An Extensible and Integrated Software Architecture for Data Analysis and Visualization in Precision Agriculture[*]

Li Tan[1,2†] Ronald Haley[1] Riley Wortman[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

Recent technology advances in information technology and other engineering fields provide new opportunities for research and practices in precision agriculture. Using these technologies, field operators can collect voluminous data from a heterogeneous network of devices that provides real-time and multiple-factor measurement of field conditions with much finer granularity. A major challenge in precision agriculture today is how to analyze these data efficiently and use them effectively to improve farming decisions. We propose an extensible and integrated software architecture for data analysis and visualization in precision agriculture, with three distinctive features: (a) a meta-data-model-based data importation component capable of importing data in various formats from a variety of devices in different settings; (b) a data-flow-driven data processing subsystem in which a user can define his/her own data processing workflows and add custom-defined data processing operators for a specific application; (c) an overall architecture design following a client-server model that supports a variety of client devices, including mobile devices such as the Apple iPad. We implemented the software architecture in an open-source decision support tool for precision agriculture. The tool has been successfully used in a USDA-sponsored project on canopy management for specialty crops.

1 Introduction

With a growing population on the planet and increased living standards worldwide, the demand for agricultural products rises quickly. Researchers and farmers around the globe are exploring and applying new technologies to agricultural operations, and an important advance in agricultural technologies is the emergence of precision agriculture [**?**, 1]. Using advanced technologies in information technology and other engineering fields, precision agriculture measures and analyzes the conditions of a field and its environment, and uses this information to optimize field operations. Precision agriculture is a data-intensive operation: in a typical setting of precision agriculture, field operators collect a large amount of data from a heterogeneous network of devices, for instance, infrared sensor arrays, Light Detection and Ranging (LIDAR) devices, soil moisture sensors etc. The success of a precision agricultural operation largely depends on its ability to analyze these data efficiently and apply the results effectively to decision processes.

By its operational characteristics, precision agriculture presents unique challenges as well as research opportunities for Computer Scientists, especially in the areas of data analysis, visualization, and decision support [9]. For instance, to process data in different formats from an array of devices, field operators need an analysis tool that can import and integrate heterogeneous data sets [6, 7]. In addition, precision agriculture operations vary widely, depending on products, fields, and many other factors. Operators need an analysis tool that enables them to define their own workflow of data processes. Last but not least, due to the increasing scale of agricultural operations, the tool has to be scalable and capable of supporting a variety of devices, including mobile devices, for on-the-fly access and in-field decision making.

To address these needs, we need a software solution with an architecture design that is extensible, scalable, and yet flexible to accommodate a variety of agricultural operations and client devices. To meet these requirements, we propose an software architecture with the following features:

1. A data importation and integration component that is capable of processing input data sets in different for-

[*]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.

mats and from multiple sources. The component uses a meta-data-model in XML schema to specific acceptable formats and semantics of data sets. A user can define his/her own data model in XML for input data sets. The meta-data-model-based approach enables the data importation and integration module to process a data set with a custom-defined data model, as long as the data model conforms to the meta-data-model defined by the component.

2. An extensible data-flow-driven data analysis subsystem. A user may define his/her own workflow of data processing for a specific farming operation. The workflow is defined as a data-flow model using built-in or custom-defined data processing operators. The user can implement a specific data processing algorithm as a custom-defined operator.

3. The overall architecture design follows a client-server model that supports a variety of client devices, including mobile devices such as Apple iPad. The architecture design follows the design principle of "separation of concerns": data intensive tasks are processed by servers, and visualization and human-computer interaction are handled by client devices. This design feature improves the scalability and extensibility of our software architecture, as data intensive tasks can be distributed among a network of servers, which may eventually migrate to a cloud-based computing platform (cf. [10]). A clearly defined server interface based on socket communication enables the architecture to support an array of Internet-connected devices, including mobile clients.

We implemented a prototype of the software architecture in an open-source data analysis and decision support tool. The tool has been successfully used in a precision agriculture project sponsored by the US department of Agriculture (USDA) under its specialty crop initiative. The project is to develop technology capability for managing canopy structures for specialty crops (fruits and nuts). Our tool has been used for importing, integrating, analyzing, and visualizing data in different formats and from different sources, and assisting decision processes.

The rest of the paper is organized as follows: Section 2 discusses data importation and integration; Section 3 describes the data-flow-driven data processing subsystem. It also describes data visualization operators used by the subsystem; Section 4 discusses the overall client-server architecture design; Section 5 discusses an implementation of the software architecture; and finally Section 6 concludes the paper with a discussion on future research directions.

2 Data Importation and Integration

Precision agriculture collects data from a heterogeneous network of devices. The formats and semantics of these data are defined by a range of factors, including sensor types, configurations of data loggers etc. A challenge in precision agriculture is how to import and integrate heterogeneous data so that the rest of data processing components may use them.

For example, we implemented the proposed software architecture as an open-source tool and use it in a USDA-sponsored precision agriculture project. The tool needs to process a variety of data collected from different sources, including PAR data collected from an infrared sensor array and GPS position of the array. The sensor array is mounted on a motorized platform *Mule*. Depending on the physical configuration of *Mule* and the setting of a data logger, the format of the data and its setting constantly change. We will use this as an example to show how our solution can support the importation and integration of data with changing formats and settings.

Our software architecture provides a two-fold approach for importing and integrating heterogeneous data sets:

1. The software architecture separates the concern of data importation and integration from the rest of data processing. The data importation component is essentially a collection of data importation operators. The variants of data formats and semantics accepted by an operator is defined by a meta-data-model in XML schema.

2. Each data set comes with a custom-defined data model in XML. The data model defines the format and the semantics of the data set. If the data model conforms to a meta-data-model defined by a data importation operator, the data set can be imported by the operator, which will integrate the data set to an uniformed internal representation accessible to the rest of data processing components.

For instance, our implementation of the proposed software incorporates a data importation operator. The format and semantics of the data sets acceptable to the operator is defined by a meta-data-model. Figure 1 shows an overview structure of the meta-data-model in XML schema. For clarity and brevity, Figure 1 only displays elements that are relevant to the discussion in this section.

The data importation operator reads in data sets collected by mobilized sensor arrays, and converts them to an uniform internal presentation for the rest of the data processing. An input data set is a collection of files with comma-separated values, recorded by a data logger. Each input data set is accompanied by a data model in XML that defines a data file's format, for example, names and column positions, as well

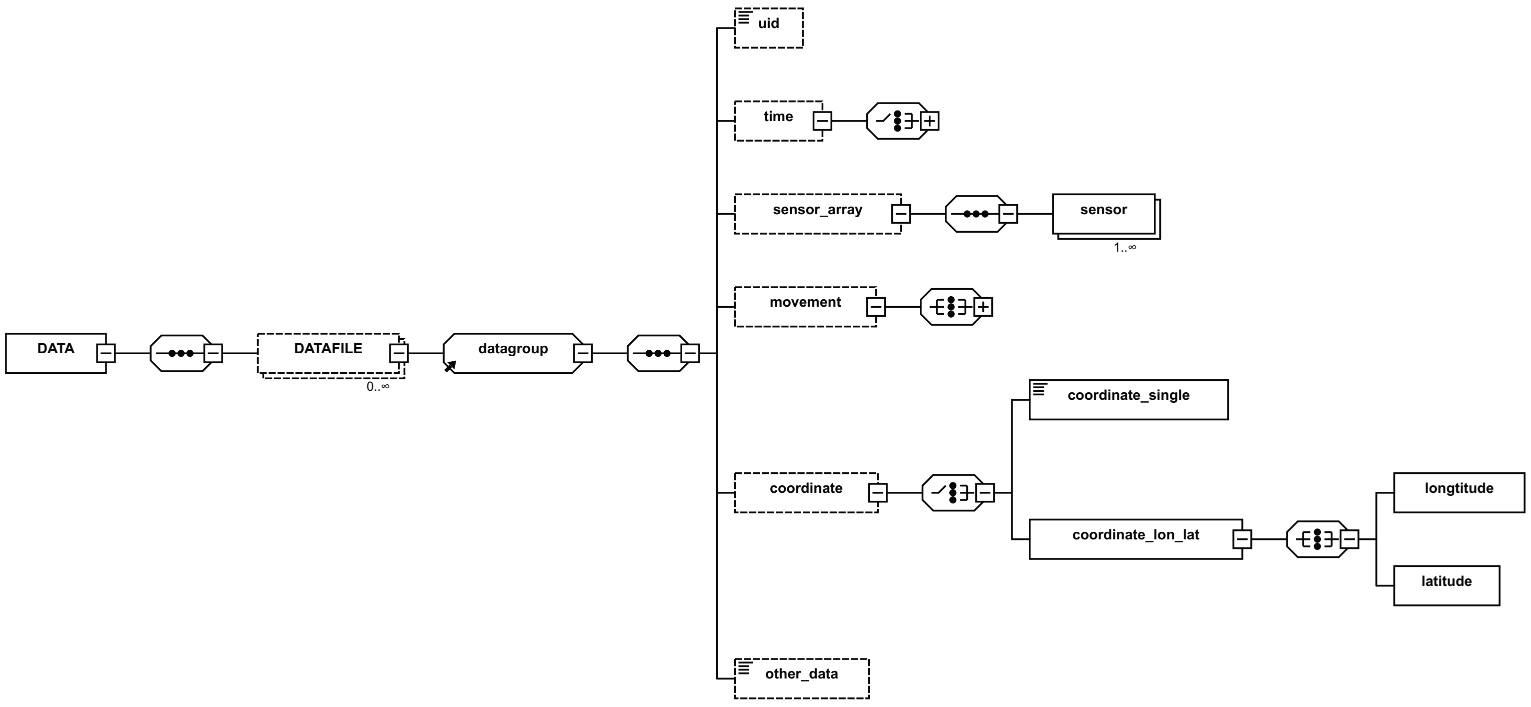

Figure 1. A meta-data-model, defined in XML schema, for data importation and integration.

as semantics, for example, measurement units and sources. In our software architecture, a data set must be accompanied by a data model. In general, one can reuse a data model for different data sets as long as the experimental setting (e.g. data loggers, devices, etc) remains unchanged. The data model of a data set must conform to the meta-data-model of the data importation operator in order for the operator to accept the data set.

Figure 2 shows a data model in XML that is used in our case study. For brevity, Figure 2 shows the content of the XML file relevant to our discussion. Figure 2 illustrates the following characteristics of the data model:

1. The data model has a tree structure with the root `DATA`, and the immediate child(ren) of the root define the names of input data files. A data set is a collection of these input data files.

2. Leaves of the tree structure are instances of (subclasses of) the *element* type in the XML schema. There is a one-to-one relation between these leaves and the columns of data files in the data set. The *in-order* traverse of leaves of a subtree with the root `DATAFILE` lists the columns of that data file referred by the `DATAFILE` node. For instance, the data model in Figure 2 indicates that a data set contains two data files `data_par.csv` and `data_gps.csv`. The order of columns in `data_par.csv` is as follows: `uid`, a list of `sensors`, `date`, and `time`. Internal nodes serve as markers of logical groups. For instance, the logical group `date_time` comprises `date` and `time`.

3. Attributes of an *element* instance define the additional semantics information of the instance. For example, for each sensor, `X` and `Y` specify its X- and Y- offsets to the center of the sensor array. Since the data set already includes GPS measurement of the center of the sensor array. The offsets may be used for computing the location of an individual sensor.

Note that in our case study every data file contains a column `uid` for unique identification numbers. `uid` serves as a reference key to rows of data files. Rows in different data files but sharing the same `uid` may be *joint*ed to provide multi-factor values for the same measurement point.

Once a data set is imported according to its format and semantics defined by its accompanying data model, the imported data is processed and integrated in an uniform internal representation. In our architecture, the internal representation of data is a network of structured data objects. The basic building block of this network are *primitive* data objects. Each primitive data object represents a measurement point, completed with the tempo-spatial information. For example, in our case study, a primitive data object represents the data collected by a sensor at a specific time and location. It consists of time, location, and sensor reading.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<DATA xmlns="AgriD"
  xsi:schemaLocation="AgriD_format.xsd">
  <DATAFILE filename="data_par.csv">
    <uid/>
    <sensor_array>
      <sensor id="1" X="mm" Y="mm" device="PAR" />
                          ...
      <sensor id="8" X="mm" Y="mm" device="PAR" />
    </sensor_array>
    <time>
      <date_time>
        <full_date format="YYYY/MM/DD" />
        <full_time format="HH:MM:SS.SS" />
      </date_time>
    </time>
    <other_data />
  </DATAFILE>
  <DATAFILE filename="data_gps.csv">
    <uid/>
    <movement>
      ...
    </movement>
    <coordinate>
      <coordinate_lon_lat>
        <longtitude unit="degree"/>
        <latitude   unit="degree"/>
      </coordinate_lon_lat>
    </coordinate>
  </DATAFILE>
</DATA>
```

Figure 2. A data definition file in XML, conforming to the meta-data-model in Figure 1.

The meta-data-model-based data importation subsystem is a important link in our software architecture. It takes heterogeneous data collected from different sensor sources and in various format, and converts them to an uniform internal representation of data so the rest of the tool can be built independently. The data modeling and integration capability provided by the data importation subsystem enables us to build a flexible and extensible analytical tool that can accommodate a variety of data collecting platforms.

3 Data-Flow-Driven Design for Data Processing and Visualization

The main purpose of our software architecture is to provide design guidelines for analytical tools that process and visualize field-collected data. Our software architecture incorporates a data-flow-driven design for the core data processing and visualization subsystem. The data-flow-driven design consists of data-flow design models defining workflows of data processing, and a run-time environment interpreting and executing data-flow models. A data-flow design model provides a high-level abstract of data processing. A user can easily change the workflow of data processing for a specific field operation by reconfiguring the design model. Furthermore, in our extensible data-flow-driven design a user can define his/her own data processing operators to data-flow design models. The data-flow-driven design includes commonly-used functions as built-in data operators, including operators for data importation and integration operators (Section 2), and visualization operators (Section 3.2).

3.1 Data-Flow Design Models

A data-flow design model describes the process of how data is imported, processed, and visualized. The data processing and visualization subsystem includes a run-time environment for interpreting and executing data-flow design models.

Building blocks of a data-flow design model are *operator*s. An operators is a basic data processing unit that takes input data from its interface, processes the data, and outputs its result. The interface of an operator consists of input and output ports. Each port is defined with its data type, which is the type of data transmitted through the port. Operators are linked by *data link*s. A data link connects an output port to an input port. A constraint is that the data types of two linked input and output ports must match. By re-routing data links, a user can re-configure a data-flow design model for a different workflow with the same set of operators.

Figure 3 shows a data-flow design model used to define a workflow of data processing in our case study. It starts with custom-defined data importation operators `Read_WS.csv` and `Read_PAR.csv`. Both operators support meta-data-model-based data importation and integration described in Section 2. Imported data are processed and finally visualized using data visualization operators (Section 3.2).

3.2 Data Visualization

Due to the importance of data visualization in precision agriculture, our data-flow-driven design includes built-in visualizations operators for addressing common needs for 2-D and 3-D data visualization in precision agriculture.

2-D data visualization displays a two-dimensional data set on a Cartesian plane. The function of 2-D data visualization is collectively implemented by a set of built-in operators including 2-D data visualization operator. The 2-D visualization operator visualizes a two-dimensional grid data as a raster image in various formats. A user may define color map used in visualization. In addition, since we adopt an open software architecture design, operators are capable of communicating external applications. The 2-D visualization operator can invoke the Google Earth application

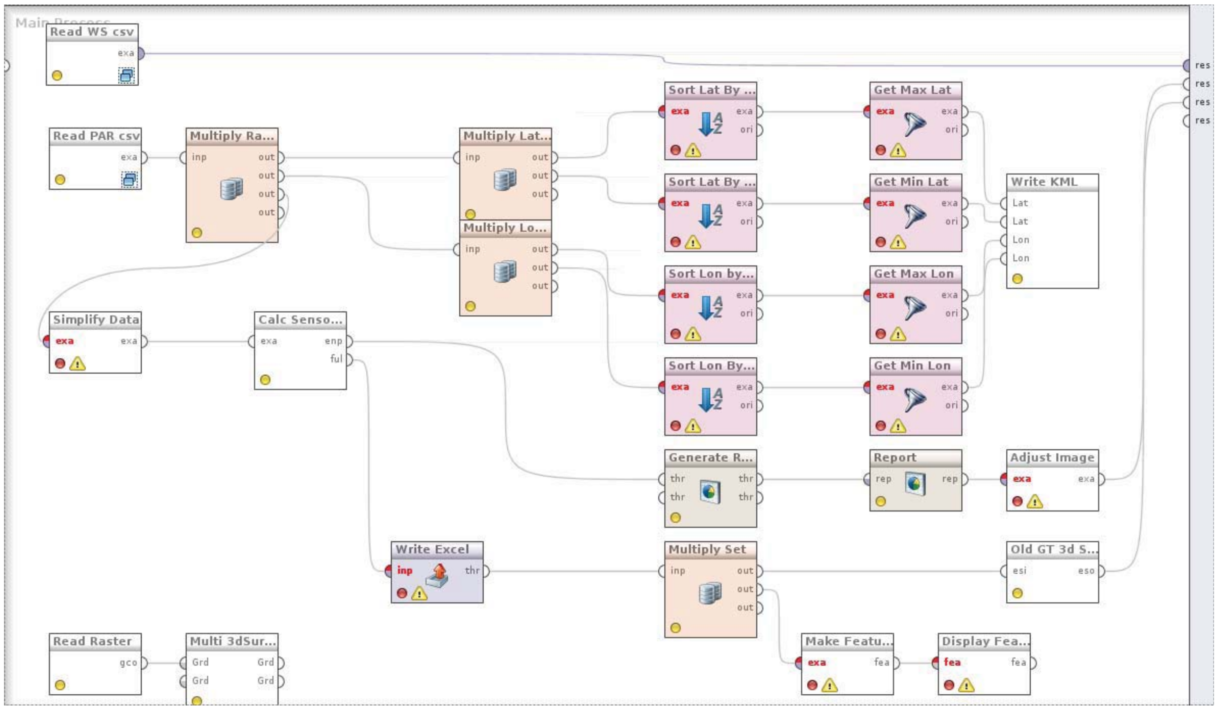

Figure 3. A data-flow-driven design model for generating 2-D image overlaid on the Google Earth.

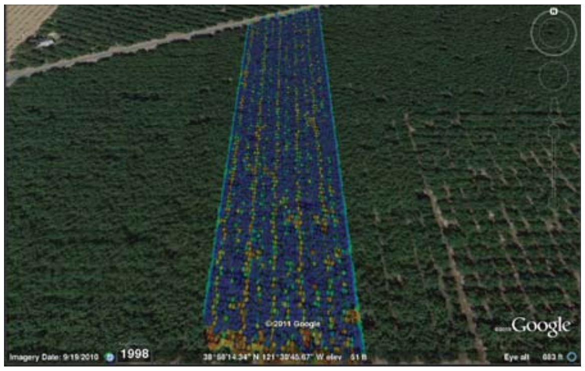

Figure 4. A 2-D visualization of Photosynthetically Active Radiation (PAR) data overlaid on the Google Earth [2].

and overlay 2-D data image on the related terrain. Figure 4 shows the Google Earth overlaid with a 2-D image of Photosynthetically Active Radiation (PAR) data. This feature provides geographic reference to visualized data sets, and has been proven very useful in our case study.

3-D data visualization visualizes multiple data sets or a data set with multiple features. Additional data sets or features may be visualized in the following ways,

1. On the additional dimension (Z-axis). Compared with 2-D visualization, the additional dimension may be used to plot a data set with two features, one using a color map and the other using Z-axis.

2. As multiple layers of images. Each data set can be plotted as an individual layer, with its own scale on Z-axis.

In addition, two options may be combined to reveal more features of a data set. Figure 5 is a 3-D visualization of Photosynthetically Active Radiation (PAR) data.

Data visualization operators may work with other operators to process and visualize data in various formats, including non-grid data. To convert a location-based non-grid data set to a grid data set, we first build a R-tree [3] to store and retrieve the non-grid data set. A user may define the dimensions of a grid used in conversion. A data interpolation operator is then used to interpolate data points on the grid: to compute the value of a data point on the grid, its neighbouring data points on the input non-grid data set are retrieved from R-tree using Sort-Tile-Recursive (STR) algorithm [5]. The data point on the grid is interpolated from these neighbouring points. As an example, in our case study the Photosynthetically Active Radiation (PAR) data comes as a series of data points in form of $\langle longitude, latitude, value\rangle$. These non-grid data are then converted to grid data, which are then visualized as 2-D raster image with a transparent

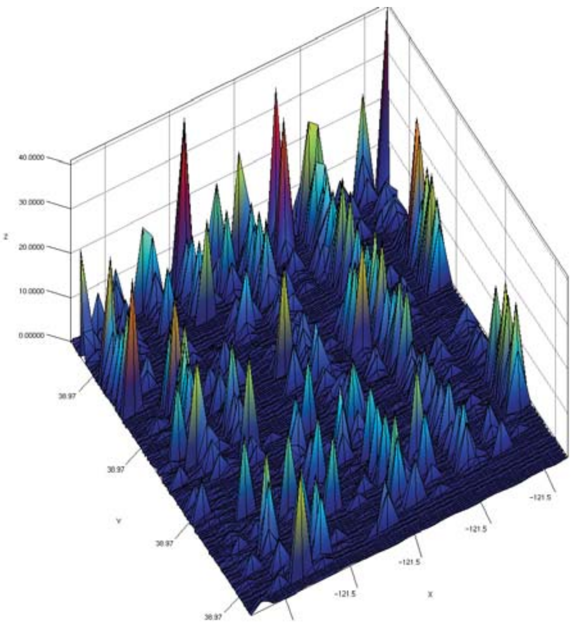

Figure 5. 3-D Visualization of a PAR Data Set. The x and z axis are coordinates for the boundry of the field and the y axis is the PAR data collected for that specific coordinate in the field

background. The 2-D image is overlaid on its related terrain rendered by the Google Earth.

4 A Client-Server Architecture Design

An objective of the proposed software architecture is that it shall support a variety of client devices, and yet it shall be scalable for handling data-intensive precision agriculture applications. Towards this end, the software architecture adopts a client-server model as its overall architecture design. The design has a clearly-defined server interface using a socket-based protocol. This enables the design to support a range of Internet-connected devices, including mobile devices, as long as they conform to the interface requirements.

In our client-server design, server(s) handles the majority of data processing, and a client device handles human-computer interactions and visualizes results from the server(s). Since most of heavy-lifting tasks are handled by the server(s), requirements on computing power and power assumption of a client device are reduced. The reduction on power assumption is specially important to a mobile device, since often battery power limits its extended field use. Furthermore, this client-server design enables us to allocate computation resources on the server end as needed, or eventually move the server end to a cloud-based computing platform for even greater scalability.

Figure 6 shows the client-server architecture design in UML deployment diagram. A client device communicates with a server using a socket-based protocol. The communication layer is realized by `Comm/Client` on the client and `Comm/Server` on the server. The server is in charge of processing data and preparing the result for data visualization on a client device. A feature of our client-server design is that the task of data visualization is distributed among the server and client devices: upon the completion of data process and analysis, the visualization component on the server (i.e. `Visualization/Server`) prepares the result as a multi-feature grid data set, ready for being visualized on the client. The visualization component on the client (i.e. `Visualization/Client`) displays the final result by using the client's native graphic capability. For example, our prototype tool includes a client application running on Linux. The visualization component of the client application optimizes data visualization using OpenGL, a computer graphic programming standard supported by many high-performance graphic cards.

We use 3-D visualization as an example to demonstrate how our client-server architecture design may optimize performance by distributing the task among servers and clients. Figure 7 gives a sequence diagram showing messages exchanged during 3-D visualization. A typical sequence of messages is initialized by a user's request. The client communicates with the server through asynchronous messages. The controller component of the server handles requests from clients and maintains session information. For instance, our system maintains a database of fields and data sets associated with these fields. A connected client may request to change the field of selection. The request is delivered through the communication layer to the controller, which keep a record for the selected field.

To further improve the scalability of our design, the analysis component on the the server is multi-threaded, and an analysis thread is created on demand per request from a client. For the example shown in Figure 7, an analysis thread is created for handling user's request for 3-D visualization. The controller sends the stored session information (i.e. `field_id` and `data_type`) to the analysis thread. The analysis thread queries the storage for data sets and parameters. It processes data based on a workflow defined in a data-flow model (Section 3.1). The result is sent back to the client. The visualization component of the client renders 3-D image using the graphic hardware on the client.

5 Implementation

To demonstrate the capability of the proposed software architect, we use it to implement AgriD, an integrated data

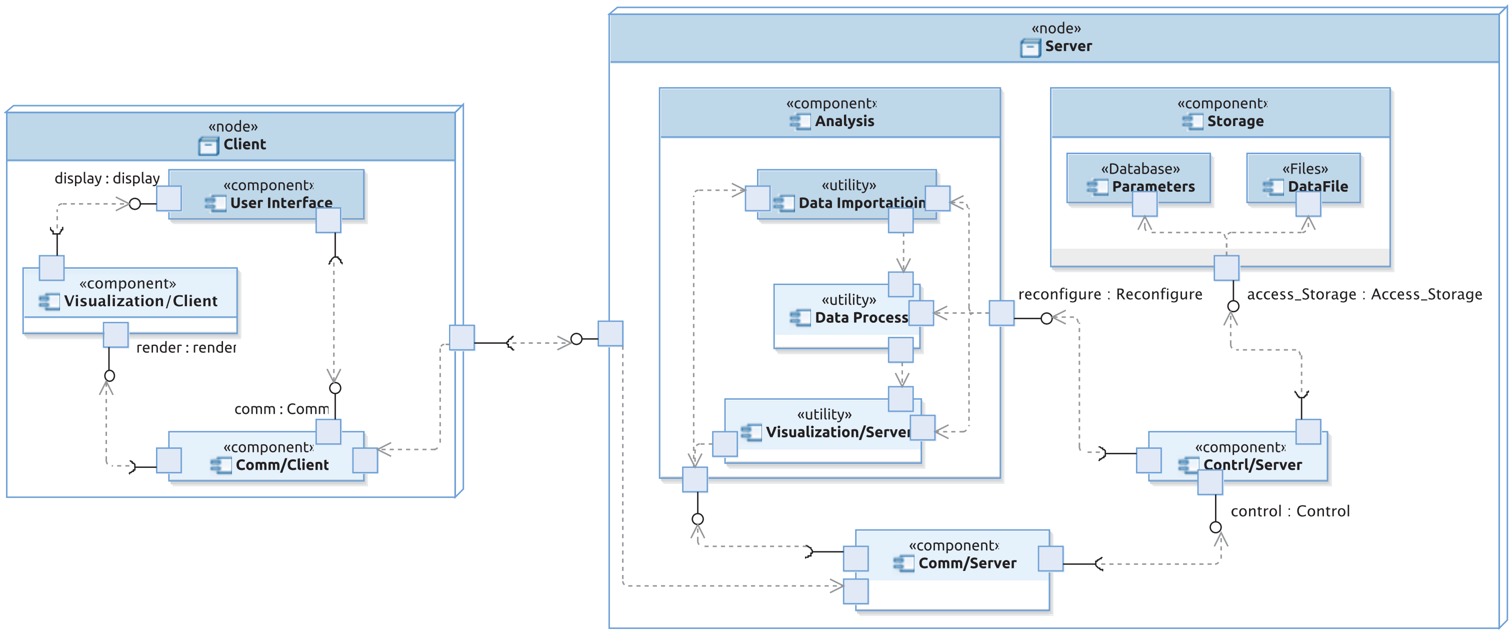

Figure 6. A Client-Server architecture design for deploying heterogeneous client devices.

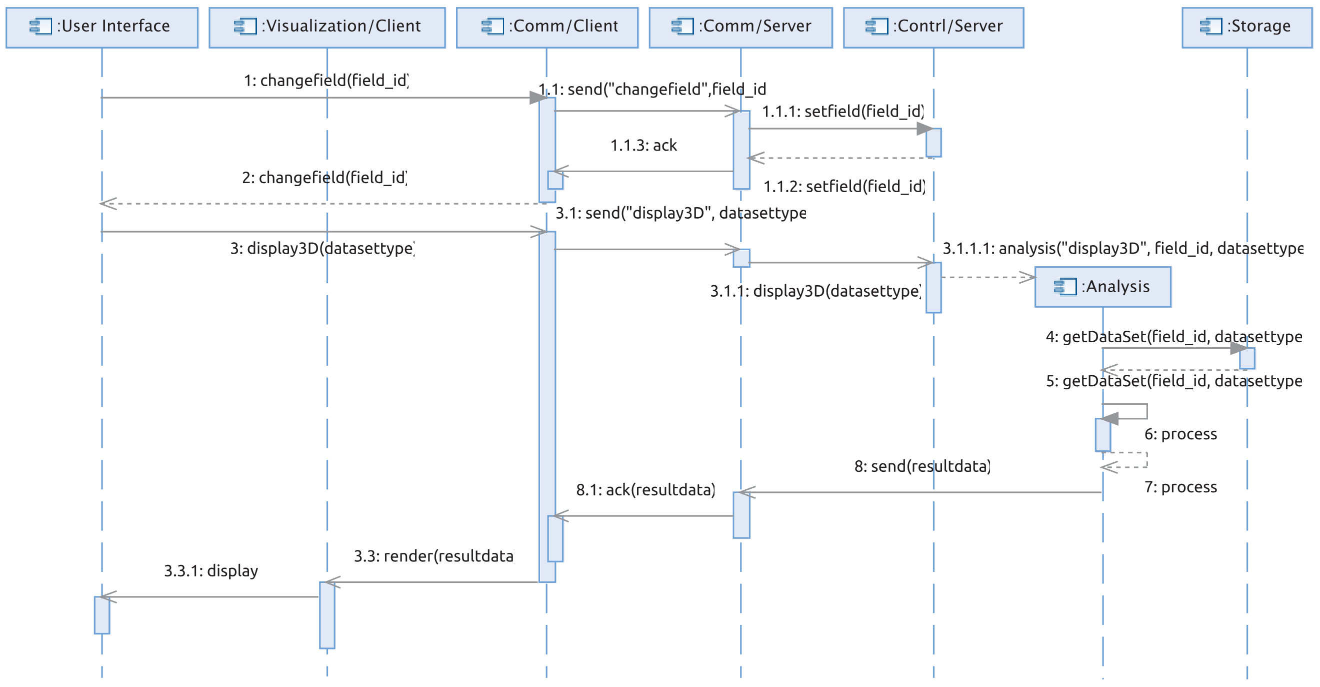
:User Interface
:Visualization/Client
:Comm/Client
:Comm/Server
:Contrl/Server
:Storage
1: changefield(field_id)
1.1: send("changefield",field_id
1.1.1: setfield(field_id)
1.1.3: ack
2: changefield(field_id)
1.1.2: setfield(field_id)
3.1: send("display3D", datasettype
3: display3D(datasettype)
3.1.1.1: analysis("display3D", field_id, datasettype
3.1.1: display3D(datasettype)
:Analysis
4: getDataSet(field_id, datasettype
5: getDataSet(field_id, datasettype
6: process
8: send(resultdata)
7: process
8.1: ack(resultdata)
3.3: render(resultdata
3.3.1: display

Figure 7. The server interface in UML sequence diagram.

analysis and visualization tool for precision agriculture. AgriD is developed in Java. Its code base consists of 2946 lines of custom-developed Java code, and 14508 lines of code from a range of external open-source tools.

The overall design of AgriD follows the client-server design in Section 4. The communication layer uses a socket-based communication protocol. The data exchanged between communication components on server and on client are encapsulated using JavaScript Object Notation (JSON). The server interface defines a list of commands and their arguments. A subset of these commands (e.g. "`changefield`" and "`display3D`" etc) and a typical message flow involving these commands have been illustrated in Figure 7.

The analysis subsystem of AgriD follows the data-flow-driven design in Section 3. The analysis subsystem is implemented on the top of an open-source data mining tool RapidMiner [8]. RapidMiner provides an interpreter and simulator for data-flow-driven models. We build on the top the RapidMiner the capability of defining, serializing, and executing data-flow design models. These models define workflows of data processing. We developed a set of operators that implemented functions central to precision agriculture, including data visualization operators (Section 3.2), and data importation and integration operators (Section 2).

AgriD has been used in a USDA-sponsored project on canopy management of specialty crops. AgriD imports heterogeneous data collected from various sensor inputs, including PAR data, weather data, and ground temperature data. Because of various types of devices and configurations used in the project, the format and semantics of input data change over time. Our meta-data-model-based approach (Section 2) provides a flexible data importation and integration schema for importing these heterogeneous data.

3-D data visualization is rendered using Jzy3d, an open source Java library that enables a rapid display of scientific data [4]. Because Jzy3d uses OpenGL to hardware acceleration capabilities of a graphic card, it significantly improves the performance of rendering 3-D images. A user can also manipulate 3-D images on-the-fly, which provides the user an interactive experience in data visualization.

6 Conclusion

Precision agriculture presents great challenges and also unique opportunities for Computer Scientists, particularly those in the areas of data analysis and decision support. Recent advances in precision agriculture call for a software tool that can process a large quantity of data collected from a heterogeneous network of devices, support a variety of client devices, and provide on-the-fly access. To address these needs, we developed a software architecture that is to significantly improve the efficiency, flexibility, and scalability of data analysis and decision support in precision agriculture. To meet these objectives, our software architecture has three distinctive features: (a) a meta-data-model-based data importation module that supports custom-defined data models. It is capable of importing data in various formats from a variety of devices in different settings; (b) a data-flow-driven data processing subsystem which supports custom-defined workflows and data processing operators; and (c) a client-server architecture design that supports a variety of client devices, including the Apple iPad.

We implemented the software architecture in AgriD, an open-source data analysis and visual tool for precision agriculture. AgriD has been successfully used in a USDA-sponsored project on canopy management for specialty crops. For the future work, we will continue to extend this software architecture, including migrating it to a cloud-based computing platform. We will also develop a browser-based client solution, which will support a range of client devices, including many internet-connected devices capable of web browsing.

References

[1] H. Auernhammer. Precision farming - the environmental challenge. *Computers and Electronics in Agriculture*, 30(1-3):31–43, Feb. 2001.

[2] Google. Google Earth. http://earth.google.com.

[3] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the ACM SIGMOD international conference on Management of data*, page 47, 1984.

[4] Jzy3d team. Jzy3d. http://www.jzy3d.org.

[5] S. Leutenegger, M. Lopez, and J. Edgington. STR: a simple and efficient algorithm for R-tree packing. In *Proceedings 13th International Conference on Data Engineering*, pages 497–506. IEEE Comput. Soc. Press, 1997.

[6] R. Nikkilä, I. Seilonen, and K. Koskinen. Software architecture for farm management information systems in precision agriculture. *Computers and Electronics in Agriculture*, 70(2):328–336, Mar. 2010.

[7] 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.

[8] Rapid-I. RapidMiner. http://www.rapid-i.com, 2011.

[9] P. C. Robert. Precision agriculture: a challenge for crop nutrition management. *Plant and Soil*, 247(1):143–149, Nov. 2002.

[10] Q. Zhang, L. Cheng, and R. Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, Apr. 2010.