Building Smart Agriculture Applications Using Low-Code Tools: The Case for DisCoPar

Isaac Nyabisa Oteyo*†, Angel Scull Luis Pupo†, Jesse Zaman†, Stephen Kimani*,
Wolfgang De Meuter† and Elisa Gonzalez Boix†
†*Software Languages Lab Vrije Universiteit Brussel, Brussels, Belgium*
**School of Computing & IT, Jomo Kenyatta Univ. of Agric. & Tech., Nairobi, Kenya*

***Abstract*—Modern farming is experiencing increased adoption of mobile and cloud computing applications as efforts are being made to automate farming processes. In this context, the mobile and cloud computing applications, that we refer to as smart agriculture applications (SAAs), can be used in data collection that can be entered directly into the applications by end-users (i.e., farmers) or via sensors. Implementing SAAs is often done using text-based techniques that require advanced skills and experience in software programming. There are low-code development tools (LCDTs) that farmers with less programming experience can use to implement their own SAAs. The LCDTs offer different features and techniques for application development e.g., some employ form-based application specification while others use graphical drag-and-drop techniques. As such, the different LCDTs are best suited for different specific tasks. For instance, a farmer may need to implement an application that connects to sensors to receive data and generate timely notifications when set thresholds are exceeded. However, to the farmer, it can be difficult to know which kind of LCDTs to choose and what category of these tools are best suited for the task. In this paper, we contrast different LCDTs and show how to use DisCoPar to develop SAAs by non-expert programmers. As a contribution, this paper presents properties for LCDTs that can be beneficial to farmers and demonstrates how DisCoPar can be used in developing SAAs.**

***Index Terms*—pervasive computing, mobile applications, SA applications, distributed computing, visual programming**

I. Introduction

Smart agriculture (SA), which advocates for using information and communication technologies (ICTs) in farming activities, is a promising revolution for modern farming to realise the global demand for food and nutrition security [1]. In this regard, SA puts emphasis on automating farming processes for increased farm productivity such as using sensors to monitor soil nutrient levels and mobile applications to predict farm yields. Broadly, the automation of farming processes can be done using mobile and cloud computing applications. In this work, we refer to such applications as smart agriculture applications (SAAs). In particular, these SAAs are gaining increased popularity among farmers in developed and developing regions [2]–[4]. This increasing popularity is continuously putting a demand for developers of SAAs that if not addressed can limit the full exploitation of ICTs in modern farming. For instance, developers are required that can program different kinds of sensor devices and at the same time develop corresponding mobile and web applications. In addition, the same developers are required to offer user support services to farmers such as configuring data collection surveys in the SAAs – tasks that can be done by the farmers themselves. On the other hand, constructing SAAs is a complex undertaking that requires advanced programming skills [5]. For instance, modern SAAs are often composed of many layers that talk to each other in a distributed way. These layers are a cross-cutting concern that can be abstracted from the application's domain. Moreover, implementing these layers can take considerable time, effort, and resources that may not be worth for low to medium-scale farmers to invest in hiring experienced programmers. As such, some tasks can be handled by farmers with less programming experience using LCDTs such as: (i) configuring SAAs for different crops in different seasons, (ii) changing data collection surveys in SAAs for various crops, and (iii) experimenting on modern farming aspects like the rate of reading data from sensors and information processing. By definition, LCDTs are visual programming environments that permit developers such as farmers that do not have strong programming experience to produce complex software systems [6], [7]. To farmers, LCDTs provide features that they can use to specify SAAs as workflows of connected components that provide application services. There exists different LCDTs in the literature such as Mendix[1] and WaveMaker[2] that offer different application development features and techniques. Moreover, these LCDTs are best suited for different tasks. However, to the farmer, being presented with a set of LCDTs, it can be challenging to know which tools to choose that are best suited for the task at hand.

This paper begins by contrasting LCDTs in terms of features and techniques that they provide for application development. We then demonstrate how to build SAAs using DisCoPar, a component-based LCDT. We extend the tool with components to register and receive data from sensors, filter the data for display and show notifications as pop-up messages. These components are important to support implementing SAAs that can benefit from sensor data. Showing notifications as pop-up messages can enhance timely communication of farm conditions to the farmers. Based on this, the goal of this paper is to depict the status of existing LCDTs that can be used to develop SAAs. This work also pursues the identification of the potential benefits that LCDTs can bring to farmers in

[1]https://www.mendix.com/low-code-guide/
[2]https://www.wavemaker.com/

terms of developing their own SAAs. Lastly, this work models communication to sensors that can be used to monitor farm environmental conditions in DisCoPar.

II. Motivation and Background

Essentially, low-code tools are based on graphical user interface (GUI) techniques in designing applications as opposed to text-based techniques. Moreover, recall from the definition of LCDTs in Section I that LCDTs are visual programming tools that can make application development more intuitive. For instance, in visual programming, an application is constructed by dragging-and-dropping blocks on GUIs and then connecting the blocks together. The blocks can be arranged either as an overlay of application elements side-by-side as shown in Fig. 1(a) or connected via links as shown in Fig. 1(b).

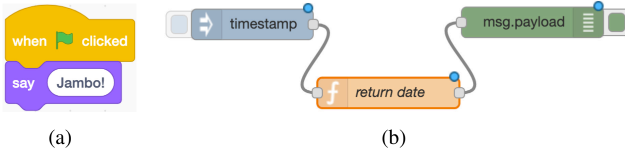


Fig. 1: (a) Overlay of blocks in Scratch programming, and (b) Connecting blocks via links in Node-RED.

The drag-and-drop feature is fundamental in LCDTS to make application development easier. This implies that with LCDTs, it can be quick to generate and deliver applications with minimum effort to both experienced and non-experienced developers [7]. For instance, some LCDTs such as Mendix and WaveMaker were designed to help experienced developers improve on their productivity to deliver business software applications. In addition to the drag-and-drop feature, LCDTs provide: (i) data centric features that can make it easy to integrate new data sources for data-centric applications, and (ii) cloud-based architectures with application development elements. The cloud-based architectures can reduce the burden of locally installing and setting up software development kits.

The GUI-based techniques coupled with the above features are making LCDTs popular to both experienced and non-experienced developers [8]. Non-experienced developers can use LCDTs to develop applications with less concern and focus on text-based techniques. Based on the features supported, different LCDTs are suited for different tasks. As such, it is important to perform a comparison of LCDTs based on the supported properties as we describe in the subsequent sections.

A. Criteria for Comparing LCDTs

This section presents the criteria for comparing LCDTs such as focus domain and application development techniques.

a) Focus domain: This feature is important to know where each tool can be applied e.g., business process modelling (BPM) and application development. BPM is aimed at identifying potential areas of improvement such as crop monitoring in modern farming. On the other hand, application development is meant to yield software products such as web and mobile applications that can be used in modern farming.

b) Application development techniques: These techniques show the kind of features that are available for application development such as visual modelling and drag-and-drop. Both of these features are supported by GUI techniques to help farmers construct applications. The drag-and-drop feature can allow farmers to select application elements and drop them on a visual programming environment, while visual programming can allow the farmers to visually model the application.

c) Application type: This intuitively explains where the application is going to run. For instance, web applications can run on browsers both on PCs and mobile devices, while mobile applications can run on mobile devices such as smartphones and tablets.

d) Application representation: This shows the mechanisms that can be used to represent an application such as flow-graphs or layers of application elements. Both mechanisms can allow the farmer to have a quick overview of the entire application structure. Intuitively, the flow-graph can show how data flows within the application.

e) Tool type: This shows whether the LCDT supports entirely visual programming or a mixture of visual and text-based programming. Purely visual LCDTs often require zero text-based development, while hybrid tools require a mixture of two worlds i.e., visual and textual programming. Whereas, the hybrid tools can be more flexible compared to the purely visual tools, they can impose an extra difficulty level to the farmers since they have to know and understand required code modifications.

f) Support for APIs: The support for third party APIs gives an indication as to whether new data sources can be added to an application or an existing application can be integrated with new services. The APIs can also help when integrating SAAs to other software systems.

g) Distributed: Often, SAAs connect to sensors that are spread across the farm and in different geographical locations. This presents a distributed setting where communication is important. However, in this setting the communication can be complex. As such, the LCDTs need to abstract the communication from farmers in components.

h) Reconfigurability: Low to medium scale farmers often practice rotational cropping where they grow different crops in different seasons. For SAAs, this implies a high rate of maintainability for the application to be reusable. As such, LCDTs should provide reconfigurability features e.g., for changing data collection surveys or changing the data reading rate from sensors.

B. State of the Art Analysis

Table I gives a summary on the state-of-the-art (SOTA) for different LCDTs based on the criteria established in Section II-A.

a) Focus domain: From the SOTA analysis, LCDTs like Aurea BPM [7] and DISME [12] are focussed on BPM, while other tools like OutsSystems[3], Mendix, Salesforce[4], Servi-

[3]https://www.outsystems.com/low-code-platforms/

[4]https://www.salesforce.com/eu/products/platform/overview/

TABLE I: Analysing and contrasting low-code development tools.

Tool	Focus area	AppDev technique	App type	App representation	Tool type	API support	Distributed	Reconfigurable
Aurea BPM [7]	BPM	VM	Web	Flow-graph	Hybrid	✗	–	–
Mendix [9] [10]	AppDev	GDD	Mobile & web	Flow-graph	Hybrid	✓	✓	✓
AppGyver	AppDev	GDD	Mobile & web	Layers of app elements	Pure visual	✓	✓	✓
OutsSystems [11]	AppDev	VM	Mobile & web	Layers of app elements	Pure visual	✓	✓	✓
DISME [12]	BPM	FBS	Web	Flow-graph	Hybrid	✗	–	–
Salesforce	AppDev	GDD	Mobile & web	Layers of app elements	Hybrid	✓	✓	✓
ServiceNow	AppDev	GDD	Mobile & web	Layers of app elements	Hybrid	✓	✓	✓
Netcall	AppDev	VM	Mobile & web	Flow-graph	Hybrid	✓	✓	✓
WaveMaker	AppDev	GDD	Mobile & web	Layers of app elements	Hybrid	✓	✓	✓
Thinkwise	AppDev	VM	Mobile & web	Flow-graph	Hybrid	✓	✓	✓
DisCoPar [13]	AppDev	VM, GDD	Mobile & web	Flow-graph	Pure visual	✓	✓	✓

BPM:- Business process modelling; AppDev:- Application development; VM:- Visual modelling; GDD:- Graphical drag-and-drop; FBS:- Form-based specification

ceNow[5], Netcall[6] and Thinkwise[7] are focussed on application development.

b) Application development techniques: Some LCDTs offer visual modelling options for application development such as Aurea BPM, OutsSystems, Netcall, and Thinkwise. Other LCDTs support graphical drag-and-drop functionalities such as Mendix, AppGyver, Salesforce, ServiceNow, and WaveMaker. From Table I, only DISME supports form-based specification technique. DisCoPar supports both visual modelling and graphical drag-and-drop techniques.

c) Application type: From Table I, most of the LCDTs support implementing mobile and web applications. Only, Aurea BPM and DISME do not support mobile applications.

d) Application representation: For LCDTs, application representation can be done in two ways: (i) using flow-graphs and (ii) using an overlay of application elements. From Table I, Aurea BPM, Mendix, DISME, Netcall, Thinkwise, and DisCoPar use the flow-graph way. In this approach, application elements are linked via connections. The connections are used as data transfer channels. AppGyver, OutsSyetems, Salesforce, ServiceNow, and WaveMaker use an overlay of application elements. As the name suggests, in this approach, applications elements are laid side-by-side.

e) Tool type: AppGyver[8], OutsSystems and DisCoPar [13], are purely visual. This implies that farmers are only required to specify configuration settings for application elements. The hybrid LCDTs such as Aurea BPM and Mendix require some textual modifications to application parts.

f) Support for APIs: From Table I, only Aurea BPM and DISME do not support APIs; all the other LCDTs offer support for APIs. As mentioned before, these APIs can be used to connect to new data sources and integrating to other software systems in modern farms.

g) Distributed: Most of the LCDTs presented in Table I support developing distributed applications. For example, DisCoPar uses components that can execute on the client-side and the server-side. Connecting client-side components to those on the server-side automatically yields distributed applications.

[5]https://www.servicenow.com/
[6]https://www.netcall.com/
[7]https://www.thinkwisesoftware.com/en/
[8]https://www.appgyver.com/

h) Reconfigurability: Most LCDTs support this feature through configurable surveys and component. For example, DisCoPar supports configurable surveys that can be reconfigured to collect data for different crops in different farming seasons.

C. Key Conclusions from SOTA Analysis

From the SOTA analysis, we make the following conclusions: (i) some LCDTs like DisCoPar combine different application development techniques which can be beneficial to the farmers, and (ii) pure visual LCDTs can be easier to use for non-experienced farmers compared to the hybrid LCDTs. Based on these conclusions, ideal LCDTs for SAAs should have a combination of features such as visual modelling, drag-and-drop, flow-graphs for application representation, distributed, reconfigurable, and purely support visual programming. We believe that this combination of features is important for farmers with less programming experience. From the LCDTs presented in Table I, DisCoPar supports all the above features and therefore we consider it ideal for developing SAAs. In the subsequent section, we describe DisCoPar and show how it can be used to build SAAs.

III. DisCoPar Explained

DisCoPar is based on the visual flow-based programming approach where applications are represented as graphs of connected components. As such, the main ingredients for an application in DisCoPar are: (1) flow-graph, (2) components, and (3) component links.

1) Application flow-graph: Fig. 2 shows an application flow-graph composed of four components i.e., $C1, C2, C3$, and $C4$. The flow-graph represents linked services offered by the components. Depending on their position in the flow-graph, components can be either in the upstream or the downstream. Upstream components come before, while downstream components come after the reference component. For instance, in Fig. 2, using $C3$ as the reference component, components $C1$ and $C2$ are upstream components, while component $C4$ is a downstream component.

2) Components and component categories: In DisCoPar, components represent services in the application. As such, DisCoPar provides two categories of components: (i) data processing components and (ii) data viewing components. Data

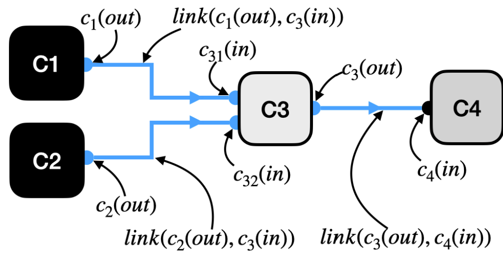


Fig. 2: Application flow-graph with four components.

processing components perform computations on data and produce some output. The data viewing component are used to build graphical user interfaces for applications and displaying data that flows through the application. Both categories can have *source* and *sink* components.

3) Component execution scope: In order to support distributed applications, components have an execution scope. As such, there are three main execution scopes for components i.e., mobile, server, and web. The components that have a mobile scope execute on the mobile client, while those with a server scope execute on the server-side. Components that have a web scope execute on the web and act as the dashboard for the server-side. In the application flow-graph, components scopes are distinguished through colours e.g., in Fig. 2, components $C1$ and $C2$ are coloured black to show that they can only execute on the mobile device. Components that execute on the server are coloured light-grey as shown in $C3$, while those that execute on the web are coloured grey as shown in $C4$.

4) Component ports: Components can have zero, one or multiple input and output ports. Input ports are used for channeling data to the component, while the output ports are used to channel computation results out of the component. In Fig. 2, components $C1, C2$, and $C3$ have one output port each (i.e., $C1(out), C2(out)$, and $C3(out)$ respectively), while component $C4$ has no output port. In this case, component $C4$ is referred to as a *sink* component since it only receives data. Similarly, components $C1$ and $C2$ have no input ports, while component $C3$ has two input ports ($C31(in)$ and $C32(in)$) and component $C4$ has one input port ($C4(in)$). Components with at least one output port are referred to as *source* components. Source components push data to components in the downstream, while sink components receive data from the upstream. Component ports have specific colours that show the kind of data they emit or receive. For example, the port colour in $C4$ is black to indicate that the component can receive `any` type of data. The blue colour in $C1, C2$, and $C3$ represents `numeric` data.

5) Component links: Components in the application flow-graph are connected together via links that originate from the source component port to the destination component port. For example, $link(C1(out), C3(in))$ in Fig. 2 that connects component $C1$ as the source component to $C3$ as the destination component. The arrows on the links show the direction in which data is flowing through the application, while the link colour shows the type of data flowing through the link. The link colour is inherited from the port colour of the source component. Linking components that have mobile or web scopes to those that have a server scope automatically yields a distributed application.

IV. Smart Agriculture Applications in DisCoPar

Often, SAAs benefit from environmental data collected via sensors. Therefore, LCDTs for SAA should provide the means to access this data e.g., by enabling plugin sensors in the application. As such, in order to support SAAs, for each incoming sensor data to the server-side, we needed a component to register to the data source and configure it to extract parts of the data e.g., temperature, humidity, and soil moisture values. In addition, for the temperature data received, we needed to perform a comparison to the set threshold and fire a notification whenever the threshold was exceeded. As such, we needed a notification component to display alerts on the dashboard by means of pop-ups. Existing DisCoPar components were not enough for the tasks that we needed SAAs to perform such as displaying notifications as pop-ups and connecting to sensors to receive data. However, the components in DisCoPar can easily be extended. Therefore, we extended DisCoPar with components to: (i) register and receive data from sensors, (ii) filter and extract respective data from the edge, and (iii) display alerts as pop-ups.

A. Application Scenario and Architecture

The application that we present in this section is aimed at connecting to sensors to receive data on environmental conditions such as temperature, humidity, and soil moisture for crops. These conditions are essential for crop development and growth; and they vary for different crops. For example, different species of the common beans (*Phaseolus vulgaris L.*) that are grown in developing regions require different environmental conditions for better development and growth. As such, having accurate information about the required conditions, can help the farmer to make decisions on when to irrigate these crops and the required amount of water based on the measured soil moisture levels. Overall, the application contributes towards the data collection process that is vital in modern farming.

Fig. 3 shows the high level architecture for the application prototype that we consider. This architecture is composed of two main parts: (i) sensor components and (ii) a visualisation dashboard which is composed of data viewing components. The sensor components run at the server from where they can connect to the physical sensors. The visualisation components execute on the mobile phone to display received data and notifications. In Fig. 3, these components are represented as *label* and *alert* on the mobile application part.

B. Communication between Application Server and Sensors

In our prototype application, we use M5StickC ESP32-PICO hardware (orange device in Fig. 4) that comes integrated with WiFi and 4MB flash memory capabilities. We attach

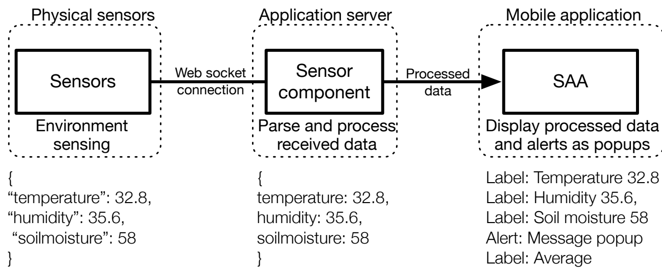


Fig. 3: Prototype application architecture.

temperature, humidity, and soil moisture sensors to the hardware as shown in Fig. 4. In order to support communication between the attached sensors and the components running on the application server, we deploy firmware code that is based on *duktape*[9] and *Arduino*. We use *duktape* to lay ground for our future iterations that will be based on JavaScript to conform to the language used in implementing components in DisCoPar. In addition, we use the web socket plugin[10] for Arduino to support communication between the sensors and the application server. Finally, we use the ArduinoJson plugin[11] to send sensor data in JSON format to the component running on the application server.

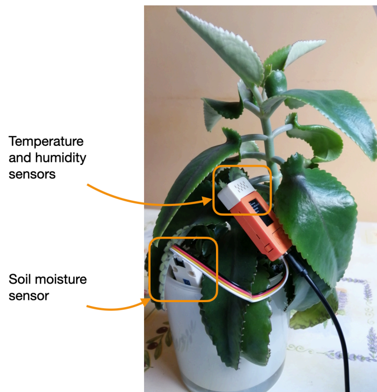


Fig. 4: Sensors deployed to a plant in a pot to send data to the prototype application.

C. Application Prototype

To validate the scenario depicted in Fig. 3, we implemented a prototype SA application using DisCoPar as shown in Fig. 5. From a software engineering perspective, the application presented in Fig. 5 has two parts that are arranged and put together as layers: (i) presentation layer which executes on the client-side and (ii) business layer which can execute both on the client-side and the server-side. The business layer which controls the application functionalities cuts across the two layers because: (i) some processing can be done in both locations, and (ii) the server-side can be used to control a large number of sensors. From the application flow-graph, the presentation layer is represented by the viewing components such as the `Label` component, while the business layer is represented by the source components such as `WebSocketInput` and `Average` components.

Fig. 5 shows the prototype application flow-graph and the resulting mobile application in use. The application flow-graph is read from left-to-right. The arrows on the links connecting the components show the direction in which data is flowing through the graph. As mentioned before, the black coloured components execute on the mobile phone and the light-grey coloured components execute on the server-side. The application registers to receive data from physical sensors using web sockets via the `WebSocketInput` component. The data received by this component is passed through the `FilterData` component that splits the received data into respective data elements. The filtered data is then passed through the `Rounding` components for precision setting. The `Label` components display the data on the application GUI. For computations, the application uses the components on the mobile side such as the `Counter` and `Average` components to keep track of the number of readings and the averages. Additionally, to compare values and trigger alerts, the application uses the `Threshold` and `Compare` components to set the limits and perform the comparison between the set limits and input data coming from sensors. The output from the `Compare` component is then used to trigger alerts that are generated and displayed on the mobile screen as pop-ups by the `Alert` component. Fig. 6(a) shows how the farmer can configure the sensor address and the data rate. Similarly, Fig. 6(b) shows how the farmer can select the data parameters to be filtered once sensor data is received. Lastly, Fig. 6(c) shows how the limit for the thresholds can be set.

V. Discussion

We believe that for farmers with less programming experience, purely visual LCDTs can be a good option for developing SAAs since they do not require text-based techniques. Furthermore, from our prototype application, the farmer needs to drag-and-drop, connect, and configure components on a visual interface. The configuration can be done to many sensor devices from the graphical interface. This can easily be done by non-expert programmers like farmers.

Programming distributed SAAs can be a complex undertaking. From our prototype application, distributed communication is abstracted into components that farmers can use to build SAAs. These abstractions can help in minimising errors that can arise when developing distributed SAAs. For example, with most LCDTs, farmers do no need to structure databases or queries to fetch data from databases. As such, query-related errors that can degrade the performance of SAAs are avoided.

To farmers, we believe that LCDTs can speed-up the development process for SAAs. The graphical drag-and-drop techniques that they provide can be a faster way to yield SAAs as compared to text-based approaches.

[9] https://github.com/svaarala/duktape
[10] https://www.arduino.cc/reference/en/libraries/websockets/
[11] https://arduinojson.org/

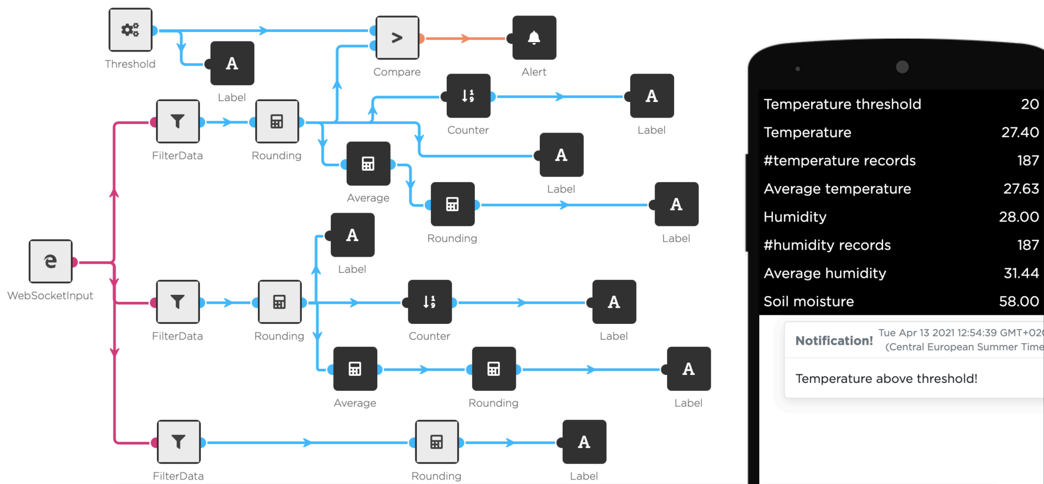
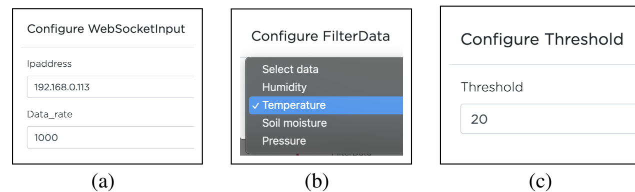


Fig. 5: SA application implemented using DisCoPar to monitor environmental conditions.



(a) (b) (c)

Fig. 6: (a) Configuring the sensor address and data rate, (b) Specifying data filters and (c) Setting the threshold.

VI. Conclusion

LCDTs can offer intuitive capabilities for developing SAAs by both experienced and non-experienced developers. In this paper, we categorise LCDTs and show their properties (such as reconfigurability, visual modelling etc.) that are important to farmers as non-expert programmers. LCDTs that combine these properties can be more beneficial to farmers when expressing and representing SAAs. As a demonstration, we show how to use DisCoPar to develop SAAs. From our prototype implementation, we believe that LCDTs can open programming to farmers. For future work, we aim to quantify the application development cost using LCDTs and revenue benefits to farmers.

Acknowledgment

This work is supported by the Legumes Centre for Food and Nutrition Security (LCEFoNS) programme which is funded by VLIR-UOS.

References

[1] M. O'Grady, D. Langton, and G. O'Hare, "Edge computing: A tractable model for smart agriculture?" *Artificial Intelligence in Agriculture*, vol. 3, pp. 42–51, 2019.

[2] S. J. Janssen, C. H. Porter, A. D. Moore, I. N. Athanasiadis, I. Foster, J. W. Jones, and J. M. Antle, "Towards a new generation of agricultural system data, models and knowledge products: Information and communication technology," *Agricultural Systems*, vol. 155, pp. 200–212, 2017.

[3] A. Walter, R. Finger, R. Huber, and N. Buchmann, "Opinion: Smart farming is key to developing sustainable agriculture," *Proceedings of the National Academy of Sciences*, vol. 114, no. 24, pp. 6148–6150, 2017.

[4] M. Michels, V. Bonke, and O. Musshoff, "Understanding the adoption of smartphone apps in crop protection," *Precision Agriculture*, vol. 21, no. 6, pp. 1209–1226, 2020.

[5] M. K. Khachouch, A. Korchi, Y. Lakhrissi, and A. Moumen, "Framework Choice Criteria for Mobile Application Development," in *2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)*, 2020, pp. 1–5.

[6] A. Sahay, D. Di Ruscio, and A. Pierantonio, "Understanding the Role of Model Transformation Compositions in Low-Code Development Platforms," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, ser. MODELS'20. New York, NY, USA: Association for Computing Machinery, 2020.

[7] R. Waszkowski, "Low-code platform for automating business processes in manufacturing," *IFAC-PapersOnLine*, vol. 52, no. 10, pp. 376–381, 2019, 13th IFAC Workshop on Intelligent Manufacturing Systems IMS 2019.

[8] R. Sanchis, Ó. García-Perales, F. Fraile, and R. Poler, "Low-Code as Enabler of Digital Transformation in Manufacturing Industry," *Applied Sciences*, vol. 10, no. 1, 2020.

[9] M. Henkel and J. Stirna, "Pondering on the Key Functionality of Model Driven Development Tools: The Case of Mendix," in *Perspectives in Business Informatics Research*, P. Forbrig and H. Günther, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 146–160.

[10] L. Michael and D. Field, "Mendix as a solution for present gaps in Computer Programming in Higher Education," in *Twenty-fourth Americas Conference on Information Systems*, 07 2018, pp. 1–5.

[11] A. N. Alonso, J. Abreu, D. Nunes, A. Vieira, L. Santos, T. Soares, and J. Pereira, "Towards a Polyglot Data Access Layer for a Low-Code Application Development Platform," 2020.

[12] M. Andrade, D. Aveiro, and D. Pinto, "DEMO based Dynamic Information System Modeller and Executer," in *10th International Conference on Knowledge Engineering and Ontology Development*, 01 2018, pp. 383–390.

[13] J. Zaman, K. Kambona, and W. De Meuter, "DISCOPAR: A visual reactive programming language for generating cloud-based participatory sensing platforms," ser. REBLS 2018. New York, NY, USA: Association for Computing Machinery, 2018, pp. 31–40.