A use case driven domain analysis for precision agriculture information systems

SARAIVA, A. M.; CUGNASCA, C. E.; HIRAKAWA, A. R.; MASSOLA, A. M. A.
Dept. of Computer Engineering - Agricultural Automation Laboratory
Universidade de São Paulo - Escola Politécnica
Caixa Postal 61548 - São Paulo - SP - 05424-970
BRAZIL

Abstract: - Precision agriculture is a recent technology which is changing agricultural practices and concepts. Since it is strongly based on huge amounts of data, one of its main components is the information system that supports decision making. Such systems are nowadays one of the gaps in the technology, since the existing ones do not cover the minimum requirements posed by the application. This paper deals with the problem of an adequate specification of the functionality of this class of systems. In order to support the development of these systems, a study was conducted to identify their desired features, based on current software engineering paradigms and techniques such as object orientation and domain analysis. The concept of use case was adopted because it allowed a clear understanding of the functions that the system should provide from its users point of view. The paper presents the method and concepts adopted for a domain analysis for this class of systems, and presents the use cases that resulted from that analysis.

Key-Words: precision agriculture, information systems, domain analysis, use cases, object-orientation, Proc.pp..4671-4676

1 Introduction

Precision agriculture (PA) is a technology, or else a set of technologies, that is considerably changing agricultural practices and concepts as it introduces the spatial variability as another factor to be taken into account in field operations and on farm management.

The basic idea of precision agriculture is to more accurately consider the differences in important parameters (such as soil fertility, soil moisture, yield) that exist within a single field. Conventional agricultural practices would deal with those differences by taking an average value for each parameter and then treating the whole field as being uniform. Based on that average value subsequent operations, such as fertilizer applications, would be carried out following a recommendation that a certain amount of fertilizer be uniformly distributed to the field.

As a consequence, every field operation, such as inputs application, might result in over or under application in most of the field except in those spots where the actual value of the parameter equals the average that was considered for the recommendation.

With precision agriculture this tends to occur in a much smaller scale, since the field variability of the parameters is now measured by electronic equipment (the data acquisition phase of PA) and the inputs are applied variably according to the actual needs of each spot of the field by means of computer controlled application equipment (variable rate application phase).

However an important question that is yet to be answered is what to do with the data collected in order to obtain an adequate recommendation for the application. In other words, this intermediate phase of precision agriculture, which can be named information management, is by far the most undeveloped. Part of the problem is the lack of knowledge on the processes related to crop growth at this unprecedented level of detail. But another important issue is the lack of adequate computer tools to turn all the huge amount of data collected into decisions and to help develop the missing knowledge. [1]

2 A Method for Domain Analysis

In order to understand the reasons for this situation and define the role of the information systems in the information management in PA, a research work was proposed. An important aim of the work was that of analyzing the requirements for this class of systems so as to help develop specific systems and to evaluate the adequacy of the ones available.

A first theoretical basis for the work was the software engineering concept of domain analysis. When the aim of a work is the development of a system, the requirement analysis is an important activity that should be carried out on the early stages of systems development. That activity is based on the problem analysis and should help define how to solve the problem, what the system should do and how to design it so as to allow its future evolution. Domain analysis can be thought of as being the equivalent to the requirement analysis phase when, instead of a problem, one is dealing with a class of problems and, instead of a single solution (a target system), a class of solutions (a class of systems or a meta-system) is being pursued. In other words, domain analysis deals with the identification, acquisition e representation of knowledge about the specification and implementation of software for classes of real world problems [2].

An important point is that this knowledge must be potentially reusable within the specific domain, since the meta-system that one is trying to identify aims at being the guideline for future implementations of (many) specific systems. It becomes evident from this statement that domain analysis and software reuse are tightly coupled.

The methods adopted for domain analysis vary widely, from formal to more practical ones. A possible approach is that derived from knowledge engineering, that adopts as basic steps the identification of suitable knowledge sources, knowledge acquisition and knowledge representation.

However, these steps are not trivial, as experience shows. Reverse engineering has played an important role in the process of search and acquisition of knowledge. Starting from a set of existing solutions, one tries to extract their common characteristics to allow a generalization.

Many other knowledge sources, as well as many different profiles of human experts, can take part in this process. Among the former, one can mention technical literature, available systems, users knowledge. Among the latter, one can list a system analyst, system users, domain analyst, domain expert [2].

The output of a domain analysis process can vary widely, ranging from standards, recommendations, interface definitions, to functional models, structural models, domain taxonomies and domain languages.

The degree of formalism found in these outputs is also very different. Besides depending on the purpose of the analysis, another important point is the difficulty of how to bridge the gap between the knowledge sources of a domain and their formal or semi-formal representation.

Object orientation has been suggested as a concept that might be of great help to requirement formalization and elicitation, as well as to knowledge acquisition [3]. Object oriented methods present some advantages to other paradigms because they structure the system in terms of objects that exist in the real world in the problem domain, and that interact with each other. This is an important characteristic because it reflects the way people see their environment, i.e., in terms of objects. Hence it becomes easy to think in terms of objects when modeling a system, and it becomes easy to understand an object model. In other words, the semantic gap between model and reality is small.

One important feature is that object technology is concerned with specifying what an object is, rather than how it is used. The uses of an object are highly dependent on application details and change frequently along the development. As the system requirements change, the object characteristics are much more stable than the way the objects are used. Thus, software based on an object structure is more stable in the long run. This potential stability is essential for a domain model of PA given the fact that the technology is in its early stages and many of the “how to” questions are still unsolved, while most of the “what” questions (the objects) can be more easily defined.

In short, object-oriented modeling and design allow for a better understanding of systems requirements. Thus, it leads to better models of the problem domain, simpler design due to software structuring in terms of abstract data concepts, more maintainable systems, and allows for the application of software reusability during development of specific systems [4].

Another concept that was borrowed from software engineering and incorporated to this approach to domain analysis, was the one of “use cases” [4]. Use cases represent a sequence of interactions related to a behavior or a function of the system, in a kind of dialog between a user and the system. They show the ways the system is used by its users and how it interacts with them. Actors represent everything that there is outside the system and interact with the system for information interchange: users (people), machines, other systems. The actors represent roles these external entities can play and hence they can be understood as classes of users. System users are instances of actors and can assume different roles at different moments.

The identification of use cases can be done for instance by asking the actors what kind of action they want to perform with (or in) the system. The use cases are represented in a use case model, whose notation is very simple [4]. Once the use case identification becomes stable, each use case must be described in detail, in text form, showing the sequence of events that constitutes it and the possible exceptions and deviations or branches. These descriptions are important as they will guide the identification of domain objects that will form the object model.

This user-based approach allows the fundamental role of the user in systems specification to be more adequately considered.

In summary, the method used to do the domain analysis included many different knowledge sources, different profiles of experts, and the concepts and notations borrowed from objects and use cases [4] [5]..

3 Results and Discussion

Following the method adopted for the domain analysis, the first activity consisted in identifying the knowledge sources, human or non-human. The acquisition of such knowledge was in a first stage based on traditional methods, such as readings all the documents obtained – mainly research papers, but also documents from existing systems, in a kind of reverse engineering. A very important role was played by the contact with system users of different profiles, such as researchers and consultants who could transmit their feelings of the weaknesses and strengths they saw on the systems with which they came across.

The huge amount of information acquired had to be structured, and the decision was made to write down a first document, Problem Definition, as suggested in [4]. The text defines an initial systems requirements set, mainly from the high level functionality point-of-view. Despite the ambiguities inherent to the textual description, it is a starting point for the models that follow, which go deeper into systems requirement formalization, using the concept and the notation of use cases and objects.

From that text, some important features the systems for PA should present can be summarized [6]:

- implement a single and coherent user interface, eliminating the need to know different packages and their operation;
- automate data exchange procedures, eliminating the need for data set preparation for execution in different packages;
- allow information tailoring according to the user;
- automate data processing and analysis, following previously validated recipes, thus freeing the user from having to understand each analysis technique and its parameters; this must take into account the different users of the system;
- allow the operator to interfere in the analysis process, creating himself his analysis recipes;
- provide means for the operator to have access to meta-data, i.e., data about the data, about the form of acquisition, and about the operations and data that generated new data;
- allow the operator to interfere on the results, inserting his impressions and knowledge directly into the maps; giving him the power of decision, and keeping him in control of the operations; the user must be able to interact with the system;
- allow simulation of different scenarios, whenever possible, through previously validated simulation models;
- be implemented in an open architecture, based on *de facto* (or *de jure*) standards in order to lessen the problems related to integration with other packages, other data sources, local or distributed;
- be structured in such a way so as to facilitate further integration of future modules, components, recipes, accompanying the evolution of PA; be expansible;

Following the generic and informal description of the Problem Definition, the next step on the domain analysis consisted in the use cases identification. This was done in a series of iterations and for that, all the information acquired with existent systems and by means of interviews with users was essential.

One important thing of use case modeling is the definition of the limits of the systems and of its interfaces to the eternal entities, the actors. So, instead of presenting as a first diagram a more complex set of use cases, it was considered more adequate to group the use cases into a few more abstract cases that could exemplify the main functions of the system and its interfaces with the external world.

Based on that point of view the use case model derived was the one presented in Figure 1. It shows that the main functions of the system – insert field data, generate maps, and configure the system. It also shows the four actors that represent the classes of users of the system: operator, manager, data-acquisition equipment and application controller equipment.

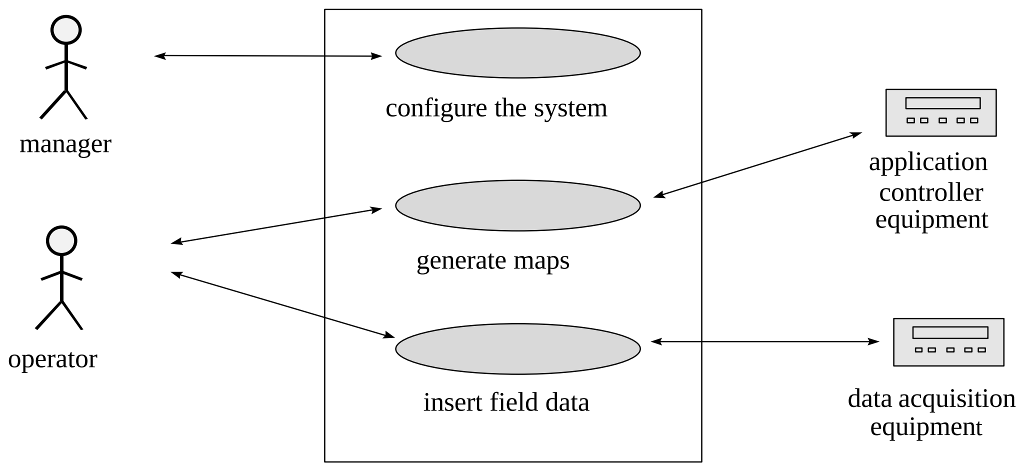

Fig.1 Use case model.

The use case "insert field data" groups all the actions related to the input of data collected from the field, both manually by an operator or automatically by means of data acquisition equipment.

The use case "generate maps" groups all the actions related to processing of the data in map form, such as corrections, editing, overlaying and deriving of further map layers. It is performed with the operator's participation and the final application map can be transferred to an application controller equipment.

The use case "configure the system" is performed only by the actor "manager", which means that this is an off-line use of the system and aims at configuring the system for different operators, crops, regions, etc.

The actor "operator" represents a class of end-users of the system, probably consultants, researchers, and maybe a few farmers.

The actor "manager" represents the class of programmers that will work on the final systems in order to customize it to each situation – a different crop, different region, different agricultural inputs, specific knowledge, etc.

Finally the actors "data acquisition equipment" and "application controller equipment" represent the classes of the proper equipment. It was decided to depart from the traditional notation [4] and use a different symbol for the non-human actors for a more clear distinction between these super classes of actors.

As mentioned before, this diagram hides a complex set of sub-use-cases and was simplified on purpose. The next step was then to describe these use cases in great detail in order to allow for the identification of objects for an object model of the domain.

.....The list of sub use cases that were grouped in the three main use cases is presented below. For each of them a detailed description of the interactions between system and user was written.

The use cases and their sub-cases are:

Use case: Insert field data
- New farmer registration
- New farm registration
- New field registration
- Sampled data reading, from external equipment
- Manual input of sampled data

Use case: Configure the system
- Management rules configuration
- Data base configuration of inputs and equipment
- Task recipes configuration
- Task automation shortcuts configuration
- New operator register

Use case: Generate maps
- Sampled data files correction
- Primary data layers creation
- Basic data layers creation
- Data layers manual edition
- Secondary data layers creation
- Scenario simulation execution
- Application maps generation
- Output files generation for application controllers
- Map printing

Each description, text or graphical, adds more structure to the knowledge of the domain as the analysis proceed. The final level adopted for this work was that of deriving object models for the domain. The objects were selected from the use cases descriptions. Although this selection is not straightforward, some guidelines can be found in many object-oriented methods, such as searching for names, substantives, etc. The choice of the objects demanded much iteration but after all it arrived to a stable and fairly complete set of objects. Instead of having a single object diagram for the whole domain, for the sake of legibility of the diagrams, the choice was to keep separate object diagrams for each use case. The objects that appear in more than one diagram show the links that exist between them. Figure 2 shows a sample object diagram, the one that represents the use case "insert field data".

4 Conclusion

In order to help understand the complexity of the domain of the information systems for precision agriculture and to provide guidelines for specific systems development, a domain analysis was conducted.

Since there are a number of approaches to domain analysis, an innovative method was proposed, having as main concepts the use of use cases for the requirements specification and object orientation for structuring the knowledge.

It was based on a more informal initial phase, in which a wide variety and a huge amount of information was gathered and studied, from which a textual description of the main needs of this class of systems resulted.

The concept of use cases was then used to define the main sequences of interactions between the systems and the external entities, the actors, which may be humans or other systems or equipment.

These use cases where further detailed and from this detailing, objects were selected to build object diagrams. This structuring of the knowledge in terms of domain objects, has the potential of being stable although the PA technology is changing.

The models are easy to understand, even for those not used to systems modeling, since the notations are simple and the objects that appear on the model have their counterpart in the real world. This helped the communication with users and domain experts and facilitates their cooperation in the modeling process.

References:

[1] Saraiva, A.M. et al. An object model for information systems for precision agriculture. *Proceedings of the Fourth International Conference on Precision Agriculture*, Saint Paul, 1998. (in print)

[2] Arango, G.; Prieto-Díaz, R. Introduction and overview: domain analysis concepts and research directions. In: Prieto-Díaz, R.; Arango, G. (ed) Domain analysis and software systems modeling. IEEE Computer Society Press, Los Alamitos. 1991. pp..9-32.

[3] Greespan, S.J.; Mylopoulos. J.; Borgida, A. Capturing more world knowledge in the requirements specification. In: Prieto-Díaz, R.; Arango, G. (ed) Domain analysis and software systems modeling. IEEE Computer Society Press, Los Alamitos. 1991. pp.53-62.

[4] Jacobson, I. et al. *Object-oriented software engineering: a use case driven approach*. 4.ed.rev. Harlow, Addison Wesley Longman/ACM Press, 1993.

[5] Rumbaugh, J. et al. *Modelagem e projetos baseados em objetos*. Rio de Janeiro, Campus, 1997. (Portuguese version of *Object-oriented modeling and design*, 1991)

[6] Saraiva A.M. et al. Object oriented approach to the development of a field information system. *ASAE Paper nº 97-3015*. St. Joseph, ASAE, 1997.

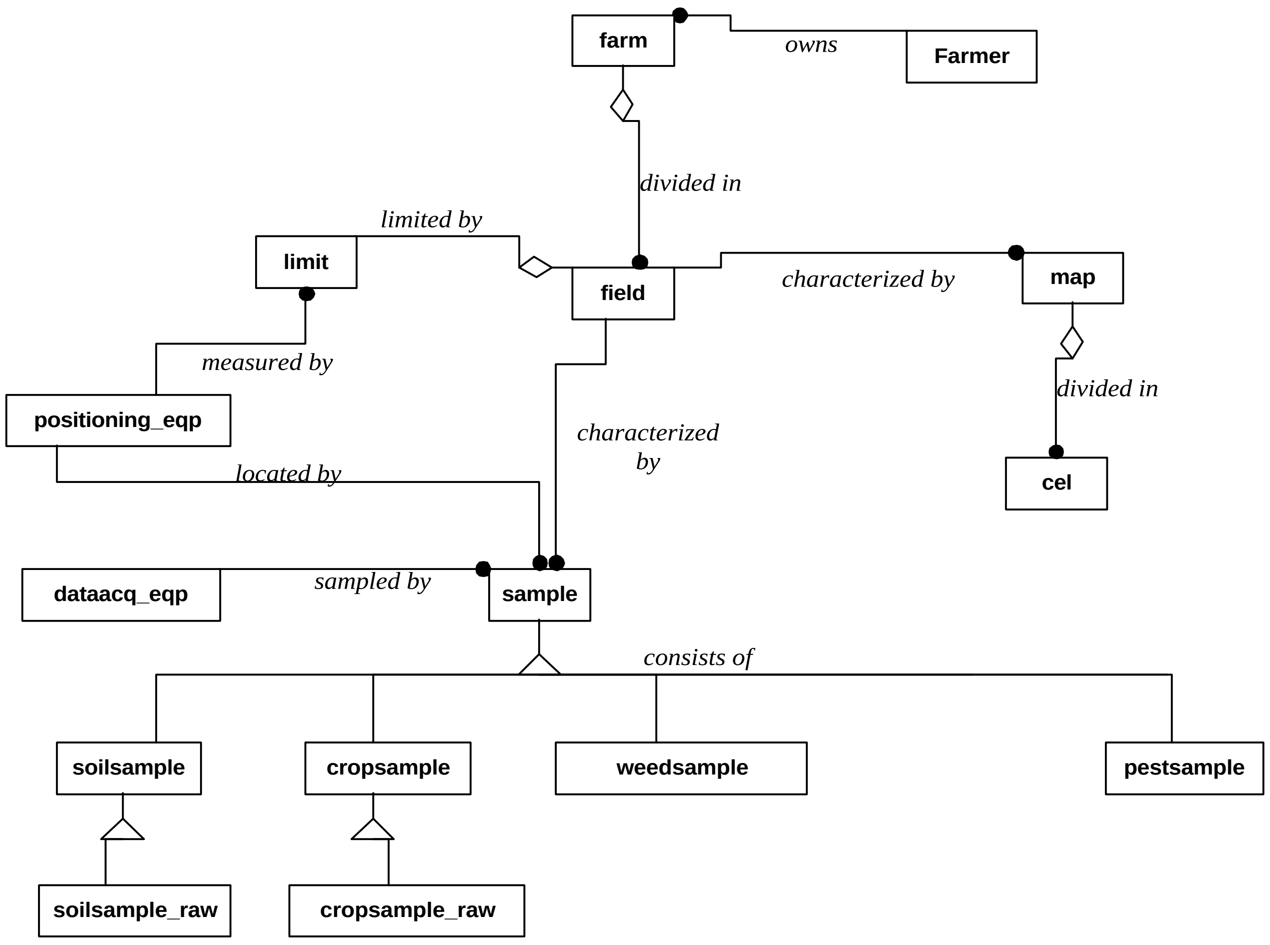

Figure 2 - Class diagram - Use case "Insert field data"