Traffic Simulation and Autonomous Driving Experiment in VIPLE

Yinong Chen(✉) and Gennaro De Luca

School of Computing and Augmented Intelligence, Arizona State University, Tempe, AZ, USA
{yinong.chen,gennaro.deluca}@asu.edu

Abstract. This paper presents the development of visual programming languages and summarizes the recent research in traffic simulation and autonomous driving experiments in VIPLE (Visual IoT/Robotics Programming Language Environment). The traffic simulator is developed in the Unity programming engine. The traffic simulator allows generation of different traffic patterns, either randomly or guided through a recorded real-world traffic dataset. Traffic experiments can be programmed in VIPLE or in Unity simulator, allowing developers to create such experiments without a deep understanding and lengthy support programming around the environment. Instead, the user can focus on implementing optimal routing algorithms that navigate a vehicle through the city traffic. The autonomous driving environment is based on TORCS simulation environment. A variety of difficulty levels of autonomous driving experiments can be created. Both traffic simulation and autonomous driving experiments can be written in visual programming language VIPLE, as well as in C# and Python within VIPLE environment. VIPLE has been widely used in different courses worldwide. This new advancement in VIPLE can particularly help students to perform traffic control and autonomous driving related programming, machine learning, and artificial intelligence research and experiments.

Keywords: Traffic simulation · dynamic routing · autonomous driving · computer science education

1 Introduction

Software engineering has evolved in several generations. The first generation was from the 1970s to 1980s, where the focus was to use structured programming languages to construct well-structure computer programs. The second generation of software engineering was from the 1990s to 2000s, where design patterns, modeling, and object-oriented programming were introduced. The third generation of software engineering was from the 2000s to 2010s, where architecture-driven development, service-oriented computing, cloud computing, and visual programming languages were introduced. The fourth generation of software engineering is based on big data processing, machine learning, and hardware-software co-design [1].

As a part of the third generation of software engineering, visual programming languages emerged in the 2000s, which allow software compositions through connecting existing components and services using orchestration and choreography styles [1, 2]. In the orchestration composition style, a central process takes control over the involved components and services and coordinates the execution of different operations. This composition style is defined by the following features.

- The involved components and services communicate with the central process only, within the application.
- The overall functionality is achieved by aggregating other components and services.
- The composition style is particularly useful for private business process, using independent components and services.
- BPEL (Business Process Execution Language) is the flagship language using this style.
- A complementary composition is choreography, which is defined by the following features.
- There is no central coordinator.
- Each component and service involved can communicate with multiple partners within the application.
- Each component and service decide how to interact with other services to consume their functionality.
- The composition style is particularly useful for public business process involving coordinated design of distributed services.
- WS-CDL is an example language using this style.

The main applications of visual programming environments and the composition styles are in business and in education, as visual programming environments help business developers and students to develop applications without deep understanding of computer organizations and programming language details at low level, such as syntax design, memory management, resource sharing, process synchronization, etc. [3]. As software engineering enters its fourth generation, some visual programming languages also include big data processing and machine learning capabilities. VIPLE is one such visual programming languages that has included fourth generation software engineering features [4].

In this paper, we introduce different visual programming environments used in business and in education. We focus on discussing the new features that are recently developed in VIPLE programming environment, including traffic simulation with machine learning capacity and autonomous driving, and quantum computing.

The basic parts of VIPLE have been used at different grade levels in middle and high schools. The advanced parts, such as traffic simulation, autonomous driving, and quantum computing are being used at senior and graduate levels at universities [5].

The rest of the paper is organized as follows. Section 2 gives an overview of the composition languages and their composition styles. Section 3 discusses VIPLE visual components and composition examples. Section 4 presents the recent development and new functions in VIPLE and their applications in traffic simulation, autonomous driving. We also briefly introduce quantum computing for machine learning. Finally, Sect. 5 concludes the paper.

2 Development of Visual Programming Languages

This section gives an overview of different visual programming language environments.

Visual programming language environments wrap components and services into visual blocks and enable drag and drop methods to compose workflows and business processes. The basic concepts of visual programming languages were discussed, and the existing attempts are surveyed in early research by Berkely scientists [6]. Another survey of visual programming languages was conducted by Mishra in 2017 [7]. New languages with practical applications were developed since 2000. Some of these languages are commercial products and others are widely used in computer science education. Figure 1 shows the major composition languages and environments that support visual composition of workflows and business processes.

There are full released and well documented visual programming languages. There are many more that are for research only. PSML-S was a research language that helped to establish the initial research of visual programming language development at ASU [8].

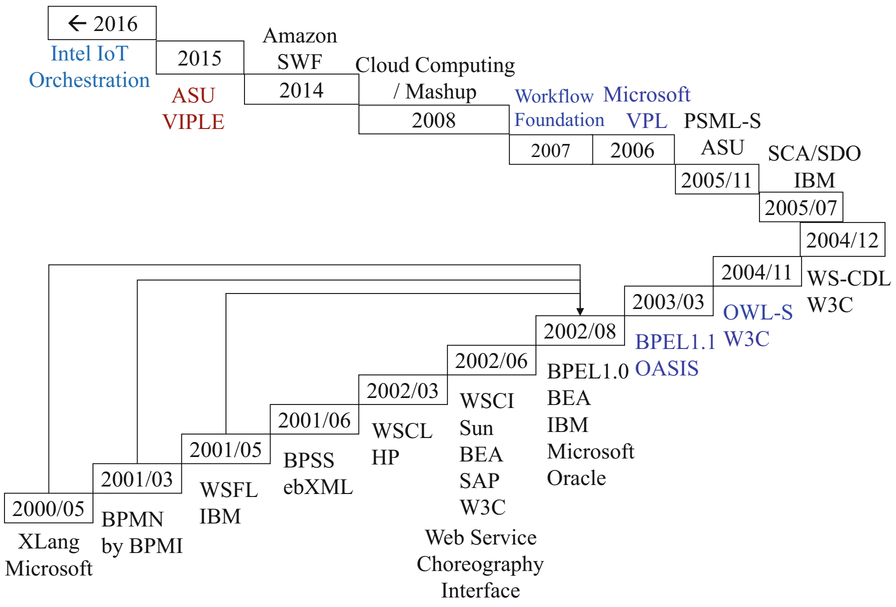


Fig. 1. Development of Visual composition languages and environments

Table 1 lists the architecture styles and the features of a few key languages that are used in business process composition. These languages are either visual notations themselves, such as BPMN [9] and WF [10], or have external visual tools to visualize the text notations, such as BPEL [11]. Some of these languages are standards defined by standard organizations, and these languages can have different implementations by different vendors, such as BPEL and BPMN. Some of the languages are proprietary to certain companies. For example, WF is part of Visual Studio by Microsoft, and SWF a part of Amazon Web Services cloud [12]. As indicated in Fig. 1, BPEL is defined by different companies jointly based on XLang, BPMN, and WSFL to make sure that it is not vendor dependent. It has become a major standard support across the industry.

Table 1. Key languages and features in business applications

Language environment	Architecture Style	Other features
BPEL 2002	Orchestration	Based on SOAP, WSDL, UDDI directory, widely used by large corps
BPSS 2001 BP Spec. Schema	Choreography	Business Process Specification Schema Based on SOAP, ebXML repository, CPP/CPA collaboration, for small biz
BPMN 2001 BP Modeling Notation	Orchestration	A superset of BPEL, supports advanced semantics & complex structures, by BPMI which merged with OMG in 2005
WSCI 2002	Choreography	WS Choreography Interface: Complementary to BPEL, submitted to W3C, not widely used
WS-CDL 2004	Choreography	Complementary to BPEL, W3C own proposal
Work-flow Foundation WF 2007	Orchestration and Choreography	A general programming environment allowing visual workflows and code activities for general computing and business processes
SWF 2014	Orchestration	Used for connecting Amazon lambda-functions and APIs to form business process in AWS cloud

In addition to business process oriented visual languages, there are a set of visual programming languages used for education. These languages allow students without programming background to implement their logic ideas and build their computational thinking concepts. These languages are used at different grade levels. Some languages are created by universities for public access, such as Scratch [13], Alice [14], App Inventor [15], and VIPLE [1, 3, 4]. Some are proprietary products associated with certain hardware products, such as Microsoft VPL [16] and Lego EV3 [17].

Table 2 lists a few key languages in this category, including their main purposes and grade level of students who can start to use the languages.

Table 2. Languages for computing education

Language	Main purpose	Starting from
MIT Scratch	Visual game and movie programming	Primary school and up
Alice	Visual game and movie programming	Middle school and up
MIT App Inventor	Phone App visual programming	High school and up
Lego EV3	Lego EV3 robot programming	Middle school and up
Microsoft VPL	Robot programming	High school and up
Intel IOT SOL	IoT and embedded system programming	College students
ASU VIPLE	General programming and robot programming	High school and up

3 VIPLE

VIPLE is a visual programming language environment designed for general-purpose programming. It particularly offers activities and services for IoT and robotics application building. This section reviews the general-purpose computing part, and the next section will present the recent new features of VIPLE.

As a general-purpose programming language, VIPLE offers the necessary programming constructs, such as assignment, calculate, if-then-else, and loop. Figure 2 shows the basic activities and general-purpose services that allow the construction of general-purpose programs.

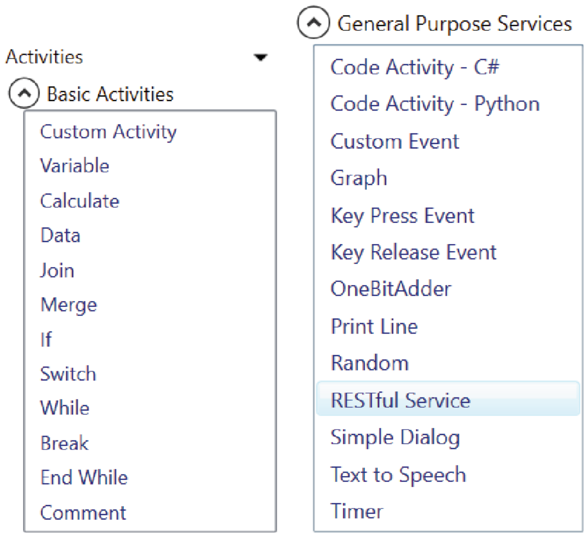


Fig. 2. VIPLE basic activities and general-purpose services

Figure 3 shows a VIPLE program that counts from 0 to 10. It uses Text to Speech service to read the current number and uses Print Line service to print "All Done" at the end of counting. A Merge basic activity is used for selecting the initial entrance of the loop and the reentrance of the loop. A variable Counter is used to store the increasing number, and a Calculate activity is used to perform add 1 operation.

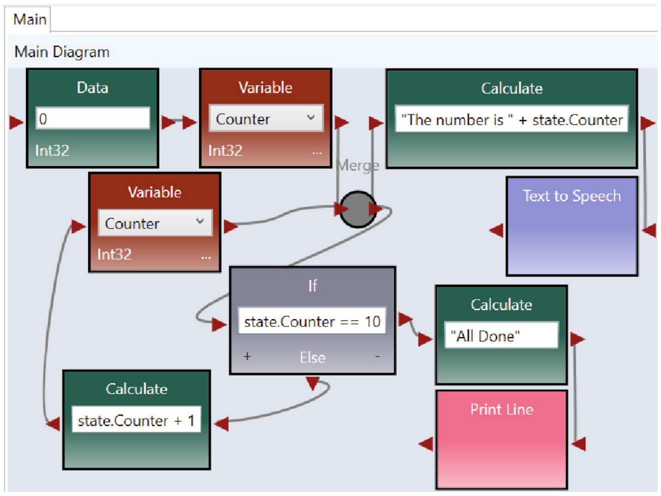


Fig. 3. A VIPLE program that counts from 0 to 10

The same program can also be implemented using a While loop activity.

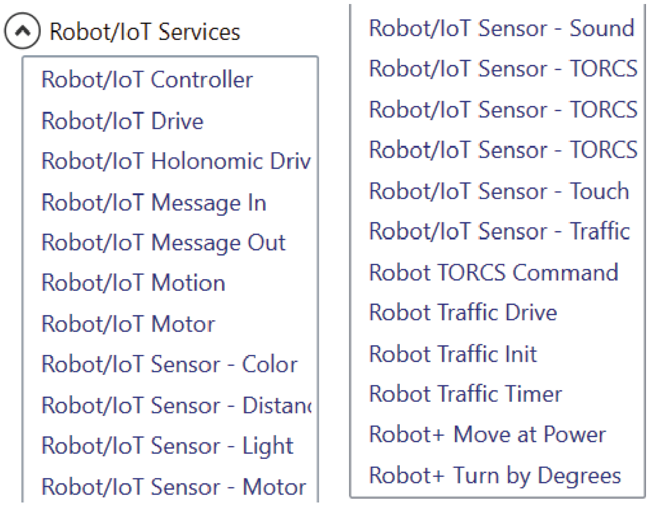


Fig. 4. VIPLE Robot/IoT services

VIPLE is called an IoT and robotics programming language, because it offers a large set of Robot/IoT services. Figure 4 shows the list of such services.

There are four sets of different types of services.

Robot/IoT Controller is the main service that defines a control center of a Robot/IoT system. One or more controllers can be used, specifying a single control center or multiple control centers.

The next set of services includes the Robot/IoT sensor, drive, and message services. These services will be associated with one of the controllers to give the controller input or to take output from the controller to perform actions.

The third set of services are TORCS services for autonomous driving experiments, which will use TOCS simulator to exhibit the driving simulation. We will discuss this set of services in the next section.

The last set of the services included is the traffic simulation services that will communicate with a Unity Traffic Simulator. We will discuss this set of services in the next section.

Figure 5 shows the commands that can start different functions.

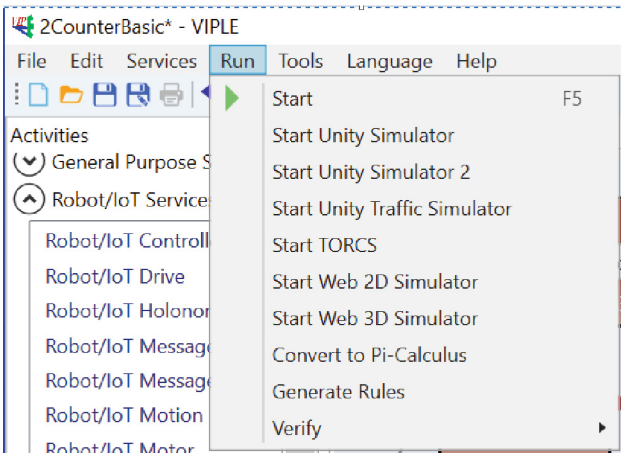


Fig. 5. VIPLE Start commands

The first command Start will start the execution of a VIPLE program. If a simulator is involved, we need to start the simulator first before we start the VIPLE program.

The next three Start commands will start one of the three Unity simulators embedded into VIPLE. We will discuss Start Unity Traffic Simulator in the next section.

The next Command is to start TORCS simulator [18].

The next two commands are to start the Web 2D maze simulator and to start the Web 3D maze simulator, respectively. These simulators are developed within VIPLE environment [1].

In addition to the simulators, VIPLE also supports Pi-Calculus based program analysis, proof, and verification [19, 20], which are shown in the next three commands in Fig. 5.

4 Latest Additions to VIPLE

This section presents the latest developments in VIPLE, including traffic simulation, autonomous driving, and quantum computing.

4.1 Traffic Simulation

A traffic simulator is recently developed using Unity game engine. The simulator will generate a city map and random or controlled traffic (cars marked in yellow) on the streets. Once started the simulation, a red car will appear in the map, and it can be programmed in VIPLE to travel from the current position to any position on the map. Figure 6 shows the overall map with clusters of dense areas connected by freeways.

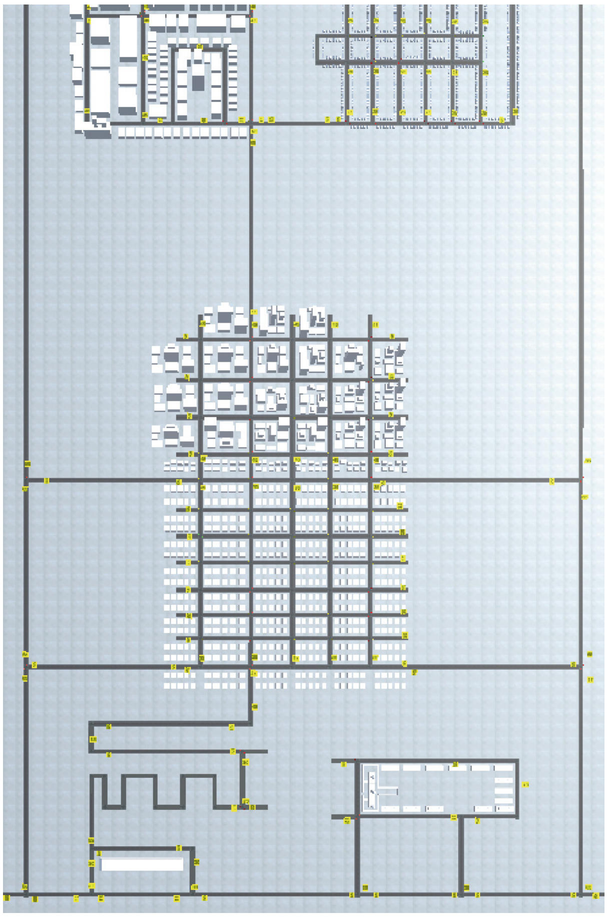


Fig. 6. The overall traffic map

Figure 7 shows a part of the map with the red car, multiple yellow cars, and the controls on the map. The controls on the map can define the traffic density (number of yellow cars in the map), destination of the red car, planning path and trace of the red car, showing different areas of the city, different views of the map, zoom in and zoom out, etc.

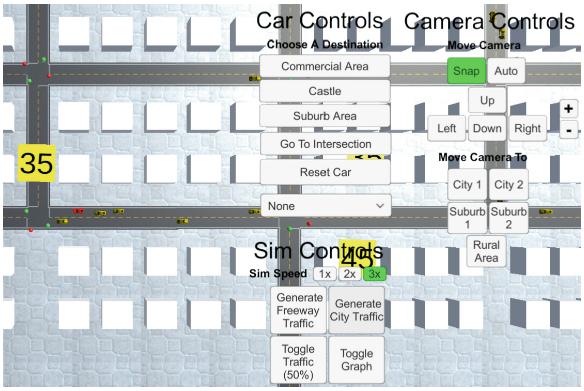


Fig. 7. Traffic map showing controls and cars in streets

The map contains traffic lights, traffic signs (e.g., stop sign), speed limits on each section of streets, distance of each section of streets, etc. These pieces of information are sent to VIPLE in real time and VIPLE can program the red car to implement the shortest-distance path to the destination and the shortest-time path to the destination.

The traffic on the streets can be generated randomly or guided by know traffic patterns. In the latest study, we have used Arizona Maricopa government published data to guide our traffic generation and write VIPLE program to find the shortest-time path [21]. A video demonstrating the traffic simulation with dynamic shortest distance computing in the simulation environment is available at [23].

4.2 Autonomous Driving

Another major development in VIPLE is the TORCS autonomous driving experiments. VIPLE makes autonomous driving experiments simple to write and easy to understand. Figure 8 shows a simple VIPLE program. The main code is in the Code Activity in C#, as shown in Fig. 9.

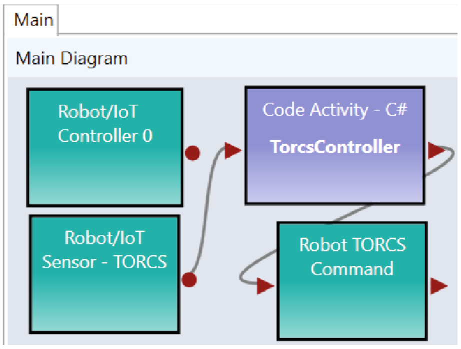
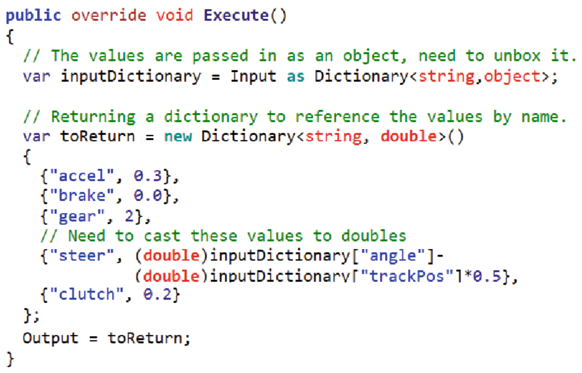


Fig. 8. Simple VIPLE code for autonomous driving

The code keeps the car in the middle of the racetrack, maintaining acceleration at 0.3, driving slowly at gear 2, and not applying brake at all. The car can complete the racetrack in about 74 s.



```
public override void Execute()
{
  // The values are passed in as an object, need to unbox it.
  var inputDictionary = Input as Dictionary<string,object>;

  // Returning a dictionary to reference the values by name.
  var toReturn = new Dictionary<string, double>()
  {
    {"accel", 0.3},
    {"brake", 0.0},
    {"gear", 2},
    // Need to cast these values to doubles
    {"steer", (double)inputDictionary["angle"]-
              (double)inputDictionary["trackPos"]*0.5},
    {"clutch", 0.2}
  };
  Output = toReturn;
}
```

Fig. 9. Code Activity in the VIPLE code

Obviously, this program is not optimal to completing the racetrack. In a real race program, we need to raise the gear as the speed increase, maximize the acceleration when the track is straight, and need to brake before the curve. An optimized VIPLE program is shown in Fig. 10.

This program can complete the racetrack in 54 s, reducing from 74 s.

VIPLE also supports two-car racing, where each car also needs to sense the other car to avoid the collision. Figure 11 shows the experiment with two cars in the same racetrack. We can ally the same algorithm or two different algorithms for the two cars.

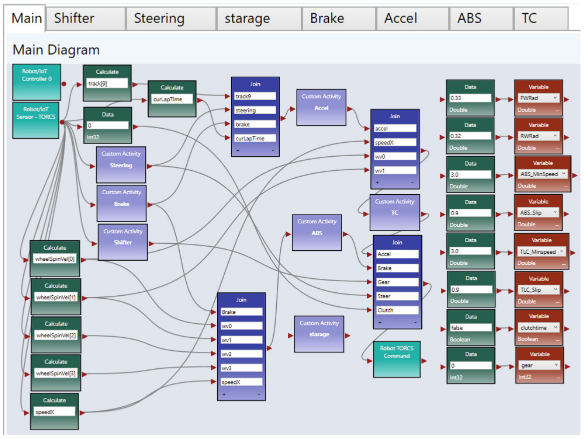


Fig. 10. Optimized VIPLE code for autonomous driving

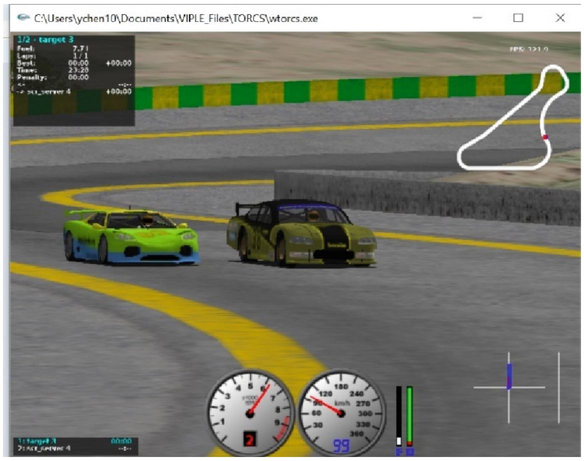


Fig. 11. Two car races in the same racetrack

4.3 Quantum Computing

The latest development in VIPLE is the inclusion of quantum computing, particularly for accelerating machine learning [22]. Figure 12 shows Quantum Basic Activities and Quantum Services that have been built in VIPLE.

From these activities and services, different computationally intensive applications can be developed, particularly for accelerating machine learning training and testing.

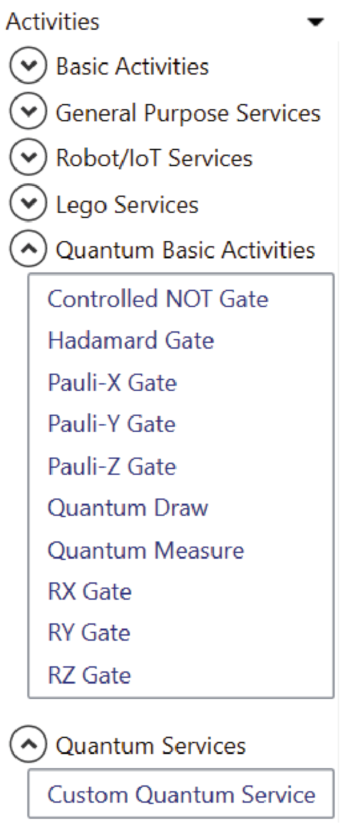


Fig. 12. VIPLE built-in Quantum Basic Activities and Quantum Services

5 Conclusions and Future Work

This paper presented the development of visual programming languages in business and education applications. VIPLE, developed by the authors, was presented and its new functions were introduced. VIPLE has been used as a teaching tool in many schools and universities around the world. The basic parts can be used at different grade levels in schools and the advanced parts, such as traffic simulation, autonomous driving, and quantum computing are being used at senior and graduate levels at universities.

Acknowledgments. This research is supported by the general fund of the IoT and Robotics Education Lab at Arizona State University.

References

1. Chen, Y., De Luca, G.: Service-Oriented Computing and System Integration: Software, IoT, Big Data, and AI as Services, 8th edn. Kendall Hunt Publishing (2022)

2. Megargel, A., Poskitt, C., Shankararaman, V.: Microservices orchestration vs. choreography: a decision framework. In: IEEE 25th International Enterprise Distributed Object Computing Conference (EDOC) (2021). https://doi.org/10.1109/EDOC52215.2021.00024
3. De Luca, G., Li, Z., Mian, S., Chen, Y.: Visual programming language environment for different IoT and robotics platforms in computer science education. CAAI Trans. Intell. Technol. **3**(2), 119–130 (2018). https://doi.org/10.1049/trit.2018.0016
4. Chen, Y., De Luca, G.: VIPLE: visual IoT/robotics programming language environment for computer science education. In: 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) (2016). https://doi.org/10.1109/ipdpsw.2016.55
5. De Luca, G., Chen, Y.: Semantic analysis of concurrent computing in decentralized IoT and robotics applications. In: Proceedings of ISADS, Utrecht, the Netherlands, pp. 95–102 (2019)
6. Boshernitsan, M., Downes, M.: Visual Programming Languages: A Survey, University of California, Berkeley, Technical Report No. UCB/CSD-04–1368, December 2004. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2004/CSD-04-1368.pdf
7. Ray, P.P.: A survey on visual programming languages in internet of things. Sci. Program. (2017).https://doi.org/10.1155/2017/1231430
8. Tsai, W.T., Paul, R., Xiao, B., Cao, Z., Chen, Y.: PSML-S: a process specification and modeling language for service oriented computing. In: The 9th IASTED International Conference on Software Engineering and Applications (SEA), Phoenix, pp. 160–167 (2005)
9. BPMN, Business Process Model and Notation 2.0 (2010). https://www.omg.org/spec/BPMN/2.0/
10. Scribner, K.: Microsoft Windows Workflow Foundation Step By Step. Microsoft Press (2007)
11. Saraswathi, R., Singh, J.: Oracle SOA BPEL Process Manager 11gR1 - A Hands-on Tutorial. Packt (2013)
12. AWF, Amazon Simple Workflow Service, Developer Guide, API Version 2012–01–25. https://docs.aws.amazon.com/amazonswf/latest/developerguide/swf-dg.pdf#swf-dg-intro-to-swf
13. Getting Started with Scratch (2022). https://sip.scratch.mit.edu/wp-content/uploads/2020/06/Getting-Started-With-Scratch-3.0.pdf
14. Rogler, H.: Alice Programming, 2nd edn. Kendall Hunt Publishing (2016)
15. Lang, K., Tezel, S.: Become an App Inventor. MITeen Press (2022)
16. VPL, Microsoft Visual Programming Language. https://en.wikipedia.org/wiki/Microsoft_Visual_Programming_Language
17. Jawaharlal, M.: LEGO EV3 Robotics: A Guide for Educators. Red Gerbera (2015)
18. TORCS, the open race car simulator, Sourceforce. https://sourceforge.net/projects/torcs/
19. De Luca, G., Chen, Y.: Visual IoT/robotics programming language in pi-calculus. In: 2017 IEEE 13th International Symposium on Autonomous Decentralized System (ISADS) (2017). https://doi.org/10.1109/ISADS.2017.32
20. Zhao, S., Li, Y., Wang, Y., Chen, Y.: Validating trustworthy service composition through VIPLE and pi-calculus. Int. J. Simul. Process Model. **15**(1–2), 76–88 (2020). https://doi.org/10.1504/IJSPM.2020.106971
21. Zhang, Z., De Luca, G., Archambault, B., Chavez, J., Rice, B.: Traffic dataset for dynamic routing algorithm in traffic simulation. J. Artif. Intell. Technol. (2022). https://doi.org/10.37965/jait.2022.0106
22. De Luca, G.: A survey of NISQ era hybrid quantum-classical machine learning research. J. Artif. Intell. Technol. **2**(1), 9–15 (2021). https://doi.org/10.37965/jait.2021.12002
23. Zhang, Z., De Luca, G., Chen, Y.: Dynamic Dijkstra algorithm traffic simulation (2022). https://venus.sod.asu.edu/VIPLE/Videos/DynamicDijkstra.mp4