uni-ruse.bg

¹Department of Computer Science and Engineering
Koç University
Sarıyer, Istanbul
TURKEY
byuksel13@ku.edu.tr

Abstract: - In this paper we discuss the unusual distinctive features of Control Network Programming as a hybrid programming paradigm. We postulate the maxim “Primitives + Control Network = Control Network Program”, and use this observation in the design of programming environments for developing Control Network Programming projects. The various possible approaches to building such environments are the main focus of the paper, together with a relatively detailed presentation of the currently most powerful locally run *SpiderCNP* environment with graphical editing and tracing, as well as two light-weight and ready-to-use cloud-based environments. An extended survey of cloud compilers and IDEs is also included.

Key-Words: - Control Network Programming, CNP, programming environments, cloud IDEs, cloud compilers, online compilers, programming paradigms, learning systems.

1 Introduction

Control Network Programming (CNP) is an unusual programming style. To mention just a few of its most important pronounced features one can point out that it is genuine visual programming, search is performed automatically due to the built-in inference engine on which the computation mechanism is based, it combines elements of declarative and procedural programming paradigms. CNP is especially advantageous for solving problems which exhibit one or more of the following traits: the problem specification or its procedural solution have a natural graph-like representation, involve nondeterminism or randomness, or are based on search. Distinguishing features of CNP are discussed in Section 2 below.

We need a CNP programming environment in order to create, modify, compile, and run CNP applications. The ultimate goal of this paper is the study of the possible approaches to building such environments, the components they must include, and the current state-of-the art in the area of CNP environments. In particular, we will discuss the structure and some implementation details of the best currently used environments, such as the powerful, fully fledged *SpiderCNP* with an embedded advanced graphic editor, and much simpler cloud-based light-weight ready-to-use versions. The cloud-based approach is not only ‘trendy’ but in fact frees a user from the burden related to installation, maintenance and updating the tool.

However, our route to the final goals is not strait. We address a number of related objectives and use

the corresponding conclusions. We discuss the recognizable characteristics of CNP and deduce the summarizing maxim “**Primitives + Control Network = Control Network Program**”. We pay proper attention to the fundamental *SpiderCNP* environment. We address the reasons for devising a cloud compiler, and acknowledging the fact that the advances in cloud-based compiling are a very recent development include an extended review of cloud IDEs and their advantages. Finally, we are at a position to explain the principles and some design and implementation details of our two cloud-based CNP environments.

A particular focus of our paper is our noticeable interest in teaching and learning – teaching major concepts of computer science and mathematics with the help of CNP tools, and learning CNP by students, programmers, and researchers.

We have been successfully using CNP as a tool for simulating various computational models and algorithms in our Computer/Software Engineering curricula at undergraduate and graduate levels. Other areas where concepts such as computation, search, inference, nondeterminism and randomness are fundamental, and where CNP could be a great teaching tool, are Computer Science, Mathematics, Industrial Engineering, Robotics, and others. A short summary of our teaching experience with CNP is given in Section 2.4.

This publication is a substantially extended version of [1].

2 Control Network Programming Unlocked

2.1 What is CNP?

The name **Control Network Programming** or **CNP** can be deciphered as ‘programming through control networks’.

In CNP, a ‘program’ consists of two fundamental parts. The first one is called a **Control Network (CN)**. It is an explicit system of graphs called **subnets**. One of the subnets is identified as a **main subnet**. Each subnet consists of **nodes (states)** and **arrows**. The node of the main subnet which is first in the description of the main subnet is called its **initial state**. When calling a subnet, by default its first node is treated as its initial state; however, which node is initial may be specified in the subnet call itself. There are a few special types of nodes; one such type is the **final state**. The arrows of the subnets are labeled with sequences of simple actions, called **primitives**. The primitives (specified by the user imperative functions) are defined

separately in the second major part of the program (or may be imported from existing libraries). A subnet can call other subnets or itself. Both primitives and subnets can have parameters.

In general, the CN might be of “nondeterministic nature”. An **interpreter** (inference engine) must implement a strategy for search (inference / computation) in a recursive network. The system attempts to find a path from the initial node of the CN to a final node, executing the primitives along the way. This process may include invoking subnets. The execution of a primitive might result in failure in which case the system starts executing the (passed already) primitives of the arrow backwards, restoring the state of the data, and attempts another path. If no more arrows outgoing from the current state exist then the system executes backwards the whole arrow through which control was passed to this state. We call this strategy extended backtracking; it can involve entering a subnet backwards. The passing of the control through the CN is thus highly intuitive and easily understandable. It is practically graphically ‘visible’ on the CN. As a matter of fact, it can be traced on the graphical representation of the CN.

As a matter of fact, no interpreter actually exists. The **Spider compiler** uses an approach similar to the recursive decent technique well known in compilation. The resulting “intermediate program” (file *SpiderUnit.SPI*) corresponds to the CN, at the same time also embodying the algorithm for interpreting it.

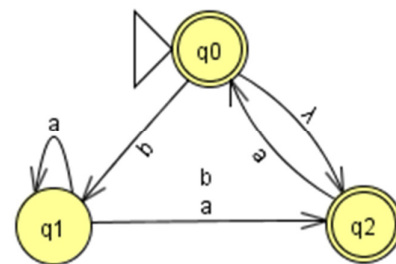


Figure 1 Exemplary NFA

Some major resources describing the technical details of CNP are [2]-[6]. How CNP can be applied for solving different types of problems is demonstrated in [7]. The computation/inference in a CN program is based on search. Therefore, the built-in powerful tools for user control of the computation can be used to implement heuristic search strategies in an unusual, non-procedural manner [8]-[10].

2.2 An exemplary CNP application

As an illustration, we show here the CNP simulation of a nondeterministic finite automaton (NFA). The

graph of the NFA is shown in Fig. 1 (screenshot from *JFLAP*). This NFA example was used in the PhD course Theory of Computation and the undergraduate course Discrete Computational Structures II at Yaşar University in the current academic year.

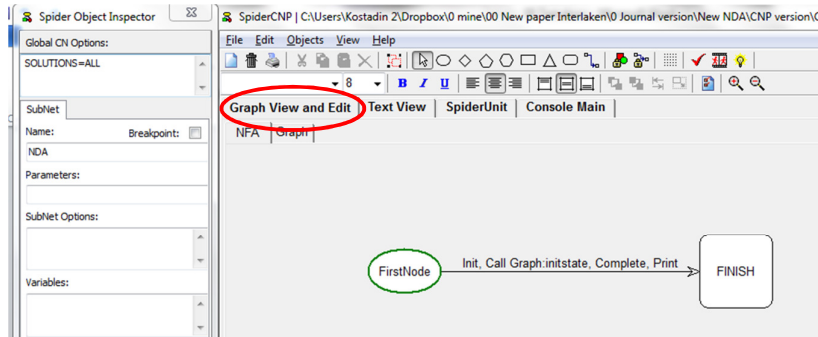


Figure 2a The *NFA* main subnet

The CN of the CNP implementation consists of two subnets. The main subnet called *NFA* is shown in Fig. 2a, and the subnet *Graph* - in Fig. 2b. The screenshots are from the *SpiderCNP* programming

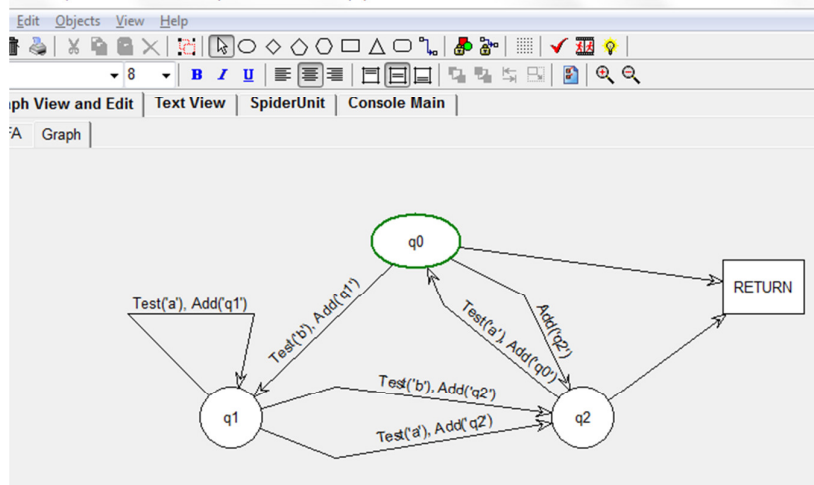


Figure 2b The *Graph* subnet

environment discussed below. There are two equivalent representations of the CN – a graphical representation and a textual representation. Either can be used for viewing and editing the CN. Figures 2a and 2b show the graphical representations of the corresponding subnets. To view the graphical representation, the user must press the “Graph View and Edit” button (which is circled in Fig. 2a). To switch to the textual view, the button “Text View” must be pressed. The textual view of the CN (both subnets) is illustrated in Fig. 3. To describe textually

the CN we use a special simple language for specifying graphs which we call *Spider*.

The primitives in a CN program are of two types. Action primitives are elementary actions. Condition primitives perform condition checks. The following primitives are used in the CN of our example.

Action primitive *Init* performs some initialization and prompts the user to enter the initial state and the input string. Primitive *Test(c)* is a typical example of a condition primitive – it completes successfully if the current input character equals *c*. Primitive *Add(n)* is an action primitive that concatenates the string *n* as the name of the current node into the

solution path. Condition primitive *Complete* checks if there are additional symbols in the input string that have not been read and used. Using this primitive in the main subnet ensures that no unused symbols have remained in the input string. Finally, action primitive *Print* displays the solution path.

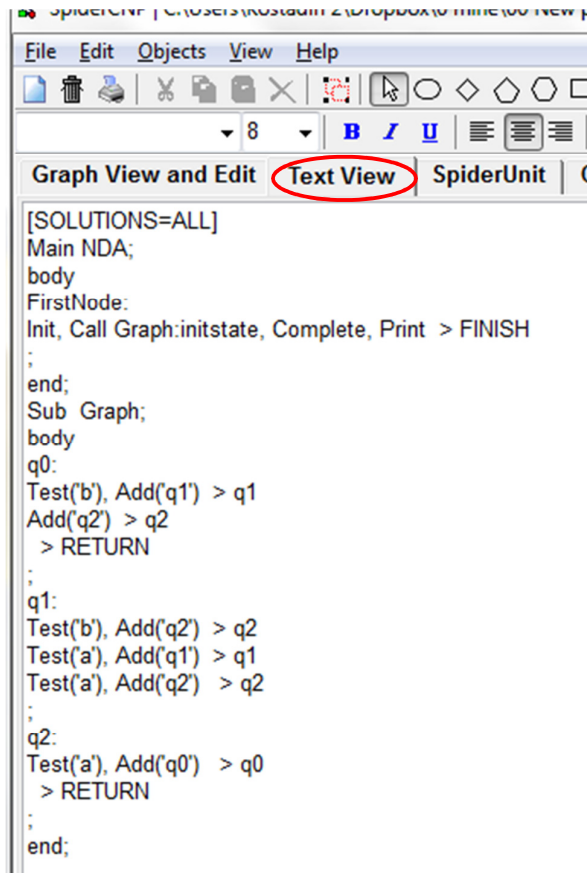
The primitives defined by the user (together with constants, helping functions, etc. if necessary) are included in the file *SpiderUnit*. It can also be displayed and modified. Fig. 4 shows this file for our example. The user may also import external modules thus using, in particular, pre-existing libraries of primitives.

Exemplary dialogues with the user are shown in Fig. 5. Note that in this CNP model of the NFA during the dialogue the user can choose different initial states. In Fig. 3, in the first three runs the selected initial state is *q0*, while in the fourth execution it is *q1*. In the

third run an empty input string was chosen – the screenshot demonstrates that, because of the existing λ -transition, initially the nondeterministic automaton can actually be in either of the states *q0* and *q2*.

The discussed NFA example is an eloquent demonstration of the descriptive (declarative) nature of CNP that will be commented on in Section 2.4. Namely, the CNP solution the major part of which is the CN, simply specifies the NFA and includes no hint at all about how the functioning of the

automaton must be simulated – there is no procedure (algorithm) at all that specifies how a solution path should be searched for. The search mechanism is actually built-in into the system; the CN program is purely declarative. The CN duplicates almost exactly the graph of the NFA.



```
[SOLUTIONS=ALL]
Main NDA;
body
FirstNode:
Init, Call Graph:initstate, Complete, Print > FINISH
;
end;
Sub Graph;
body
q0:
Test('b'), Add('q1') > q1
Add('q2') > q2
> RETURN
;
q1:
Test('b'), Add('q2') > q2
Test('a'), Add('q1') > q1
Test('a'), Add('q2') > q2
;
q2:
Test('a'), Add('q0') > q0
> RETURN
;
end;
```

Figure 3 Textual view of the CN

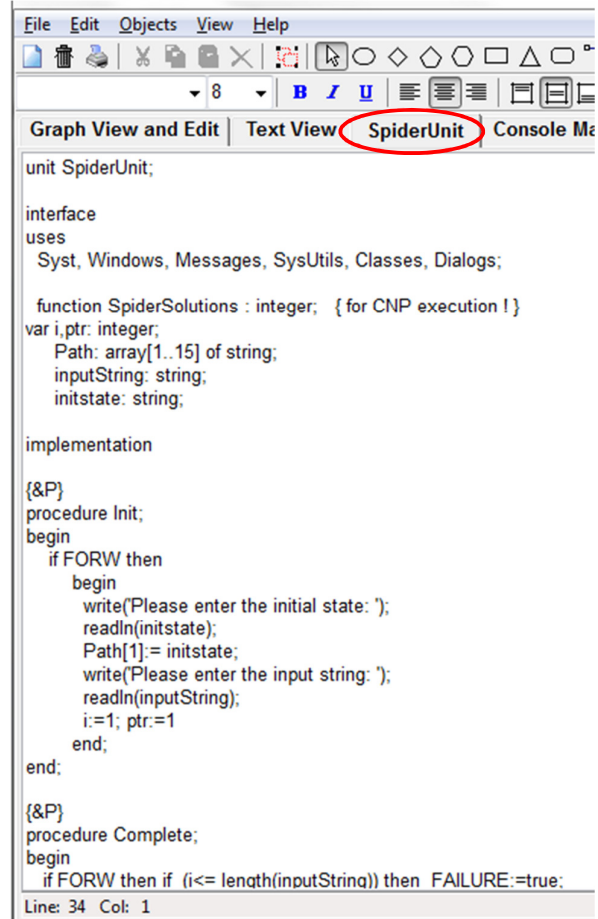
2.3 “Primitives + Control Network = Control Network Program”

Back in 1975, Niklaus Wirth proclaimed: “Algorithms + Data Structures = Programs” [11]. First of all, this applies to structured, imperative programs. In 1979, sighting logic programs Robert Kowalski responded: “Algorithms = Logic + Control” [12]. Modifying the famous Wirth’s statement, Zbigniew Michalewicz added in 1992: “Generic Algorithms + Data Structures = Evolution Programs” [13].

In [1] we notice and proclaim: **“Primitives + Control Network = Control Network Program”**.

Our statement has a very direct and literal meaning: a Control Network (CN) program consists of two parts: definitions of primitives, and a CN using these primitives. Physically, a CN

programming (CNP) project includes two main files – one that contains the primitive definitions, and a second one with a textual representation of the CN. In more detail this will be discussed in the following sections.



```
unit SpiderUnit;

interface
uses
  Syst, Windows, Messages, SysUtils, Classes, Dialogs;

function SpiderSolutions : integer; { for CNP execution ! }
var i, ptr: integer;
    Path: array[1..15] of string;
    inputString: string;
    initState: string;

implementation

{&P}
procedure Init;
begin
  if FORW then
  begin
    write('Please enter the initial state: ');
    readln(initstate);
    Path[1]:= initState;
    write('Please enter the input string: ');
    readln(inputString);
    i:=1; ptr:=1;
  end;
end;

{&P}
procedure Complete;
begin
  if FORW then if (i<= length(inputString)) then FAILURE:=true;
end;
```

Figure 4 *SpiderUnit*

A note is needed. We “apologize” to the data for somehow neglecting its importance. The data does exist and is important. We refer to the first major component of a CN program as “Primitives”. To be maximally precise, we should’ve called it “Data and Primitives”. The primitives act on data. The data is usually declared in the same file where the primitives are (or in other project modules used). However, for simplicity, we’ll keep the shorter name “Primitives”. It is understood by default that there is data processed by these primitives. Generally, in CNP one focuses more on the CN (which specifies the problem) rather than on data.

2.4 CNP distinguishing features

We describe CNP as a new programming paradigm extending and integrating declarative programming,

imperative programming, and programming rule-based systems. As all other programming paradigms CNP is universal – that is, it can be used for implementing any algorithm. However, it is especially effective when solving problems which can be naturally represented in a graph-like manner, and/or whose descriptions exhibit nondeterminism and declarativeness.

```

C:\Users\Kostadin 2\Dropbox\0 mine\00 New paper Interlaken\
Hello from lazarusCNP!
Please enter the initial state: q0
Please enter the input string: baa
One of the possible paths = q0 q1 q1 q2
One of the possible paths = q0 q1 q2 q0 q2
One of the possible paths = q0 q1 q1 q0

Bye from lazarusCNP!
Number of solutions: 3
Press enter to exit!

C:\Users\Kostadin 2\Dropbox\0 mine\00 New paper In
Hello from lazarusCNP!
Please enter the initial state: q0
Please enter the input string: b

Bye from lazarusCNP!
Number of solutions: 0
The input string is NOT accepted
Press enter to exit!

C:\Users\Kostadin 2\Dropbox\0 mine\00 New pap
Hello from lazarusCNP!

Please enter the initial state: q0
Please enter the input string:
One of the possible paths = q0 q2
One of the possible paths = q0

Bye from lazarusCNP!
Number of solutions: 2
Press enter to exit!

C:\Users\Kostadin 2\Dropbox\0 mine\00 New paper Interlaken\0 J
Hello from lazarusCNP!
Please enter the initial state: q1
Please enter the input string: baa
One of the possible paths = q1 q2 q0 q2 q0 q2
One of the possible paths = q1 q2 q0 q2 q0

Bye from lazarusCNP!
Number of solutions: 2
Press enter to exit!

```

Figure 5 Console

Often the CN can be looked at as a **declarative (descriptive, nonprocedural) description** of the

problem to be solved. This was already discussed in the case of our CNP simulation of a NFA. Other typical illustrations of this viewpoint are the CNP solutions to the Animals classification problem in [7], the Map traversal problem in [4], the non-recursive heuristic solutions to the same problem in [9] as well as its iterative and recursive solutions in [4] and [9], the iterative and recursive solutions to the Wolves and sheep problem in [7], expression grammar in [4] and [14], the algorithms for solving constraint satisfaction problems in [10], etc.

CNP can be successfully used for typical **procedural solutions** as well. Such an example is the SelectionSort algorithm in [7]. Procedural solutions to search problems are described in [9]. Here, the CN is an explicit graphical representation of the program control (as understood in imperative programming and depicted, for example, by an activity diagram or a flowchart). In other words, in CNP the program control is extracted from the imperative program and made explicit. The actions on data are defined in the simple and well-understood primitives. This helps for easier understanding, creating, modifying, or verifying the algorithm.

In both cases of considering a CN as a declarative problem description or as an explicit description of the control in a procedural solution, CNP can be also described as a type of **graphical programming**. Indeed, the CN (being the leading principal part of the CN program) is a recursive set of graphs. Depending on the development environment used this net (the CN) may be actually seen and edited in a graphical editor, or may be coded textually using our simple language for describing graphs, *Spider*. Especially in the case of using a graphical editor of CNs, it would be fair to say that CNP is a genuine graphical programming.

‘Executing’ the CN is a kind of search. CNP has been equipped with powerful means to control this search – namely numerous system options and control states [4], [5], [14]. Some of the latter can also introduce randomness. This makes CNP a very powerful tool for realizing a great number of search approaches. Especially interesting and unusual are implementations of algorithms based on **local search**, as the execution of the CN is in fact a particular type of local search itself. This approach is called non-procedural implementation [8], [10], [14]. It does not involve writing any search algorithm in the usual sense – a behavior equivalent to the corresponding search algorithm is achieved “automatically” through the appropriate usage of the

built-in search control tools.

2.4 CNP in education

At Yaşar University, we have been systematically using CNP for three years in teaching the courses of Artificial Intelligence (4th year undergraduate) and Theory of Computation (PhD.) in our Computer/Software Engineering curricula, and we have found it to be a very useful tool in simulating various models and algorithms. Other possible areas include Formal languages and automata, Compilers, Algorithm analysis and design, Concepts of programming languages, Discrete mathematics, Logic, Digital design, Algorithms and data structures, and more.

In general, it is the prevailing view of the educators in areas such as computer science and mathematics that students tend to have substantial difficulties in apprehending the ideas behind nondeterministic and randomized (also referred to as stochastic) computation models and algorithms. CNP can be a great instrument to help understanding and getting confident with these concepts [14], [15]. In fact, our experience and surveying results strongly suggest that understanding and using CNP for our students (who have already developed a strong procedural way of thinking) is substantially easier than Prolog.

3 Programming environments for CNP

3.1 SpiderCNP – a CNP IDE for visual programming

To practice CNP, i.e., to create, edit, compile, and run CNP applications one needs an appropriate development environment. A number of such environments have been created.

The most powerful one is *SpiderCNP* [16]. It has two versions. They are integrated as a tool in the *Delphi* and *Lazarus* IDEs, respectively. A fundamental advantage of the chosen approach is the possibility to use all the features and tools of the larger encompassing environment and the latest

versions of the programming language around which the IDE is built. This programming language is also used to program the primitives. The installation process consists of three steps: install the *Delphi* or *Lazarus* IDE, then run a simple installation program that installs *SpiderCNP* as a

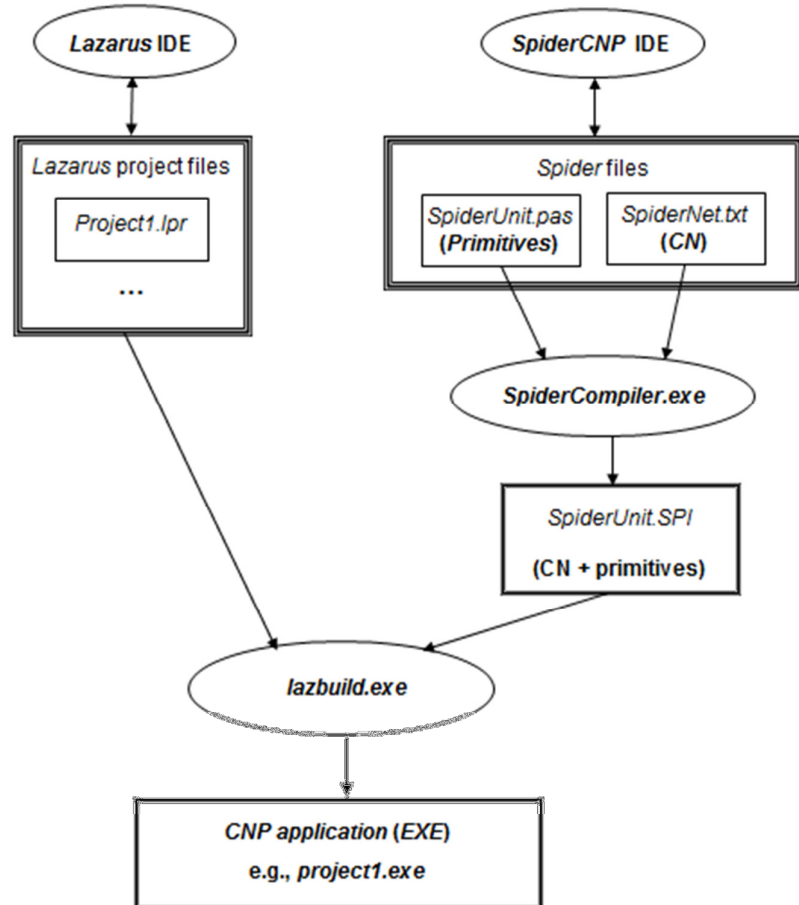


Figure 6 Structure of a CNP application

tool of the IDE, and finally fix some settings.

Delphi [17] is a sophisticated, ambitious, advanced professional environment which, of course, is an important advantage of this approach. However, this is also its main disadvantage, especially when using it in teaching. The IDE is rather expensive, free academic versions are very difficult or impossible to obtain, updates in the IDE are difficult for the same reason. The second disadvantage is the size of the software product and correspondingly the difficulties in its installation. Difficult installation was the single most important drawback of the CNP approach identified by the students in their surveys and comments during the first year of using CNP in teaching.

In our opinion the switch to the *SpiderCNP*

version based on *Lazarus* [18] (called also *LazarusCNP*) brought essential advantages. *Lazarus* is a free product; it is pretty easy to download the latest version of the product, and much easier to install it. The IDE is also quite advanced and stable. We have been using *LazarusCNP* for two years and it is our most advanced and well-tested CNP development environment.

Although the installation still requires the same three steps, it now takes less than 5 minutes, and all the components to install are free. The corresponding installation instructions and download files are available at [19].

Before discussing the CNP IDE further we need to understand the architecture of a CNP application.

3.2 Architecture of a CNP application

There are two types of applications in *Free Pascal* (also known as *OO Pascal*) which is the core of *Lazarus*: console applications and window applications. Correspondingly, we have the same two types of CNP applications. Console applications are simpler; the I/O is realized through a DOS-type console. Such an I/O console is shown in Fig. 5. The I/O of a window application is performed through various built-in or user-defined windows. This allows the creation of applications with more attractively looking and modern I/O. In practically all cases of teaching applications the console I/O is enough and easier to follow. The online CNP compiler that we will be describing further in this report allows console applications only. Therefore, we will introduce the structure of a CNP application for the case of console applications only (although the differences are minimal). It is shown in Fig. 6.

A CNP application is created as a *Lazarus* project. All the files of the project (i.e., the CNP application) are placed in a folder. As we know, a CN program consists of two fundamental parts: primitives and CN. The file with the primitives is *SpiderUnit.pas*. Technically, it is an *OO Pascal* file in which the primitives are defined as procedures. The file may contain also definitions of global data. The CN is specified in the text file *SpiderNet.txt* using the *Spider* language – see [2], [6], [7] for a description of this language.

The CNP compiler (called *SpiderCompiler.exe*) uses the above two files (the CN and the Primitives) to produce a Pascal program as its object file *SpiderUnit.spi*. This program includes directly a copy of the definitions of the primitives and the global data. The behavior of the program corresponds to that of an interpreter that would execute the CN according to its semantics. The CNP

compiler has been developed using ideas similar to the well-known recursive decent parsing strategy.

As any *Lazarus* project, the project folder contains some other files. The only one important for our description here is *Project1.lpr*. A CNP user may well survive without knowing about it, but if (s)he wants to change the initial and concluding text in the output, the user may do corresponding modifications there.

All *Lazarus* files included in the project, as well as the *SpiderUnit.spi* file, are used by the *lazbuild.exe* file (which is part of the *Lazarus* IDE) to create the file *project1.exe*. This executable file is the CNP application. It can be called (executed) from inside the IDE, or directly as a stand-alone executable file.

The general view of the *SpiderCNP* IDE can be seen in Fig. 2. In the main window, the user may switch between displaying the CN (graphical view or textual view; the latter is the file *SpiderNet.txt*), primitives (file *SpiderUnit.pas*), or console (file *Project1.lpr*). Normally, the CN is studied and edited in the graphical view. However, if preferred, the textual file may be modified (currently, for a given project, only one of these options is possible). The textual view of the CN shown graphically in Fig. 2, is given in Fig. 3.

Clearly, the most distinguishing feature of *LazarusCNP* is the existence of graphical editor of CNs. The CNP IDE has many other features, including tracing the execution within the graphical editor on the graphical view of the CN.

Working with a CNP project means creating and modifying as needed the three components (files) described: CN, primitives and *project1.lpr*.

3.3 Possible principles for designing a CNP development environment

Historically, the first CNP were run using **interpreters** which interpreted the CN. However, for over fifteen years all CNP environments use a **compiler** (e.g., [2]) – an approach which we found to have substantial advantages.

As we emphasizes earlier, a CN program consists of primitive definitions and a CN. For representing the CN we have always used the simple *Spider* language mentioned before. The second question arising is how to specify/program the primitives.

It is possible to use a **special, defined by us programming language for primitives** which, in a way, will make CNP self-contained. We have decided against this approach, however, as it will imply constant improvements, extensions, and

modifications to the language and the corresponding environment, for which we must be responsible, together with the corresponding documentation, installation files, etc.

Following the approach we have chosen, we must **integrate our CNP development environment with the external IDE** of a programming language. As already mentioned, our main CNP development environment, *LazarusCNP* is integrated as a tool inside the *Lazarus* IDE. Therefore, a CNP developer can avail of all the powerful features of *Lazarus*. Also, a developer working in *Lazarus*, can in principle create a 'regular' *Lazarus* project (e.g., in *OO Pascal*) using CNP only for subtasks where CNP would be most effective. Naturally, this approach also has drawbacks, the major one being the necessity to obtain, download and learn the basics of *Lazarus*. This disadvantage is not so severe as *Lazarus* is freely available and well maintained; however it still exists. For example, we need to have *Lazarus* installed in our teaching labs.

An alternative promising approach is to develop a **'stand-alone' CNP environment** which does not depend directly on any specific external development environment. However, in contrast to the approach with a specially design programming languages for coding the primitives which was mentioned above, a natural and highly appealing approach would be to use primitives programmed in different programming languages and eventually developed in different IDEs. This idea is aligned with the modern-day idea of language interoperability [20] - [24]. The idea could be implemented on the base of any of the two major groups of managed languages – the .NET SLI compliant languages and JVM compliant languages [25], [26]. Creating such a light-weight and highly-flexible CNP development environment is in our plans for the near future. In particular, being able to write primitives in a language of the student's choice is a highly desired feature of a CNP environment used in teaching. The learners of CNP come from different backgrounds, have different personal preferences, and would like to be able to use the language they feel most confident with. A main idea behind creating the declarative-driven approach of CNP is that 'programming' can be done by any user, including those with very limited or even non-existing experience in programming.

Finally, we can develop the previous approach even further by using a **cloud CNP development environment** the core component of which is a cloud CNP compiler.

A short survey of cloud compilers and development environments follows. The sections afterwards describe our current two working CNP cloud compiler called *Bouquet* and *SpiderCloud*, respectively.

4 Cloud-based IDEs

4.1 Cloud IDEs and compilers: state-of-the-art

It is widely accepted that cloud computing is undoubtedly one of the biggest buzzwords in the technology world today. According to Fast Company [27], the cloud is a vast network of low-cost, high availability computing resources. Almost 60% of companies are already in the cloud and an additional 20% are planning to do so within the next 12 months. Cloud computing refers to application service provisioning where typical client server software is run at a remote location. Such services are given popular acronyms like 'SaaS' (software as a service), 'PaaS' (platform as a service), 'IaaS' (infrastructure as a service), 'HaaS' (hardware as a service) and finally 'EaaS' (everything as a service) [28].

Cloud computing has well-known advantages and challenges which we are not going to address here in general. Instead, we discuss below the advantages and limitations of cloud IDE's, simpler development tools, and compilers.

Software development, and in particular its most important component – compiling - can also be shifted from being performed on a user's physical computer into being done in the cloud using available remote software resources. The result of this approach is the so called cloud development environments (most developed ones are referred to as **cloud IDEs**). Some authors use the phrase online environment/platform or online compiler – this is technically correct but does not emphasize well enough the nature of a cloud application – it is not simply contents available from the internet but it is actually an integrated resource/service available remotely, most often from a dynamic website. A cloud development environment may include much more than a single compiler – editors, libraries, online execution facilities, user storage, etc.

For some reason lifting the code production into the cloud is happening later than many other types of business and other applications. Advancement in cloud compiling and cloud IDEs is a comparatively recent development, an emerging technology.

Some of the most interesting representatives of

currently available cloud development environments are [29] - [44].

They have different features. Some are simple and free; many of the best ones are (as it should be expected) paid and highly professional. Some sites support many (e.g., 60) languages, others are highly specialized and focus on one particular programming language or tool. Some allow for the execution of the compiled object program on the cloud server or even for the deployment of the compiled embedded code into a particular device. Some IDEs also allow the developed code to be deployed into cloud platforms such as Windows Azure, Amazon Cloud Services, or Heroku.

Jimenez, founder of [32], summarizes: “The online IDE is one of the final frontiers of apps ported to the web. I would like to be able to develop from any computer or operating system and have the same experience without having to install software or install anything.”

The following advantages of cloud compilers may be identified (not all sites possess all advantages).

It can be frustrating to have to install volumes of software just to write a little bit of code. Cloud IDEs keep it simple by making all these tools available in the cloud with the click of a mouse. Some cloud IDEs come equipped with almost every tool, library, etc. that the code developer may ever need. With an online IDE, one can get their projects up and running faster than ever by skipping over tedious installations, and getting right down to the programming of the project itself.

The local computer is not loaded with large-size software, neither is computer time and other resources used for compilation and other related tasks.

Cloud IDEs allow the code to be accessed and edited from just about any computer worldwide, freeing the programmer from the need to have constant access to a single computer where all the tools and files are. Typically, cloud IDEs are cross-browser and device-friendly. They have been tested across all modern desktop and mobile web browsers like Internet Explorer, Firefox, Chrome, and Safari. With support for touchscreen interaction one could write code all from their mobile or tablet device. It is possible to log into one’s online IDE with a smartphone or tablet, edit the code, test it and send it off to a client in a matter of minutes.

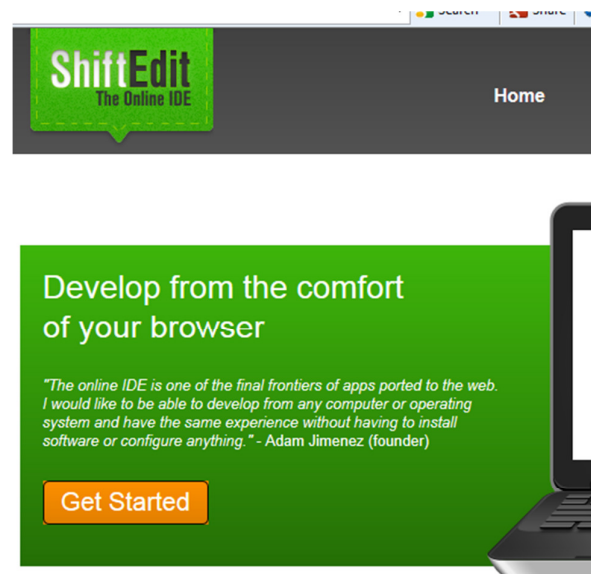
A developer is now able to program for a wide selection of devices, without actually needing to go out and purchase them. They can write for Mac,

Windows, Linux or even an iPhone or iPad without spending the money on buying one of each.

A cloud IDE can integrate features that can hardly be achieved by a compiler installed on a local computer.

The remote server can be more powerful (in terms of speed, storage and memory) than a local computer. The compiler, editor and other components may be most advanced and possess most useful and increasing the developer’s productivity features. E.g., the cloud compiler may include advanced code optimizing features, the editors may support features such as autocomplete, syntax checking, multiple cursors, keyboard shortcuts, etc. Live editing might be supported where you can see real-time updates as you tweak your code. Advanced tracing may be included, even simulating the state of data. Powerful latest popular and non-standard libraries may be made available. The software components on the cloud-based site will be kept up-to-date at any time.

You can have a user account at the cloud IDE – a central depository and a virtual console. You can forget about Dropbox, USB sticks, external drives — with a cloud IDE central depository your code is always accessible online. You can access files directly from your folders on the IDE or from Dropbox, Google Drive, Amazon S3, etc. You can keep a revision history. You do not have to compile into local native images or re-download pre-compiled native images.



Everything you need

Figure 7 A screenshot from shiftedit.net

An especially important feature is the possibility to easily share your code and collaborate. Some cloud IDEs enable developers around the world such as teammates or peers to edit the same code and chat together in real time.

Some IDE's allow the developer to integrate with many other services that are important within the development life cycle. Thus the user can work with the tools they know and like, or even develop and integrate their own extensions.

Some sites offer for download local servers that can automatically synch with the cloud workspace.

Fig. 7 is a screenshot from the home page of [32] and summarizes some of the major advantages that cloud IDEs offer. We are tempted to also reproduce the memorable statement from the video at [29]: "And just like everything else important in your life, c9 IDE is cloud-based".

Of course, cloud IDE's also have shortcuts and limitations – some of them are inherent, other relate to the fact that the technology is only emerging. One of the major issues is security – both in

relation to the user code, and in relation to the integrity of the servers of the provider. Another issue is the client-server communication, in particular during the editing. Usage of graphical tools such as graphical editors or complex graphical I/O in the developed programs might be a serious challenge. Reaction to errors on the server and auto-safe functionality are another issue to address.

4.2 Cloud IDE's and learning systems

It is often claimed that online IDEs and compilers are a perfect learning tool for students and other

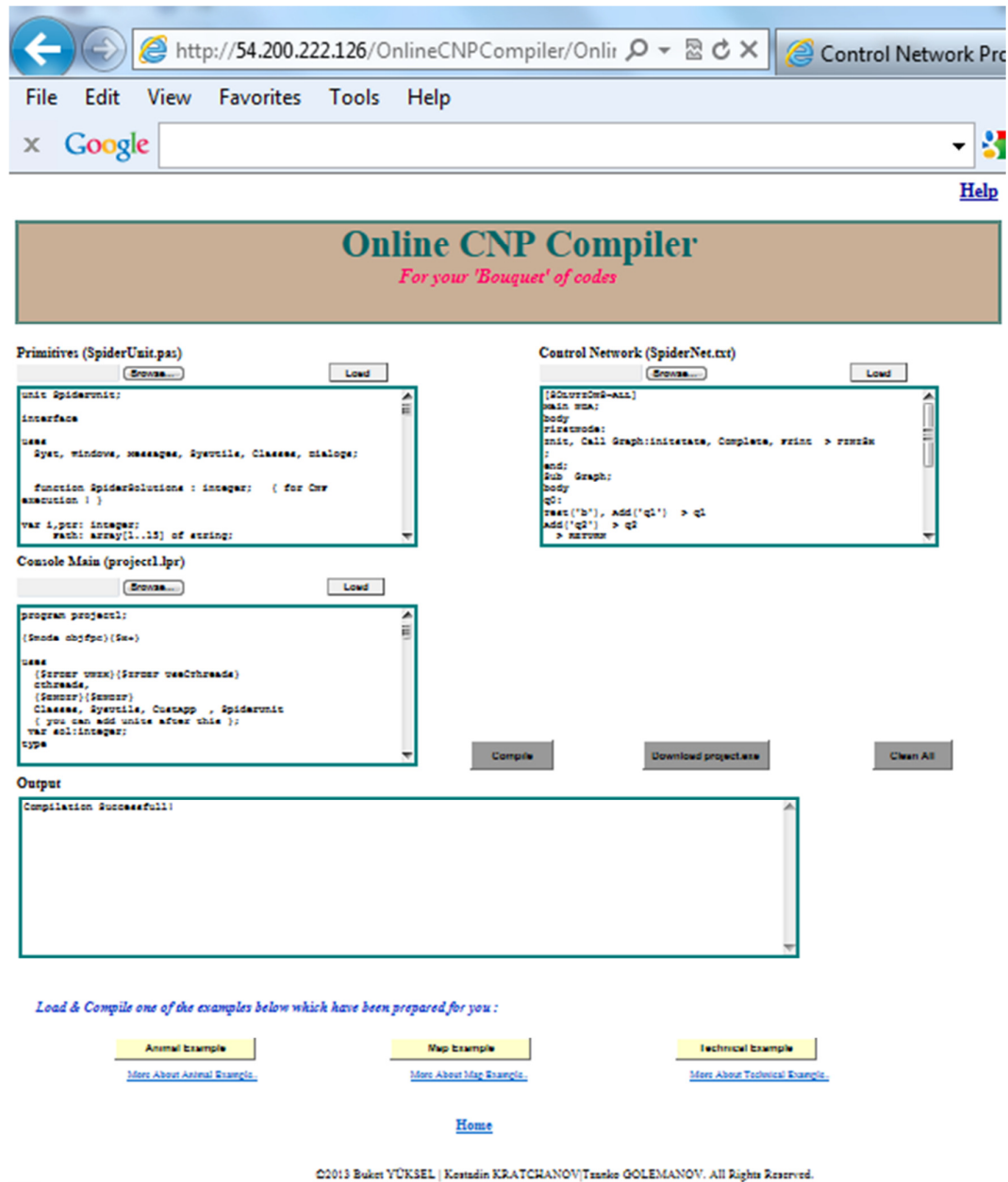


Figure 8 The Bouquet cloud CNP compiler

persons learning programming with a particular programming language or tool. Some of the systems mentioned in the previous subsection have been actually created mainly as a learning tool.

A cloud IDE is easy to be used by students. They can create their codes all from the comfort of their browsers. All the heavy lifting has been done by the creators of the cloud IDE, so students and learners can just focus on writing and learning code. A cloud IDE is a natural sandbox for learning a programming language. You can write code in the computer lab and pick up where you left off at home. You can learn programming by visualizing code execution, use advanced editors, etc.

Many of the other assets which cloud IDEs demonstrate and we discussed in the previous subsection, can be considered as essential advantages from the viewpoint of learning systems.

It is also natural to integrate a cloud IDE/compiler within an integrated learning system. In addition to the IDE, an integrated learning environment may include reference guides, online interactive tutorials, pdf and video tutorials, demos, tests, problems, projects, other resources, etc.

Cloud IDE sites often include blogs and forums. You can easily get help from the programming community as well as from other students and learners.

In addition to teaching and learning, cloud IDEs may be used in training courses and certification, recruitment, programming contests, and similar activities.

5 CNP cloud compilers

Using a cloud-based CNP development environment is especially convenient and advantageous for a user who wants to learn the basics of CNP and use it for running demos or creating their own small-size CNP applications. Generally, our students belong to this class of user, together with other students, researchers or programmers whose aims are to get basic awareness of CNP and its possibilities but are not yet its heavy users.

5.1 The *Bouquet* CNP cloud compiler

Following the modern trends in ‘lifting’ compilers into the cloud described above, and first of all understanding the substantial advantages of cloud IDEs, we have developed two online CNP compilers described in the current and the next sections, correspondingly.

We have called the CNP cloud compiler presented in this section *Bouquet* which is the

English translation of the name (Buket) of the programmer most actively involved in its coding. The compiler can be accessed at [46].

The general view of the cloud compiler is shown in Fig. 8. The data that can be seen are from the NFA application described earlier in Sec. 2.2 (where *SpiderCNP* was used).

The webpage includes three input and one output forms. An input area exists for each of the three files that specify a CNP application: *SpiderUnit.pas* for the primitives, *SpiderNet.txt* for the textual representation of the CN, and the console file *project1.lpr*. These files were discussed in Sec. 3.2.

In order to compile a CNP application we must enter (e.g., write or copy) the corresponding files into the three input windows. We can also use the *Browse* button under each form. Three exemplary applications are prepared in advance and their files can be loaded with the click of a single button – the Animals classification, Map traversal, and the Technical example. These are the major examples used for introducing CNP in [2]-[4],[6],[7], as well as in teaching CNP at university.

When the three input files of an application are ready, the user can activate the *Compile* button. This triggers the following sequence on actions. A project folder with a partially random name is created on the virtual server. The randomness of the folder name allows multiple independent users to work simultaneously with the cloud CNP environment. A new CNP project is created in that folder. The input files from the forms are uploaded into the project folder. Then compilation/building is started by executing the file *SpiderCompilerCloud.exe* on the virtual server. This file is a cloud version of *SpiderCompiler.exe* (see Fig. 3). The generated name of the project folder is given as an argument to *SpiderCompilerCloud.exe*. In addition to the application executable file, a second output file, *CompResult.txt* is produced which contains details of the compilation and is most useful in the case of errors. Finally, *lazbuild.exe* is run which in absence of errors generates the CNP application executable file *project1.exe* and saves it in the application folder in the cloud. The user can now push the *Download project1.exe* button and download the executable file of the CNP application to their local computer. Execution of the application file directly in the cloud is not offered due to the security policy of the cloud services provider.

In case the CNP compiler or the *Lazarus* builder encounters errors, corresponding messages will be displayed in the output window. The text displayed

is a filtered version of *CompResult.txt*. After fixing the errors the user can initiate compilation again.

Currently the *Bouquet* CNP environment is hosted using Amazon Elastic Compute Cloud (Amazon EC2) web services [47]. We found this advanced but still convenient and user-friendly cloud services platform suitable and attractive. However, this is a paid service and we will have to find an alternative solution in the future.

The cloud CNP environment is installed on a remote virtual server (rented from Amazon WS). Windows 2008 R2 server with IIS7 is installed on the virtual server instance. The application is coded in ASP.NET and C# using web forms with code behind. Setting properly the server configuration actually proved to be probably the most difficult part of developing the CNP compiler as allowing an executable file (the CNP cloud compiler) to run on the server was a hard job.

5.2 The *SpiderCloud* CNP online compiler

Our second CNP online compiler, *SpiderCloud* has been described in [45]. Its look and feel remind those of the *Bouquet* compiler – basically, the user interface has the same three input and one output areas: CN editor area (for loading and editing the textual description of the CN), Primitives editor area (for loading and editing the file *SpiderUnit* which typically contains definitions of primitives and global variables), Main console editor area (for loading and editing the main program of the console application), and Messages area (for output messages related to the project compilation and building). The main menu consists of buttons for starting a new project, compiling a project, and downloading the generated executable file. This can be considered as a fifth, control area.

The general underlying idea of the *SpiderCloud* compiler functioning is also similar to the one used in *Bouquet*. However, its implementation is based on a very different technology.

For implementing the *SpiderCloud* CNP development environment we use the advanced IntraWeb technology [48], [49]. IntraWeb is a platform for easy and fast development of web applications (usually called weblications) [50] – [52]. A weblication is any type of application that is similar to what you would put into an executable, but instead deployed using a web browser and the Internet [50]. In other words, a weblication is similar in terms of interface and functionality to an ordinary application to be run on a local computer, but instead it is run on a web server and accessed from an Internet browser. This is exactly what an

online CNP development environment must be.

IntraWeb is the only tool specifically geared towards this type of development [50]. Its usage allows a developer to create complicated weblications extremely quickly and easily. While in a typical web application the developer must take care of all the details of communication (input/output) using different protocols and approaches, in IntraWeb all these tasks are automated. IntraWeb developers design and create their weblications in the same manner a usual Windows application is developed. They use control components and IntraWeb forms and define their properties and create corresponding events, similarly to using a visual control library (VCL). Based on [49], we can give the following concise definition of IntraWeb: IntraWeb is a component-based Web development framework combining the visual development normally associated with RAD tools with a robust collection of dynamic Web site creation features. It is included in Delphi 7 Studio, and is available for Delphi 5 and 6, Kylix 2 and 3, C++ Builder 5 and 6, and JBuilder 7. The code behind may be written in C#, VB, Java, C++, or Delphi.

Above, we have talked about web applications only. Actually, the other type of web site development mode is page mode (as distinct from application mode). In application mode, an IntraWeb application is a self-contained, state-maintaining executable that generates HTML and JavaScript for rendering by a Web browser.

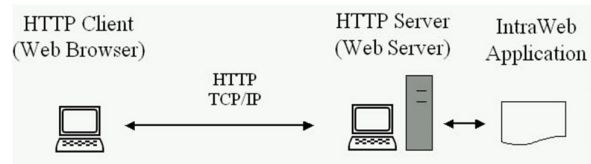


Figure 9a ASAPI type application

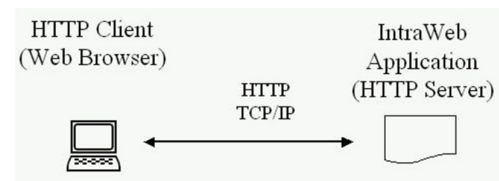


Figure 9b Stand-alone server-type application

Web applications developed in IntraWeb can be now divided into two application types depending on their deployment in order to be made available to

the Internet users. The first standard approach which is used in all other web application development platforms and can be used in IntraWeb is illustrated in Fig. 9a (borrowed from [49]). Typically, a web server such as Apache or Microsoft IIS must be installed and configured first on the web hosting machine. In IntraWeb this type of applications are called web server extensions or ASAPI type applications.

In the stand-alone mode (Fig. 9b), the IntraWeb application is a server itself, listening for HTTP requests on a specified port. In this mode, the IntraWeb server processes the browser's HTTP requests directly. The SA Server is a web application that includes three parts: an embedded web server, the IntraWeb core code, and the specific user code. The main goal of this approach is to allow the developer to easily deploy the application, without all the issues of installing and configuring a web server like IIS or Apache. As the stand-alone server application includes its own web server (which is an Indy based web server), one can simply execute the application and it is ready to receive requests and send responses.

Because of the simplicity and convenience of the second type of IntraWeb applications discussed above, the *SpiderCloud* CNP development environment has been implemented as a stand-alone server-type webapplication. It includes an embedded web server, IntraWeb core code, and the actual code of the Spider compiler. In order to become functioning and ready for use by Internet users, *SpiderCloud.exe* must be started on a cloud Windows server. As the HTTP server is embedded into the application, during the coding the ports for

receiving external requests and sending responses must be set. Usually IntraWeb applications are compiled for work with port 8888; however, setting port 80 for HTTP communication will avoid problems related to firewalls, proxies, etc.

The access to the so installed cloud CNP development environment may be done from any web browser using a link such as <http://ec2-54-241-120-135.us-west-1.compute.amazonaws.com:8080> or <http://54.241.120.135:8080>.

We can note that practically there is no need to install *SpiderCloud* on a sophisticated advanced cloud computing platform such as, for example, Amazon EC2. Multiple-user access to the environment can be achieved by simply using a home computer through a communication port with outside access permission.

The implemented architecture of the *SpiderCloud* webapplication is illustrated in Fig. 10. It consists of two main modules. *SpiderCodeCL* is a set of tools for initialization, upload/download, and text editors for the CN program components. *SpiderBuilder* includes the standard Spider compiler which is reused from the stand-alone CNP development environment *SpiderCNP* (Sec. 3.1, [16]). It also embodies *SpiderIntegrator* which creates the final executable file of the application.

5.3 Further development of cloud CNP development environments

As an emerging technology, today's online compilers are certainly not without their limitations. Most online compilers have yet to integrate reliable version control capabilities which are necessary on production projects, as well as to enhance their integrated auto-save functionality to temporarily make up for lost ground, for example when the internet connection is lost.

Bouquet and *SpiderCloud* are more than just compilers, more precisely – they are simple development environments. They are, however, still not cloud IDEs. They are missing many of the advanced features of cloud IDEs discussed in Sec. 4.1 and 4.2. In particular, we plan on adding users' registration and history control, tools for testing, error elimination and fine tuning of projects, means

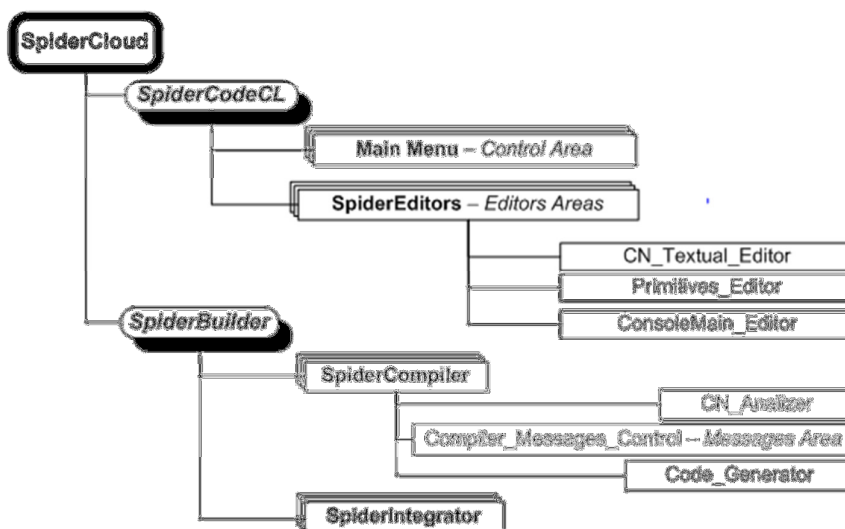


Figure 10 *SpiderCloud* architecture

for remote execution of the compiled projects. Development continues.

The first most important feature that needs further research and improvement would be the possibility to use the graphical editor in the cloud version of the environments.

Another important task is to integrate a cloud environment into an advanced learning system for studying and improving skills in CNP. Such a system is currently under development.

6 Conclusion

Different approaches to creating programming environments for developing and running CNP applications have been considered. Particular attention has been paid to our most powerful visual environment *SpiderCNP* which is run on a user's local computer, as well as to two versions of light-weight cloud-based environments ready to be run on remote servers which are especially suitable for use by persons learning CNP. It has been emphasized that "Primitives + Control network = CN program", with this postulate playing a fundamental role in designing a CNP environment.

References:

- [1] K.Kratchanov, B.Yüksel, T.Golemanov, and E.Golemanova, Learning Control Network Programming with the *Bouquet* Cloud Compiler. In: *Recent Advances in Educational Technologies and Education: Proc. 2014 Intl. Conf. on Educational Technologies and Education (ETE 2014), Interlaken, Switzerland, 2014*, pp. 29-36. Available also at: <http://www.europment.org/library/2014/interlaken/bypaper/EDU/EDU-02.pdf>.
- [2] K. Kratchanov, T. Golemanov, and E. Golemanova, Control Network Programming. In *Proc. 6th IEEE/ACIS Conf. on Computer and Information Science (ICIS 2007), July 2007, Melbourne, Australia*, pp. 1012-1018.
- [3] K. Kratchanov, E. Golemanova, and T. Golemanov, Control Network Programs and Their Execution. In *Proc. 8th WSEAS Int. Conf. on AI, Knowledge Engineering & Data Bases (AIKED '09), Feb 2009, Cambridge, UK*, pp. 417-422.
- [4] K. Kratchanov, T. Golemanov, and E. Golemanova, Control Network Programming: Static Search Control with System Options. In *Proc. 8th WSEAS Int. Conf. on AI, Knowledge Engineering & Data Bases (AIKED '09), Feb 2009, Cambridge, UK*, pp. 423-428.
- [5] K. Kratchanov, T. Golemanov, E. Golemanova, and T. Ercan, Control Network Programming with SPIDER: Dynamic Search Control. In *Knowledge-Based and Intelligent Information and Engineering Systems, Proc. 14th Intl Conf. (KES 2010), Cardiff, UK, Sep 2010, Part II, Lecture Notes in Computer Science (Lecture Notes in Artificial Intelligence)*, vol. 6277, Springer, 2010, pp. 253-262.
- [6] K. Kratchanov, E. Golemanova, T. Golemanov, and Y. Gökçen, Implementing Search Strategies in Winspider I: Introduction to Control Network Programming and Search. In *Knowledge-Based Automated Software Engineering*, I. Stanev, and K. Grigorova, Eds., Cambridge Scholars Publ., 2012, pp. 87-113.
- [7] K. Kratchanov, E. Golemanova, and T. Golemanov, Control Network Programming Illustrated: Solving Problems With Inherent Graph-Like Structure. In *Proc. 7th IEEE/ACIS Int. Conf. on Computer and Information Science (ICIS 2008), May 2008, Portland, Oregon, USA*, pp. 453-459.
- [8] K. Kratchanov, E. Golemanova, T. Golemanov, and T. Ercan, Non-Procedural Implementation of Local Heuristic Search in Control Network Programming. In: *Knowledge-Based and Intelligent Information and Engineering Systems, Proc. 14th Intl Conf. (KES 2010), Cardiff, UK, Sep 2010, Part II, Lecture Notes in Computer Science (Lecture Notes in Artificial Intelligence)*, vol. 6277, Springer, 2010, pp. 263-272.
- [9] K. Kratchanov, E. Golemanova, T. Golemanov, and Y. Gökçen, Implementing Search Strategies in Winspider II: Declarative, Procedural, and Hybrid Approaches. In *Knowledge-Based Automated Software Engineering*, I. Stanev and K. Grigorova, Eds., Cambridge Scholars Publ., 2012, pp. 115-135.
- [10] E. Golemanova, Declarative Implementations of Search Strategies for Solving CSPs in Control Network Programming. *WSEAS Transactions on Computers*, vol. 12, No.4, 2013, pp. 174-183.
- [11] N. Wirth, *Algorithms + Data Structures = Programs*. Prentice-Hall, 1975.
- [12] R. Kowalski, *Algorithms = Logic + Control*. *Comm. ACM*, vol. 22, 1979, pp. 424-436.
- [13] Z. Michalewicz, *Generic Algorithms + Data Structures = Evolution Programs*. Springer, 1992.

- [14] K. Kratchanov, E. Golemanova, T. Golemanov, and B. Kùlahçiođlu, Using Control Network Programming in Teaching Nondeterminism. In *Proc. 13th Int. Conf. on Computer Systems and Technologies (CompSysTech'12)*, Ruse, B. Rachev and A. Smrikarov, Eds., ACM Press, New York, 2012, pp. 391-398. Also, ACM Digital Library, <http://dl.acm.org/citation.cfm?id=2383333&dl=ACM&coll=DL&CFID=169141915&CFTOKEN=28327026>.
- [15] K. Kratchanov, E. Golemanova, T. Golemanov, and B. Kùlahçiođlu, Using Control Network Programming in Teaching Randomization. In *Int. Conf. Electronics, Information and Communication Engineering, Macau (EICE 2012)*, ASME, 2012, pp. 67-71. Also, in ASME Digital Library: <http://dx.doi.org/10.1115/1.859971.paper14>.
- [16] T. Golemanov, SpiderCNP - an Integrated Environment for Visual Control Network Programming. University of Ruse Annual, vol. 51, 2012, ser. 3.2, pp. 123-127 (in Bulgarian).
- [17] <http://www.embarcadero.com/products/delp hi>
- [18] <http://www.lazarus.freepascal.org/>
- [19] <http://cnprogramming.com>.
- [20] [http://msdn.microsoft.com/en-us/library/vstudio/a2c7tshk\(v=vs.100\).aspx](http://msdn.microsoft.com/en-us/library/vstudio/a2c7tshk(v=vs.100).aspx).
- [21] <http://msdn.microsoft.com/en-us/library/a2c7tshk.aspx>.
- [22] <http://forums.codeguru.com/showthread.php?369066-NET-Framework-IL-What-is-Language-Interoperability>.
- [23] M. Enevoldsen, *Object Oriented Language Interoperability* (Master's Thesis), Uni. Of Aarhus, 2004 (available at <http://users-cs.au.dk/beta/eclipse/mbeOOLI.pdf>).
- [24] http://en.wikipedia.org/wiki/Language_interoperability.
- [25] http://en.wikipedia.org/wiki/List_of_CLI_languages.
- [26] http://en.wikipedia.org/wiki/List_of_JVM_languages.
- [27] <http://www.fastcompany.com/3001010/cloud-computing>.
- [28] http://en.wikipedia.org/wiki/Cloud_computing.
- [29] <https://c9.io/>.
- [30] <http://www.compileonline.com/>.
- [31] <https://compilr.com/>.
- [32] <https://shiftdit.net/>.
- [33] <http://codepad.org/>.
- [34] <http://pythontutor.com/>.
- [35] <http://ideone.com/>.
- [36] <http://www.onlinecompiler.net/>.
- [37] <http://www.tutorialspoint.com/>.
- [38] <http://www.w3schools.com/>.
- [39] <http://codebender.cc/>.
- [40] <http://cloudcompiling.com/>.
- [41] <http://mbed.org/>.
- [42] <https://ludei.com/>.
- [43] <http://www.silverlightshow.net/items/Windows-Phone-8-Compile-in-the-Cloud.aspx>.
- [44] <http://www.codeproject.com/Articles/199537/What-are-Online-Compilers-Online-IDE-s>.
- [45] T. Golemanov, K. Kratchanov, and E. Golemanova. SpiderCloud – A Control Network Programming Environment Cloud-Based Environment. *Ruse Univ. Annuals*, v. 52, ser. 3.2, 2013, 131-136 (in Bulgarian).
- [46] <http://cnprogramming.com/download.html>
- [47] <http://aws.amazon.com/ec2/>.
- [48] <http://www.atozed.com/intraweb/index.en.aspx>
- [49] <http://edn.embarcadero.com/article/29650>
- [50] <http://www.atozed.com/intraweb/Weblications/index.EN.aspx>
- [51] <http://docs.atozed.com/docs.dll/getting%20started/IntraWeb%20Application%20Types.html>
- [52] <http://docs.atozed.com/docs.dll/Deployment/Deploying%20your%20application%20as%20a%20Stand%20Alone%20Server.html>