An Extensible Software Platform for Cloud-based Decision Support and Automation in Precision Agriculture*

Li Tan[1,2†] Hongfei Hou[1] Qin Zhang[2]
[1] School of Electrical Engineering and Computer Science, Washington State University, Richland, WA 99354
[2] Center for Precision and Automated Agricultural Systems, Washington State University, Prosser, WA 99350

Abstract

Precision agriculture is a data-driven farming practice that uses intra- and inter-field information to optimize farming operations. The "brain" of precision agriculture is a decision support system (DSS) that acquires data from various sources, analyzes them, and recommends actions to farmers. Recently cloud computing has been used to improve the scalability and reliability of a DSS. Cloud-based DSSs present some major challenges for software design:(1) how can a cloud-based DSS process a diversified profile of intra- and/or inter-field data from various sources? *(2)* how can a cloud-based DSS accommodate and support the diversity of farming operations? *(3)* how can a cloud-based DSS automate the entire decision process and control field devices directly? *we proposed an extensible cloud-based software platform that integrated 3 novel components to address these questions: (1) a meta-model-based data acquisition and integration module that accepts data in different formats and semantics; (2) an adaptive software architecture supporting on-the-fly re-configuration of decision modules; and (3) software-defined control, a new software design paradigm we proposed for handling control diversity. It enables a DSS to control various field devices through unified software-defined interfaces. We implemented the platform in Agrilaxy, a cloud-based DSS, and deployed it on Amazon Web Services (AWS). An early version of Agrilaxy has been used in a USDA-sponsored project on canopy management for specialty crops.*

*This research was supported in part by the U.S. department of Agriculture grant 20101304802. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the view of the U.S. Department of Agriculture.

†Corresponding author. Email: litan@wsu.edu.

1 Introduction

Modern agriculture is increasingly becoming a data-driven operation. On the frontier of such trend is precision agriculture. Precision agriculture is a farming practice using technologies to measure and to respond to inter- and intra-field conditions. Precision agriculture plays a central role in emerging "smart farming" movement [4], which makes farms more "intelligent" using information, sensing, and other technologies. At the core of precision agriculture are Decision Support Systems (DSSs). A DSS works as the "brain" of a smart farm: it acquires inter- and intra-field measurement from different sources, analyzes the data, and recommends actions to farmers. A next-generation DSS is also expected to automate the entire decision process, and to control field devices directly with optimal decision. The success of precision agriculture, and hence "smart farming" is largely hinging on how *intelligent* a DSS is.

Our software platform is developed to address challenges in designing a cloud-based DSS. The scalability and reliability of cloud computing makes it particularly attractive for DSSs. To achieve economies of scale, a DSS is expected to serve a large number of farms, and their demand for decision support may vary drastically on and off a growing season. A key benefit of cloud computing is that it can pool computing resources and allocate them on demand. A cloud-based DSS can therefore scale up or down its resource usage on-the-fly. Compared with a desktop DSS, a cloud-based DSS also enjoys many other benefits. For example, software upgrade is carried out in the cloud, transparent to users. A cloud-based DSS also provides an online platform to connect farmers and developers. Nevertheless, designing a cloud-based DSS is a paradigm shift from designing a desktop DSS, and it presents several major design challenges.

From the perspective of system engineering, a modern farming operation is a complex dynamic system, with many

inputs and variables to consider (e.g. field condition and crop variety) and many decisions to make (e.g., irrigation and fertilization schedules). Because of so many variations, each farming operation has its unique character, for which a cloud-based DSS must be tailored. The diversity of data sources, operations, and field devices presents some major challenges for designing a cloud-based DSS:

(i). A DSS needs to work with different sources to acquire various intra- and inter-field measurement data. These sources may include field sensors (e.g. soil moisture sensor [10]), remote sensing data (e.g. Drone imaging [5]), and online data services (e.g. *AgWeathernet* [16]). Each data source may have its own data format and/or semantics. A research question is *how a DSS can process a diversified profile of intra- and/or inter-field data from various sources*.

(ii). Each farming operation has its own character, and hence it has a unique need for decision support. For example, cherry and apple growers have different sets of decisions to make. Even within cherry orchards, decisions may vary drastically due to crop variety and terrain condition etc. A research question is *how a DSS can accommodate the diversity of farming operations*. The design of a DSS needs to be adaptive to existing and future farming operations.

(iii). An ultimate goal of precision agriculture is to automate the entire decision process, from collecting data, to synthesizing decisions, to applying actions to field devices. Existing DSSs are focusing on the former two. A major research question is *how a DSS can automate the entire decision support process and control a variety of field devices directly and safely*.

To address these research questions, our software platform integrated three novel technological components.

1. Meta-model-based data acquisition and integration. In [12] we proposed a meta-model-based integration technique for a DSS with a client/server architecture. We extended our previous work in [12] to support cloud-based data acquisition and integration. We developed a RESTful interface for cloud-based data acquisition through a unified JSON-based interface.

2. An adaptive software architecture for decision modules. The architecture enables on-the-fly reconfiguration of decision modules. A user can customize decision logics through re-configuration mechanisms including dataflow re-routing and block parameterization.

3. Software-defined control (SDC). Inspired by the concept of software-defined network (cf. [7]), we proposed SDC as a new software design paradigm to handle the diversity of field devices. SDC provides a layered abstraction of devices through software-enabled adaptation. Each layer provides an abstract control interface featuring two-way communication. A control interface in SDC is defined by software, and communication via interface is carried out via events and variables. SDC enables a DSS to work with existing and future devices through a unified software-defined control interface.

The rest of the paper is organized as follows: Section 2 provides an overview of our platform; Section 3 discusses our meta-model-based data importation and integration technique; Section 4 introduces an adaptive software architecture for decision modules; Section 5 proposes software definition control, a novel software design paradigm to tame control diversity; Section 6 discusses our implementation of the platform and its deployment on Amazon Web Services (AWS); and finally, Section 7 concludes the paper.

2 Platform Overview

Figure 1 shows an overview of the platform. It has 3 major subsystems: (1) a data acquisition and integration subsystem, which acquires and integrates data from different sources. The subsystem provides a web interface to work with both active (i.e. sources posting data to a URL) and passive (i.e. sources accepting queries) data sources. In precision agriculture, each data source may have its own data format and semantics. To accommodate the diversity of data sources, the subsystem uses a meta-model-based approach to define and interpret data coming from a source. Meta-model-based data integration will be discussed in Section 3; (2) an adaptive decision subsystem, which synthesizes device-independent decisions using acquired data. The decision subsystem comprises decision modules. It deploys an adaptive software architecture (Section 4) that enables on-the-fly re-configuration of decision modules through dataflow re-routing and block parameterization; and (3) a device control subsystem. The subsystem utilized software-defined control (SDC, Section 5), a novel software design paradigm proposed in [11]. SDC translates a device-independent decision to commands for field devices through multi-stage software-enabled adaptation.

3 Meta-Model-Based Data Acquisition and Integration

A major challenge for cloud-based DSSs is how to handle the diversity of data sources. To capture intra- and inter-field variabilities, precision agriculture uses data

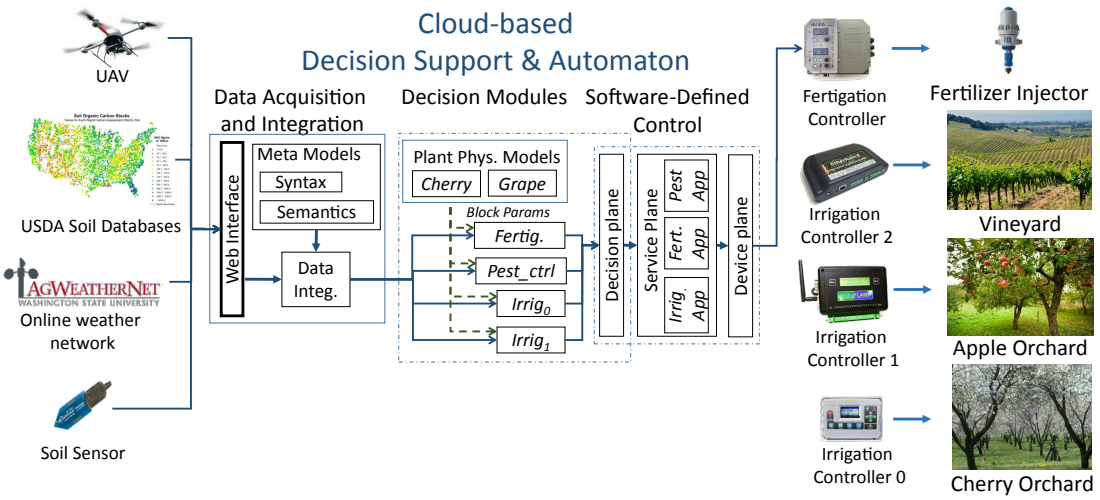

Figure 1. Extensible software platform for cloud-based decision and automation

from a variety of sources, for example, sensors measuring soil moisture [1] and organic matters [6], UAVs surveying biomass [5], mobile infrared sensor platforms collecting Photosynthetically Active Radiation (PAR) data [8], and online databases storing weather [16] and soil composition data [14]. These data come in different formats and semantics. Moving a DSS to the Cloud makes the challenge even greater: a cloud-based DSS is expected to serve many farms, each of which may have different data sources.

To handle data diversity, we developed a meta-model-based data acquisition and integration technique. The technique enables a flexible data acquisition through custom-defined data formats and semantics. Users define data formats and semantics for a data source using a two-tier data modeling system. An input data set is interpreted based on a custom-defined data model. The data set is then translated into a unified internal format acceptable to decision modules. By separating data diversity from the rest of the DSS, the technique makes it possible to develop a flexible and extensible analytical tool accommodating a variety of data sources.

3.1 Meta Data-Model-Based Data Integration

The data integration module uses a meta-model-based technique for defining and interpreting data from different source. The technique is based on a layered abstraction of data syntax and semantics. A data model in XML is used to specify the format of an input data set. A meta-model in the XML schema (XSD) defines the scope of valid XML data models. And finally, a data importer specifies operational semantics for the XML meta data model. Operational semantics define how data associated with each element of a meta model shall be interpreted and translated into a unified internal format.

To illustrate the workflow, consider an example in which the DSS platform needs to work with a new type of soil sensors. To complicate the matter further, soil sensors' data loggers can be configured to produce data in custom formats, e.g., data columns in different orders. Traditionally a DDS's data importation algorithm needs to be reprogrammed to accept the new type of sensors. Using our technique, a developer only needs to provide a new data importer along with a XSD file specifying data models supported by the importer. Figure 2 illustrates the XSD model. The XSD model specifies acceptable XML data models. For example, a valid XML data model must define at least one "DATAFILE" element, and each "DATAFILE" element must contain a "UID" element. Note that the "Data_time" segment has one of two types: a "Date_seg" type comprising 6 elements ("year" $\cdots$ "second"), and a "Date_single" type. Figure 3 shows a XML data model conforming to the XSD meta data model in Figure 2. The XML data model specifies the exact structure of a data set generated by a soil sensor. For instance, the XML file in Figure 3 specifies that each data set contains just one data file ("data_par.csv"), and inside the data file, 5 data columns are organized in 3 logical segments: "UID", "Measurement", and "Data_time". In this example, a data file following the XML data model in Figure 3 is a CSV file (data_par.csv) comprising rows of data, each of which has 5 columns ("UID", "Soil_moisture", "Soil_temperature", "Soil_nitrogen", and "Date_single").

Operational semantics for a data set are operations that interpret the data set and translate it into an uniformed internal format. For example, in an implementation of our platform, the internal format for date/time is a multiple-

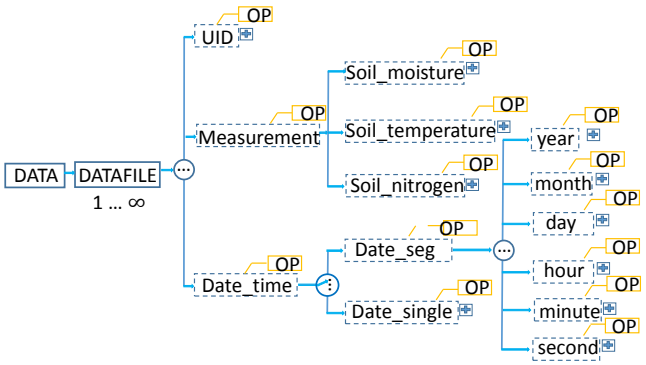

Figure 2. A XML schema file defining a meta-model for a type of soil sensors.

```
<?xml version="1.0" encoding="UTF-8"?>
<DATA xmlns="Agrilaxy"
  xsi:schemaLocation="Agrilaxy_format.xsd">
    <DATAFILE filename="data_par.csv">
      <UID type="string"/>
      <Measurement>
        <Soil_moisture unit="percentage" type="float"/>
        <Soil_temperature unit="f" type="float"/>
        <Soil_nitrogen unit="percentage" type="float"/>
      </Measurement>
      <Date_time format="YYYY/MM/DD_HH:MM:SS" type=
          "datetime"/>
    </DATAFILE>
</DATA>
```

Figure 3. A XML file defining the data model for a soil sensor. The XML file conforms to the meta data-model in Figure 2.

field record, in which date and time are kept separately. The operational semantics for "data_single" is hence an algorithm that translates a string format of "YYYY/MM/DD_HH:MM:SS" into a record with separate fields for date and time. In our approach, operational semantics associated with each element of the XSD meta-model (i.e. "OP" in Figure 2) is defined in the data importer.

As a summary, to support a new type of data sources, the DSS platform can be extended with a new data importer and its meta-model in XSD. A data source of the new type provides a XML model conforming to the XSD meta-model. The XML model is used by the data importer to interpret the data format of the new data source.

3.2 Cloud-based Data Acquisition

Before a cloud-based DSS can process data, it must acquire them from a data source. Data Acquisition Module (DAM) in our platform provides a web portal for communicating with data source. The web portal is a collection of RESTful (Representational State Transfer) APIs [2]. These RESTful APIs support data transfer through standard HTTP requests (POST, GET, DELETE etc). The APIs support data acquisition in batch mode and in real time. In the batch mode, a data source may upload a data set as a file, along with a XML model specifying its format. In the real-time mode, a data source may request a session by posting its XML data model to the web portal. DAM communicates with the Data Integration Module (DIM) to find the right data importer accepting the XML data model. Once the XML model is validated, the web portal returns a session ID to the data source. The data source then posts data in real time, using the session ID as its identifier. The data will be interpreted by the data importer according to the XML data model provided by the data source.

The web portal works with both active and passive data sources. An active data source, such as Internet-connected soil sensors, can initiate communicate and post data to the web portal. An passive data source, such as USDA's web-based soil databases [14], does not initiate communication. Instead, it accepts data inquiries. For each passive data source, DAM includes a proxy for the data source. The proxy queries the passive data source according to a predefined schedule. The data from the data source can then be posted to the web portal either in batch or in real time.

4 Adaptive Software Architecture for Decision Modules

Agriculture operations are complex dynamic systems with many variables, such as field conditions and crop variations. Each farming operation has its own character. This operational diversity presents even a greater challenge to a cloud-based DSS than to a traditional site-specific DSS. A site-specific DSS, running on a desktop or a dedicated server, is configured for a specific operation. In comparison, a cloud-based DSS serves many farms concurrently, each of which has its own need for decision support. A key challenge in designing a cloud-based DSS is *how it can accommodate the diversity of farming operations*.

To handle this operational diversity, we developed an adaptive software architecture for decision modules. The architecture enables individual farmers to customize decision logic on-the-fly through re-configuration mechanisms. The example in Figure 4 illustrates how an irrigation decision module may be adapted for a specific farming operation, using re-configuration mechanisms including dataflow

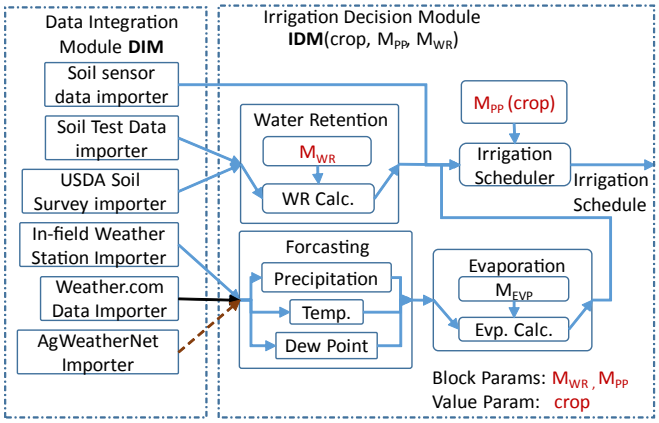

Figure 4. An adaptive irrigation decision module (IDM), where M_{WR} and M_{PP} are a water retention model and a plant physiology model respectively, and $crop$ specifies the type of crop.

re-routing and block paramterization.

Dataflow re-routing The adaptive architecture follows a data-flow-driven design. Blocks are connected through dataflows, represented as directed edges in Figure 4. The data-flow-driven design gives users an intuitive way to view and configure a decision logic: a farmer may change the decision logic by re-routing its data flow. In the example shown in Figure 4, the farmer may choose to replace *weather.com* with *AgWeatherNet* as the data source for weather information. To do so, the farmer simply reroutes the source of the data flow from the data importer for *weather.com* (shown as a dark line), to the one for *AgWeatherNet* (shown as a brown dash line). *AgWeathernet* is a network of weather stations in the Pacific Northwest region [16]. By switching to *AgWeathernet*, the same irrigation decision module (IDM) may produce an irrigation schedule based on real-time on-the-ground weather data. All these changes through dataflow re-routing can be done on-the-fly without requiring additional programming.

A constraint for dataflow re-routing is that new and old data sources/destinations must agree on the same data format. In the example in Figure 4, the constraint is satisfied using the meta-model-based data integration technique in Section 3.1. Data importers translate input data from different data sources into a uniform internal format, making it easier to compliant to the requirement for data re-routing.

Parameterization by values Another re-configuration mechanism we used is *parameterization by values*, in which a block has one or more parameters. These parameters are bounded with values as part of run-time setting. A parameter value may be supplied by a user via the run-time environment. The value may be passed down block hierarchy: a block may bind the value with a parameter of its child block. In the example in Figure 4, the irrigation decision module (IDM) takes $crop$ as its parameter, where $crop$ specifies the type of the crop for an agricultural operation. IDM then passes $crop$ to the plant physiology model M_{PP}.

Parameterization by blocks Our adaptive architecture also supports *parameterization by blocks*. in which the domain of a parameter is a set of blocks sharing the same interface. The technique is developed to promote the reuse of decision logic. Often time decisions for similar farming operations share many common features, and hence underlying decision logics share common components. Fitting a decision logic to a similar operation may require changes to only part of the logic. Parameterization by blocks allows modification of a sub-logic, while preserving the overall structure of a decision module. Parameter blocks may be passed down a module hierarchy. For instance, $C(B)$ is denoted for a parameter block C with B as its parameter. During an execution, B is bound with a real block B'. The resulting block $C(B')$ is the same as C, except that parameter B is bounded with block B'.

As an example, consider the IDM in Figure 4. IDM has two block parameters: M_{WR}, a parameter block for a water retention model, and M_{PP}, a parameter block for a plant physiology model. These block parameters are used as place holders in the construction of IDM. At run time, M_{PP} is bound with a specific plant physiology model, such as the *Water-GreenLab* [9], and M_{WR} with a specific water retention model, such as the *Van Genuchten* model [15].

5 Software Defined Control

Today's DSSs are designed to recommend decisions to farmers. An ultimate goal of a next-generation DSS is to automate the entire decision process. This new generation of DSS, which we refer to as Decision Support and Automation System (DSAS), will be capable of acquiring data from field and other sources, analyzing the data, and controlling field devices directly with an optimal decision. In agricultural practice, farmers deploy a variety of devices in their field applications, for example, irrigation and pest control. These field devices generally come with their own control interfaces. To complicate the matter further, a cloud-based DSS is expected to serve many farms concurrently, each of which may have its own set of field devices. A question in cloud-based control automation is *how to control a variety of field devices directly and safely from the cloud*.

To address this research question, we proposed *software-defined control* in [11], a new software design paradigm that tames control diversity through software-enabled adaptation. In a nutshell, software-defined control makes devices

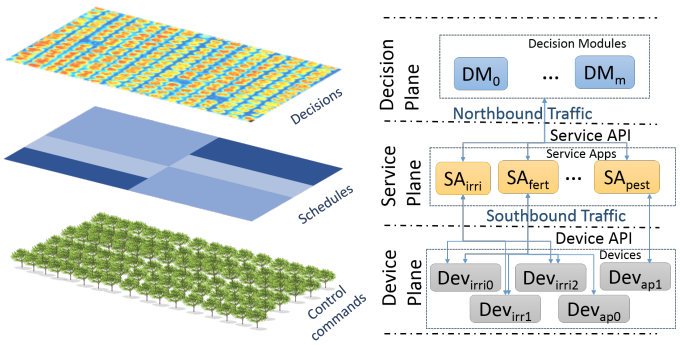

Figure 5. Software defined control: an illustrative example

interoperable by wrapping them with a unified control interface defined by software. Software-defined control gives a device manufacturer a flexible and non-intrusive way to work with different DSSs. To work with a new DSS, a device manufacturer implements a software-defined control interface for a device. The software-defined interface encapsulates the device's actual control interface, and it provides an abstract and unified interface of the device, which the DSS can access uniformly.

The objective of software-defined control is to translate device-independent decisions synthesized by a DSS, to device-specific control signals. The translation is carried out through a 3-step software-enabled adaptation.

1. A decision plane synthesizing device-independent decisions. These decisions specifies field actions at an abstract level. For example, a device-independent irrigation decision specifies how water is distributed in a field, regardless of the layout of the irrigation system in the field;

2. A service plane translating a decision to commands for a device. Commands are delivered to a device through the device's own control interface;

3. A device plane executing commands it received and applying actions (e.g. switching on/off an irrigation valve) to a field.

Figure 5 shows the 3-layer architecture implementing software-defined control, using a multi-zone irrigation system with fertigation attachment as an example. Fertigation is a practice of applying fertilizers by injecting them to an irrigation system [3]. In this example, the underlying irrigation system has 3 irrigation controllers (Dev_{irri0}, Dev_{irri1}, and Dev_{irri2}) and 2 fertigation controllers (Dev_{ap0} and Dev_{ap1}). Each irrigation controller is capable of controlling multiple zones through irrigation control valves. Each fertigation controller controls an injector attached to the main water supply line, and each injector can be supplied with a specific fertilizer. For simplicity, we did not show control valves and injectors in Figure 5.

The decision plane is embedded in a decision module. In our example, it is part of an irrigation/fertigation decision module. A decision plane translates a decision made by the decision module into a unified and abstract format. In our example, the decision produced by the decision plane is a geological distribution of water volume, as well as fertilizer, as illustrated by the color map in Figure 5. The device-independent decision does not take into the account irrigation system layout such as zoning, nor device-dependent information such as maximal flow volume.

The service plane translates the abstract decision made by the decision plane, to commands accepted by the device plane. Here a *service* refers to a set of logically related functions. The service plane comprises a set of service apps, and each app implements the functions for its service. In our example, the service plane includes service apps for irrigation (SA_{irri}), fertigation (SA_{fert}), and pesticide application (SA_{pest}). Each service app is responsible for communicating with the device plane. Service apps implement logic functions on top of physical devices, and they may share same physical devices among each other. In our example, SA_{irri} and SA_{fert} both communicate with irrigation controllers (Dev_{irri0}, Dev_{irri1}, and Dev_{irri2}). In addition SA_{fert} also controls Dev_{ap0}. Upon receiving a fertigation decision, SA_{fert} computes a control schedule for underlying devices, and sends the schedule to the controllers in the device plane.

The device plane wraps actual field devices with an API accessible by a service plane. For an off-the-shelf web-enabled device such as an Internet-connected irrigation controller, the device plane simply translates function calls by the service plane, to HTTP requests to the irrigation controller. For a stand-alone irrigation controller, the device plane also needs to implement a web interface, for example, an Internet gateway for the irrigation controller.

It shall be noted that all 3 layers and their interfaces support two-way communication. Borrowing a similar notation in *software-defined network* [7], we denote *southbound traffic* for data and control flowing from a higher layer to a lower one, and *northbound traffic* for data and control flowing in the reverse direction. Previously our discussion focused on southbound traffic. Northbound traffic carries feedback provided by a lower layer. Northbound traffic is crucial for run-time monitoring and adaptation, enabling a higher layer to monitor lower layers and adjust its operation accordingly.

Software-defined control interface A feature of software-defined control is that its control interface is defined purely

by means of software. In general, the decision plane is part of a decision module, and the service plane is part of a device module. The control interface in software-defined control refers to the interface between the decision plane and the service plane. Formally, the control interface is defined as a tuple $\langle V_{in}, V_{out}, F_{in}, C_{out} \rangle$, where V_{in} and V_{out} are the set of input and output variables, respectively, F_{in} is a set of input functions, and C_{out} is a set of call-back registration functions. Input variables and functions are used for implementing southbound traffic, i.e., data and control flowing from the decision plane to the service plane. Formally a function $f \in F_{in}$ represents a type of events: to pass an event $e(p_0, \cdots, p_n)$ to the service plane, the decision plane call the function f_e with parameters $p_0 \cdots p_n$.

Output variables and call-back registration functions are used to implement northbound traffic, i.e., the data and control flow from the service plane to the decision plane. In order for the service plane to pass an event $e'(p_0, \cdots, p_m)$ to the decision plane, the latter has to register a call-back function $f_{e'}$ with the service plane. To do so, the decision plane calls the call-back registration function $c_{e'}$ for event e' with the parameter $f_{e'}$. When event $e'(p_0, \cdots, p_m)$ occurs, the service plane calls $f'_e(p_0, \cdots, p_m)$ to deliver the event to the decision plane.

6 Implementation

To test the feasiblity of our proposed platform, we are using it to guide the implementation of Agrilaxy, a cloud-based decision support and automation system for precision agriculture. Agrilaxy provides farmers, developers, and developers an online platform for developing, exchanging, and applying web apps for precision agriculture. These web apps customize and extend Agrilaxy for a precision agricultural application. Farmers may select and configure web apps to fit to their own needs for decision support and automation, and a researcher/developer may develop a web app that extends the functions of Agrilaxy. Examples of web apps include data importers (Section 3), decision apps (Section 4), and software-defined service apps (Section 5).

Agrilaxy is implemented using Ruby on Rails. Ruby is a dynamic typing language popular among in the web development community. Part of its popularity is due to the Rails framework. Rails is a collection of Ruby libraries, known as GEMs, that support web development using Model-View-Controller (MVC) design pattern. Communication between Agrilaxy and external entities, such as data sources and field devices, are carried out through a set of RESTful APIs implemented using Rails. The RESTful APIs support data exchange in Javascript Object Notations (JSON) and XML, two open-standard data exchange formats. The current version of Agrilaxy comprises approximately 8056 lines of Ruby code.

Agrilaxy has been tested on Amazon Web Services (AWS). We used PostgresSQL as our underlying databases. One of important design issues for a cloud-based DSS is data privacy. Farmers often see field data as the extension of their physical properties. To improve data privacy, we used a multi-tenancy database design similar to the one we proposed in [13]. For each farmer, his/her field data are partitioned and stored in separate sets of data tables. When a farmer is authenticated to access Agrilaxy, he/she is given only the permission to access data tables storing his/her own farming data. An early version of Agrilaxy has been used to process field data in a USDA sponsored project for canopy management.

7 Conclusion

Precision agriculture plays a central role in transforming agriculture into a data-driven industry. At the core of precision agriculture is a decision support system (DSS). Recently Cloud computing is being used to improve the scalability and reliability of a DSS. Nevertheless, cloud-based DSSs present some major challenges in software design, particularly when dealing with the diversity of data sources, farming operations, and field devices. We proposed an extensible software platform to address these challenges. The platform comprises three major components: (1) a meta-model-based data acquisition and integration technique enabling a DSS to work with a variety of data sources; (2) an adaptive software architecture for decision modules supporting on-the-fly customization of a DSS for a specific operation; and (3) a novel software defined control technique enabling a cloud-based DSS to work with a variety of field devices. We are implementing the platform in Agrilaxy, a cloud-based DSS. An early version of Agrilaxy has been deployed on Amazon Web Services, and it has been used to process field data in a USDA-sponsored project on canopy management.

It is worth noting that the technologies developed in this work are not limited to precision agriculture. For example, software-defined control can be used to handle control diversity for cloud-based smart Internet-of-Things (IoT) systems. It enables software-defined integration of IoT, allowing these devices to have a unified control interface defined entirely by software. The meta-model-based integration technique may also be applied to other cloud-based data-driven applications working with different data sources.

References

[1] R. Cardell-Oliver, M. Kranz, K. Smettem, and K. Mayer. A Reactive Soil Moisture Sensor Network: Design and Field Evaluation. *International Journal of Distributed Sensor Networks*, 1(2):149–162, feb 2005.

[2] R. T. Fielding and R. N. Taylor. Principled design of the modern Web architecture. In *Proceedings of the 22nd international conference on Software engineering - ICSE '00*, pages 407–416, New York, New York, USA, jun 2000. ACM Press.

[3] A. Gärdenäs, J. Hopmans, B. Hanson, and J. Šimnek. Two-dimensional modeling of nitrate leaching for various fertigation scenarios under micro-irrigation. *Agricultural Water Management*, 74(3):219–242, jun 2005.

[4] F. Guerrini. The Future of Agriculture? Smart Farming. http://www.forbes.com/sites/federicoguerrini/2015/02/18/the-future-of-agriculture-smart-farming/, feb 2015.

[5] S. Herwitz, L. Johnson, S. Dunagan, R. Higgins, D. Sullivan, J. Zheng, B. Lobitz, J. Leung, B. Gallmeyer, M. Aoyagi, R. Slye, and J. Brass. Imaging from an unmanned aerial vehicle: agricultural surveillance and decision support. *Computers and Electronics in Agriculture*, 44(1):49–61, jul 2004.

[6] J. Hummel, K. Sudduth, and S. Hollinger. Soil moisture and organic matter prediction of surface and subsurface soils using an NIR soil sensor. *Computers and Electronics in Agriculture*, 32(2):149–165, aug 2001.

[7] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, jan 2015.

[8] B. Lampinen, V. Udompetaikul, G. T. Browne, S. G. Metcalf, W. L. Stewart, L. Contador, C. Negrón, and S. K. Upadhyaya. A mobile platform for measuring canopy photosynthetically active radiation interception in orchard systems. *HortTechnology*, 22(April):237–244, 2012.

[9] B. Pallas, C. Loi, A. Christophe, P. H. Cournède, and J. Lecoeur. Comparison of three approaches to model grapevine organogenesis in conditions of fluctuating temperature, solar radiation and soil water content. *Annals of botany*, 107(5):729–45, apr 2011.

[10] S. Peets, A. M. Mouazen, K. Blackburn, B. Kuang, and J. Wiebensohn. Methods and procedures for automatic collection and management of data acquired from on-the-go sensors with application to on-the-go soil sensors. *Computers and Electronics in Agriculture*, 81:104–112, feb 2012.

[11] L. Tan. Cloud-based Decision Support and Automation for Precision Agriculture in Orchards. In *5th IFAC conference on Sensing, Control and Automation Technologies for Agriculture*, Seattle, WA, 2016.

[12] L. Tan, R. Haley, and R. Wortman. An Extensible and Integrated Software Architecture for Data Analysis and Visualization in Precision Agriculture. In *the proceedings of IEEE Int'l Conf. on Info. Reuse and Integration*, 2009.

[13] L. Tan, R. Haley, and R. Wortman. Cloud-Based Harvest Management System for Specialty Crops. In *Proc. of IEEE 4th Symp. on Network Cloud Computing and Apps.*, 2015.

[14] USDA Natural Resources Conservation Services. Web Soil Survey. http://websoilsurvey.sc.egov.usda.gov/, 2015.

[15] M. T. van Genuchten. A closed-form equation for predicting the hydraulic conductivity of unsaturated soils. *Soil Science Society of America Journal*, 44(5), 1980.

[16] Washington State University. AgWeatherNet. http://weather.wsu.edu/awn.php, 2015.