A Knowledge-Intensive Approach to Computer Vision Systems

Nicole Koenderink

Stellingen

behorende bij het proefschrift

Knowledge-intensive Computer Vision Applications

door Nicole Koenderink,

8 februari 2010

1. Alleen door kennis van domein experts te formaliseren kan een geautomatiseerd visueel inspectieproces transparant worden opgezet. (Dit proefschrift)
2. De vakgebieden kennismanagement en beeldverwerking zijn beide cruciaal voor het automatiseren van transparante kennis-intensieve computer vision systemen. (Dit proefschrift)
3. 'White-box' computer vision systemen voor kennis-intensieve beeldverwerkingsapplicaties bieden voor zowel de ontwikkelaar als de eindgebruiker significante voordelen ten opzichte van 'black-box' systemen. (Dit proefschrift)
4. Positieve discriminatie is een onwenselijk instrument om meer vrouwen in topposities te krijgen daar het de positie van de aangenomen vrouwen ondermijnt.
5. Architecten dienen alvorens hun auteursrechten op een gebouw gerechtelijk uit te oefenen er eerst drie weken te wonen of te werken.
6. Door de aanschafbelasting op voertuigen uitsluitend te koppelen aan de CO_2-uitstoot en niet langer te compenseren voor het gewicht van het voertuig, kan de Staat der Nederlanden laten zien dat ze de Kyoto-doelstelling wel serieus neemt.
7. De betekenis van het woord 'geloven' in "geloven in God" en "geloven in de evolutietheorie" wordt ten onrechte als hetzelfde beschouwd.
8. Een uitvaartverzekering is een ongebruikelijke spaarvorm, aangezien mensen een warm gevoel krijgen bij een rentepercentage dat gelijk is aan de inflatie.
9. In een winstgevende projectorganisatie kan urenregistratie geen juiste weergave zijn van waar uren werkelijk aan worden besteed.
10. Met de opkomst van Wikipedia doet de kennisdemocratie zijn intrede.
11. Het gemiddeld jaarlijks rendement van een beleggingshypotheek wordt over het algemeen correct door hypotheekadviseurs voorgespiegeld. Indien het beloofde gemiddelde rendement daadwerkelijk gehaald wordt, zegt dit echter niets over de uiteindelijke geldsom.
12. De gemeente Utrecht zou in haar luchtvervuilingsmodel ten minste rekening moeten houden met enkele simpele behoudswetten. Volgens de huidige parametrisering verdwijnt al het busverkeer tussen Nieuwegein en Utrecht in het niets. (Bron: *Giftige cijfers*, april 2009, `http://utrecht.sp.nl/weblog/wp-content/uploads/2009/04/giftige-cijfers-20.pdf`)

Deze stellingen worden verdedigbaar geacht en zijn als zodanig
goedgekeurd door de promotoren, prof. dr. ir. L.J. van Vliet en prof. dr. J.L. Top.

A Knowledge-Intensive Approach to Computer Vision Systems

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. ir. K.C.A.M. Luyben,
voorzitter van het College voor Promoties
in het openbaar te verdedigen op maandag 8 februari 2010 om 10:00 uur
door

Nicole Josephina Johanna Petronella KOENDERINK-KETELAARS

doctorandus in de Wiskunde en
Master of Technological Design in Wiskunde voor de Industrie,
geboren te Helmond.

Dit proefschrift is goedgekeurd door de promotoren:

Prof. dr. ir. L.J. van Vliet
Prof. dr. ir. J.L. Top

Samenstelling promotiecommissie:

Rector Magnificus, voorzitter
Prof. dr. ir. L.J. van Vliet, Technische Universiteit Delft, promotor
Prof. dr. ir. J.L. Top, Vrije Universiteit Amsterdam, promotor
Prof. dr. C.M. Jonker, Technische Universiteit Delft
Prof. dr. ir. P.P. Jonker, Technische Universiteit Delft
Prof. dr. G. Schreiber, Vrije Universiteit Amsterdam
Prof. dr. P.C. Struik, Wageningen Universiteit
Dr. ir. J.C. Noordam, TNO

This work was supported by the Dutch Ministry of Economic affairs (IOP Beeldverwerking, IBV02010; ICT innovation program), the Dutch Ministry of Agriculture, Nature and Food Quality, the Dutch Ministry of Education, Culture and Science (BSIK grant), and by the involved companies in the Glasgroentegroep of Plantum. Part of the work was written within the context of the Virtual Laboratory for e-Science (VL-e) project (http://www.vl-e.nl), sub-programme Food Informatics.

ISBN 978-90-8585-564-4

Contents

Chapter 1

Introduction

Visual inspection plays a crucial role in today's industry. It is used for quality inspection of raw materials and produced goods, for guiding logistical processes, for automatically assigning products to quality classes, and for assistance in various other tasks. Some of these tasks are simple checks on the presence or absence of a part, but others, such as determining the quality of plants, are more complex and are at present usually performed by highly trained and experienced human operators. In the more complex tasks, the knowledge and expertise of the operators is crucial for correctly performing the task. Automation of such tasks requires access to the involved expert knowledge in such a way that it can be used in a computer vision system. In many cases, the required expert knowledge is not available in an explicit format. As a result, automation of a knowledge-intensive task in a way that the application benefits from the existing knowledge and expertise is difficult.

Mostly, automated applications perform the task in an opaque way in the sense that the knowledge used by the experts cannot easily been identified in the code. This makes *e.g.* trouble-shooting and performance checking difficult tasks. This thesis describes how knowledge-intensive computer vision tasks can be automated in such a way that the *knowledge of the experts is explicitly used*. To this end, we combine techniques from the fields of knowledge engineering and computer vision. We aim to contribute to both fields.

1.1 Automation in practice

Mankind has been using tools and techniques to assist in a wide variety of tasks since very early times. The use of flint for creating axes, the use of bone for creating needles and the use of sticks for creating bows are well-known examples of tools used by prehistoric people. These tools were hand-powered.

Mechanisation, the next step in using tools and technology, introduces the use of *machines* for replacing manual labour or to the use of powered machinery to assist a human operator in performing a task [36]. Some examples of mechanised tasks from early history are the irrigation systems in the Mesopotamian

area and the implementation of automatically opening temple doors, devised by the Greek Heron. In the Middle Ages mills and pumps were designed and employed in *e.g.* textile, paper and mining industries. In 1769, James Watt patented his steam engine principle and mechanisation really took off.

A further development in the area of mechanisation is *automation*. Automation can be defined as 'the use of control systems such as computers to operate industrial machinery and perform processes, replacing human operators' [1]. In this thesis, we focus on *computer vision*, a key technology in the automation of visual inspection tasks. The term computer vision is used to refer to automated systems in which input is acquired from a camera or a visual sensor system. Computer vision tasks generally consist of three sub-tasks: *seeing*, *thinking* and *acting*. 'Seeing' is the image acquisition process that deploys a camera to obtain a representation of the object in the memory of the computer. 'Thinking' is the image assessment process in which the recorded object is modelled, all relevant information is processed by the computer vision software and appropriate action is decided upon. 'Acting' is the final process of the computer vision application in which it communicates with external actuators and indicates which actions should be undertaken. We illustrate this computer vision paradigm with an example of automated rose harvesting. A bush of cut roses is 'seen' with a colour camera. The position and the ripeness of individual roses in the recorded bush are identified in the 'thinking' process. The position of the ripe roses is next communicated to the automatic harvester robot. This robot 'acts' by cutting and transporting the ripe roses.

1.2 Impact of computer vision on industrial tasks

Computer vision is considered for use in industry. Automating tasks is of interest when costs can be reduced, when the desired level of detail, precision, or accuracy can be improved, or when speed can be increased. Improved precision plays a role when human operators cannot meet the quality standards that are demanded by industry. Increased speed is important when the human operators cannot achieve the desired throughput. Both aspects offer the possibility to increase profit for a company. However, process automation is not always feasible. This thesis aims to develop an approach that increases the viability of automating computer vision tasks.

Automation of inspection processes can be employed across a wide range of applications. Machines can be used to automate the handling of objects, as in car assembly industry. Automation can be used to implement and guide feedback modules in a production process. An example is the measurement of paper wetness in the paper production process to adjust the speed of the paper pulp being deposited on a conveyor belt at the beginning of the process. In general, computer vision can be used to replace human inspection and sorting tasks, to support medical teams in performing precision surgery or to facilitate research

[1] `http://en.wikiquote.org/wiki/Automation`

by assisting in visual analysis of experiments. To illustrate the broad range of applications that are based on the use of computer vision technology, we now summarise a number of examples from practice.

Car assembly industry

Car assembly is an industrial process in which the first automation steps have been taken as early as 1902. In 1913, Ford introduced the first automated assembly line. Nowadays, the assembly of cars is largely taken over by robots. For example, in 2003, a new General Motors car factory opened [114]. In this factory, 338 programmable welding robots are used to perform over 2000 welds per car. They are assisted by more than 50 vision-enabled robots to place tools and hold parts in place. All in all only 1500 workers are employed in this factory.

Car assembly is a process for which automation enables a much higher production speed than manual assembly[2]. Flexibility is an issue as well; by allowing a flexible work flow, more than one type of car can be assembled at a time. This allows a flexible catering to customer demand [114].

License plate recognition

Automated license plate recognition involves taking pictures from passing cars. Typically, the automated license plate recognition process consists of two phases: (i) the location of the license plate on the recorded car and (ii) the reading of the license plate. [13]

Recognising license plates is automated for example to assist in speed detection over longer trajectories. This system permits automatic speed monitoring of all cars on a motorway by measuring the elapsed time between entry and departure of the monitored zone. Afterwards the license plates can be automatically read and the registered owners of the speeding cars can be fined. Manual analysis of the pictures of speeding cars is a tedious job.

Automatic surveillance systems

In many places, surveillance cameras record people and their activities. In most cases, no image analysis is performed on the recorded data and all recorded data are stored for a period of time. This includes video streams in which nothing out of the ordinary has happened. If a crime has been committed, all recorded data are retrieved and are searched for indications of this crime.

Recently, research has been done to apply image understanding techniques to such video streams [2]. By storing only 'interesting events', storage capacity and search time for abnormal events can be reduced significantly [101].

Automated egg inspection

In supermarkets eggs are offered in boxes with 6 or 10 eggs. Eggs in one box have approximately the same colour and are relatively clean. This is not because

[2] http://news.thomasnet.com/IMT/archives/2003/08/the_car_factory.html?t=archive

all hens lay perfect eggs, but because quality inspection and sorting steps take place between egg-production and the supermarket. In industrial egg inspection lines, all eggs are selected for their shape and inspected on the presence of dirt, due to excrements, egg yolk or blood. Eggs that are too dirty or odd-shaped are discarded for the consumer market, the remaining eggs are sorted on colour and size. Until recently, this job was performed by human operators.

Today, egg sorting lines have a throughput of around 100.000 eggs per sorting line per hour. This means that a staggering amount of 28 eggs need to be inspected per second, a number too high even for skilled egg inspectors. Therefore, an automated computer vision system for visual quality inspection has been developed and is now successfully used in a commercial setting[3].

1.3 Man-made objects and biological products

In the examples in the previous section, we describe a number of computer vision systems that are implemented and used in real-life situations. Each example has its own characteristics and its own level of complexity. Computer vision systems can be considered from many different viewpoints. We can distinguish roughly two categories of computer vision systems. The first category of computer vision systems deals with the processing of *man-made objects*, while the second category deals with the processing of *biological objects*. For this thesis, we are specifically interested in the inspection of *biological objects*.

Man-made objects typically have the property that each manufactured product in a product category is identical to the previous product in shape and colour. By this we mean that although more types of *e.g.* screws exist, in the category of 'hexagon socket head cap screws'[4], all correctly produced screws look the same. Detection of objects that have a lower quality is typically done by looking at the deviation of the object under inspection from the template object for the quality class.

Biological objects on the other hand have an inherent natural variation. Where man-made objects that are assigned to the same quality class are all similar in shape and colour, biological objects in the same class may differ. The in-class variation – the variation between two arbitrarily chosen objects in the same quality class – for biological objects may seem larger than the between-class variation – the variation between two arbitrary objects in different quality classes. In other words, two biological objects in the same quality class may look less alike than two biological objects in different classes.

To illustrate this point, we look at an example in the world of insects. In Figure 1.1 three insects are displayed. When an image classification algorithm without additional domain information would classify the images in two classes, image (a) and image (b) are likely to be assigned to the same class and image (c) would be added to a different class. This assessment is based on the obvious

[3] http://www.greenvision.wur.nl/documents/folders/up2date/GV05_Handout\%20Visionsysteem \%20voor\%20de\%20uitwendige\%20inspectie\%20van\%20eieren.pdf

[4] http://www.3dcontentcentral.com/ShowModels/MISUMIDATABASE/Hexagon\%20Socket\%20Head\%20Cap\%20Screws/pdf-LB.PDF

(a) (b) (c)

FIGURE 1.1: *Three images of insects. Image (a) depicts a common wasp, image (b) depicts a hoverfly, image (c) depicts a cuckoo wasp. These images were taken from Wikipedia, image (a) is owned by Trounce, (b) by Joaquim Alves Gaspar, and (c) by Kurt Kulac.*

black-and-yellow pattern on the insects and would be a correct classification if the division was to be made on insect colour. However, if the task is to classify the insects in classes that represent insect genera – a dimension in which specific shape aspects are more relevant than colour aspects – the classification proposed above is incorrect. Both the common wasp in image (a) and the cuckoo wasp in image (c) are of the order Hymenoptera, whereas the hoverfly in image (b) is of the order Diptera. When the classification task is considered more closely, other information than general shape and colour plays an important role. Important features are antenna length and the connection between thorax and abdomen. These features result in a classification of image (a) and (c) in the 'wasp' class and image (b) in a different class. For the feature 'colour pattern' the in-class variation for wasps is larger than the between-class variation of wasps and hoverflies.

Biological objects are usually characterised by a combination of non-obvious features. This makes automated inspection more complex than the automated inspection of man-made objects. To perform a correct classification, we need to know which features and feature values of the object determine the class to which the object belongs and which margins are allowed. To make a correct classification, knowledge of the biological objects and their *quality features* is required. Biologically motivated computer vision is a knowledge-intensive task.

1.4 Knowledge-intensive tasks

Automation of routine tasks for identifying man-made objects is relatively easy, in the sense that decision algorithms need to deal with few criteria and small deviations there off. The more knowledge is needed for a correct performance of the task, the harder it is to determine the precise decision criteria. In this respect, biological objects are knowledge-intensive, since the correct inspection of natural objects requires a high level of expert knowledge of the relation between object features and quality of the object.

Our focus on biological objects implies that we selectively target the automation of *knowledge-intensive* computer vision tasks. We can define such tasks as *tasks that need a high level of specialist knowledge for a correct performance of*

the task. Such tasks are generally performed by a task specialist or task expert. These experts are a valuable source of information when automating knowledge-intensive tasks due to a lot of experience in performing their task. Unfortunately, in most cases part of their task knowledge is tacit knowledge. Through praxis, knowledge has frequently been applied for the task and has become the base for the expert's routines [85]. The task knowledge has been internalised and as a consequence, the expert is not consciously aware of this knowledge. Typically, the expert can only partially describe the knowledge needed for a correct performance of the task. In the field of knowledge acquisition, several approaches can be used to explicate as much of this tacit knowledge as possible.

With the focus of this thesis set on knowledge-intensive computer vision tasks for biological objects, we now continue with a brief description of the chosen approach for designing such tasks. First, we look at related work in the field of machine vision and cognitive vision. We show aspects of applying black-box, white-box and grey-box design strategies and choose to use ontology-based white-box design as the basis for the knowledge-intensive computer vision tasks that are the focus of our thesis. This choice leads to a number of research questions.

1.5 Knowledge-intensive computer vision

The subject of knowledge-intensive computer vision is closely related to the fields of machine vision, machine perception, robotics and cognitive vision. Machine perception, a subdiscipline of both artificial intelligence and engineering, focuses on the possibility to interpret aspects of the world based on visually obtained input. Machine vision is the field in which computer vision is applied in industrial settings. Robotics is the discipline that focuses on the design and application of robots. For a robot to interact with the world, it needs to have an explicit representation of task and object knowledge, and the need for 'grounding' the meaning of the recorded data in terms of real physical objects, parameters, and events (the so-called symbol grounding problem) [45]. Cognitive vision is the field aiming at creating practical vision systems that combine vision with reasoning, task-control and learning. One of the main research issues in this field is the impossibility of designing *a priori* a complete model of all possible objects, activities, and actions, and the need to build systems that can deal dynamically with previously unseen image data, characterize previously unseen objects, and address ad hoc tasks, not just ones that have been prespecified [110]. Each of these disciplines deals with complex information that has to be interpreted in real-world terms. Below, we briefly describe the use of semantic information in the four fields related to knowledge-intensive computer vision, so we may use their approach as inspiration for our own work.

When a robot interacts with the world, it 'sees' the object and its environment, it 'thinks' or interprets the recorded scene, and based on this interpretation, the robot 'acts'. The 'thinking' process usually takes place on three

different levels. It uses a level for vision knowledge to segment the recorded scene, an intermediate level with 'semantic' object parts – *i.e.* recognised object features that are grouped together based on domain knowledge instead of on appearance alone – and a reasoning level that helps in interpreting and acting [67, 87]. The intermediate level contains semantic knowledge. This may be phrased explicitly, but often such knowledge is specified in terms of (object oriented) programming structures that are integrated in the programming code [90, 111]. The semantic knowledge can consist of different levels of explicit knowledge. Crevier *et al* [17] categorise the different approaches in their survey on knowledge-intensive computer vision as generic image processing knowledge, image processing knowledge actually in use, generic knowledge and task knowledge.

The generic and task knowledge as specified in [17] describes the world in 'interpretable objects'. Roughly, two kinds of object description are used. The first kind consists of objects specified in terms of visual descriptors [11, 62, 74], resulting in an object representation. An example taken from the paper of Boissard *et al* [10] is the sentence 'a white fly has a rectangularity vector of [0.5 0.6 0.8 0.85]'. In such applications, the object knowledge is closely linked to the image processing features used to identify the described entity.

The second kind of object descriptor uses specifications that corresponds to domain descriptors. Albusac *et al* [2], for example, uses a description of traffic situations that may be encountered in his surveillance system. He uses machine learning to automatically derive knowledge rules that are framed in terms of expert knowledge. An example of such a rule is 'if *object* is not {person, group of people} $\wedge$ *object speed* is {slow, medium} $\wedge$ *sidewalk intersection degree* is {very low, low} then *situation* is normal'. The concepts 'person', 'group of people', or 'slow' are understandable terms for a domain expert.

1.6 Design strategy

To design a computer vision application that performs a task automatically, the application designers have to choose a modelling approach. In this section, we discuss three types of different design philosophies: black-box, white-box and grey-box design.

A black-box method for solving a problem is a method for which the inner workings are hidden from or mysterious to the user and the software developer[5]. The internal workings of the method may be known in a technical sense, but they are not defined in terms that relate to the task domain [24]. Below, we describe two well-known classes of black-box methods: (i) artificial neural networks, a subclass of artificial biological networks, and (ii) genetic algorithms, a subclass of artificial evolutionary algorithms. Neural networks obtain their internal models by learning from multiple cases. Genetic algorithms adapt their internal models to match the data. Both types of methods are frequently used in a computer vision context.

[5] `http://www.merriam-webster.com/dictionary/black+box`

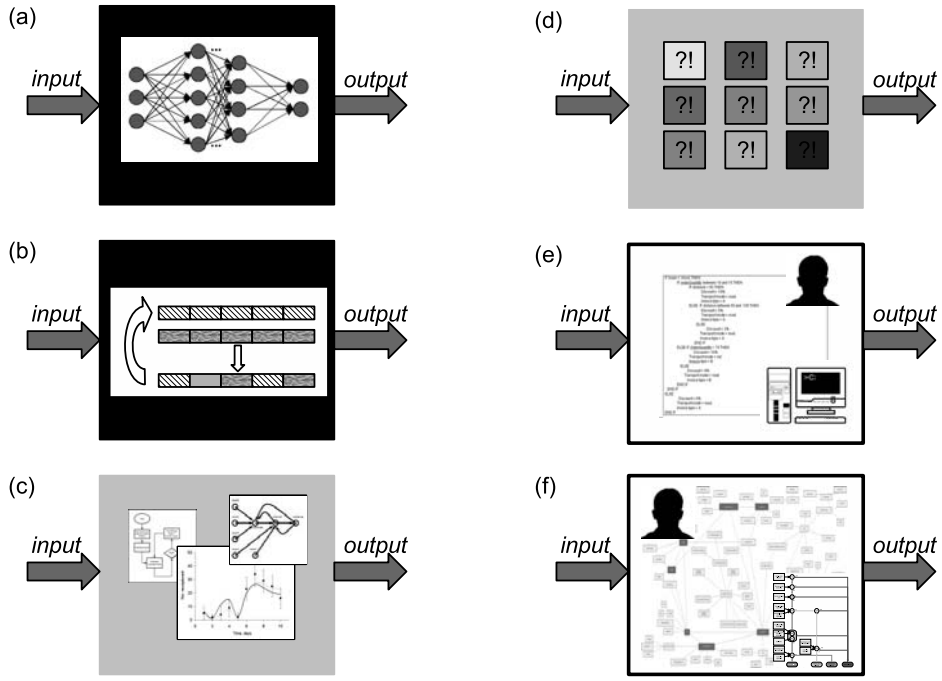


FIGURE 1.2: *A schematic representation of the several types of design methods discussed. Figures (a) and (b) depict an application based on an artificial neural network, respectively on a genetic algorithm. Figures (c) and (d) depict an model fitting approach, respectively a case-based reasoning approach. Figures (e) and (f) represent a white-box design method based on an expert system, respectively an ontology-based system.*

White-box methods have the property that the internal workings of the system are known and transparent. They can be understood in terms of the task domain. The underlying conceptual model is explicitly defined. Crevier and Lepage note that white-box systems in general have the following advantages over black-box systems: knowledge rules can be added easily without modifying existing rules, knowledge can more easily be validated when laid out explicitly than when it is buried into opaque code, and knowledge pertaining to different domains can be stored separately [17]. Below, we discuss two types of white-box methods: (i) rule-based expert systems, and (ii) ontology-based methods. These methods can also be applied to computer vision applications.

Where black-box methods require no formal specification of input, internal workings, or output, and white-box methods require a full specification of these aspects, grey-box methods are in between. Examples of grey-box methods are inductive logic programming (ILP), case-based reasoning methods and model-fitting methods. ILP uses explicit background knowledge to inductively and automatically derive new knowledge [81, 66]. For case-based reasoning methods, related cases to the problem are sought based on explicit knowledge

(white-box), but the actual problem-solving mechanism is black-box. Model-fitting methods are based on a physical, mathematical or statistical model that describes the input data (white-box), but for which the actual parameters need to be found in a black-box way.

In Figure 1.2, a schematic representation of two black-box, two grey-box and two white-box design methods is given. The black-box methods depicted are a neural network approach and a genetic algorithm approach. The grey-box methods shown are based on model fitting respectively case-based reasoning. The white-box methods in the figure are based on expert systems and on ontologies. These six methods are elaborated upon below.

Artificial neural networks (black-box)

An artificial neural network is a black-box method that is loosely based on an understanding of the structure and functioning of the human brain [94]. The brain consists of many neurons that communicate with each other to process information and to control corresponding actions. An artificial neural network is modelled analogously, albeit with less neurons. It consists of interconnected simple computational elements – representing the neurons – that calculate an output model based on the input that is offered. A neural network has to be trained to properly perform a task. This can be done by supervised learning, unsupervised learning or reinforced learning. In each case a function is to be found that maps an instantiation of an input model to the corresponding output model in such a way that the error made is minimised. When an artificial neural network has been trained, it can be used to derive the output for arbitrary input using the function that has been found during the training process.

The class of neural networks that is commonly used is the back-propagation feed-forward multi-layer perceptron [15, 94]. Other classes of neural networks exist as well [44, 71, 107], but are not discussed here. Back-propagation feed-forward multi-layer perceptrons consists of a layer of input neurons, a number of hidden layers of neurons, and a layer of output neurons (see Figure 1.3). Each layer receives information from the previous layer as input, calculates new information and feeds this new information to the next layer (feed-forward). The function that is to be determined in the training procedure consists of finding the optimal set of weights of the interconnections between the nodes. This is performed by applying a backpropagation method, that calculates the difference between the actual output and the desired output. The found error is propagated backwards by using the required output layer weight changes to calculate the required weight changes for the nodes in the previous layer. In this way the neural network is trained and ready for use.

When a neural network is properly designed and trained, it can be very successful in performing a computer vision task [26]. Examples can be found in many areas of computer vision, such as tumor recognition [27, 46], computer chip manufacturing [70], or food classification [12, 22].

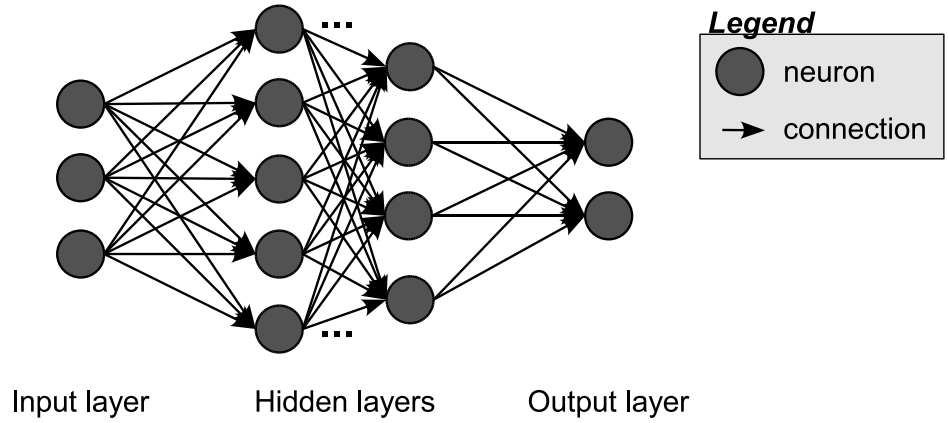


FIGURE 1.3: *A schematic representation of a back-propagation feed-forward multi-layer perceptron. The number of neurons in the input layer is equal to the number of input variables provided for the task, the number of output neurons is determined by the number of desired output values [115]. The number of hidden layers and the number of neurons in each of these layers is a design issue for the neural network engineer.*

Genetic algorithms (black-box)

Genetic algorithms are a second frequently used class of black-box modelling methods. Genetic algorithms are loosely based on the theory of evolution [48]. A genetic algorithm has an 'initial population' of solutions as input. The algorithm allows this population to develop into next generations. For each generation, combinations of selections, mutations and recombinations change the individuals in the populations. For each individual, its fitness is determined. Based on the fitness, individuals are stochastically selected and changed. The goal of the genetic algorithm is to obtain a population of solutions that are sufficiently fit to form solutions to the problem.

Genetic algorithms have been successfully applied to the fields of image segmentation and pattern recognition [24]. Examples include the use of genetic algorithms for tumor detection [41], food quality evaluation [14, 40], or shape reconstruction [23, 47].

Case-based reasoning methods (grey-box)

Case-based reasoning methods are problem-solving mechanisms that use the explicitly defined specific knowledge of previously encountered problems. New problems are solved by identifying similar past cases and reusing the corresponding solution in the new situation [1]. This process is usually performed in a cyclic process consisting of four steps: retrieve, reuse, revise, retain [113]. The first step searches for past cases that are potentially of interest for solving the problem. Past cases are stored as a problem-answer set annotated with terms

that allow the case-based reasoning system to assess the applicability of the case. A retrieved case is used to solve the problem by providing the answer of the case as the answer to the new problem. Unless the case is very similar to the problem at hand, the solution that is provided by this case has to be revised in a domain-specific way. Adaptation can use various techniques, including rules or further case-based reasoning on the finer grained aspects of the case [75]. The problem, the tested cases and their success rate in solving the problem are stored for future use.

Case-based reasoning is a technology that is applied in a growing number of applications. In the field of computer vision, case-based reasoning is used as well [89]. Applications can be found in *e.g.* rail inspection [52], brain/liquor segmentation [88], or soil analysis [55].

Model-fitting methods (grey-box)

In many cases, a system behaves according to a well-known model. An example is the growth of micro-organisms on food. This growth can be modelled using an S-curve. The model can be explained in terms of the behaviour of the micro-organisms. At first, the micro-organisms go through a lag-phase. During this phase, the cells are adjusting to the specific growth conditions present in the new environment. Next, the exponential phase is entered. The micro-organisms start to divide and grow exponentially. At a certain point, the growth is slowed down due to *e.g.* competition between cells, dying cells and increasing toxicity, until a stationary phase is entered [93]. In such data modelling tasks, the shape of the model is known beforehand. The parameters of the model, however, are not known. The lag phase may not be noticeable in the data, the food may not be sufficient for the group of micro-organisms to enter the exponential phase. Statistical methods are needed to find the best fitting model for the measured data to optimally reflect the actual behaviour of the colony of micro-organisms.

The model describing a general scenario for a certain task can be given by a mathematical, a physical-chemical, or *e.g.* a biological model, and can therefore be considered to be white-box. The fitting of the data to the scenario, or in other words, the identification of the specific instance of the model for the actual situation, is a statistical process and is black-box. Therefore, we consider the fitting of models as a grey-box method.

Computer vision is used in combination with model-fitting methods to semi-automatically estimate model parameters and analyse the behaviour of the observed system. Examples are the monitoring of embryonic cell growth [84], of bacterial morphology [69] or bacterial community structure [82].

Expert systems (white-box)

Expert systems aim to use explicated human task-specific knowledge for advising users of the system and explaining the logic behind the advice [106]. Rule-based expert systems represent the human knowledge in the form of rules [25].

These rules can be used to infer new facts from existing facts [68]. Many expert systems were developed for commercial application in the 1980s [28, 76].

Expert systems are based on rules that have been obtained from human experts. Whereas human experts can use their common sense, expert systems can only work with the rules that have been defined. Due to the intertwining of the knowledge rules with the software, software based on expert systems is hard to maintain.

Applications of expert systems can be found in fields such as medicine, mortgage advice and human resource management. Several computer vision applications employing expert systems have been described as well [78]. Examples include segmentation and enhancing of video images [6], weld assessment [97], and fire detection [32].

Ontology-based methods (white-box)

A second type of white-box methods is based on applying ontologies. For ontology-based design, the domain and task knowledge is modelled using an ontology. In the context of computer and information sciences, an *ontology* is defined as a set of classes, attributes, and relations among class members with which a domain of knowledge is modelled [39].

Ontologies are currently obtaining massive attention from computer scientists working on the next generation of the internet, the Semantic Web [4, 7]. They enable clear communication and common understanding of information among people or software agents by formally defining the vocabulary of one or more specific domains. Ontologies are useful models to encode domain knowledge in a formal, machine-readable way and hence can be used in computer vision applications.

Ontology-based systems are different from expert systems in the sense that the semantics of the knowledge is explicitly defined. Not only rules, but also descriptive domain knowledge is expressed, the knowledge is stored separately from the implementation, and the knowledge can be used for other tasks than defining rules and reasoning. In this thesis, we show how ontology-based design can be applied to automate knowledge-intensive tasks.

For automating knowledge-intensive tasks, we have to make a decision concerning the design method of choice. The criteria that we use for this decision are the following. First, we require a system to be accurate and fast enough for use in practice. Second, it is important that the computer vision system is trusted by the end users. Third, we aim for a system that can be changed locally to correct for mistakes or to adapt it to different situations. Finally, it would be preferable if the system can be used to support organisational learning.

The first criterion can be met by white-, black- or greybox systems alike. We believe that the second criterion is best met when the system is set up in a transparent way, such that its function and its output is understandable by the end users. This implies that white- or grey-box systems are preferable over black-box systems. The third criterion asks for a flexible system in which local

changes can be made relatively easy. Again, white- and grey-box systems meet this criterion best. Supporting organisational learning, the last criterion, implies that the level of human expertise can be increased by explaining the underlying reasoning of the system, not just the outcome of the system.

Based on these criteria, we opt for a design method that is fully white-box. More specifically, we choose for an ontology-based design method in which the domain expert's knowledge is explicitly modelled and used in the design of the computer vision system.

1.7 Research questions

The main research question of this thesis is *Is an ontology-based white-box design method suitable for designing knowledge-intensive computer vision systems for biological objects?* To answer this research question, we have formulated the following subquestions, each dealing with an aspect of the problem:

1. What are the characteristics of knowledge-intensive tasks? In which cases is automation opportune? What is special about automating *knowledge-intensive* tasks?
2. What are possible approaches for automating knowledge-intensive tasks? What are the benefits of the chosen approach?
3. How can we obtain the knowledge that is relevant for successfully automating the task?
4. How can we systematically decide on the right level of transparency in the design of knowledge-intensive computer vision applications?
5. How well does the proposed method perform in terms of a number of predefined criteria?
6. How applicable is the developed method to the automation of other knowledge-intensive computer vision tasks?

In the subsequent chapters of this thesis, we attempt to answer these research questions and describe a method to design knowledge-intensive computer vision systems. Before we proceed, though, it is necessary to set the range of tasks and domains that are covered by this thesis. The scope of this thesis is set by the following observations and limitations.

- We constrain ourselves to tasks which are to be automated with a computer vision system. These tasks are typically aimed at quality assessment, quality inspection, or quality-based sorting of products.
- The tasks of interest are knowledge-intensive tasks that are at present executed by task experts. Task experts are individuals who are well-versed in performing the specified task in a certain domain. They have acquired detailed expertise that is relevant for their task. This knowledge is used by them to assess the quality of the object under inspection.

- The focus of our research is directed at biological objects instead of man-made objects.

1.8 Main contributions

The main contributions of this thesis to the field of knowledge-intensive computer vision are the following. First, we have developed an ontology-based method to systematically design transparent computer vision applications to perform knowledge-intensive inspection tasks. In short, this method consists of decomposing the inspection task into subtasks and explicitly formulating for each of the subtasks the expert knowledge required in the form of an ontology. In this process, software engineers and knowledge engineers form an interdisciplinary team to obtain a white-box ontology-based computer vision system.

Second, we argue that internal transparency of procedural knowledge makes it easier to adapt a computer vision system to new conditions and to diagnose faulty behaviour. At the same time, explicitness comes at a price and is always bounded by practical considerations. Therefore we introduce for the software engineer a method to make a balanced decision between transparency and opaqueness. The method provides a set of pragmatic objectives and decision criteria to decide on each level of a task's decomposition whether more transparency makes sense or whether delegation to a black-box component is acceptable.

Third, we provide the knowledge engineer with a method to acquire explicit expert knowledge. We propose an interview and observation based method to obtain a task-specific, multi-domain, multi-expert knowledge model. With this method, individual experts are interviewed, the individual knowledge models are merged, an outsider's perspective is added and the merged model is presented to the interviewed experts in a joint teach-back session. We complement the interview-based method with the ROC-method. We have developed this method to allow domain experts to take a more prominent and active role in the knowledge modelling process. The ROC-component supports the domain expert in identifying relevant knowledge, for ROC incorporates a prompting process that offers the domain engineer terms associated with the terms that he has selected in earlier iterations. In this way, the domain expert has a better opportunity of covering all aspects of his knowledge. For the prompting process, existing triple-based knowledge sources are used. Thereby, we benefit from already existing sources in our ontology engineering process.

In this thesis we use a case study in the horticultural sector to illustrate our work on automating knowledge-intensive tasks. In the next section we describe the context of the case study, the various viewpoints on the quality inspection task, and the complexity of seedling sorting.

1.9 A case study in seedling inspection

1.9.1 Seedling inspection in horticulture

Our case study is centered around seedling inspection in horticulture in the Netherlands. This task is performed to predict the amount of high quality fruits or vegetables that the mature plant will produce. The goal is to maximise the probability for a high yield of the whole crop. The quality assessment is performed by seed growers and plant nurseries. Seed growers produce seeds and sell these to plant nurseries. Plant nurseries grow the seeds until the young plants are viable enough to be sold to a vegetable grower.

In seed growers' companies, a sample from a batch of produced seeds is taken and the seeds are grown. When the plants are in the seedling stage, they are inspected and the inspection results are used (i) to determine the quality and price of the produced seed batch and (ii) to provide feedback to the seed breeding department for the development of future seed batches. The produced seed batch is marked with the determined quality assessment and is sold to plant nurseries. It is important that the quality indication matches the actual growth of the seedlings; if the deviation is too large, the buyer will file a claim for compensation with the seed grower.

Plant nurseries obtain seeds from the seed grower to grow them until the young plants are large enough to be delivered to a vegetable grower. Plant nurseries have to make sure that enough plants are grown for the grower; they use the quality assessment of the seed growers to determine the number of plants that are to be produced. When the plants have reached the seedling stage, a quality assessment is performed to sort the batch of grown seedlings into quality classes. The best plants are sold to the grower, the plants of lowest quality are discarded. Again, it is important that the seedlings are correctly assessed, since this assessment determines which plants are viable enough to result in a high yield crop for the vegetable grower.

1.9.2 The inspection process

Quality inspection of seedlings is a complex task. Seedlings occur in various shapes and sizes. To predict the yield of a mature plant, a set of heuristically validated quality criteria is used. Some criteria are simple and concern *e.g.* leaf area, stem length, and leaf curvature. Other criteria deal with more complex issues such as the likelihood that a plant is budless, or the regularity of the leaf shape.

To illustrate the diversity of occurring seedlings and the broad range of quality features, we show in Figure 1.4 several examples. The depicted plants represent only a small sample of possible variations. Plant (a) is a typical example of a plant of good quality, since the plant has two cotyledons and two true leaves that form more or less a cross. The area of the true leaves is approximately equal to the area of the cotyledons and the cotyledons originate from the stem on the same height at opposite sides. Plant (b) is also a plant with two seed lobes and two true leaves, but the area of the true leaves is sig-

FIGURE 1.4: *A small sample of seedlings to be assessed on their quality. All ten displayed seedlings are equally old but vary in quality.*

nificantly smaller. Plant (c) has three cotyledons instead of two, which implies a larger probability to become budless. Plant (d) is a 'Siamese twin'; on one stem, two buds have been growing, each developing its own cotyledons. Plant (e) is a plant with wrinkled cotyledons, causing them to have a different shape than expected. Plant (f) has not discarded its seed well enough: the tips of the cotyledons are still stuck within the seed coat. For plant (g) the same observation holds. Plant (h) has a disfigured cotyledon, as has plant (i). Plant (j) is missing part of a cotyledon.

In horticulture, each nursery company is specialised in producing seedlings selected for the needs of their customers. Some specialise in growing plants for the Northern market, others cater to the Southern European market, some produce seedlings suited for grafting, others are interested in seedlings for tomato production, et cetera. The different business foci result in correspondingly different quality inspection rules, and hence in different ways to perform the quality assessment process. Depending on the specific business focus, plants (b) to (j) may be classified as first choice, second choice, too small, or even as abnormal plants.

At present, quality assessment is performed by highly-trained experts. They use their knowledge to assess seedlings on all relevant aspects. However, task experts are only human, and therefore their assessment is not as consistent as desired. For example, the expert assessment depends on the quality of the seedlings that were inspected just prior to the current batch. Moreover, after a while, experts show signs of fatigue and sloppiness. Finally, we note that even though each expert is intensely trained by the company in which he is employed, the assessment of experts within one company will typically differ up to ten percent. This is in part caused by the fact that the quality inspection knowledge is only partly explicitly defined. The horticultural companies indicated that they use the small set of eight classification rules specified by Naktuinbouw [83], whereas from interviews with experts we found that over 60 quality determination rules are actually used.

To obtain a consistent quality assessment, the horticultural sector is interested in automating the inspection process. Individual companies in the horticultural sector have already invested in the process of automating the seedling sorting process. These first efforts have only been partially successful. Three visual inspection systems are commercially available. One of these takes a picture of a tray of very young seedlings to give an estimation of the average cotyledon area. This feature is used as a rough quality indication. A second system looks at individual seedlings. Two cameras – one for a view from above, one from the side – are used to determine the cotyledon area. The third existing machine is better attuned to the seedling sorting reality, since it can not only determine cotyledon area but also true leaf area.

Neither of these computer vision systems is accurate enough to inspect the seedlings as well as the experts. The existing computer vision systems are hampered by two problems: (i) the lack of detailed expert knowledge and (ii) the use of a too simplified model of the plant. The existing systems only look at leaf area as a quality indicator. The other more detailed quality features that are used by the experts are disregarded by the machine. A second simplification in the inspection systems is the use of 2D images of the plant. The projection causes an imprecise view of the plant under inspection. We show in this thesis how these limitations can be overcome by using 3D image analysis and introducing a computer vision approach that is based on explicit expert knowledge. Still, the 3D computer vision system may experience differences between two recordings of the same batch of seedlings. These differences caused by the image acquisition phase will impact the subtasks that follow. It is important that the computer vision system is robust enough to deal with such differences.

In this thesis, we consider a computer vision system to be successful when its decisions are acceptable for the domain experts. Since two experts do not always obtain the same answers when assessing an object, we allow for a deviation between the computer vision system and a task expert that is at most as large as the inter-expert variability. Moreover, the quality assessment that is assigned to an object by an expert may differ from the 'scientifically sound quality' of that object. To survive, each company needs to specialise on specific quality requirements set by their customers. Therefore, each company is focused on specific characteristics of the seeds or seedlings. The quality assessment procedure partly depends on the business focus of the company and hence differs per company. We do not concern ourselves with the notion of objective quality, but consider the quality assessment made by the task experts as valid.

1.9.3 The roles in knowledge-intensive computer vision design

In the design of knowledge-intensive computer vision tasks, four different roles are involved. Since these roles are used throughout the whole thesis, we briefly introduce them here.

Domain expert, task expert, or task specialist. The person that fulfills this role is an expert with respect to the task and domain for which the computer

vision application is developed. For the case study, this person would be an expert in the seedling inspection domain. He has detailed knowledge about the plant domain and the seedling assessment process.

Knowledge engineer. The person that fulfills this role is schooled in the field of knowledge management. He has interviewing skills, and can create knowledge models such as ontologies. Moreover, the expression of procedural knowledge in declarative statements is also done by the knowledge engineer.

Software designer: computer-vision specialist, computer-vision engineer. The software designer creates the computer vision application. He has sufficient knowledge to implement a computer vision application.

Problem owner. The commissioner of the application. In the horticultural case study, the problem owners are the seed growers and the plant nurseries.

1.10 This thesis

To automate a knowledge-intensive task using image analysis, several choices have to be made. In Chapter 2 we focus on the design choices that are to be made for a successful automation of such a task. We create a framework consisting of subtasks and intermediate models. The required knowledge models are touched upon in this chapter, but the detailed description of creating application-specific knowledge models is elaborated upon in Chapter 3. We illustrate the proposed design method in this chapter by applying it to the seedling inspection task from the horticultural case study.

Chapter 3 shows a method to obtain a formalised version of the task knowledge in the form of application ontologies. To this end, traditional interview-based knowledge acquisition is enriched with a reuse-based ontology construction component. This component re-uses existing knowledge from general sources and supports the expert in defining concepts and relations that are relevant for the task. It also increases the role of the experts in providing explicit task knowledge.

We describe in Chapter 4 a method to support a software engineer to make a balanced decision on the optimal level of transparency in the computer-vision application. We propose a set of criteria to facilitate this decision and show how this can be elaborated in our case study.

In Chapter 5, we show how well the developed method performs in a prototype application for the case study. We focus in Chapter 6 on the predicted advantages of the proposed implementation of the knowledge-intensive computer vision task. Chapter 7 contains a discussion and our conclusions.

Chapter 2

Knowledge-Driven Computer Vision: Designing the Framework

In this chapter, we propose a method to systematically design computer vision applications that perform knowledge-intensive inspection tasks. The novelty of our work is the explicit use of expert knowledge in the form of application ontologies in the design of a knowledge-intensive computer vision system. We show how the use of expert knowledge influences the design of the computer vision system. We illustrate the proposed method by designing the computer vision framework for the task in the case study. We show how the task can be broken down into subtasks that are interspersed with knowledge models. A knowledge model, in this thesis, is defined as a set of instances of formal concepts that describe a domain. This chapter is based on a paper [60] published in the International Journal of Human Computer Studies in 2006.

2.1 Properties of an ontology-based computer vision system

Knowledge-intensive computer vision tasks are currently typically performed by highly trained experts. We aim to design an ontology-based computer vision application that can perform the task as well as these experts. To ensure its practical applicability, it should – as any computer vision system – have at least the following two properties:

Correctness

The application should assess the objects correctly. If the computer vision application makes too many errors, it will not be used in practice. The maximum acceptable percentage of errors is to be determined by the companies.

Speed

The application should have a high enough speed to be of economic interest. When this is not the case, the application will not be used in practice.

Besides these two properties, the proposed ontology-based design of the computer vision system also facilitates the following aspects.

Expert acceptance

The application should produce results that are acceptable for the task experts [21]. When the assessment of the objects under inspection by the application is the same as the assessment by the experts, the expert will have no difficulty in accepting the results of the application. However, since the inter-expert variability to assess objects such as seedlings is 5 to 10% according to the companies, it is reasonable to expect that the application will occasionally generate a response that is different from the initial assessment of an expert. This should not be a problem, as long as the system can explain to the expert in his own terminology why – and based on what – a decision was made. That way, the expert can understand the reasoning of the system. He can concur that the interpretation of the system is reasonable or can indicate to the application developers which part of the reasoning structure is flawed. It is our expectation that this property helps in building the expert's confidence in the application.

Corrigibility

Flaws should be easy to track down in the automated inspection application. When the automated application makes an incorrect quality assessment, it should be possible to identify the cases incorrectly assessed by the application and to correct the application accordingly.

We stress that this property of the computer vision system only guarantees the easy and accurate pinpointing of flaws in the implementation phase; actually changing the algorithms and making the system perform better may still be a difficult task.

Adaptability

Additional features for assessing the objects under inspection should be easy to implement. When additional features are defined by the task experts, it should be possible to implement these features in the application in an easy way. The advantage of the adaptability property is that the system can be easily adapted to different yet similar objects or to changing quality rules.

Robustness

The application should be reliable with respect to both frequently and rarely occurring objects. As long as an object lies within the scope of the task and domain, the

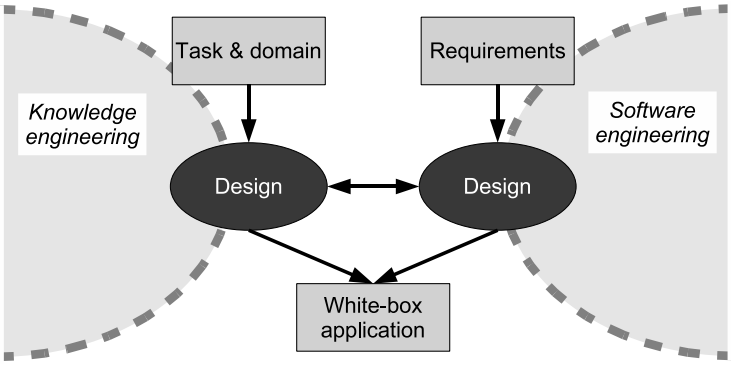


FIGURE 2.1: *The software engineering and knowledge modelling task interact at the design level. Tasks on the left-hand side are mostly in the knowledge engineering domain, tasks on the right-hand side are mostly in the software engineering domain.*

application should correctly assess objects that it has never been seen before, even when such an object does not look like any other object that has ever been offered to the application. This property is important, since it is often difficult to obtain a data set that contains all possible quality variations of the objects to be inspected. Correctly assigning less-frequently occurring objects to the correct quality class increases the overall performance of the quality prediction.

Reliability

The application should be able to indicate when an object lies outside its area of expertise. In some cases objects may be offered to the application that are outside the scope of the application. Instead of blindly assessing such an object, it would be preferable if the system would reject the object.

2.2 The design of the computer vision application

In Chapter 1, we have discussed the differences between black-box, grey-box and white-box design. To design a knowledge-intensive computer vision application that performs well and has the properties described above, we submit that an ontology-based white-box approach can be applied. The main characteristic of such a method is that we explicitly use formalised expert knowledge to make the computer vision application as transparent as possible. The expert knowledge is leading in the design of the computer vision system.

In this chapter we propose a method to systematically design ontology-based computer vision applications. We note that both software engineering and knowledge engineering are important fields for the design of an ontology-based computer vision method (see Figure 2.1).

The methodology that we propose for designing ontology-based computer vision applications is depicted in Figure 2.2. The application design process

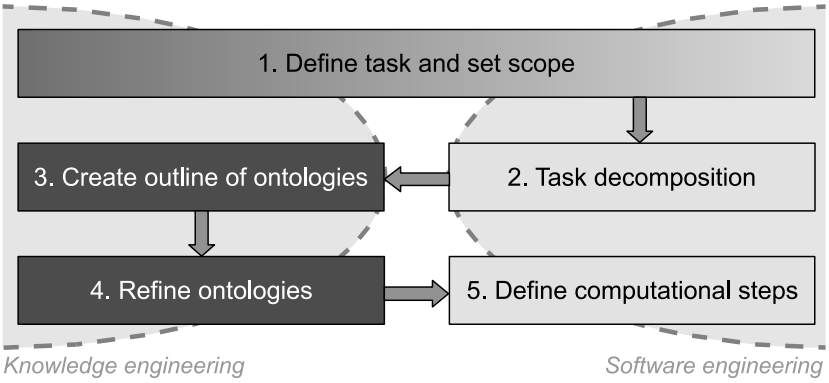


FIGURE 2.2: *A schematic representation of the design process.*

starts with the specification of the task and the domain in which the task resides to set the *scope* for the software application. When modelling expert knowledge, the task and domain specification are used to set the scope of the knowledge to be modelled. Next, the task is broken down into *subtasks*. These subtasks mimic the expert's process of performing the task. Each subtask transforms an input model of the object into an output model of the object. These models are described using *application ontologies* that can be defined between the subtasks. The third step is to define a rough setup of these application ontologies. Next, these ontologies are *refined* to ensure that they contain all relevant knowledge for successfully performing the task. This ontology construction process is described in Chapter 3. Finally, we note that the design of the ontologies has an influence on the *computational steps* that represent the internal mechanism of each subtask. We specify these steps accordingly.

2.3 Specifying the task and domain, and setting the scope

After performing a requirement analysis in which the stakeholders and their requirements are determined, the first step is to determine the limitations that are set by the scope of the task. This aspect is used to identify for which area of expertise the knowledge model has to be developed.

The observation that the scope of an application is an important factor in the design of a computer vision application is not new. Annett *et al* [3] proposed in 1971 the method of hierarchical task analysis, as a means of describing a system in terms of its goals. In the software engineering field, requirements specification is a standard part of the design process [109]. The specification of requirements is aimed at getting a complete specification of the task and the limitations set by the environment in which the application should function. Several authors who present a computer vision application, remark that their application is intended for a specific task. Examples are Jeliński *et al* [53] who developed a computer vision application for cheese inspection, Bhandarkar *et*

al [8] who developed an application for fractured bone alignment, or Llano *et al* [72] who developed an application to automatically classify DNA. However, although the task is usually mentioned, the boundaries of the domain of interest are often not made explicit. We believe that an explicit scope definition is important both for the specification of the ontologies, and for a transparent foundation of the choices that are made in the computer vision framework. Therefore, we present the specification of the task, its domain and its scope as a standard part of the proposed design method.

Setting the scope of the application consists of two parts. First, the scope of the application should indicate what task must be executed. Second, the boundaries with respect to the task and domain have to be set. Examples of limitations placed by the scope of a computer vision task are listed below.

- The scope determines the type of objects that are to be recorded.
- The scope could indicate the type of features that should be detected. One could think of colour features, shape features, texture features, features describing the components in the object, *et cetera*.
- The scope could set conditions on the size of the object parts to be recognised. In some cases *e.g.* object parts should be recognised that are visible with the naked eye, in other cases microscopic features should be detected.

By setting the scope of the task, the domain expert and problem owner have put limitations on the computer vision application. These limitations have to be taken into account in the design process of the computer vision system. Based on the scope, the computer vision experts can decide whether the task requires a representation in pixels/voxels, points, height maps, *etc*. Another decision that is made based on the scope is the choice of the image acquisition technique. In some cases laser range imaging would be suitable, in other cases simple 2D imaging, stereovision or volumetric intersection would be required. The scope also influences the wavelength that is required in the recordings. Moreover, the scope helps to determine which expert knowledge is relevant and which is not. This helps in the ontology specification and the process to define the computational steps.

For the case study at hand, the task that has to be executed by the computer vision system is to *determine the quality of tomato seedlings*. The following observations from the task experts determine the scope of this task. Note that these observations set both the scope for the manual seedling inspection task as for the automated task.

- The quality of the seedlings predicts the yield of the seedling as an adult plant in terms of tomatoes.
- The quality of the seedlings is assessed when they are approximately 12 days of age.
- The quality of the seedlings is determined solely by looking at external properties of the plant. Quality aspects that are only observable by looking at internal properties of the plant are not taken into account. Of the

external plant features, we only take those into account that are visible with the naked eye.
- Only visible plant aspects that are related to the shape of the plant are taken into account.
- We focus on the quality assessment of tomato plants in general, we are not interested in cultivar specific quality aspects.
- The quality assessment of a seedling is always executed relative to the quality of the tray of plants from which the seedling is taken.

These observations have an impact on the design of the computer vision application.

2.4 Task decomposition

With the scope and the task specified, the task can be broken down into subtasks. Task decomposition occurs in many different areas: project management [77, 104], building projects [65], and modularisation in software engineering [109]. A frequently used method is hierarchical task analysis that breaks down the operation at hand into sub-operations [3]. In this thesis, we base ourselves on the description given by Schreiber *et al* in the CommonKADS methodology [95]. They describe a method for systematic breakdown of a task into subtasks in the field of knowledge engineering. They indicate that decomposition of a task in subtasks results in a number of consecutive elementary steps, each with their own explicit input and output knowledge and an associated person or software system that performs the subtask.

To obtain a white-box design, we aim to decompose the task into subtasks in such a way that the computer vision expert's way of performing the task is followed closely[1]. An expert looks at an object, interprets it and makes an assessment of this object. Since we want to be able to explain to the expert in his own terminology why a certain decision is made by the computer vision application, we follow the strategy of observation-interpretation-assessment in the design of the computer vision application. This division results in the following three subtasks:

- **The 'see' or 'image acquisition' part**: the computer vision application 'sees' the object by making a recording and an internal representation of it. Obviously, without a recording of the object, the image analysis application has no internal representation of the object and cannot continue with inspecting and assessing the object.

- **The 'interpretation' or 'segmentation' part**: the assessment task can only be executed when the image analysis application knows what the recording signifies. In other words, the assessment task has to know what the object is that is represented by the recording and what its pose, its relevant

[1]Note that we do not presume to model the exact thought processes of the expert. Instead, we follow the description that the expert gives of his task.

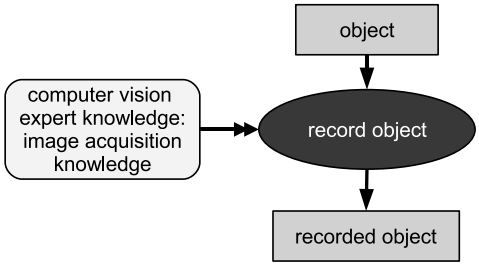


FIGURE 2.3: *The decomposition of the image acquisition task in its subtasks.*

parts *et cetera* are. Segmentation is the task that divides the representation of the acquired object in regions that belong together, *i.e.* form an object part. The first part of the 'thinking' process takes place in this subtask

- **The 'assessment' or 'classification' part**: the part of the computer vision application that mimics the assessment performed by the task expert. In this task relevant characteristics of the object under inspection are determined and the recorded object is assessed. The 'thinking' is completed with this task. The findings of the application can be communicated to the machine that 'acts' on the decision made in the assessment task.

The knowledge that the domain expert formulates will typically focus on the assessment of the object under inspection. For the segmentation and image acquisition subtasks expert knowledge from computer vision experts is required.

2.4.1 Image acquisition

The image acquisition task makes a recording of the object under inspection in the real world to create a representation of the object in the computer vision application. The scope of the task determines the type of recording that is required. This has an influence on the choice for a recording method, the required precision of the recording and the allowed signal-to-noise ratio. The expert knowledge needed for this task is delivered by a computer vision expert. This knowledge entails recording techniques, recording devices, lighting methods, calibration procedures *et cetera*. In Figure 2.3, the image acquisition task is schematically depicted.

For the case study, the scope sets limitations on the image acquisition method. Three of the aforementioned observations are relevant for the image acquisition task: (i) the seedlings are 12 days of age; this gives an indication of the size of the plant under inspection, (ii) only plant features that are visible with the naked eye are to be detected (iii) only shape-related quality aspects are taken into account. These observations require an image acquisition method that can record images of approximately 10 cm of height in a resolution with details of approximately 1 mm^2 of size. Moreover, the resulting plant model of the image acquisition method should be geometric instead of *e.g.* colorimetric. Based on these observations, we have chosen an image acquisition method that

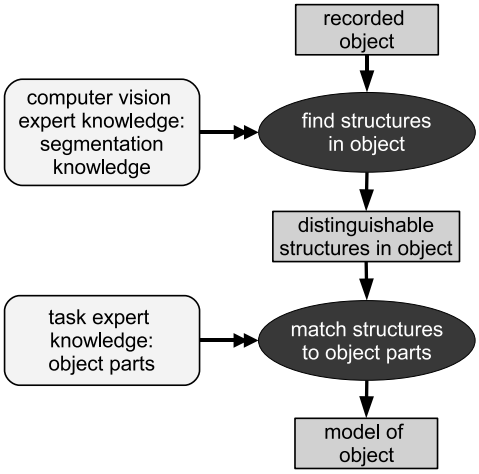


FIGURE 2.4: *The decomposition of the image segmentation task (top row) and image analysis & decision task (bottom row) in its subtasks.*

uses volumetric intersection [79] to obtain 3D point clouds that represent the seedlings.

Image acquisition is a process that is needed to obtain point clouds for further processing. However, we do not consider the image acquisition process itself as part of this thesis. Therefore, we have chosen to use an existing device to generate a point cloud for each seedling based on a set of 24 colour images. The representation of the seedling is given as a 3D point cloud. In this thesis, we do not discuss the internal workings of the image acquisition task.

2.4.2 Segmentation and analysis

When a knowledge-intensive task is executed by a task expert, the object under inspection is *known and recognised* by the expert. When a recorded object is present in a computer vision system, it could be a recording of an arbitrary object. Therefore, one of the tasks that has to be executed by the computer vision system is to study the recorded image, divide it into meaningful regions and interpret its findings.

To ensure that the result of this task can be used by the quality determination task, we need to know which object parts are part of the task domain. These parts are specified by the task expert. To automatically determine the objects parts from the input model, the aforementioned task needs computer vision knowledge. Hence, a computer vision expert is involved to specify the required knowledge.

In Figure 2.4, we see that the knowledge of the task expert and of the computer vision expert are both used to successfully interpret the recorded object. Note that the computer vision expert can only select the correct segmentation

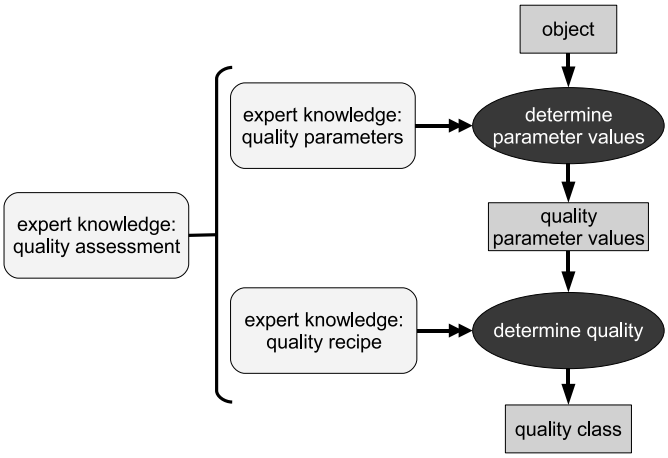


FIGURE 2.5: *The decomposition of the expert knowledge into parts and the corresponding subtasks.*

knowledge if (i) the scope of the computer vision task is known and (ii) the required input for the classification task is known.

For our case study, the domain expert specifies the plant parts that are of interest for the quality inspection task. Plant parts that are mentioned (and modelled in the plant ontology) are Stem, Plug, True leaf, Cotyledon, *etc*. The computer vision expert provides 'structures' – geometrical shapes – that match these plant parts. For the case study, these structures are Thin cylinder, Thick cylinder, Paraboloid and Surface (see the geometry ontology specified in Chapter 5).

2.4.3 Decisions

The decision task performs the actual assessment of the object. This task can be divided into two subtasks:

- Quantifying the *quality parameters*. The *quality parameters* are the characteristics of the object that are relevant for the quality assessment.
- Applying the *quality rules*; the rule structure that determines the quality indication of the object under inspection. Individual rules using the measured quality parameters are specified by the task expert.

These two tasks form the decomposition of the decision process. This decomposition is displayed in Figure 2.5.

For the case study, we note that the scope sets us some limitations: (i) the quality measures have to predict the yield of the adult plant, (ii) quality assessment is of tomato plants in general, not of cultivar specific quality aspects, (iii) quality assessment of a seedling is always executed with respect to the quality of the tray of plants from which the seedling is taken. These limitations guide the domain expert in specifying the quality rules.

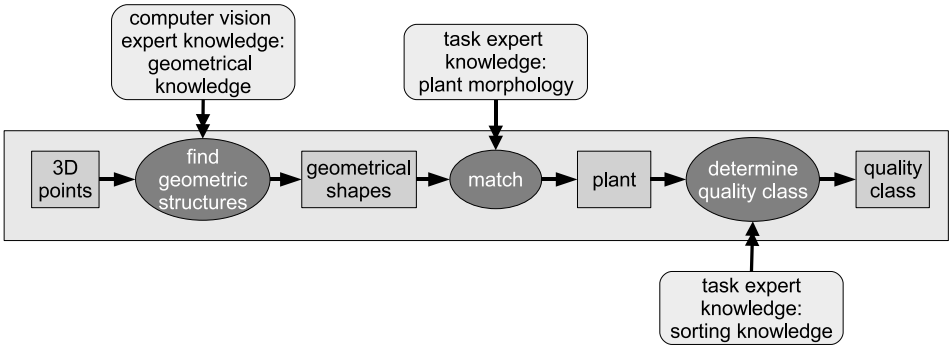


FIGURE 2.6: *The decomposition of the segmentation and classification task in its subtasks elaborated for the horticultural case study.*

The overall task decomposition of the seedling inspection task results in the framework as depicted in Figure 2.6. We see that the image segmentation task starts with a subtask that fits geometrical shapes to point groups. The geometrical shapes are matched to plant parts to form a model of the plant. The quality determination tasks consists of determining the value of the relevant parameters and deciding on the correct quality class based on the value of the combined quality inspection rules.

2.5 Specifying the subtasks

In the previous section, we have defined the decomposition of the knowledge-intensive task into subtasks interspersed with knowledge models. Each of these subtasks can be represented by a sequence of components that perform a clearly defined part of a task [103] – a so-called computational workflow. In this section, we define the workflows that are used for the case study. These workflows are based on concepts from the application ontologies as specified in Chapter 3.

Workflows and workflow management systems have recently gained much attention, specifically in the field of bioinformatics. Bioinformatics researchers frequently use workflows to define the research protocols used to repeatedly perform a specific scientific analysis [103]. Such workflows are generally composed of Web Services [63]. Web services are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes. [29].

The concept of workflows consisting of modular applications is useful in the context of computer vision applications, but invoking such modular components over the Internet may not be feasible due to speed requirements. At its most abstract level, though, all workflows are a series of functional units – components, tasks, jobs or services – and the dependencies between them which define the order in which the units must be executed [20]. The functional

units need not be *Web* services. For the method that we propose, we assume that computational workflows consist of *services*: self-contained modules that perform a specific job as part of the subtask workflow.

To specify workflows in the context of knowledge-intensive computer vision applications, we explicitly identify for each service in the computational workflow (i) the input concepts, (ii) the output concepts, (iii) a human readable (high level) description of the service. By explicitly describing the services in this way, corrigibility, adaptability and acceptability are made possible, as we show in Chapter 6. In the next subsections, we specify the workflows for the subtasks in the case study.

2.5.1 Find structures in object

The first part of the seedling inspection task is to find geometrical shapes in the recorded 3D point cloud to assist the segmentation algorithm in creating the plant model. Colloquially speaking, the plant under inspection consists of a cylindrical plug, possibly with a paraboloid-shaped cap consisting of vermiculite, from which a thin long cylindrical stem grows. Attached to this thin cylindrical stem, one or more surfaces – normally known as leaves – can be found. To find the thick cylinder, the paraboloid, the thin cylinder and the surfaces, we use the workflow represented in Figure 2.7 to infer all information from the 3D point cloud in the point ontology. Moreover, the actual mapping between instances of regions from the point application ontology onto instances of concepts in the geometrical application ontology takes place. Not all concepts in the point application ontology are used in the mapping process; only the instances of the regions are mapped to concepts in the geometry ontology. A 'Linelike region' is mapped to an instance of 'Thin cylinder' and the 'Planar regions' are mapped onto either 'Thick cylinder', 'Paraboloid' or 'Surfaces'.

The services that compose the workflow are described next.

Create 'Point groups'

- Input: All instances of the class 'Point'.
- Output: An equal number of newly created instances of the class 'Point group'.
- Description: For each instance of the class 'Point' an instance of the class 'Point group' is created, of which the 'Point' is the 'central point' and for which the 'neighbours' are determined.

Determine point type

- Input: An instance of the class 'Point' with no value for 'type'.
- Output: An instance of the class 'Point' with a value for 'type'.
- Description: For each instance of the class 'Point' its 'type' is determined. If the 'Point group' of which the point instance is the central point is a planar area, then the 'type' is set to 'planar'. If the 'Point group' of which

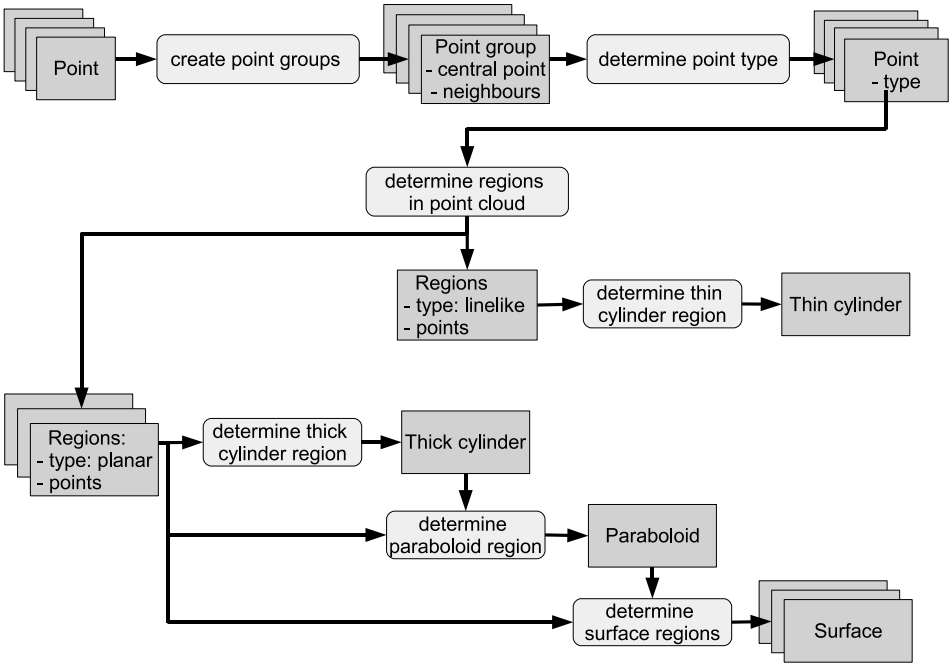


FIGURE 2.7: *The workflow that performs the subtask to find structures in the point cloud.*

the point instance is the central point is a linelike area, then the 'type' is set to 'linelike'.

Determine 'Regions' in the point cloud

- Input: All instances of the class 'Points'.
- Output: A set of 'Regions'.
- Description: For each set of point instances that form a region, an instance of the class 'Regions' is created.

Determine 'Thin cylinder' region

- Input: The instance of the class 'Linelike region'.
- Output: An instance of the class 'Thin cylinder'.
- Description: If an instance of 'Region' has the value 'linelike' for the property 'type', then this instance is mapped onto an instance of the 'Thin cylinder'. This means that the points corresponding to the 'Region' also correspond to the 'Thin cylinder'.

Determine 'Thick cylinder' region

- Input: All instances of the class 'Planar region'.

- Output: An instance of the class 'Thick cylinder'.
- Description: The instance of 'Region' with the value 'planar' that best fits to a cylindrical shape with the expected radius of a plug is assigned as 'ThickCylinder' The points corresponding to this 'Region' also correspond to the found thick cylinder.

Determine 'Paraboloid' region

- Input: All remaining instances of the class 'Planar region'.
- Output: One or zero instances of the class 'Paraboloid'.
- Description: For all input 'Planar regions', the region that is best represented by a paraboloid and that has a small enough distance to the 'Thick Cylinder' is mapped to an instance of 'Paraboloid'. If none of the 'Planar regions' fits well enough, no instance of the class 'Paraboloid' is created.

Determine 'Surface' regions

- Input: All remaining instances of the class 'Planar region'.
- Output: An equal amount of instances of the class 'Surface'.
- Description: Each planar region is mapped to an instance of the class 'Surface'.

The underlying algorithms

To implement the services, algorithms or logical rules are needed. In the cases where algorithms are needed, the computer vision expert is consulted on how to calculate the desired values. To illustrate the underlying structure, we give a description of the algorithms for this workflow. For the other workflows, we omit this information.

- *Determine neighbours of point group*: The algorithm that is used by this service calculates the distance between the central point and all other points, and assigns a point to the set of neighbours when the distance between the central point and this point is below a certain threshold.

- *Determine point type*: The algorithm that is used by this service calculates the principal components of the central point in its 'Point group'. Based on the values of the eigenvalues a 'planarity measure' is calculated. The point is classified as 'linelike' or 'planar' based on this planarity measure.

- *Create region instances*: The algorithm takes all instances of the class 'Point' as input and divides these into a group of planar points and a group of linelike points. Some correction steps (*e.g.* $k$nn correction [98]) are taken to make sure that no isolated planar points are located in a group of solely linelike points and vice versa. Then, all linelike points are assigned to one 'region'. The planar points are separated into coherent groups using a 3D flood fill algorithm. For each coherent group of planar points a 'region' is created and these points are assigned as 'points' in this region.

- *Determine thin cylinder* and *Determine thick cylinder*: With the Levenberg-Marquardt non-linear least squares fitting routine [91], a thin cylinder of radius of 1 mm is fitted to the linelike region and a thick cylinder of radius of 1 cm is fitted to the planar region. Then, the region that fits best to a thick cylinder is selected.
- *Determine paraboloid*: This algorithm performs the Levenberg-Marquardt algorithm on the remaining planar regions to find a paraboloid-shaped region that is close enough to the thick cylinder.

2.5.2 Match structures to object parts

The previous workflow has resulted in a set of geometric shapes, each with a set of points assigned to them. In this workflow, we aim to find the plant parts that correspond to the identified geometric shapes. The first step is to calculate the value of the 'top point' of the Thin cylinder, which is the only missing information in the geometry ontology. Next, the actual mapping of the geometrical concepts onto the plant parts takes place: the instances of 'Surface' are mapped onto instances of 'Leaf', the instance of 'Thick cylinder' onto an instance of 'Plug body', the instance of 'Paraboloid' onto an instance of 'Plug head', and the instance of 'Thin cylinder' onto an instance of 'Leaf or stem'. This mapping has transformed the instances of the geometry ontology to instances in the plant ontology. However, the identified instances in the plant domain do not yet correspond to plant parts as recognised by the domain expert. The 'Plug body' and 'Plug head' should together be assigned to 'Plug'. Moreover, the 'Leaf or stem' instance has to be identified as either a 'Leaf' or a 'Stem'. In case of a 'Leaf', this instance has to be specified as 'True leaf', 'Cotyledon' or 'Connected cotyledons'. All identified plant parts together form the 'Plant'. The workflow corresponding to this subtask is depicted in Figure 2.8

Recognise 'Plug body'

- Input: instance of 'Thick cylinder'.
- Output: instance of 'Plug body'.
- Description: The 'Thick cylinder' is recognised as 'Plug body'.

Recognise 'Plug head'

- Input: instance of 'Paraboloid'.
- Output: instance of 'Plug head'.
- Description: The 'Paraboloid' is recognised as 'Plug head'.

Recognise 'Leaf or stem'

- Input: instance of 'Thin cylinder'.
- Output: instance of 'Leaf or Stem'.
- Description: The 'Thin cylinder' is recognised as 'Leaf or Stem'.

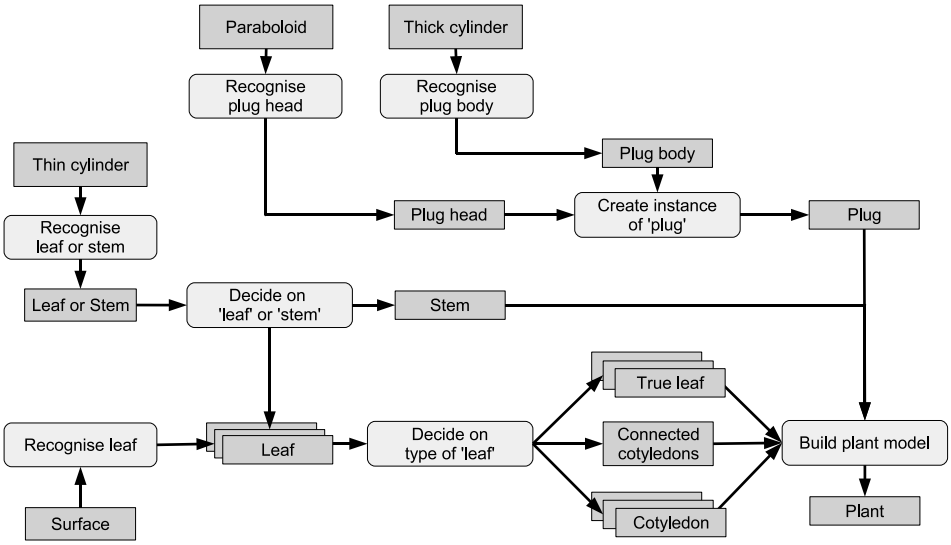


FIGURE 2.8: *The workflow that performs the subtask to match geometrical structures to concepts in the plant domain.*

Recognise 'Leaf'

- Input: instance of 'Surface'.
- Output: instance of 'Leaf'.
- Description: The 'Surface' is recognised as 'Leaf'.

Create instance of 'Plug'

- Input: instance of 'Plug body', instance of 'Plug head'.
- Output: instance of 'Plug'.
- Description: Together, the instances of 'Plug body' and 'Plug head' form the 'Plug'.

Decide on 'Leaf' or 'Stem'

- Input: instance of 'Leaf or stem'.
- Output: Either an instance of 'Leaf' or an instance of 'Stem'.
- Description: This service decides whether the 'Leaf or stem' instance represents the 'Stem' or a small 'Leaf'.

Decide on leaf type

- Input: instance of 'Leaf'.
- Output: one or more instances of 'Cotyledon', 'True leaf', or 'Connected cotyledons'.

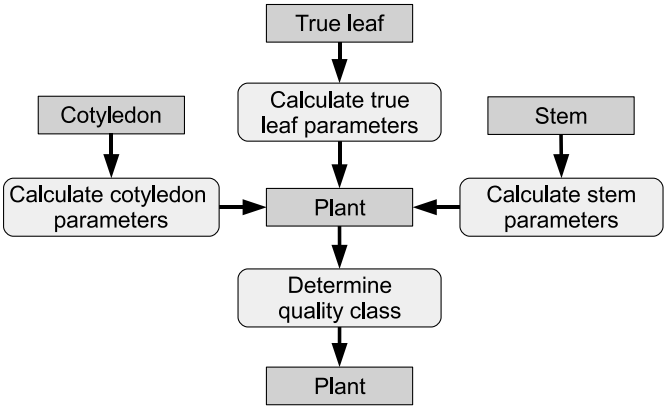


FIGURE 2.9: *The workflow that performs the subtask that calculates the parameters of the plant parts and that determines the quality class of the Plant. The quality class is stored as property of the Plant.*

- Description: This service determines whether an input leaf is a compound leaf, or not. It splits the leaf into proper leaf areas and creates a new instance for each area corresponding to the actual type of leaf: 'Cotyledon', 'Connected cotyledons', or 'True leaf'.

Build plant model

- Input: instance of 'Plug', instance of 'Stem', instances of 'Cotyledon', instance of 'Connected cotyledons', instances of 'True leaf'.
- Output: instance of 'Plant'.
- Description: The recognised instances of the plant parts are assigned to a new instance that represents the 'Plant'.

2.5.3 Determine quality class

The next subtask calculates those parameter values of the plant parts that are required for the quality assessment process. For the seedling assessment process, parameters of the stem, the cotyledon and the true leaves are required. Next, the calculated parameters are combined to determine the quality of the plant. The workflow representing this subtask is depicted in Figure 2.9.

Calculate stem parameters

- Input: instance of 'Stem' without stem parameters.
- Output: instance of 'Plant' with stem parameters.
- Description: This service calculates the thickness and the length of the stem.

Calculate true leaf parameters

- Input: instance of 'True leaf' without true leaf parameters.
- Output: instance of 'Plant' with true leaf parameters.
- Description: This service calculates the 'true leaf area', the 'true leaf length', the 'curvature' measure of the true leaf on the stem side, and the 'curvature' measure of the true leaf on the tip side.

Calculate cotyledon parameters

- Input: instance of 'Cotyledon' without cotyledon parameters.
- Output: instance of 'Plant' with cotyledon parameters.
- Description: This service calculates the area, length, width, indent area, circumference and curvature of the cotyledons.

Determine quality class

- Input: instance of 'Plant' with parameters.
- Output: instance of 'Plant' with quality value.
- Description: This service determines the quality class to which the plant belongs.

2.6 Discussion and conclusions

In this chapter, we have presented a method to systematically design ontology-based computer vision applications by decomposing the task into subtasks until the level of inferences has been reached. An ontology-based computer vision application models the expert's knowledge in ontologies that are used to perform the computer vision task automatically. Moreover, the computer vision application is designed in such a way that it mimics the way that the expert has described the task execution.

The proposed method supports software engineers and knowledge engineers to implement a knowledge-based computer vision system that closely follows the expert's way of inspecting objects. In this sense, it is a user-centered design method [105, 112]: the assessment by the experts, the usability of the application with respect to changing needs. The method starts with setting the scope of the computer vision application. The scope supports the domain expert in formalising the application ontologies. Moreover, it helps the domain expert in accurately specifying the inspection task. Finally, the specification of the scope helps the computer vision expert to determine the type of image acquisition procedure that is to be used.

Next, the method decomposes the task into subtasks. This decomposition models the domain expert's "observe – interpret – assess" way of performing a visual inspection task. This decomposition leads to a generic framework. The list of consecutive subtasks – record object, find structures, identify object

parts, determine quality – can be reused for any visual inspection task. The decomposition of the task into subtasks results in a framework in which the mentioned subtasks are alternated with consecutive application ontologies in which models of the recorded object exists.

The last step in the method is to design the various application ontologies that exist between the subtasks. This process is discussed in Chapter 3. The application ontologies contain those concepts that are required for a successful execution of the computer vision task; they strongly influence the workflows of which the subtasks are composed. Information about the object under inspection can be captured and propagated through the subtasks due to the concepts of the corresponding application ontology. To this end, the workflow must be created in such a way that it enables the concepts from one ontology to be transformed into the next ontology in a meaningful way. Expressing processing knowledge in a declarative way is discussed in more detail in Chapter 4.

In this chapter, we have illustrated each step with the horticultural case study. We submit, however, that the proposed method can be used for the design of any ontology-based computer vision application aimed at automating a visual inspection task. For each visual inspection task, a specification of the task and its domain can be made. This is a joint activity of the problem owner and the domain expert. With the scope in mind, the image acquisition task can be designed. Recorded camera data are stored in the computer vision application. It does not matter if the chosen structure is a point cloud, as in the horticultural case, or a set of pixels, a set of line-scan images, etc. As long as an application ontology is created by the computer vision expert (together with a knowledge engineer), the application can handle the obtained input format.

The segmentation phase starts with looking for structures in the image. For the horticultural case these were geometrical structures. For other applications, they may be structures composed of areas with similar texture or colour, edges, clusters, *et cetera*. As long as an application ontology is created for the chosen structures, the model of the object can be dealt with by the computer vision application. After this low-level segmentation, the next step recognises the object under inspection. The recognised structures transform into a model of the object under inspection, *e.g.* a plant, a parcel, a tumor, but again only into objects that can possibly exist in this specific task context. This is ensured by the explicit definition of the application ontology, delineated by the specified scope. The segmented object in turn is mapped to an assessment class, such as quality, price level, style, again specifically selected for this task.

The method proposed gives a conceptual, generic framework that sets the stage for the content of the application ontologies and for the algorithmic implementation of the procedural knowledge. At the lowest conceptual level, task-specific design steps are taken. By setting up the computer vision application in the proposed way, we predict that the application has some important additional properties besides the required correctness and speed: expert acceptance, corrigibility, adaptability, robustness and reliability. We show this in Chapter 6.

Chapter 3

Obtaining Task-Specific Knowledge

In the previous chapter, we have proposed a method for developing ontology-based computer vision applications. One of the key aspects of the proposed method is the availability of task-specific ontologies that contain the relevant knowledge for representing the specific perspectives needed to fulfill this task. In this chapter, we describe a method to develop such dedicated ontologies. The knowledge modelling activity starts off with a traditional interview-based knowledge acquisition method that is extended with a reuse-based component for domain experts. We submit that the presented method is in particular suited to create application ontologies for *task-specific, multi-domain, multi-expert settings in which expert knowledge may be partly tacit*.

This chapter is based on three publications: a paper for the Theory and Applications in Knowledge Management workshop (TAKMA 2005) [59], a paper for the European Federation of Information Technology in Agriculture (EFITA 2005) [58] and a paper for the Asian Semantic Web Conference (ASWC 2008) [57].

3.1 Application ontologies in computer vision applications

The use of ontologies as a means to transfer knowledge of the environment to a computer vision application has been reported in several studies in the field of cognitive computer vision [30, 38]. Thonnat [102] for example, shows two applications of using explicit expertise for complex image processing. Hudelot and Thonnat [50] use ontologies to support their generic (domain-independent) cognitive vision platform. Maillot *et al* use ontologies in their 'methodology that is not linked to any application domain' [73]. The ontologies described in these cognitive vision publications are generic ontologies. Such ontologies are most commonly used, both in the context of cognitive vision and in the context of the Semantic Web. Domain ontologies are frequently used to improve the accuracy of web searches, to assist in the organisation and navigation of web sites, or

to enable the communication of intelligent agents with their environment [4]. These types of tasks require a vocabulary that covers a broad domain.

For modelling knowledge that is used by a group of experts to perform a dedicated knowledge-intensive task, domain ontologies are too generic. First, generic domain ontologies commonly contain more information than necessary for a task. For example, the description of all possible kinds of defects and diseases of plants in the generic plant domain ontology is irrelevant for the seedling inspection task. It would cause unnecessary overhead and complexity for the automated seedling inspection application if a generic plant ontology was used. Checking a plant for all listed defects and diseases will cause unnecessary delays in the high speed computer vision application. The computer vision application is expected to handle approximately 15,000 plants per hour, a number which is on the limit of current processor speed. Any delay ensures that this number is not met and hence that the computer vision system is of limited economic interest. Second, generic domain ontologies are not detailed enough for fulfilling a knowledge-intensive task successfully. For any knowledge-intensive task, task-specific information must be included in the ontology in addition to general knowledge about the domain. Examples of attributes that are relevant for our case study but that will not be found in a generic plant ontology are the *curvature of leaf tips* and the *angle between true leaves*.

We introduce the idea of *application ontologies*[1] to find a balance between general domain knowledge and specific task details. An application ontology consists of (i) general domain knowledge that is relevant for the task at hand, and (ii) additional task-specific knowledge that is not necessarily valid or available for the generic domain. Hence, application ontologies are formal knowledge models that are dedicated to a specific task.

We continue this chapter with a detailed description of the interview-based knowledge acquisition method that has been used to create the plant application ontology (Section 3.2). This method can deal with multi-expert, multi-domain, task-specific settings. Next, we introduce an alternative knowledge acquisition method, the ROC-method, that is based on reusing existing sources (Section 3.3). ROC is developed for single-expert settings and has not been applied to the seedling inspection case study. Instead, this method is illustrated by a 3D geometry ontology for a drawing application, and a supply-chain ontology for an expert finding application. The interview-based knowledge acquisition method and the ROC-method are briefly compared in a cost-benefit analysis in Section 3.3.5. In the discussion section (Section 3.4), we reflect on the strengths and weaknesses of the interview-based method and the ROC method. We describe in which settings which method can be used. We conclude that a combination of both methods leads to an efficient knowledge acquisition process.

[1] 'Task-specific ontology' is used as a synonym for 'application ontology'.

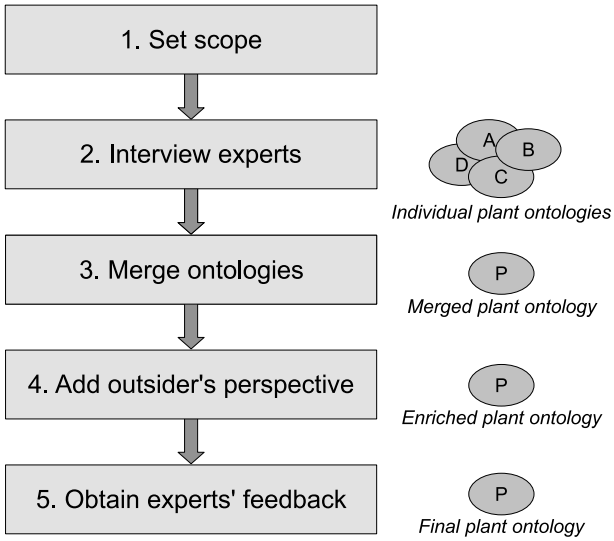


FIGURE 3.1: *A schematic representation of the interview-based knowledge acquisition process, illustrated for the plant ontology case study. This process can be used as a template for interview-based knowledge acquisition for multi-expert, multi-domain, task-specific situations.*

3.2 Interview-based knowledge acquisition

For knowledge-intensive computer vision applications, application ontologies have to be developed. Such application ontologies are *multi-domain*; not only the domain knowledge of a domain expert, but also the domain knowledge of a computer vision expert has to be taken into account. They are *task-specific* as well, since the application ontology should contain only those concepts and relations that are relevant for the computer vision task. By involving *multiple domain experts* in the knowledge acquisition process, the tacit knowledge from some of these experts can be filled in by the explicit knowledge from other experts.

For creating application ontologies we propose a knowledge acquisition process that can deal with these three characteristics of the task (see Figure 3.1): individual domain experts are interviewed, the individual models are merged, an expert from a different domain is asked to add knowledge to the application ontology, and the enriched ontology is presented to the domain experts for validation.

For the seedling inspection application, we are interested in two types of knowledge from the plant inspection expert: *seedling assessment rules* and *plant morphological knowledge*. Both types of knowledge can be obtained at the same time by interviewing domain experts that assess seedlings on a regular basis. In this section we focus on the plant morphological knowledge. In the next sections, the interviewing process is indicated in *italics* interspersed with observations and lessons learned in roman font.

3.2.1 The interviewing process

According to Schreiber *et al* [95], knowledge acquisition or knowledge elicitation is the process of capturing and structuring knowledge from a human expert through some form of interaction with that expert. The knowledge needed to perform visual inspection tasks consists of *procedural knowledge* – how is the task performed in terms of methods, rules and heuristics – and *descriptive knowledge* – what are the relevant concepts. Procedural knowledge is used to model the task decomposition and corresponding work patterns. Descriptive knowledge has to be acquired to allow the creation of the corresponding knowledge models.

There are several well-known methods to elicit descriptive knowledge. Schreiber *et al* provide a good overview of such knowledge acquisition techniques in Chapter 8 of the CommonKADS book [95]. For the case study, we have used the open interview technique proposed by Scott *et al* [96]. Interview-based knowledge acquisition is an activity for which a knowledge engineer interviews one or more domain experts. The knowledge engineer asks questions to learn and understand as much as possible from the domain experts. It may be useful to think of this type of knowledge acquisition as an apprenticeship process by which knowledge engineers make the transition themselves from that of a novice to becoming well-versed in the domain [64]. We have also employed the observation-based 'Think aloud method' as described by Van Someren *et al* [100]. Observation-based knowledge acquisition is the activity of observing the domain expert while he executes the task. By combining these interview-based and observation-based techniques, the knowledge engineer aims to get a full description of the domain expert's knowledge.

For the case study's knowledge acquisition sessions, we asked 13 experts in individual sessions to list plant features relevant for the quality assessment task. Part of each session consisted of the expert showing us his task while explaining his actions aloud. Hereto, each expert had access to two trays of 240 seedlings each to illustrate their assessment routines. The trays were provided by the companies that employ the experts. This ensured an optimal correspondence between the plant material used in the interviews and the material that is usually inspected by each expert.

In the knowledge acquisition process, three (idealised) roles are distinguished, each role with its own responsibilities and skills: the problem owner who explicates and monitors the purpose of the ontology, the domain expert who explains the relevant domain knowledge and the knowledge engineer who creates the application ontology. By asking the problem owner to state the purpose of the application ontology in advance, the domain expert has a way of checking that the knowledge that he expresses is relevant for the application ontology.

Each expert was interviewed once. Each interview-observation session lasted approximately two hours. After each session, a report in natural language was written and the expert used approximately 30 minutes to read it and verify its content.

By comparing the thirteen interviews, we noticed that each expert described the

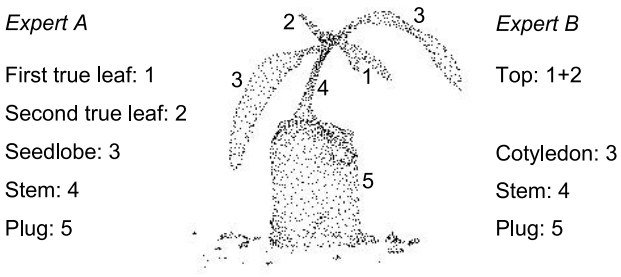


FIGURE 3.2: *The plant structure as seen by two different sorting experts.*

plant structure in a different way. Figure 3.2 depicts part of the plant morphology as sketched in two of the thirteen interviews.

Individual experts give a description of their domain as they see it. Differences between two experts that perform the same task in the same domain are caused by three effects. First, two experts may have a different granularity level for the terms with which they describe their domain. One expert considers for example the two top leaves of a seedling as separate entities *First true leaf* and *Second true leaf* respectively, whereas a second expert describes them as one entity, the *Top* of the plant. Second, the terms used for the same concept may differ between two experts. For example, the leaves that are indicated with a '3' in Figure 3.2 are referred to as *Cotyledons* by Expert *A* but as *Seed lobes* by Expert *B*. Third, two experts may use the same term to indicate a different concept. The term *Growth point* for example is used by some experts to indicate the tiny plant part from which the next true leaf will sprout, while it is used by other experts to indicate the *Top* of the plant. The knowledge engineer should be aware of these differences when he models the experts' knowledge.

Based on the information obtained in the interview-observation sessions, we created for each expert an individual plant ontology. As a first step, we determined in each interview all plant parts (the concepts) and used these to form the rudimentary structure for the individual knowledge models. The second step in creating the ontologies consisted of further studying the interviews and adding all mentioned properties (attributes and relations) of the plant parts to the initial ontology.

In Figure 3.3, we see a representation of part of the two ontologies that were depicted as rudimentary structure in Figure 3.2. This time, the relations between and the attributes of the concepts are displayed as well. We see that the structures of the two ontologies differ, since not all concepts of ontology A are present in ontology B and vice versa. Another difference stems from the variation in emphasis on the relevance of the parameters: in ontology A, damage may only occur at the level of the 'Seed lobe'; in ontology B, damage to the 'True leaves' is also taken into account and the location (close to the 'Stem' or not) and degree (heavy, medium, light) of the damage is also important. Finally, we noticed that the concepts in the ontologies may contain different attributes. Sometimes, these attributes are synonyms of each other. This holds e.g. for

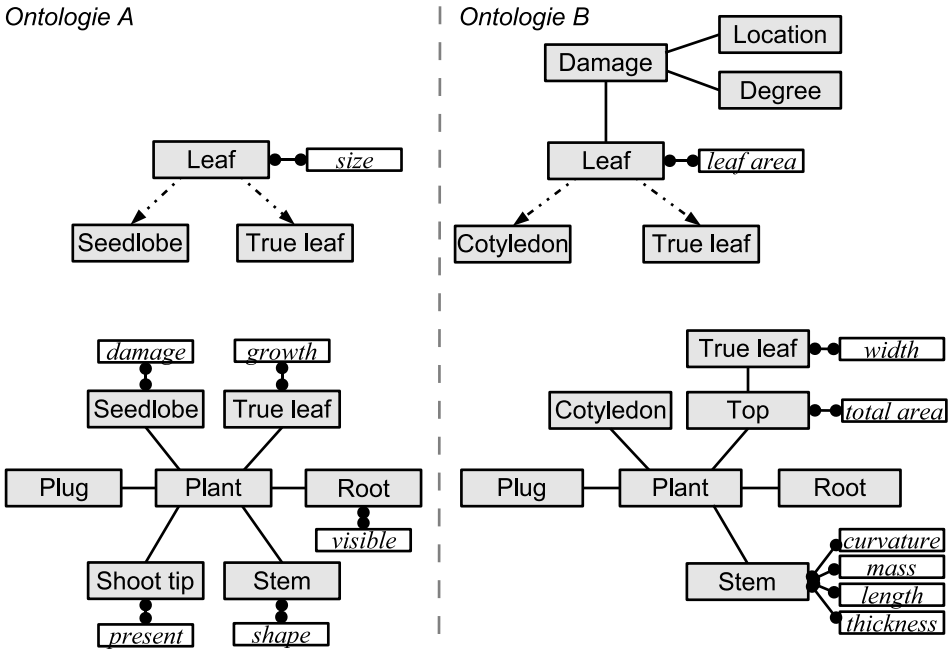


FIGURE 3.3: *Schematic representation of a part of two individual plant ontologies.*

the attributes 'size' and 'leaf area' of the concept 'Leaf'. In other cases, attributes exist in only one of the two ontologies, as e.g. the attribute 'curvature' of the 'Stem'.

3.2.2 Merger of individual ontologies

Although individual ontologies contain different descriptions of the domain, each description represents a domain model with which the expert can successfully fulfill his task. Each individual ontology corresponds to a valid expert's view on his domain. Since we are interested in finding a complete overview of experts' domain knowledge, we perform a merging step in which the differences between the individual ontologies should be incorporated.

To merge the thirteen individual ontologies, we identified the concepts in the individual ontologies, chose appropriate names for them and indicated other names as synonyms. The concept 'Cotyledon', for example, had seven different synonyms in the interviews[2]*: 'Cotyledon', 'Seed lobe', 'Lobe', 'Ear', et cetera.*

Retaining synonyms of concepts expressed by different domain experts is important for the merged ontology in order to allow each expert to recognise his

[2]Not all names can be translated. In Dutch they are: cotyl, kiemlob, kiemlobbe, lob, lobblad, oor, zaadlob

own terms in the final knowledge model. This facilitates the experts in easier understanding and acceptance of the created knowledge model, allows knowledge from one expert to be expressed in terms of another expert, and supports the communication process between experts.

We noticed that on the concept level most differences between the individual ontologies resulted from different levels of detail in the plant description. Hence, we incorporated the different plant models from all individual ontologies by introducing hierarchical levels of plant concepts. The concept 'Top' for example consists of all 'True leaves'. Even though the concept 'Top' did not occur in the ontology of expert A, by adding it to the merged ontology we did not fundamentally change the structure of the plant as described by expert A, but added additional details in the plant structure.

Next, we defined the attributes for each concept. We found that experts differed significantly in the expression of these attributes due to the different business foci of the companies. Partly, though, some experts had forgotten to express some attributes, even when they did use those attributes in the quality inspection task.

Part of the phase in which the attributes were defined consisted of deciding the range of the attributes. Some attributes were defined qualitatively in some individual ontologies and quantitatively in other ontologies. This was the case for the attribute 'length' of the concept 'Stem'. The value of this attribute was expressed quantitatively as a number (at least 3 cm), or qualitatively in comparison to the average stem length of all plants in the tray. By deciding to use the latter range, an additional attribute 'average stem length' had to be added to the concept 'Tray'.

The interview-based knowledge acquisition process expects from experts that they can give a full overview of relevant knowledge to a knowledge engineer. This may be a too optimistic representation of the domain experts capacity of communicating about their task knowledge. The knowledge engineer can ask questions, but due to his lack of domain knowledge, he may not cover all parts of the relevant domain. This problem can be solved by interviewing multiple experts that fill in each other's gaps. Another way is to prompt the expert with possibly relevant concepts taken from existing sources. We elaborate on this procedure by introducing the reuse-based ontology construction component in Section 3.3.

With the merging of the concepts and relations, a rough outline of the final merged ontology was obtained. The last step in the modelling process dealt with a refinement of the merged ontology in order to accommodate implicit nuances that were used by some of the experts. An example of such a refinement is to subdivide the concept 'Damage' by the notions of 'Localised Damage' and 'Global Damage'. The experts indicated that for certain defects, such as 'leaf curvature', the location of the defect is important in the quality assessment, while this was not true for other types of defects, such as 'indent area'.

In the merging process, the knowledge engineer is forced to think about details of the individual ontologies that may at first have been hidden. As a result,

additional relevant information can be identified that was not expressed by any of the originally interviewed and observed experts.

3.2.3 Addition of 'trivial' aspects

The first two steps of the interview-based knowledge acquisition process ensure that the views of multiple experts are combined into one rich knowledge model. The resulting model is created for use in a multi-domain setting; it is used in a computer vision application. To successfully support the computer vision task, we involve a second type of expert in the creation of the plant ontology: the computer vision expert. The role of this expert is to identify relations or concepts that exist in the knowledge domain of the first experts, that are crucial for the computer vision system but that were not mentioned by the task experts (see step 4 in Figure 3.1). Such knowledge may be too trivial for task experts to mention, or may not be part of the usual perspective of the task experts.

More generally speaking, in a multi-domain application, any expert E_i in a different domain than expert E can add a task-relevant outsider's perspective to the domain of expert E to reduce the probability that too trivial or otherwise forgotten facts are omitted in the domain description.

The plant ontology plays an important role in two of the identified subtasks; the output of the segmentation task is defined in terms of the plant ontology as is the input of the quality determination task. The computer vision expert based his input on the merged ontology and a set of recorded seedlings. Properties that were observed by the computer vision expert and added to the plant ontology were e.g. the relation that a cotyledon is connected to a stem, and the fact that a tomato plant can have only one stem.

3.2.4 Verification of obtained models

To ensure that the merging of individual ontologies and the adding of outsiders' perspectives do not result in inaccuracies, it is important to verify the resulting application ontology. A well-known verification method is a *teach-back session*. The essence of such a session is that the knowledge engineer presents to all participating domain experts at the same time the created knowledge model and asks for feedback from the experts.

At the teach-back session we presented to all domain experts interviewed the merged plant application ontology that was enriched with the knowledge needed for the computer vision task. During the teach-back session, several concepts had to be refined on instigation of the experts. The experts indicated for example that the concept 'Leaf stem', that connects the leaves to the main stem, must be considered as part of the 'Leaf' and not of the 'Stem'. Some relations and attributes were removed from the merged ontology. The attributes 'is beautiful' and 'mass', for instance, were covered by combinations of other attributes and were therefore removed from the ontology. In two cases, an attribute was discussed for which the sorting experts could not reach an operational definition.

These were the attributes 'uniformity' of a 'Tray', which was outside the scope of the knowledge used by the sorting specialists, and the attribute 'delicacy' of a Plant, which is so complex that it needs to be defined for each cultivar separately. Since these two attributes lie outside the scope of the seedling inspection task, they were removed from the ontology. The same holds for the concepts 'Root' (inside the plug and hence not visible) and 'Shoot tip' (too small for detection).

Part of the discussion was about the additional relations and attributes of the concepts that were added to support the computer vision task. Since these were of almost trivial nature to the experts, there were no problems in approving them.

The teach-back session with the experts took approximately 1.5 hours. The proposed changes were made to the ontology.

Interview- and observation-based knowledge acquisition entails intensive interaction between knowledge engineer and domain experts. This human factor often leads to a willingness of the experts to make the project a success. Because of this, the involved experts may be inclined to adapt their own individual knowledge model and accept a richer, all-encompassing domain model. This effect is stronger for domain knowledge that is background knowledge, like the plant morphology in our example, than for knowledge concerning economically critical processes, like the inspection rules that are used for the quality inspection.

3.3 The reuse-based ontology construction component

In the previous section, we presented an interview- and observation-based knowledge acquisition process to obtain a multi-domain, multi-expert task-specific ontology. This process consists of four steps: (i) create knowledge models based upon interviews with and observations of individual experts, (ii) merge the individual ontologies into a merged ontology, (iii) add the perspective of a second type of experts to the merged ontology, (iv) organise a teach-back session to check the validity of the created ontology. We identified three roles that are involved in the knowledge acquisition process: the problem owner, the domain expert, and the knowledge engineer.

The interview-based knowledge acquisition method has a number of advantages. First, by interviewing experts, these experts tend to get involved in the knowledge modelling activity. This commitment improves the quality of the final model. Second, by eliciting the knowledge of several domain experts, we increase the probability that tacit knowledge of one expert is covered by explicit knowledge of another expert after merging the individual ontologies. This results in a particularly rich application ontology. Third, the involvement of experts from various task-relevant domains results in an application ontology that is honed to the task for which the application ontology is created. Finally, the teach-back procedure allows for corrections when needed, discussions between the involved experts to obtain a common solution to possible contradictions, and commitment of the experts to the final model.

Knowledge acquisition based only on interviews and observations has some disadvantages as well. First, it is a process that takes a lot of time, since the knowledge engineer has to be trained to become a domain expert himself before he can create a model of the task domain. For the creation of the plant application ontology with only 37 concepts, for example, the domain experts spent a total of 58 hours and the knowledge engineer even 88.5 hours. Second, for the domain expert, the knowledge elicitation process is a new task. It may therefore be difficult to give a structured and full overview of his task knowledge. Finally, interview-based knowledge acquisition typically creates a model from scratch, even though reusable sources may exist.

Based on these observations, we conclude that involving domain experts is beneficial for obtaining a usable task-specific knowledge model. To overcome the disadvantages of pure interview-based techniques, we propose to add a reuse-based ontology construction component (ROC) to our toolkit of interview- and observation-based knowledge acquisition that gives domain experts a more prominent and active role in the knowledge modelling process. This way, we aim to make the knowledge acquisition process more efficient. Moreover, the ROC component supports the domain expert in identifying relevant knowledge, for ROC incorporates a prompting process that offers the domain expert terms associated with the terms that he has selected in earlier iterations. In this way, the domain expert has a better opportunity of covering all aspects of his knowledge. For the prompting process, existing knowledge sources are used. Thereby, we benefit from already existing sources in the ontology engineering process.

3.3.1 Related work on ontology engineering

Knowledge acquisition is part of the research field of ontology engineering. Besides traditional knowledge acquisition activities such as interviewing and observing, the field of ontology engineering covers methods involved in ontology development, management and support. Gomez *et al* [37] present a good overview of well-known ontology engineering methods, such as KACTUS, Methontology, SENSUS, On-to-Knowledge etc.

Most ontology engineering methods support the *knowledge engineer* in creating the required ontology. Ontology editing tools such as Protégé [56] are aimed at professionals with a background in formal representation languages such as OWL or RDF. This makes them less suited for use by domain experts who lack such a background.

Some methods, such as ACE, Rabbit and CLone [54, 42, 34], have been presented that transfer the knowledge modelling process to domain experts. The chosen approach is to force the domain experts to use controlled natural language and subsequently parse the constructed sentences into appropriate (OWL) statements. However, sentences that express domain expert's knowledge in such languages can only be formed in predefined ways and restrict the domain experts' freedom of expression. The reason is that only by using highly constrained sentence structures, it is possible to unambiguously translate them to OWL statements.

We do not think that the solution to making the knowledge acquisition process more efficient is obtained by training the domain expert to become knowledge engineer. Instead, with the ROC component we explicitly aim to support the domain expert in identifying the knowledge that is to be included in the ontology and expect the knowledge engineer to transform the identified knowledge into the required ontology in a separate modelling step. Therefore, we do not target a specific set of OWL constructs, but allow the domain engineer as much freedom as possible to express knowledge. The only prescribed structure is that knowledge has to be entered in a 'subject–predicate–object' triple format, the most basic sentence structure in natural language and at the same time the basis for the RDF ontology language. There are no additional restrictions on the triple's content.

The integration of the newly created knowledge model with existing sources is traditionally handled by the knowledge engineer. Semi-automatic support for reuse activities such as mapping is provided by tools such as Falcon-AO [49]. A novel idea in ROC is to reuse existing sources by offering the domain expert possibly relevant concepts from a preprocessed repository. Hence, we explicitly support the associative process of knowledge elicitation (for example, by helping the expert in remembering to include related concepts). It is expected that such associations speed up the acquisition process. Some work in this direction is also done by D'Aquin *et al* [19]. They have developed a plug-in for both the Protégé and NeOn toolkits which enables the search for related RDF-triples on the Web using the Watson semantic search engine[3]. The user of this plug-in can select relevant RDF-triples for inclusion in the current project. With ROC, we provide similar functionality but (i) base ourselves on an information repository focused on the domain at hand that contains triples gathered from RDF-sources and semi-structured web documents, and (ii) integrate the reuse-functionality in the domain expert-friendly ROC-environment.

3.3.2 Proto-ontologies

For the knowledge identification activity of the domain expert, we introduce the concept of a purpose-specific 'proto-ontology'[4] Proto-ontologies are knowledge models that solely consist of concepts and relations; formal term definitions, knowledge rules and logical constructs are not part of the proto-ontology. We believe that proto-ontologies are well-suited for the domain expert to easily capture relevant knowledge as basis for the final application ontology.

In the creation of proto-ontologies, we do not require the use of 'good modelling practices', as for example defined in [86], since concepts like 'subclassOf', 'datatypeProperty', or 'inverseProperty' as used in the ontology language OWL are mostly meaningless to the domain expert and may even hinder the knowledge identification process. Instead, we stay close to natural language expressions to assist the domain expert. The resulting proto-ontology is a useful

[3] `http://watson.kmi.open.ac.uk/editor_plugins.html`

[4] A proto-ontology in this thesis is always application specific. For readability reasons, we use proto-ontology instead of 'application proto-ontology'.

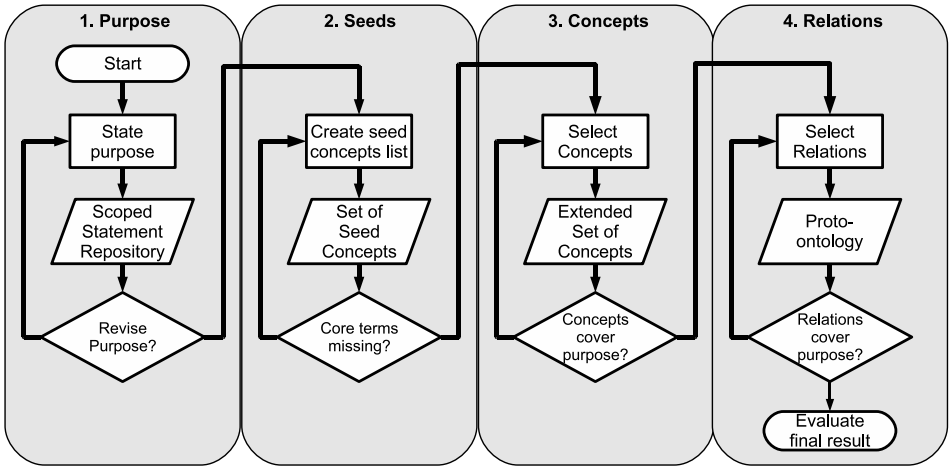


FIGURE 3.4: *Workflow of the proto-ontology creation process. Setting the purpose and identifying seed terms are tasks that are performed by the problem owner and the domain expert, selecting and specifying concepts and relations is done by the domain expert. The problem owner is responsible for evaluating the concepts and relations that are added to the proto-ontology. After the evaluation of the final result, it is possible that a new ROC-cycle is required.*

intermediate product consisting of interconnected informal RDF-triples that the knowledge engineer can work on.

3.3.3 Reuse-based ontology construction in five steps

In this section, we explain the steps used to support the domain expert in identifying relevant knowledge for the proto-ontology. This process is divided into four main parts; stating (1) purpose, (2) seed terms, (3) concepts, and (4) relations, not unlike existing methods such as Methontology [31]. The progression through these tasks is presented in Figure 3.4, but the domain expert is free to switch back and forth between different steps as he sees fit. In the 'concepts' and 'relations' part, the ROC component prompts the domain expert with possibly relevant concepts or relations. Below, these steps are presented in more detail.

Orthogonal to these four steps is the process of evaluation. While each step has a decision criterion to decide whether or not to progress to the next step, Step 4 finishes with an overall evaluation of the resulting proto-ontology. The domain expert and the problem owner evaluate the created proto-ontology with respect to the specified purpose. They decide whether the result is satisfactory. If this is not so, any step in the proto-ontology construction workflow can be revisited.

Step 1: Source identification.

For the prompting process, the ROC component requires the presence of semi-structured knowledge in an information repository. We are interested in reusing

existing knowledge sources that have been processed into simple natural language statements in a 'subject – predicate – object' format. Such statements are understandable to the expert as 'term – association – term' statements and can, in a later stage, easily be mapped onto RDF/OWL expressions by the knowledge engineer.

To create the required information repository, we have to identify sources that may contain relevant information for the proto-ontology. We recognise two types of usable sources: (i) *semi-structured sources*, like web pages in a structured layout (tables, pages with rigid section structures, etc.), and (ii) *existing ontologies and thesauri*, typically formalised in OWL, RDF(S), or SKOS.

To effectively use the semi-structured sources, a *triple extraction* step is required. Various techniques can be employed to do so. The possibility used in this thesis is to create a custom parser and triple extractor for each identified semi-structured source. This 'tailor-made' approach yields relatively high-quality triples, but is labour-intensive for the knowledge engineer, since it requires adaptation of the extraction tool for each new source. Alternative approaches using more generic and robust tooling are supported as well, for example in the form of more generic parsers and crawlers, but also the integration of text mining and named entity recognition software (*e.g.* Calais[5]). Note that our claim is not to have created a particularly novel triple extraction technique, but that when a source is harvested for triples, those triples can be efficiently used to support the proto-ontology creation process.

Existing ontologies and thesauri, the second type of existing sources, are already formalised and structured as triples. The triples from both types of sources can be used as the basis for the prompting process. The triples are stored in an information repository, in our case realised using the Sesame RDF framework[6]. To ensure that triples can be presented to the domain expert in a format that is intuitive to him and that does not burden him with formal knowledge representation terminology, informal labels of the formal relations are required. Hence, the knowledge engineer has to review the selected sources and provide such mappings. For example, the formal 'rdfs:subClassOf' relation could be mapped to the natural language expression 'is a'.

Step 2: Defining the scope.

Although ontologies are typically considered purpose-independent artifacts, in creating application ontologies, we take the purpose as defined *a priori* by the problem owner into account as a crucial aspect during development, since it helps the domain expert to keep focus and to decide on issues such as coverage and level of detail of the proto-ontology. The scope of the proto-ontology is determined by three dimensions: the application perspective, represented by the problem owner; the domain perspective, determined by the domain expert, and the discipline perspective, covering the multi-domain aspect of application ontologies (see Figure 3.5). The problem owner is the main stakeholder for the

[5] `http://www.opencalais.com/`
[6] `http://www.openrdf.org/`

application that is to be supported by the ontology. Hence, he takes the lead in defining the application's scope and purpose. He first identifies the type of task that is to be performed, using the CommonKADS [95] task templates. During the proto-ontology construction process, the domain expert and problem owner interactively refine the purpose and adapt the proto-ontology to converge to a knowledge model that is well suited to support the envisioned application. A domain expert selects concepts from the domain; the problem owner may reject some of them and indicate that other concepts are to be explored further. This process of delineating the scope with respect to the domain and task dimension takes place for every discipline involved.

Aspects that help to specify the scope of the proto-ontology are:

- The application for which the proto-ontology is created. In the ROC-component, the problem owner can enter a description of the application in natural language.
- The type of end users of the application, and thereby the level of expertise required in the proto-ontology. In ROC we distinguish 'general public', 'professionals', and 'experts'.
- The disciplines covered by the proto-ontology. The ROC-component offers a list of general subjects, such as food, agriculture, mathematics, etcetera.

Moreover, in the process of setting the scope of the application, the problem owner and domain expert can specify terms that should be included in the proto-ontology, and terms that should not be included. This way, the scope definition is gradually refined.

For example, we imagine developing the plant proto-ontology for our case study using the ROC component. The description of the application is: *A computer vision application that uses expert knowledge to automatically determine the quality of recorded tomato plants*, a classification task. The system will typically be used by *Experts*. The subjects of relevance that are chosen by the problem owner may be *Agriculture*, *Plants* and *Computer vision*. During the process, the problem owner and domain expert indicate concepts that are to be explored in more detail, such as *Cotyledon* and *True leaf*, and reject concepts that are too detailed for inclusion such as *Petiolule* (too detailed on the domain axis) and *Plant diseases* (out of focus of the classification task). The domain expert elaborates by finding related concepts to *True leaf* such as *Terminal leaflet* and *Vein structure*. The problem owner can accept or reject the proposed concepts. In this way, problem owner and domain expert work together to find the concepts that best suit the purpose of the proto-ontology within the plant subject layer. Next, the computer vision expert is asked for his input. In a similar process, he and the problem owner will identify that geometry concepts such as *Cylinder* and *Surface* are important, but texture features such as *Edge* and *Pattern* are out of scope. In Figure 3.5 this scope refinement process is depicted. Concepts C, D and F are within scope, the other concepts are too generic or specific in one of the dimensions.

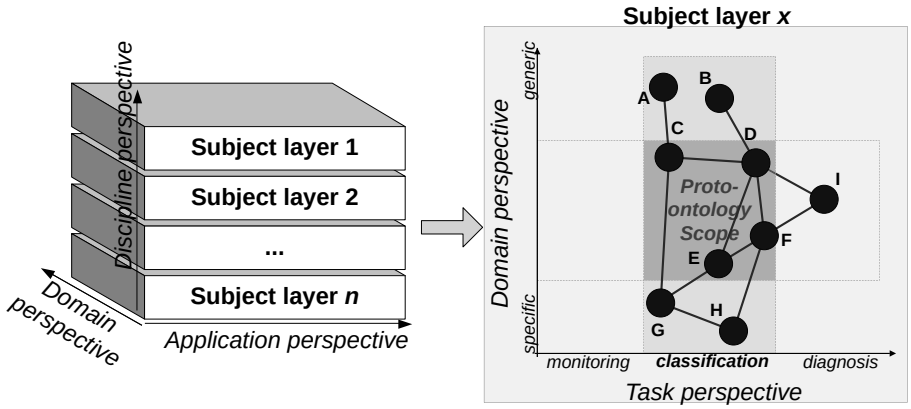


FIGURE 3.5: *Scoping of the proto-ontology. The scope is determined by the three dimensions task perspective, domain perspective and discipline perspective. For each subject layer on the discipline axis, a domain expert is asked to identify the relevant concepts. In the figure, concepts C, D, E and F are within the scope. Concepts A and B are concepts like 'Flower' and 'Disease' that are too generic for the task, concepts G and H are concepts like 'Stem hair' and 'Petiole' that are too specific for the task. Concept I may be a plant disease such as 'Necrosis' or 'Fungus', useful for the diagnosis task, but not for the classification task.*

Step 3: Seeding the proto-ontology.

When the scope is set, the domain expert is asked to compile a list of terms that are relevant to the proto-ontology domain, the so-called *seed concepts*. The seed concepts can be obtained in several ways. The domain expert can just list them at the beginning of the proto-ontology construction process, the knowledge engineer can extract them from an intake interview or from the description of the application as specified in the scoping process, or they can automatically be extracted from relevant texts. The problem owner has the possibility to identify terms that are *not* to be included in the proto-ontology. This list of non-concepts serves to restrict the proto-ontology to its intended scope. Both the approved terms and the rejected terms will grow in the proto-ontology construction process. The domain expert and problem owner can revisit the seeding step whenever they think of concepts that should or should not be included in the proto-ontology.

For the example in the plant domain, the domain expert chooses the concepts *Stem*, *Cotyledon* and *Leaf* as initial seed concepts (see Figure 3.6).

Step 4: Extending the set of concepts.

The purpose of this task is to identify relevant terms from a pool of terms associated with the previously defined seed concepts. The method uses either the seed concepts or previously associated and approved concepts to *automatically look up associated terms in the information repository*. The identified terms are of-

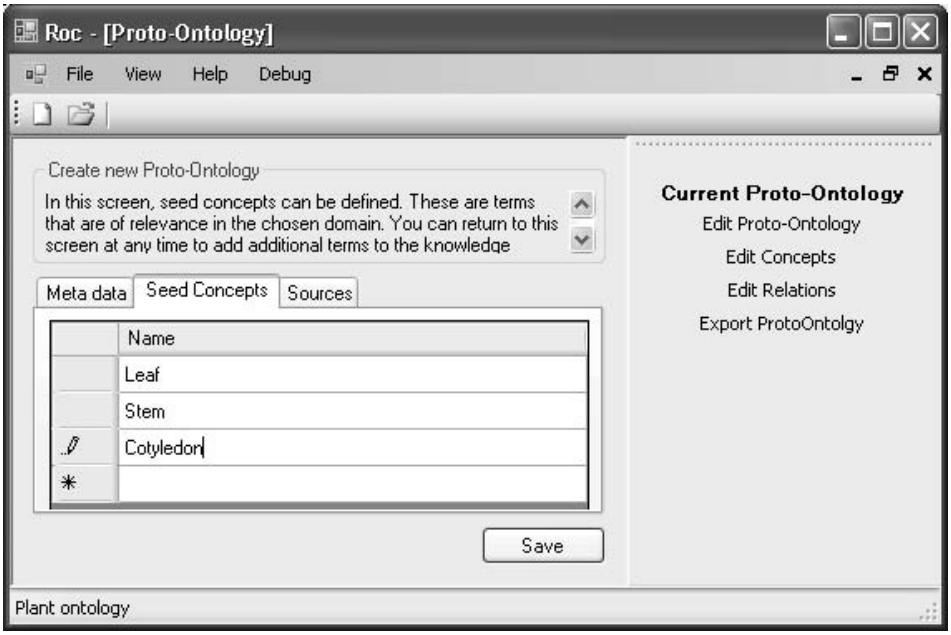


FIGURE 3.6: *A screenshot of the ROC component during the process of seeding the proto-ontology. The domain expert is asked to name relevant terms for the proto-ontology.*

fered to the domain expert and the problem owner for inspection. If a new term is relevant to the domain and task of the proto-ontology, both the domain expert and problem owner accept the term; otherwise, the term is rejected. Accepted terms may be adapted to better reflect the domain knowledge.

If a term from the seed list is not found in the information repository, it is added to the proto-ontology as a single concept. In Step 5, the domain expert can link this concept to the rest of the proto-ontology by defining appropriate relations.

For the plant proto-ontology, the information repository may yield for the seed concept *Leaf* terms such as *Leaf blade*, *Leaf vegetables*, *Leaf area*, *Leaf angle*, *Tomato leaf crumple virus* and *Compound leaf* (see Figure 3.7). The domain expert chooses the latter four, which automatically transfers the non-chosen terms to the list of non-approved concepts. These terms are not offered again to the domain expert. In the example, the problem owner may reject the term *Leaf abscission* as being too detailed and change the name of *Leaf-stem angle* to *Leaf angle*. Next, the domain expert assesses all associations for *Stem* and *Cotyledon*. By updating the list of associations, the information repository identifies all terms connected to the newly approved concepts. The domain expert and problem owner can iterate until they have identified all relevant concepts for the proto-ontology.

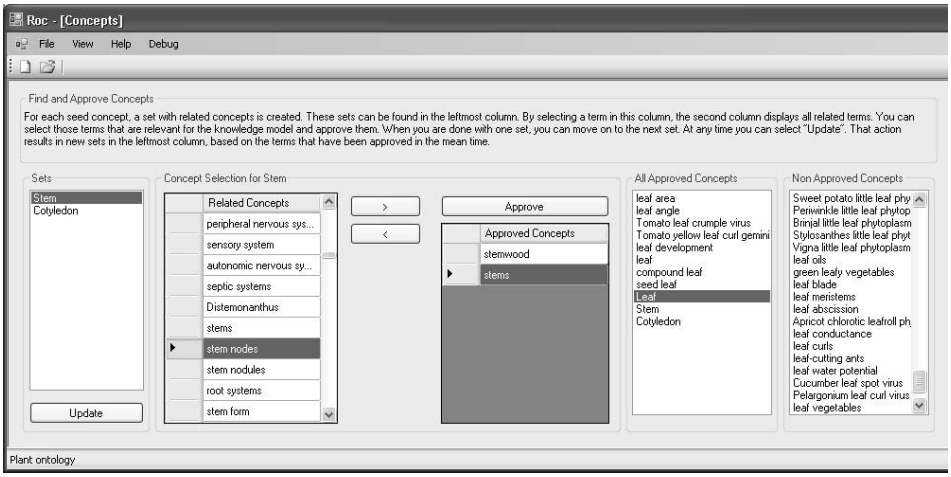


FIGURE 3.7: *A screenshot of the ROC component during the process of identifying concepts for the proto-ontology. The seed concepts and approved concepts are listed on the left. For each concept in this list, the related concepts are identified and presented in the second column. The expert is asked to indicate which related concepts are relevant for the proto-ontology. These concepts can be selected and approved, the other non-selected concepts are automatically added to the list of non-approved concepts.*

Step 5: Adding Relations.

In this task, the domain expert identifies relevant relations and labels them properly. To do this, *relations between approved concepts are retrieved from the information repository and offered for review* to the domain expert. The domain expert is asked to either approve or reject the relations. If a relation is approved, the domain expert can change the label of the relation. The domain expert can also add new relations to the proto-ontology.

With respect to the plant example, the system may present the expert with the relations *Stem – grows from – Plug* and *Stem – develops into – Cotyledon*. The first relation is accepted by the expert, but the relation *Stem – develops into – Cotyledon* is replaced by the relation *Stem – is connected to – Cotyledon*.

Note that monitoring the purpose of the proto-ontology is important for the process of defining labels for the relations. Depending on the type of task, the extent to which the labels have to be specified differs. In some cases, it suffices to know that a relation exists, but the type of relation is irrelevant. In other cases, as *e.g.* the seedling inspection case, the precise specification of the type of relation is required for a useful deployment of the proto-ontology.

At the end of the ROC-process, a proto-ontology has been obtained that reflects the domain expert's task knowledge. Due to the close cooperation of the problem owner and the domain expert in setting the scope, the obtained model is relevant for the knowledge-intensive application.

3.3.4 Case studies for the ROC component

To evaluate the efficiency of the ROC component, we have created proto-ontologies for several application domains. Here, we present two cases that are not related to the seedling-inspection application from the case study. The first case study that we present deals with the creation of a geometry proto-ontology for a drawing program. The second case study creates a proto-ontology for identifying supply chain experts. Below, we briefly present these case studies, discuss the lessons learned and evaluate the efficiency of the ROC component in the knowledge acquisition process.

Case study 1: the geometry proto-ontology

As a first test case for the ROC component, a proto-ontology containing geometry concepts has been created. The problem owner had indicated that the resulting knowledge model should be used to support an imaginary 3D drawing program. In this case study, we used a *preliminary* implementation of the ROC component: the case study was used to improve the ROC-process.

To prepare the ROC repository, we have asked the expert to indicate semi-structured sources relevant for the intended drawing application. The expert mentioned the Geometrical Classroom on Mathworld[7]. Besides this source, we used the sources that were already present in the information repository. Note that these sources were not application specific. We had not yet added the possibility to actively include or exclude sources in this early version of the ROC component. Below we give a short description of the used sources.

- The CABI thesaurus[8], consisting of terms related to applied life sciences.
- The NAL thesaurus[9], containing agricultural and biological terminology.
- OUM, the ontology of units and measures[10], containing units of measure, quantities, dimensions, and systems of units.
- The OpenCyc thesaurus, a generic knowledge base[11].
- Mathworld Geometrical Classroom, containing an overview of geometrical terms, their definitions and the categories to which they belong.

The Mathworld Geometrical Classroom is a semi-structured source for which we have created a tailor-made parser. We have used the structure of the Mathworld page to find triples like <term> isPartOfCategory <category name>, <term> hasDefinition <definition> and <word> isRelatedTo <term>. Examples of identified triples are: *Triangle – is part of category – Polygon*, *Triangle – has definition – A three-sided (and three-angled) polygon*, and *Hypothenuse – is related to – Triangle*. The process of creating the parser and harvesting the triples took

[7] http://mathworld.wolfram.com/classroom/classes/Geometry.html
[8] http://www.cabthesaurus.info
[9] http://agclass.nal.usda.gov.agt/agt.shtml
[10] http://www.afsg.nl/foodinformatics/index.asp
[11] http://www.opencyc.org

the knowledge engineer approximately 0.5 days. The other sources were harvested in a similar way. The retrieved triples have been added to the information repository.

For this early test of the ROC component, both the knowledge engineer and the domain expert were present at the proto-ontology creation process. The knowledge engineer operated the ROC system, and the domain expert provided the input. In this first implementation of the ROC component, the purpose was only defined in terms of a global description of the application and of the domain of interest.

The domain expert started the seeding process with two concepts in the seed list: *cubes* and *cylinders*. In the 'concept step' these terms were looked up in the repository. In this first implementation of ROC, we did not distinguish between the 'concept step' and the 'relation step'. Therefore, the domain expert was asked to assess the statements and to adjust relations and concepts when necessary. At certain points in time, the created intermediate proto-ontology was visualised; the knowledge engineer manually mapped the concepts in the information repository to classes and the relations to properties and used the TGViz plug-in [12] of Protégé to show the intermediate proto-ontology to the domain expert. The domain expert needed thirteen iterations to reach a satisfactory proto-ontology. For this process, approximately 20 hours have been used by the domain expert and 25 by the knowledge engineer. The resulting proto-ontology contains 453 triples.

Case study 2: the supply chain proto-ontology

For a knowledge institute, it is important that the expertises of its employees are known to properly answer questions from *e.g.* journalists. We have developed a prototype system for Wageningen University and Research Centers in which a search term can be entered to find the corresponding expert.

For the prototype of the expert finder system, we have performed a pilot study that focused on the areas of expertise of 'agri-food supply chains'. To this end, we have invited a domain expert in this area to participate in a number of ROC sessions. In these sessions, two knowledge engineers were present: one to guide the domain expert through the ROC process, the other to operate the second version of the preliminary ROC tools.

To prepare the information repository, the domain expert was asked to identify relevant Web-based sources on 'supply chains'. The expert indicated that 'chain logistics' and 'supply chain management' are more appropriate terms and presented us with relevant sources. Below we give a short description of the identified sources.

- The CABI and NAL thesaurus as in the first case study.

[12] `http://users.ecs.soton.ac.uk/ha/TGVizTab/`

- The MeSH vocabulary[13]: a controlled vocabulary in the area of life sciences.
- The UMLS vocabulary[14]: a controlled vocabulary in the biomedical sciences.
- The AGROVOC thesaurus[15]: covering concepts in the agri-food domain.
- Agrologistics list[16]: a structured list with terms in agrologistics.
- Sustainability list[17]: a structured list with terms in sustainability.
- Expertise list: a structured list with the expertises of Wageningen UR.
- Wikipedia supply chain management[18]: containing information on supply chain management.

A tailor-made parser for the last three sources was used to harvest triples. This parsing process resulted in triples of the form <term> isSubcategoryOf <chain logistics term>, <term> isSubcategoryOf <supply chain management term>, and <term> isRelatedTo <term>. Examples are *Cost-benefit analysis – is related to – costs*, *Chain integration – is a subcategory of – organisation*, and *Food safety – is a subcategory of – chain transparency*. The other sources were available in SKOS-format and could therefore directly be added to the information repository.

The process of identifying appropriate additional knowledge sources and writing parsers for these sources, took the domain expert 0.75 days. The harvested statements were added to the information repository.

The purpose of the proto-ontology was defined as *being useful for expert identification within Wageningen UR*. The proto-ontology domain was *supply chains* and *food domain*, the expert type of end users was defined as *general public*.

The domain expert started with a seed concept list of 49 terms specified in Dutch. These terms were translated into English. The translation to English terms was needed since the used sources were partly in English. The remainder of the first session was used for the 'concept step'. All terms that were automatically retrieved from the information repository were presented to the domain expert in separate sets; each set centered around a seed concept. The advantage of this set-based way of presentation is that the list of retrieved terms is presented to the domain expert in manageable chunks instead of in an overwhelmingly large list. For each set, the domain expert checked all terms and indicated whether they had to be included in the proto-ontology.

The proto-ontology construction step was concluded after a second iteration of the 'concept step'. Since the purpose defined for this task did not require any further specification of the relations – a simple 'has-relation-with' label sufficed

[13] http://www.nlm.nih.gov/mesh/filelist.html

[14] http://www.nlm.nih.gov/research/umls/documentation.html

[15] http://www.fao.org/aims/faq_aos\#30.htm

[16] Internal reports 'kenniskaart agrologistiek en visie agrologistiek' (in Dutch), and 'platform agrologistiek' (in Dutch)

[17] Internal report 'Vitaal en samen' (in Dutch)

[18] http://en.wikipedia.org/wiki/Supply_chain_management

– the 'relation step' was not entered. In total the process of creating the proto-ontology took 5 hours for the domain expert and a little less than 17 hours for the knowledge engineers. The resulting proto-ontology contains 248 triples.

Experiences and lessons learned for the ROC component

One of the goals of the ROC component is to support domain experts in optimally performing their task of defining concepts and relations between concepts, while staying in their frame of reference. The domain experts indicated that the use of 'term – relation – term' statements was clear to them. Both experts[19] were well capable of performing the knowledge specification activity within the predefined knowledge format. They indicated that the separation of roles of domain expert and knowledge engineer was satisfactory, since it made sure that the domain expert was responsible for the knowledge specification part of the process.

In the first case study, we noted that the combination of the concept step and the relation step was cumbersome for the domain expert. He had to look at the visualisation of the intermediate proto-ontologies to recall what earlier decisions had already been made. In the second implementation of ROC, the separation of these steps was embedded. The domain expert in the second case study indicated that already discarded concepts should not show up in relation to other concepts in other sets. As a result, we adjusted the method (step 4) in such a way that a list of discarded concepts is maintained and a filtering step removes such concepts from the sets that are offered to the domain expert.

3.3.5 Evaluation of the ROC component

We evaluate the ROC component with respect to its original objectives and the chosen solutions.

Firstly, we claimed that the ROC component is more efficient than interview-based-only knowledge acquisition, since it allows the problem owner, domain expert and knowledge engineer to focus on their own strengths: the problem owner sets the scope in close cooperation with the domain expert; the domain expert gathers knowledge and creates a proto-ontology; the knowledge engineer focusses on the knowledge modelling aspects. We can evaluate the ROC component with respect to the first two parts of this process. The last part has not been elaborated yet. To evaluate the efficiency of using ROC compared to only using interview-based knowledge acquisition, we performed a cost-benefit analysis in which we compare proto-ontologies developed with respectively without the ROC-component.

Secondly, the ROC-component aims to support the domain expert in gathering the required knowledge and benefitting optimally from existing sources. To see whether the use of multiple sources in the information repository indeed led to reuse of existing knowledge in the proto-ontology, we performed

[19] These experts were academically trained, but had no previous experience in the field of knowledge modelling.

an analysis on the two proto-ontologies from the presented case studies. The usefulness of the information repository can be measured by identifying how often the automatically proposed statements are selected by the domain expert and how often the knowledge engineer added knowledge not taken from the repository.

Finally, we intend to use the ROC-component as a means of creating task-specific knowledge models. We evaluate the usability of the created knowledge model for one of the examples, namely the expert recommendation application for which the supply chain proto-ontology was developed.

The above mentioned evaluation methods provide us with a preliminary judgement of the potential of the approach. The full evaluation of the interview- and observation-based method complemented with the ROC component for application ontology construction awaits the development of the knowledge engineering part of the ROC component.

Comparison of interviews and ROC

To obtain an impression of the costs and benefits of the ROC component, we compared a ROC-only method to interview-based-only proto-ontology creation. In other words, we compare the effort for creating a number of interview-only proto-ontologies with the effort to create proto-ontologies with the use of the ROC-component.

Within our group, we have developed for example the Plant Ontology for the seedling-inspection task, the Healthy Food Components Ontology[20], and the Potato Ontology [43] using interview-based knowledge acquisition. We can compare these with the Geometry Ontology and the Supply Chain Ontology as described in the ROC use cases. In Table 3.1, an overview of these ontologies and their development efforts is given. We see that both the knowledge engineer and the domain expert are involved in the knowledge acquisition process, be it in the interview-based or in the ROC-based component. The problem owner was not explicitly included in the process. The process of creating the potato ontology and the supply chain ontology are most similar, since in both cases only one domain expert from only one discipline was interviewed and the proto-ontologies are of comparable size. The geometry proto-ontology was hampered by the immature character of the initial ROC tools and is therefore less suited for comparison.

When we compare the construction of the supply chain ontology with the construction of the potato ontology, we see that for the supply chain ontology, the knowledge engineer needs three times the time of the domain expert, whereas in the potato proto-ontology this ratio is 1 to 4. Although these results are far from statistically conclusive, they suggest that the ROC component could reduce the amount of time required by the knowledge engineer to develop the proto-ontology. It is indeed one of the design criteria of ROC to not require the knowledge engineer to study the domain of the proto-ontology. The numbers for the domain expert are more difficult to interpret. In the supply chain

[20]The application based on this ontology can be found at `www.afsg.nl/icgv`

proto-ontology	# sessions	$\sum$ time DE	$\sum$ time KE	total time	# concepts	# disciplines
Plants	13 (13 DEs)	58 hrs	88.5 hrs	146.5 hrs	37	2
Food components	4 (4 DEs)	10 hrs	30 hrs	40 hrs	120	1
Potato	8 (1 DE)	8 hrs	32 hrs	40 hrs	279	1
Geometry	12 (1 DE)	20 hrs	25 hrs	45 hrs	208	1
Supply chains	2 (1 DE)	5 hrs	16.5 hrs	21 hrs	236	1

TABLE 3.1: *In this table, an overview is given of the properties of three manually developed proto-ontologies and the two proto-ontologies covered in the case studies (Geometry and Supply Chains).*

proto-ontology, the domain expert is involved in fewer sessions of the ROC component than for the development of the traditional interview-based knowledge acquisition for the potato ontology. Whether this is a generic trait of the ROC component is still to be seen. The total time used to develop the potato respectively the supply chain ontologies differs a factor of two in favour of the ROC component.

Evaluation of ROC in an application

One of the evaluation measures of a proto-ontology is to see how well it supports an envisaged application. For the geometrical case study no specific application was created. For the second case study, we developed an expert recommendation system that can be used to identify experts in the field of supply chains.

For the expert recommendation system, we assumed that experts publish on subjects that are within their area of expertise. These publications can be used to create individual 'fingerprints'[21], containing a set of characteristic terms describing the expertise of each expert. To link a search term to a fingerprint, we need ontologies for a number of expert domains. If, for example, a journalist needs information about avian influenza, the appropriate expert will probably be known as expert on bird diseases. When only text based search is used, the journalist will not find the desired expert. When a (light-weight) ontology is used in which the link between avian influenza and bird disease is made, the corresponding expert can be identified.

To find an expert, a user enters a free text string indicating the topic for which the expert is required. This term is matched with the terms in the proto-ontology. If a matching term is found, its *related terms in the proto-ontology* are collected. The application uses these terms to scan the publications' fingerprints for the original search term and its related terms. The authors from the publications are identified and ranked. The user can see which related terms have been found, and which experts best match the original query.

To evaluate the usability of the supply chain proto-ontology, we selected ten terms from the proto-ontology. With this set of terms, we queried the system for experts. Next, we checked with the involved domain expert whether the Top 3 of scientists returned are indeed experts in the indicated areas. For the terms *food supply*, *supply chains*, *supply chain management* and *food production*, the

[21] `http://www.collexis.com`

(a) Case 1: Geometry

Source	# Triples
CABI	2
Mathworld	164
OUM	8
Domain expert	279

(b) Case 2: Supply Chains

Source	# Triples	Source	# Triples
CABI	81	Intranet sustainability	3
AGROVOC	61	Intranet agrologistics	2
Wikipedia	42	UMLS	1
MeSH	30	Domain expert	0
NALT	28		

TABLE 3.2: *Contribution of sources to total triple size of proto-ontology.*

identified experts were the persons expected by the domain expert (*i.e.* high precision). For the terms *logistics* and *food safety* the domain expert did not know all identified experts personally. After looking into the experts' affiliations, though, the domain expert concluded that it was reasonable to assume that the unknown persons were indeed experts in the indicated fields. The terms *trade barrier* and *chain governance* were on the border of the expertise of the domain expert; he could not give an indication of the correctness of the selection. The terms *quality* and *networks* are important for the food supply field of expertise, but also have a meaning in other areas. The found experts indeed related to these terms, but were not specifically linked to the area of supply chain quality or networks. Overall, the expert finder tool seems to indicate the expected experts or related persons in the expected departments. This suggests that the proto-ontology supporting the expert finder tool fulfills its expectations well.

Use of the information repository

In this section we look at the amount of information from the repository that was actually reused in the case studies. We see in Table 3.2 that in the geometrical case, the domain expert has mainly taken concepts and associations from Mathworld, and has added many new statements (*e.g.* equations, parameters, etc.). The skewed ratio between reused and new concepts was caused by (1) spontaneous associations by the domain expert at the presented concepts – a desirable effect – and (2) a limited amount of available dedicated statements in the information repository. This shows that identifying sufficient and adequate sources is important to profit optimally from ROC. For the supply chain case more sources were available. The ratio between triples from these sources is more balanced and no new concepts were added by the domain expert.

3.3.6 Conclusion

We have observed that the ROC component may accelerate proto-ontology construction by supporting different players in the process. First, the problem owner is assisted in defining the application context. Second, the domain expert specifies a proto-ontology without being hindered by technical modelling details. Third, the time spent by the knowledge engineer to get to know the domain is minimized. With ROC, association rate, focus and readability during ontology development is enhanced. Existing knowledge sources are used from

the start of the construction process and the purpose of the proto-ontology is continuously monitored. Furthermore, we use natural language-like statements generated from a triple format as intermediate representation.

Even though ROC and its tools are still under development, we have already used them successfully. In the two case studies we have constructed proto-ontologies in a relatively short time. The evaluation of the results shows that combining multiple sources works well, as they all appear in the resulting proto-ontology. Not many additional triples need to be added by the domain experts when sufficient reusable sources are available. We also analyzed the costs and benefits of developing ontologies for five cases. Two of these ontologies were developed with ROC, the other three with an interview-only method. The comparison shows that the time needed by the knowledge engineer and to a lesser extent the domain expert is reduced by using ROC. Measuring the quality of the proto-ontology, though, remains difficult and is ultimately expressed by the effectiveness in applying the model in some context.

Future developments around the ROC-component involve the following aspects:

- An aspect of ROC that needs further attention is ensuring that domain experts stay motivated during the process. This can be achieved by certifying that the domain expert is committed to the intended application and by minimizing the amount of manual editing.
- Another issue we will attend is to include the selection of appropriate domain experts as a step in the ROC component, not unlike choosing appropriate text sources.
- A third extension that we presently investigate is the inclusion of existing triple extraction tools in the ROC toolkit.
- The fourth proposed addition to the ROC component is a proto-ontology structure dashboard giving continuous feedback on a number of performance indicators, to guide the problem owner and domain expert in creating balanced, high-quality proto-ontologies.
- The fifth aspect is to systematically support the identification of ontologies that may serve as reusable sources based on a problem description provided by the domain expert.
- Lastly, the steps for refinement of a proto-ontology towards *e.g.* a full-blown OWL model, a SKOS model or any other form requires additional work.

3.4 Discussion and conclusions

In this chapter, we have discussed the knowledge elicitation process to obtain application ontologies for knowledge-intensive computer vision tasks. The discussed methods can be used to obtain other application ontologies as well.

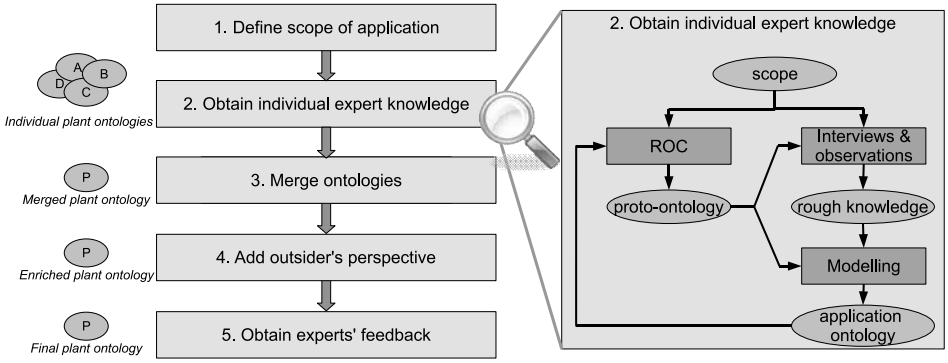


FIGURE 3.8: *A schematic representation of the interview-based knowledge acquisition process enhanced with the ROC-component, illustrated for the plant application ontology case study. This process can be used as a template for any interview-based knowledge acquisition for multi-expert, multi-domain, task-specific situation.*

Ontologies for knowledge-intensive computer vision tasks typically require task-specific, multi-expert and multi-domain knowledge. We have proposed an interview-based method that can deal with these aspects. First, the scope of the application is set and task-specific knowledge from individual experts is obtained. Then, the created knowledge models are merged into one multi-expert ontology. Next, expert(s) from other domain(s) are asked to provide their input and finally the created knowledge model is presented to all experts for feed-back and error correction.

Setting the scope for a knowledge-intensive computer vision task is a process that takes place in three dimensions: the application perspective, domain perspective, and discipline perspective are to be considered. The problem owner covers the application perspective and the domain expert the domain perspective. The problem owner is involved in the scoping process on all application-domain slices, the domain expert is only concerned with the discipline that he is expert in. By setting the scope explicitly and ensuring that checking the scope is relevant in the whole knowledge acquisition process, the problem owner ensures that the created knowledge model is dedicated to the application.

For obtaining individual expert knowledge, interview & observation-based knowledge acquisition is a good approach. This approach has the benefit that domain experts are involved in the process. The effect is that the experts work with the knowledge engineers to create a high quality application ontology. By involving multiple experts, it frequently happens that the tacit knowledge of one expert is covered by the explicit knowledge of another expert. As a result, the coverage of the application domain is better as well. The involvement of experts from different domains that are relevant for the task ensures that the created ontologies are well suited for the application.

However, ontology construction only based on interviews has some disadvantages. Interview sessions allow the knowledge engineer to obtain sufficient

understanding of the task domain to identify the knowledge needed for the modelling process. It would be more efficient if the domain expert could fulfill this role. Moreover, knowledge models built in interviews assume creation from scratch, even when reusable knowledge sources are available. The ROC-based ontology construction method was developed to deal with these issues.

The ROC component helps the domain expert in expressing his knowledge by giving suggestions for relevant terms. It prompts the domain expert with related terms taken from existing preprocessed knowledge sources. The preliminary evaluation of ROC indicates that the ROC component indeed results in reuse of existing knowledge, efficiency in the knowledge acquisition process and task-specificity of the created proto-ontology.

The ROC component in its present form has some disadvantages. First, if no reusable knowledge sources can be found, the ROC component can obviously not be used. Second, task experts may find the process of creating a knowledge model on their own less appealing than participating in an interviewing session. At present, we have no experience with collective ontology development. We expect that this process would be more appealing for experts. Finally, the quality of the proto-ontology cannot yet be checked automatically.

To combine the benefits from both approaches, we propose to combine interview-based methods with the ROC-component in the same process (see Figure 3.8). The scope of the application is input for the interview & observation process or the ROC-process. The result of the ROC method is a proto-ontology. This proto-ontology can be sufficiently rich to immediately create an application ontology. In some cases, though, the proto-ontology will be used as input for the interviewing process. The domain expert and knowledge engineer can focus on some parts of the proto-ontology that have not been covered yet. The application ontology obtained from the interviewing process can serve as input as one of the reusable sources in the ROC component. The domain expert gets the opportunity to enrich the application ontology with existing sources. By choosing this setup for the knowledge acquisition process, we use the benefits of the interview approach and combine them with those of the ROC component.

The method proposed in this chapter is well-suited for creating application ontologies for knowledge-intensive computer-vision tasks, and even more widely, for knowledge-intensive tasks in general. It gives domain experts an active role in the knowledge capturing process, ensures that the created ontology is on target by involving the problem owner in the process, and enables the knowledge engineer to focus on the modelling process.

Chapter 4

Transparent Procedural Knowledge

In agriculture, automating knowledge-intensive tasks is a frequently occurring problem. Task-performing software is often opaque, which has a negative impact on a system's adaptability and on the end user's understanding and trust of the system's operation. A transparent, declarative way of specifying the expert knowledge required in such software is needed.

We argue that a white-box approach is in principle preferred over systems in which the applied expertise is hidden in the system code. Internal transparency makes it easier to adapt the system to new conditions and to diagnose faulty behaviour. At the same time, explicitness comes at a price and is always bounded by practical considerations. Therefore we introduce a method to find a balanced decision between transparency and opaqueness. The method proposed in this paper provides a set of pragmatic objectives and decision criteria to decide on each level of a task's decomposition whether more transparency is sensible or whether delegation to a black-box component is acceptable.

We apply the proposed method in our case study and show how a balanced decision on transparency is obtained. We conclude that the proposed method offers structure to the application designer in making substantiated implementation decisions.

In this chapter, we focus on the research question "How can we systematically decide on the right level of transparency in the design of knowledge-intensive computer vision applications?" We show how the transparency decision depends on (secondary) task objectives. A paper 'Transparent Procedural Knowledge for Computer Vision Applications' which covers this chapter has been submitted for publication.

4.1 Introduction

In traditional applications, the software written to perform a task is opaque in the sense that the knowledge used in the application cannot easily be identified in the code. Such a black-box setup works well in those cases where the role of the software developer and the domain expert completely overlap and when the developers remain responsible for the application over its entire lifetime for all use cases. However, the central message of this thesis is that for knowledge-intensive applications, opaque software is not satisfactory. The considered domains are hard to master for software engineers, new application areas and unforeseen conditions may arise over time. Opaqueness makes it hard to maintain and adapt such a system.

In Chapter 2, we have focused on using explicit descriptive knowledge to get a clear image of the task and domain of the computer vision application. We have created a decomposition of the inspection task into subtasks and intermediate models. For this decomposition, we have chosen to follow the CommonKADS method [95] of decomposing until the primitive task level has been reached. In this chapter, we introduce the idea that the desired level of transparency depends on the secondary objectives of the application. For some applications, a decomposition of the task until the primitive level is not necessary and perhaps not even desired.

In this chapter, we focus on designing a computer-vision method that is transparent *to the level that is feasible and sensible in practice*. This chapter is organised as follows. We set out general objectives that ask for a degree of transparency. We next propose a set of criteria that can be used to decide for a component in the application whether transparency is desired. These criteria are used as guidelines to support the making of practical design choices. Moreover, we introduce mechanisms for actually adding transparency to the targeted component. We apply the proposed method to our case study in Section 4.4 and evaluate how the criteria help us to design an application with the desired level of transparency in Section 4.5. We conclude in Section 4.6.

4.2 Related Work

The subject of knowledge-intensive computer vision has been studied in the fields of machine vision, machine perception, robotics and cognitive vision, since each of these disciplines deals with complex information that has to be interpreted in real-world terms. For the context of this chapter, we focus on computer vision applications that use explicit procedural knowledge.

Procedural knowledge is used on various levels of explicitness. Crowley *et al* [18], for example, have chosen to use case based reasoning for their system that can independently diagnose and repair itself. Cases of repair strategies are used to apply procedural knowledge to a problem; the knowledge itself, though, is embedded in the cases and is not formulated explicitly. The work of Clouard *et al* [16] focuses on automatic generation of image processing programs. The resulting computer vision code does not explicitly reflect the domain expert's

procedural knowledge either. Albusac *et al*[2] use both an explicit description of the domain (traffic situations in a surveillance system) and knowledge rules that are stated in terms of expert knowledge. An example of such a rule is 'if object is not <person, group of people> AND object speed is <slow, medium> AND sidewalk intersection degree is <very low, low> then situation is normal'. The concepts 'person', 'group of people', or 'slow' are understandable terms for a domain expert. They explicitly stress that allowing the domain expert to understand the system output is very important. In that sense, their work is closest in spirit to our work.

4.3 Implementing Transparency

Introducing transparency in an application automatically implies making choices as to the level where adding transparency stops. Transparency is a property of a software system that is not strictly required for a correct functioning of the application. We have to make a decision on the level to which transparency is beneficial for the system. After all, from a certain level additional explicit knowledge introduces unwanted overhead or may even cause confusion for developers and users. Instead of blindly adding more transparency, the use of black-box components may be the best implementation choice at certain points in the application.

To get a clear image of the trade-offs between black-box and transparent design, we need to take a step back from technical considerations and focus on underlying design objectives and criteria.

4.3.1 Design Objectives

Software systems have to comply to both functional and non-functional requirements. The following requirements are relevant for our discussion: (i) robustness and reliability, (ii) trust, and (iii) speed. A system should be *robust and reliable*. This means that it should handle all possibly encountered situations properly, and signal cases that fall outside the scope of the system. This requirement asks for a scope of the system and its subsystems that is modelled as accurately as possible. Moreover, the system should be *trusted* by its end users. Transparency of the system can help end users to build confidence in the application. Another objective is that the system should respond *fast enough* to a given input.

Besides affecting these general system properties, transparency may also contribute to the support of tasks associated with the system and its development. Supporting these tasks is not vital for the execution of the task per se, but it provides the users and developers of the application with additional benefits. We consider two types of tasks that are of interest for our discussion.

The first task is system modification [17, 35], entailing maintenance, modification, trouble shooting, correction, testing, diagnosis, et cetera. These tasks aim at sustained system improvement. Task support can be given by tools that detect the cause of an error and point to its location in the code or by tools

that pinpoint modifications needed for a new application area. In general these tasks can be said to contribute to the objectives *corrigibility* and *adaptability*.

The second task is organisational learning [21], including education, discussion, elicitation, externalisation, et cetera. This refers to all tasks that are performed by the system to lift the level of *human* expertise in the organisation, summarised as the objective *understandability*. Here, task support builds on the fact that system behaviour is paired to explanation of the underlying reasoning. For example, novices in a particular field of expertise can learn to see the effect of decisions made by the system, reflecting consented knowledge from leading experts. Another example is that other experts may become aware of the implicit reasoning they apply in cases similar to the ones demonstrated by the system.

We submit that a clear view on these tasks helps to properly set the objectives for transparency.

4.3.2 Transparency criteria

The decision on whether or not to further specify task-specific knowledge depends on (secondary) objectives of the application. To help decide where and to which level transparency is required and feasible, we present the following set of criteria:

Availability of explicit domain expertise.

If the expert knowledge required for the component can be modelled explicitly, then the component is eligible for explicit specification. Some tasks are typically implicit and hard to express verbally. For example, it is difficult for people to express how they recognise faces.

Application range.

This factor is related to the scope of the considered software component, both in terms of task and domain. The application range indicates the envisaged future tasks and domains for which it might be applied. A specification of the input and output types that the component covers is asked for.

Common sense and trivial knowledge.

Not all knowledge underlying an application should be made explicit. Detailing trivial facts (for example how an average value is computed) clutters the system with irrelevant knowledge and makes it less transparent.

Explanation.

In order to support tasks such as diagnosis and education, the system should be able to provide a clear explanation of its reasoning process. It may be necessary

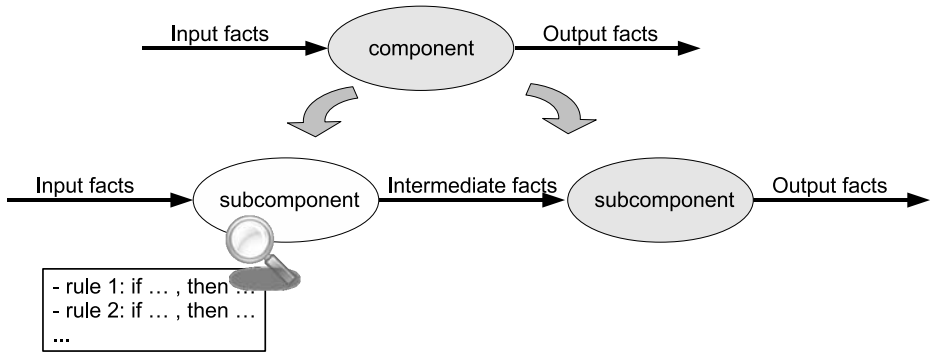


FIGURE 4.1: *A graphical representation of adding transparency to an application. A component can be broken down into subcomponents. A component can be explicitly defined by expressing its internal workings in a declarative fashion. Conceptual refinement is not displayed in this picture.*

to give more information about the underlying knowledge than strictly needed to perform the original task.

Third party expertise and trust.

It often occurs that components developed by a third party are used. System developers may or may not trust the origin of such a component. If one believes that the component performs as promised, no need for further specification exists. For example, many predefined mathematical computing libraries from respected software developing companies have been validated and verified extensively. Third party components are often optimised for speed of execution, memory footprint and scalability.

The identified criteria are a guideline in deciding on the correct level of transparency. They offer structure in designing knowledge-intensive computer vision applications, but some aspects of the decision process will be undecided by the criteria and remain subjective.

4.3.3 Realisation

When the criteria from the previous section indicate that transparency is asked for, we have to actually realise transparency in the component. To encode a component in a transparent way, the knowledge in the component must be made explicit and encoded separately from the processing code.

For every subtask on every level of the application, a decision on whether more transparency is desired can be made. In this section, we introduce three patterns for making conceptual knowledge explicit (see Figure 4.1).

Task and domain decomposition.

The first manner in which we can add transparency to an application is by performing task and domain decomposition. Task decomposition is a technique that is used in many different areas, such as project management [77, 104], building projects [65], and modularisation in software engineering [109]. A frequently used method is hierarchical task analysis that breaks down the operation at hand into sub-operations [3]. Without decomposition, these would remain hidden. The CommonKADS methodology [95] has formalised this type of decomposition as a way to model knowledge-intensive tasks. At the lowest level of task decomposition it links domain models to the inputs and outputs of so-called inferences. In creating transparency, it is not so much the decomposition of the activity that is important, but the associated identification of intermediate concepts and relations.

Conceptual refinement.

The second way to add transparency to an application is by providing conceptual refinement of its input facts. These input facts are instances of an application ontology. The classes in the application ontology may be too imprecise for the component under inspection, especially when future use of the component is envisaged. If a component is only suitable for a subset of the instances of the input class, then the application ontology should be refined with an additional subclass corresponding to this subset. For example, instead of allowing all plants as input for a component that is only suitable for tomato plants, we create a subclass 'tomato plant' of the class 'plant' and indicate that only instances of the class 'tomato plant' can be used as inputs for the component. Besides refining facts that are related to the input for a component, facts that express the conditions under which the component can be applied also provide transparency.

Logical inferences.

The third manner to add transparency to an application is by providing explicit knowledge rules for the inner workings of a component. Knowledge is essentially a mechanism to derive new facts from existing facts. Logical rules describe the relations between the facts specified in the other two steps. They provide additional insight on how a task is performed, but without resorting to imperative coding. This manner of adding transparency separates the inner workings of a component into explicit expert knowledge and processing steps.

The three methods listed are different ways to add transparency to an application. With the first two methods, descriptive domain knowledge can be explicated; the ontologies in which the domain knowledge is modelled are refined. With the last method, procedural knowledge is made transparent. By defining procedural knowledge in the form of knowledge rules, the relations between

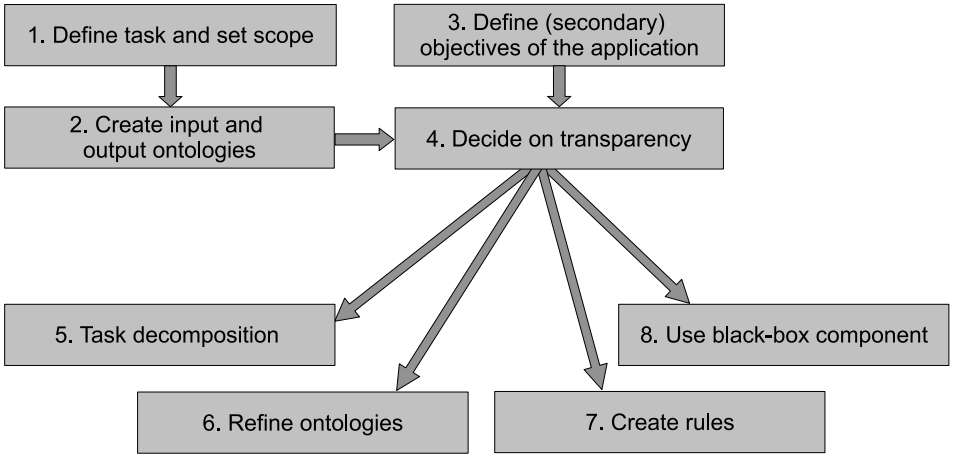


FIGURE 4.2: *A schematic representation of the design process of transparent knowledge-intensive applications.*

consecutive models of an object are made transparent. In this way, both descriptive and procedural expert knowledge can be embedded in the computer vision design.

The proposed design process for knowledge-intensive computer vision applications can be summarised as follows (see Figure 4.2): first the task is defined and the scope is set, as proposed in Chapters 2 and 3, next the input and output ontologies are defined. Then, the secondary objectives of the application are charted, since they are leading in deciding on the transparency of a component. If a component requires additional transparency, either the component and the involved ontologies are further decomposed, the input and output ontologies are further refined, or the inner workings of the component are made explicit using knowledge rules. If no additional transparency is needed, a black-box component is introduced. The process of deciding on transparency is repeated until for all components in the application a decision has been made.

This design process is a further elaboration on the process proposed in Chapter 2 in the sense that two new components – define objectives and decide on transparency – have been added to the design process, and the components task decomposition, refine ontology, and create algorithms (computational steps) no longer have to be applied for all components. Moreover, the new setup distinguishes between white-box logical rules and black-box algorithms, instead of always choosing the black-box computational steps.

The five decision criteria proposed support the designer of a computer vision application in deciding which transparency decision can be implemented using which transparency mechanism:

The criteria concerning *explicit expertise*, *trivial knowledge* and *third-party expertise* can lead to a further decomposition of task and domain:

- The *explicit expertise* criterion investigates whether it is at all possible to express the knowledge in a component in an explicit way. When this criterion is not met, the decision for the component is easy: it has to remain a black-box.
- The criteria concerning *trivial knowledge* and *third party expertise* decide when it is necessary to decompose a task: if the component under inspection is not sufficiently primitive, adding transparency could make sense; if a trusted third-party component is available, then further decomposing of the task may not be necessary.

Based on the *application range* criterion, conceptual refinement is required: this criterion ensures that if a component is only suitable for a limited set of input objects, this fact is stored with the component. By setting conditions on the input objects, we prevent the component from accidentally be used for other objects.

Adding transparency by explicitly defining logical inferences is done based on the criteria *trivial knowledge*, *explainability* and *third party expertise*:

- The *trivial knowledge* criterion indicates that making too detailed knowledge explicit leads to cluttering. The *explainability* criterion indicates that the explication of knowledge contributes to gaining insight in the inner workings of the component.
- The *third party expertise* criterion indicates that existing trusted components can be reused and need not be specified. The combination of these three components allows us to decide on the necessity for formulating procedural knowledge in a declarative format.

4.4 Transparency in the computer vision application

In this section, we apply the decision criteria and the means to create transparency introduced in the previous section to the seedling inspection case study. The computer vision application to assess tomato seedlings on their quality is created with several of the mentioned objectives in mind. Suppose[1], that we are interested in *adaptability*, since that allows the experts to use the application for other objects, such as bell peppers seedlings, or cabbage seedlings, and for other tasks such as grafting. We are also interested in *corrigibility*, since that allows the tracking down of errors in the application. Moreover, we are interested in *reliability*. Whenever a batch of seeds is polluted with seeds from a different species, it is important that this is noticed. Without reliability as property, the different seedlings would simply be rejected as 'abnormal plants', and would not be identified as plants outside the scope. Also *understandability* and *building trust* are desired objectives. The horticultural sector is interested in an application that is trusted by all companies and that can be used to train new personnel.

[1] In reality, we are also interested in the objective *speed* for the application.

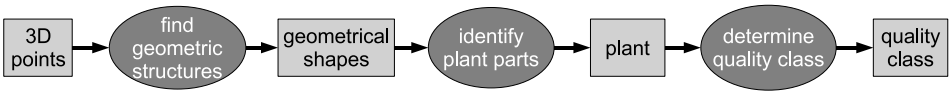


FIGURE 4.3: *A graphical representation of the subtasks and intermediate models at the highest level. Boxes indicate instances of models, ovals represent processing components.*

At the highest level, the application transforms the recordings of the tomato seedlings into quality classes. This task consists of the following steps: (i) a segmentation of the point cloud to obtain a representation of the object in geometric shapes, (ii) an interpretation of the geometric shapes in terms of plant parts, and (iii) a classification of the plant into a set of quality classes (see Figure 4.3). For each of the steps, we single out a number of components. For each of these components, we use the five transparency decision criteria to decide whether more transparency is required for the component, and show how the decision can be implemented. For demonstration purposes, we use a pseudocode representation of the required knowledge rules. The actual implementation takes place using semantic web technologies such as OWL [80] and Jess [33]. OWL is used to encode ontologies, Jess is used to encode declarative rules. We use Java to deal with the opaque components in the application. We show the actual implementation for the example 'create point group'.

4.4.1 Selected subtasks of the segmentation component

The segmentation component has a set of points as input and a set of geometric shapes that are closely connected to plant parts as output. This component is decomposed in a number of subtasks. We elaborate three of them: (i) create point group, (ii) determine point type, and (iii) determine thick cylinder region.

Create point groups

The first component that we discuss performs the subtask labeled as 'create point groups'. For each point P, it results in a point group that has point P as its central point and that contains all points P_n that are close enough to P as neighbours. We want to decide whether this component needs further specification or not. We can apply the five criteria.

First, the available of *explicit expertise* is studied. The knowledge required to make this component work is covered in an explicit manner by the computer vision expert. Hence, this criterion is met; the means to create a white-box component are available. Second, we consider the *application range*. As far as we can foresee, the component is applicable for all computer vision tasks in which an object is recorded as a point cloud and for which the point cloud has to be segmented. Assessing bell peppers and cabbages – or even airplanes and chairs – does not differ for this component from assessing tomatoes, and neither do quality assessment and grafting. Therefore, we state that the application range is sufficiently met; no further specification of the input and output of the

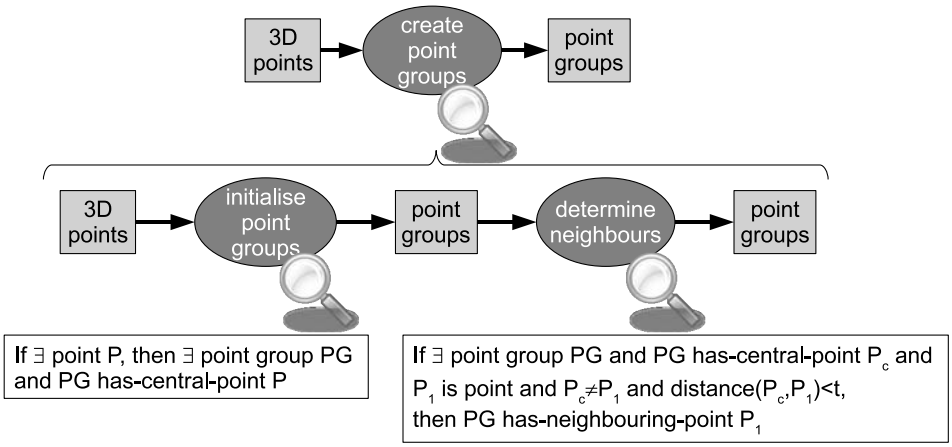


FIGURE 4.4: *A graphical representation of the specification of the 'create point group'-component.*

task is asked for. Third, we consider the *triviality level* of the knowledge in the component. At this level of the application, the knowledge needed to create point groups and identify neighbours is domain-dependent and hence not trivial. The definition of when a point is a neighbouring point is not made explicit yet. Based on this criterion, it makes sense to further decompose the task into subtasks. Fourth, the need for *explaining* the internal workings of the component is considered. For the plant expert, explicit knowledge of this component is not required. Explicit knowledge will not facilitate training or acceptance, since this component is outside the scope of the plant expert's knowledge. Finally, we are interested to know whether a third-party component is available that can be used. At this level of the application, no reusable third-party tools exist. A further decomposition of the component would lead to more primitive components that may be obtained as third-party components.

So, based on the expectation of more trivial knowledge and the possibility to reuse existing components on a lower level, we decide to further decompose the 'create point groups' subtask into more detailed subtasks (see Figure 4.4). The subtasks identified in the decomposition are (i) 'initialise point groups' and (ii) 'find neighbouring points'.

The 'initialise point groups' component indicates that for each point a point group must be created with the point as central point; no further decomposition is possible, no existing third-party tools are available. Expressing the knowledge in a white-box fashion does not really contribute to explainability, but does not clutter the code either. We conclude that the decision criteria give no clear indication on whether a white-box or a black-box expression would be preferred. In this case, we make the (subjective) decision to create a declarative implementation of this component:

If ∃ point P, then ∃ point group PG and PG has-central-point P.

For the second subtask, 'find neighbouring points', the distance between points

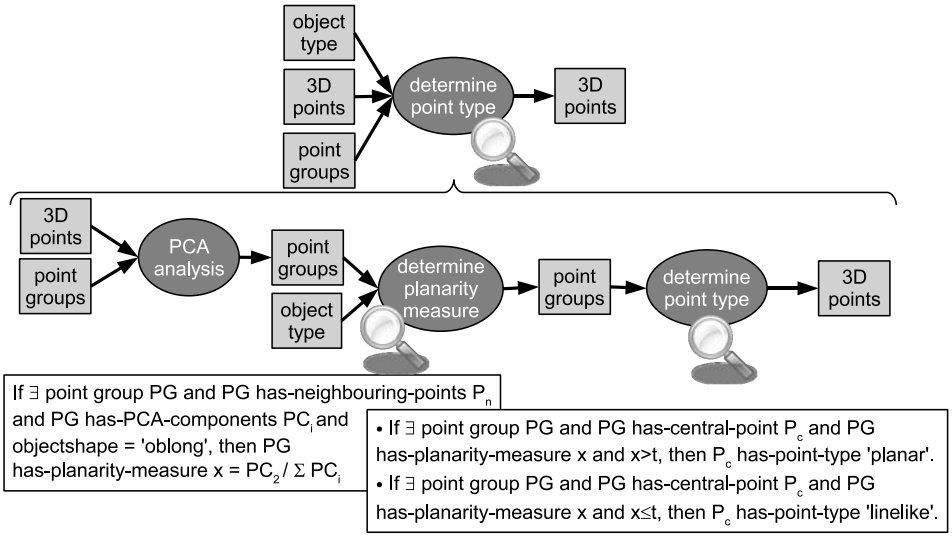


FIGURE 4.5: *A graphical representation of the specification of the 'determine point type'-component.*

has to be calculated. This functionality is available as a trusted third party tool and can be reused. The actual assigning of points to a point group is based on a domain-specific threshold value t on the distance. We therefore specify it in a declarative fashion with a call to the black-box component 'distance':

If $\exists$ point group PG and PG has-central-point P_c and P_1 is a point and $P_1 \neq P_c$ and $\underline{\text{distance}}(P_1, P_c) < t$, then PG has-neighbouring-point P_1.

Determine point type

The second component that we discuss is the subtask 'determine point type'. This component determines for an input point P, belonging to a point group PG the value for 'point type'. This value can be linelike or planar. The relevant criteria for this component are the application range, the triviality of the knowledge, and the third party components.

At this level of decomposition, the component is not yet so primitive that we can use third-party trusted software or that the knowledge is so trivial that there is no use for further decomposition. We therefore decide to zoom in on the component and look at it in more detail (see Figure 4.5). The component can be decomposed into three subtasks: (i) 'perform a PCA analysis', (ii) 'calculate the planarity measure', and (iii) 'decide on the point type'.

Performing a PCA-analysis is a standard procedure for which adequate third-party components are available. There is no need to further specify this component. Calculating the planarity measure, on the other hand, is a domain specific component; depending on the type of objects under inspection,

the equation may change. With respect to the application range, we note that the component is only valid for seedlings with oblong leaves. For example for banana plants (with elongated leaves) and Christmas cacti (with round leaves), the calculation of the planarity measure will probably need a different equation or threshold value. For this component, we refine the value of the 'has-leaf-shape' attribute of object-type from 'any' to 'oblong'. It makes sense to make the inner workings of the component 'calculate the planarity measure' explicit:

If $\exists$ point group PG and PG has-neighbouring-points P_n
and PG has-PCA-components PC_i and object-type has-leaf-shape 'oblong',
then PG has-planarity-measure $x = \frac{PC_2}{\sum PC_i}$.

Deciding on the point type involves a simple comparison with respect to a domain-specific threshold value t. We choose to express the component in a declarative fashion, so that the use of the domain-specific threshold value is explicitly available. The corresponding knowledge rules are:

If $\exists$ point group PG and PG has-central-point P_c and
PG has-planarity-measure x and $x > t$,
then P_c has-point-type 'planar'.

If $\exists$ point group PG and PG has-central-point P_c and
PG has-planarity-measure x and $x \leq t$,
then P_c has-point-type 'linelike'.

Determine thick cylinder region

The third component that we discuss is the subtask 'determine thick cylinder region'. For this component, the planar regions are used as input, the region that most resembles a cylinder with a fixed radius is selected and identified as a thick cylinder shape. The relevant criterion for this component is the criterion concerning third-party expertise; the other criteria do not give a preference for white-box or black-box. The region that best resembles a cylinder can be determined by performing a Levenberg-Marquardt non-linear least squares fit to the points in the region. This function is available as a third party component. We therefore decide to implement this component in a black-box fashion using the trusted Levenberg-Marquardt algorithm.

4.4.2 Selected subtasks in the interpretation subtask

The main task, assessing the quality of the seedling, has an 'interpretation' subtask as component (see the introduction of Section 4.4). This component has as input a set of geometrical shapes and as output a plant model consisting of a set of plant parts with quality features determined. Based on the criteria, we further decompose the subtask into subcomponents. In this section, we discuss the component 'recognise plug body'.

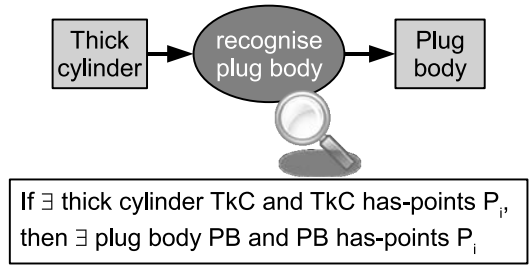


FIGURE 4.6: *A graphical representation of the specification of the 'recognise plug body'-component.*

Recognise plug body

One of the components in the interpretation subtask is the subtask that recognises the plug body. This task has as input a thick cylinder shape consisting of a number of points, and as output a 'plug body'-plant part consisting of the same points. The computer vision expert has sufficient knowledge to provide an explicit description of the component. As far as the application range criterion is concerned, this component is applicable to all plants that grow in rock wool plugs. In all foreseeable automated inspection applications, seedlings grow in these plugs. Therefore, this criterion has no preference for a white-box or a black-box implementation. The knowledge used in this component is trivial; it concerns a one-on-one mapping of a geometric shape on a plant part. Therefore, this criterion requires no further decomposition of the subtask. No explainability is required, and no third party component is available that implements this subtask (see Figure 4.6).

Concluding, no preference exists for implementing this component in a declarative or an imperative way. We choose to use a declarative knowledge rule:

$$\begin{aligned} &\text{If } \exists \text{ thick cylinder } TkC \text{ and } TkC \text{ has-points } P_i, \\ &\quad \text{then } \exists \text{ plug body } PB \text{ and } PB \text{ has-points } P_i. \end{aligned}$$

4.4.3 Selected subtasks in the classification subtask

The classification subtask has as input a plant model consisting of plant parts and quality parameters, and has as output a quality class that corresponds to the quality of the plant model. In this section, we focus on the component 'determine quality class' and decide on the level of detail required for optimal transparency.

Determine quality class

The subtask 'determine quality class' decides for an input plant model what the corresponding quality class is. The manner in which this subtask operates is of interest for understandability; the need for explainability is strong. Explicit

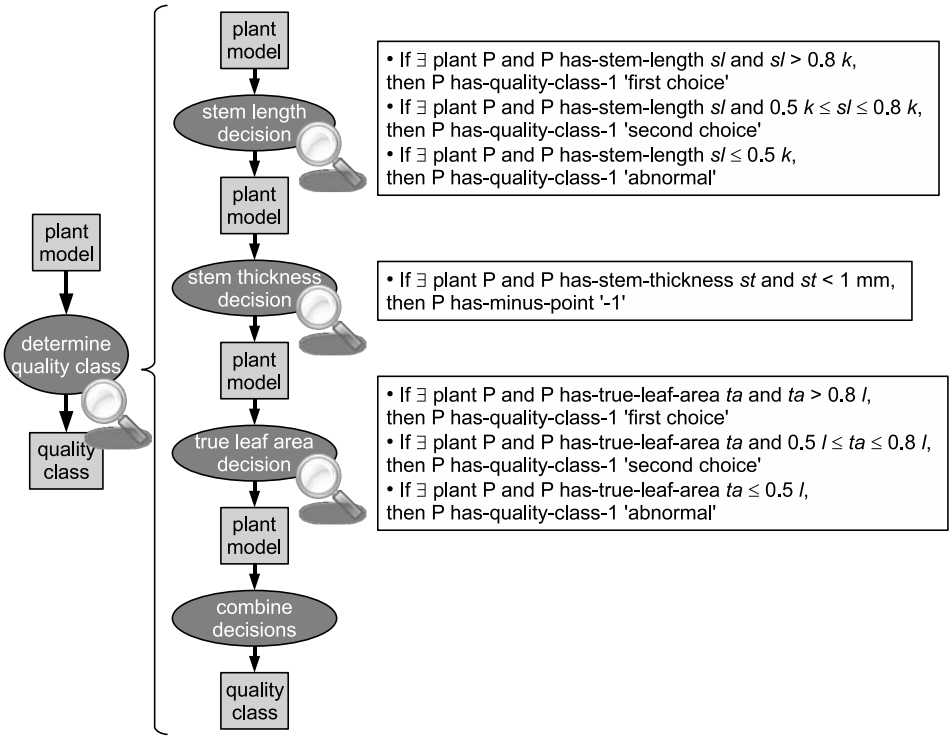


FIGURE 4.7: *A graphical representation of the specification of the 'determine quality class'-component.*

expertise on how the decision is made is available from interviews with domain experts, so explicit expression of knowledge is not inhibited. To further specify the application range, it is important to have insight in the precise decision rules. Different subsets can be used for different tasks or domains. Based on these criteria, we decide to further decompose this task into subtasks[2]: (i) stem length decision, (ii) stem thickness decision, (iii) true leaf area decision, (iv) combination of individual decisions (see Figure 4.7).

The 'stem length decision' indicates for which stem length a plant is still a candidate for a first class assessment, for which only second class is obtainable, and when a plant should be regarded as an abnormal plant. This rule is valid for all plants and tasks for which stem length is an important quality criterion. The stem length is compared to the average stem length k of all seedlings. We express this component as a set of three declarative knowledge rules for optimal support of explainability and adaptability:

[2]For conciseness reasons, we give only a subset of the classification subtasks.

If plant P and P has-stem-length sl and $sl > 0.8 * k$,
then P has-quality-class 'first choice'.

If plant P and P has-stem-length sl and $0.5 * k \leq sl \leq 0.8 * k$,
then P has-quality-class 'second choice'.

If plant P and P has-stem-length sl and $sl < 0.5 * k$,
then P has-quality-class 'abnormal'.

For the same reasons as above, the 'stem thickness decision' is expressed as a declarative rule as well. This rule is valid whenever stem thickness plays a role in the quality assessment of the plant:

If plant P and P has stem thickness st and $st < 1$ mm,
then P has-minus-point -1.

The decision about the true leaf area of a plant is also expressed declaratively. Again, this rule is only used in domains and tasks where true leaf area is an important quality aspect:

If plant P and P has true-leaf-area ta and $ta > 0.5 * l$,
then P has-quality-class 'first choice'.

If plant P and P has true-leaf-area ta and $ta \leq 0.5 * l$,
then P has-quality-class 'second choice'.

The 'combination of individual decisions' takes the decisions made by the previous three rules and combines them to find the appropriate class of the plant. The knowledge for this subtask is available in an explicit way, the application range suffices for all foreseeable applications, the combination of quality classes and finding the maximum allowed class is trivial knowledge, explainability is not a very high priority for this component, and no third-party components are available that automatically perform this task. These criteria do not lead to a strong bias for using declarative or imperative knowledge at this point. We decide to implement this subtask in a imperative fashion.

4.4.4 Implementation in OWL, Jess and Java

We show the implementation details for the component 'create point groups'. This component has as input a set of instance of the class Point, and as output

a set of instances of the class Point Group. Both classes are defined in the Point Ontology and are encoded in OWL. The corresponding part of the OWL code is displayed below. For conciseness reasons, we have omitted all labels of the concepts and relations. Moreover, only the relevant properties of the concepts Point and Point Group have been listed.

```
<owl:Class rdf:ID="Point">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>

 <owl:Class rdf:ID="Point_group">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="has_x_coordinate">
  <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger"
          >1</owl:cardinality>
  <rdfs:domain rdf:resource="#Point"/>
  <rdfs:range rdf:resource="#Coordinate"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="has_y_coordinate">
  <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger"
          >1</owl:cardinality>
  <rdfs:domain rdf:resource="#Point"/>
  <rdfs:range rdf:resource="#Coordinate"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="has_z_coordinate">
  <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger"
          >1</owl:cardinality>
  <rdfs:domain rdf:resource="#Point"/>
  <rdfs:range rdf:resource="#Coordinate"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="has_central_point">
  <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger"
          >1</owl:cardinality>
  <rdfs:range rdf:resource="#Point"/>
  <rdfs:domain rdf:resource="#Point_group"/>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >has_central_point</rdfs:label>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="has_neighbours">
  <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger"
          >5</owl:minCardinality>
  <rdfs:range rdf:resource="#Point"/>
  <rdfs:domain rdf:resource="#Point_group"/>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >has_neighbours</rdfs:label>
</owl:ObjectProperty>
```

We see that the class Point has as properties an x, y, and z coordinate. The class Point Group has the properties has-central-point and has-neighbours, both with the class Point as range.

With these concepts formally defined, we can define the two white-box rules 'initialise point groups' and 'determine neighbours' in Jess. The description of the rules is added between quotes, the Jess-representation of the rules is given in the rule body. We require a black-box component (distance) in the decomposition and therefore introduce a load-function command to link to the corresponding method in Java.

```
;; Loading the required java classes
(load-function afsg.me.Distance)

;; The rules
(defrule initialise-point-groups
    "If a Point P exists, then a Point group PG is created, of which
    P is the 'central point'"

    ?P <- (Point (x-coordinate ?x) (y-coordinate ?y) (z-coordinate ?z))
    =>
    (assert (Point_Group (has_central_point ?P)))
)

(defrule determine-neighbours
    "If a point group PG exists and PG has central point Pc and P1 is
    a point and P1 is unequal to Pc and distance(Pc,P1)<t, then PG
    has neighbouring point P1."

    ?Pc <- (Point (x-coordinate ?xc) (y-coordinate ?yc) (z-coordinate ?zc))
    ?PG <- (Point_Group (has_central_point ?Pc))
    ?P1 <- (Point (x-coordinate ?x1) (y-coordinate ?y1) (z-coordinate ?z1))
    (test (neq ?Pc ?P1))
    (test (< (distance (create$ ?Pc ?P1)) t)
    =>
    (assert (Point_Group (has_neighbours ?P1)))
)
```

The Java-code required for calculating the distance is implemented in a standard way in Java. To communicate between Java and Jess, the Java program has to import the `jess` package to enable the use of the Jess-interpretable class Userfunction.

4.5 Evaluation

In this section, we reflect on the results obtained in the case study. We evaluate how the decision criteria have helped us to meet the transparency objectives of the application. Moreover, we show how some decisions would have been different if the set of objectives changes. Next, we show how the decisions made help to realise the desired objectives for the seedling inspection case.

4.5.1 Objectives and criteria in the case study

For each of the five decision criteria used, we have seen how they contribute to the five objectives of the case study (see Table 4.1). The *criterion concerning explicit expertise* deals with the possibility to express knowledge in a transparent

criteria / objectives	Explicit Expertise	Application Range	Trivial Knowledge	Explanation	Third Party Tools
Reliability & Robustness		•	•	•	
Build trust		•	•	•	
Correction		•		•	•
Adaptation	•	•			
Understandability	•			•	

TABLE 4.1: *Overview of how the method criteria defined in section 4.3.1 support the five case study objectives identified in section 4.4.*

way. By explicitly describing the domain knowledge in terms of objects and properties in the application, adaptability, reliability and understandability are supported. In the case study, we have applied the explicit expertise criterion to all tasks where additional transparency was required.

The *criterion concerning application range* deals with the input objects for which a component is well suited. By explicit specification of the input objects and their attributes, the application knows for which objects the component can be used; it supports building of trust, adaptability and reliability. For the case study, the application range criterion lead to a declarative specification of 'calculate the planarity measure' and a further decomposition of 'determine quality class'.

The *trivial knowledge and common sense criterion* indicates whether decomposing and specifying a component leads to more insight in the component or to more cluttering. This criterion helps to find the right level of decomposition and specification; it supports finding the right level of transparency. For example for the components 'create point groups' and 'determine quality class' in the case study, no further decomposition is required, whereas for the components 'decide on the point type' and 'stem length decision' specifying the components' knowledge in an explicit way leads to more transparency.

The *criterion concerning explanation* is strongly linked to further explaining knowledge rules. Explanation allows for a specification of a component's knowledge in domain relevant terms. It therefore contributes to the objectives of understandability and building trust. In the case study, the component 'determine quality class' was decomposed based on explanation. By decomposing this component, the subcomponents are made visible and the objectives are served. For the same criterion and objectives, a component like 'stem length decision' was explicitly specified.

The last criterion deals with the *availability of third party expertise* and the trust one has in a component created by a third party. Since the makers of the components are trusted, using these components contribute to the desired level of trust of the application. Due the availability of third party components, the components 'perform PCA analysis', 'determine thick cylinder region' and 'combination of individual decisions' are implemented as black-box components.

	Explicit Expertise	Application Range	Trivial Knowledge	Explanation	Third Party Tools
Speed	•				•

TABLE 4.2: *This table indicates how the criteria support the speed objective.*

It is interesting to look at the influence of an additional objective on the design process. Suppose that the system would need to perform at high speed. In Table 4.2, we see that the availability of explicit expertise and of third party components have an influence on the speed objective.

In the case study, the adding of the speed objective leads for example to a clear advise in the case of 'initialise point groups'. Based on the explicit expertise criterion, we decide that the component is not eligible for further decomposition. Using explicit expertise would lead to an explicit specification of the knowledge that is not profitable for the other objectives. The declarative specification of the knowledge may influence the speed in a negative way. Therefore, we choose to implement this component in a black-box fashion. The same argument holds for all non-biased components in the original case study.

In the same way, we can look at the influence of removing an objective. Let us look at a component that is used in the quality assessment process[3]: 'splitted heads decision'. A 'splitted head' is a defect in a seedling that shows itself as a plant with two sets of true leaves. The splitted head component can be explained in domain specific rules:

$$\text{If plant } P \text{ and } P \text{ has-number-true-leaves } 4 \text{ and}$$
$$P \text{ has-true-leaf-area } > 0.7 * l, \text{ then } P \text{ has-splitted-head-probability 'true'.}$$

$$\text{If plant } P \text{ has-splitted-head-probability 'true',}$$
$$\text{then } P \text{ has-quality-class 'abnormal'.}$$

The first of these rules is an explicit specification of *how* the value for the property 'has-splitted-head-probability' is determined. If understandability is not an objective, the determination of the splitted head probability is treated as a black-box component, not as an explicit knowledge rule.

4.5.2 Design consequences for the transparency objectives

In Section4.4, we have made transparency decisions to support the objectives *adaptability*, *corrigibility*, *reliability*, *understandability* and *building trust* in our case study. In this section, we look at the practical consequences of the transparency decisions made. Our claim is that due to a correctly defined level of transparency, we can offer tool support to the end user of the application with respect

[3]This component was not mentioned before due to conciseness reasons

to the desired objectives. In the remainder of this section, we describe the manner in which this is possible. We sketch the possibilities to use the transparent setup of the application to build trust and to support corrigibility. Next, we show that adaptability and reliability can also be supported. The actual tools are presented in Chapter 6.

4.5.2.1 Leafing through the application

Transparency can be used to build trust, to support corrigibility and to assist in organisational learning. In the design process, the transparency criteria have supported us to design the application such that the components are either explicitly explained, so trivial that they need no explaining, or available as trusted third-party components. The side effect of this design process, is that we can show the inner workings of the application to the end user in just the right amount of detail. By visualising the sequence of models, by pointing out the trusted black-box components and by presenting the declarative rules to the expert in the expert's own terminology, the expert can gain insight in the steps in the algorithms used and his trust in the application will increase.

With the explicit task knowledge captured in declarative rules and task-specific ontologies, the expert knowledge has been formalised. In the case study this means that experts have indicated which plant features are relevant for assessing the quality of the plants and for which values of these features the plant should be assessed as a first class, second class or abnormal plant. Such explicit task knowledge can easily be represented in a flow diagram that is interpretable by the experts. Such a diagram gives a concise overview of the task knowledge and is useful in training new employees and in discussing about the seedling inspection process. An example of such a flow diagram can be found in Figure 4.8. It is a valuable tool for the organisational learning task.

4.5.2.2 Adaptability and reliability

The adaptability objective requires that the computer vision application can easily be adapted to changing situations. The application has been designed with this objective in mind. For each component, the software designer has indicated for which input objects it is valid. This allows the expert to identify the components that have to be adapted for a different task or domain.

Moreover, due to the understandability objective, the explicit knowledge rules employed by the expert are known. This allows domain experts to *autonomously* adapt quality assessment rules according to their needs. Since the quality rules are expressed in a declarative format, a user-friendly tool can be implemented to show the assessment rules to the experts. Experts can adapt the set of quality rules by changing the decision value of a feature, add or remove a quality class, introduce new quality features or reject features that are already in use. When new features are introduced, the software team obviously has to play a part in implementing the algorithms to calculate the new features. In all other cases, the domain expert should be able to make the changes on his

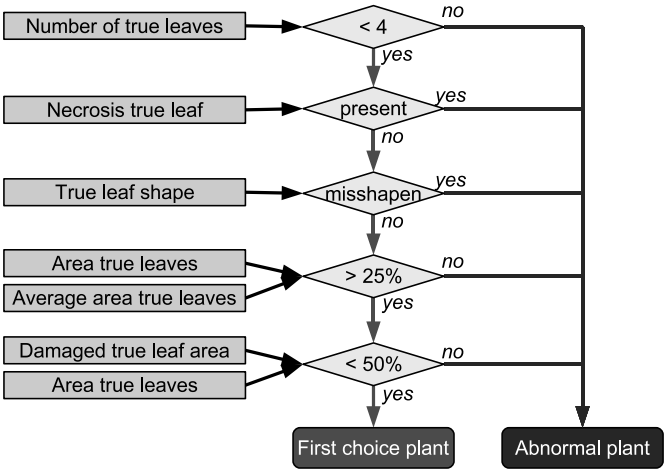


FIGURE 4.8: *A flow diagram containing the task knowledge for seedling assessment. With this diagram, new employees can be trained to assess seedlings.*

own. A custom-made Adaptation Module can be used to change the quality assessment rules.

A Reliability Module can be set up as well. The function of this module is to contain relevant exclusion criteria expressed using knowledge rules. Due to the similarity of the reliability tool to the adaptability tool, we do not show its implementation in Chapter 6.

4.6 Conclusion

For knowledge-intensive task, the availability of explicit expertise has an influence on trust in the application, robustness and reliability of the application, speed of the application, sustained system development, and organisational learning. Software developers have to make decisions on the correct level of transparency in the application. This level depends on the objectives that are indicated as relevant.

The transparency objectives identified in this chapter partly overlap with the white-box aspects defined in Chapter 2. Corrigibility, adaptability, robustness, reliability and speed are overlapping objectives. Expert acceptance in Chapter 2 has been interpreted as trust in this chapter. Organisational learning and sustained system improvement are new objectives introduced in this chapter. These latter two objectives can only be obtained by allowing transparency of the processing steps; the other objectives can already profit from transparency of descriptive knowledge alone.

In this chapter, we have shown how the choice of design objectives for a task influences the decision on when transparency is desired for a task component. We have presented a set of decision criteria that can be used by the

software developer to decide on the need for transparency at every level of the application. Based on the decision, the component and the domain ontologies have to be further decomposed, the input and output ontology has to be refined, knowledge rules have to be defined, or a black-box component can be used.

We have shown that the objectives of the case study play an important role in design decisions to obtain the desired level of transparency. A different set of objectives leads to different decisions. The theory in this Chapter offers structure to the application designer in making transparency decisions. This ensures a careful weighing of costs and benefits in the implementation process.

Chapter 5

The white-box seedling inspection system

In this chapter, we show the results of applying the proposed method for designing white-box, knowledge-intensive computer vision applications to the seedling inspection case study. To this end, we have created three application ontologies: the point ontology, the geometric ontology, and the plant ontology. These ontologies have been created using interview-based knowledge acquisition. Moreover, we have implemented a prototype version of the computer vision application that inspects and assesses tomato seedlings following the method proposed in the previous chapters.

5.1 The ontologies created for the vision application

The main application in this thesis – the seedling inspection system – employs three ontologies to express different views on the recorded seedling: (i) the plant, (ii) the geometry, and (iii) the point application ontology. The plant application ontology is based on plant expert knowledge and is focused on the set of quality inspection rules that the domain experts use to decide on the quality of the recorded plants. It contains knowledge that represents the morphology of the plants under inspection. The concepts in the geometry ontology are limited to the shapes of the possibly occurring plant parts. The specification of the point application ontology partly depends on the chosen image acquisition method and partly on the information required in the geometry application ontology. In the next paragraphs, we discuss the content of these ontologies.

5.1.1 The point application ontology

The point application ontology contains all concepts that are related to three dimensional point clouds. The point ontology corresponds to the 'see' task of a computer vision application: the image acquisition results in a set of points that

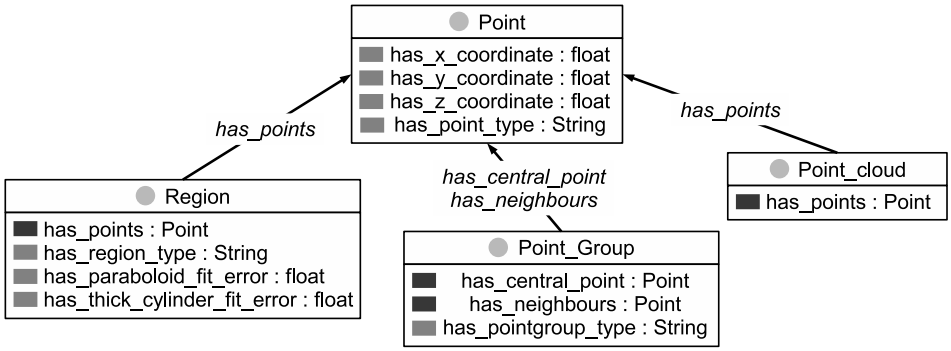


FIGURE 5.1: *A graphical representation of the concepts and relations in the Point Application Ontology.*

are represented in this ontology. Moreover, the lowest level processing, where points are tentatively grouped together in regions is performed in terms of this ontology.

The point application ontology is depicted in Figure 5.1. The concepts in this ontology are (i) 'Point cloud', (ii) 'Point', (iii) 'Point group', and (iv) 'Region'. For each plant that needs to be classified, an instance of the 'Point cloud' is created. This instance corresponds to the whole plant. The point application ontology contains the class 'Point'. Each instance of this class has as attributes a 'x, y, and z-coordinate' and a 'type' that can have the value 'linelike' or 'planar'. For each 'central point' p a 'point group' is created. Each instance of a 'point group' contains the 'central point' p and its 'neighbouring points'. 'Regions' consist of points of the same type that are grouped together. A 'Region' has two properties that indicates how well the 'Region' matches a cylinder and a paraboloid.

5.1.2 The geometry application ontology

The geometry application ontology contains all geometrical concepts that correspond to a more sophisticated segmentation of the recorded object into units that can be used to identify plant parts later on. The geometry concepts in this application ontology are a subset of all possible geometry shapes; the set of shapes is restricted by the seedling inspection task.

We give a graphical representation of this ontology in Figure 5.2. The four concepts in this ontology all represent a geometrical shape: (i) 'Thick cylinder', (ii) 'Thin cylinder', (iii) 'Paraboloid', and (iv) 'Surface'. An instance of a 'Thick cylinder' consists of a set of 'Planar points' that roughly have the shape of a cylinder with a radius of 20 mm. An instance of a 'Thin cylinder' consists of a set of 'Linelike points' that roughly form a cylinder shape with a relatively small radius. A 'Paraboloid' consists of a set of 'Points' with point type 'planar' that together have an 'umbrella' shape. An instance of a 'Surface' has a set of 'Points' that form neither a 'Thick cylinder' nor a 'Paraboloid'. The geometry application ontology always contains exactly one instance of a 'Thick cylinder',

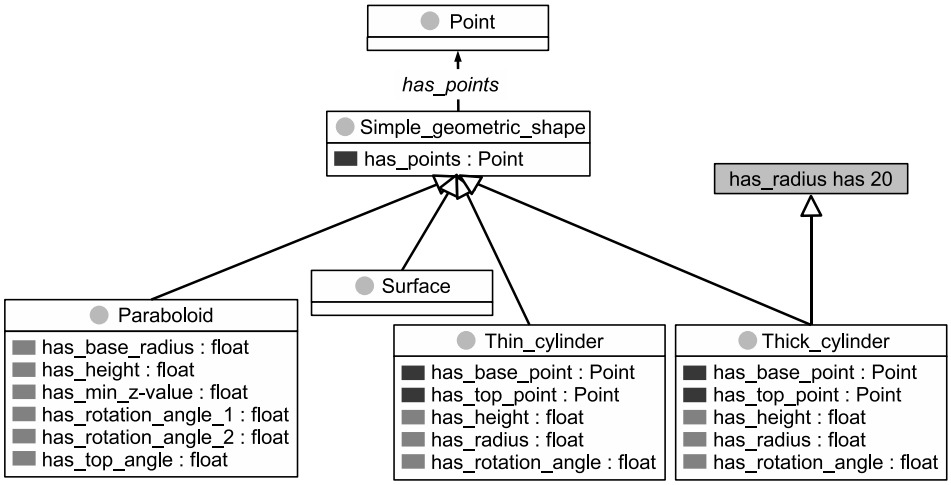


FIGURE 5.2: *A graphical representation of the concepts and relations in the Geometrical Application Ontology.*

at most one instance of a 'Paraboloid', and an arbitrary number of instances of 'Thin cylinders' and 'Surfaces'.

5.1.3 The obtained plant application ontology

The plant application ontology obtained in the knowledge acquisition process consists of concepts representing the plant parts of the seedlings under inspection. The concepts in this ontology are (i) 'Plug', (ii) 'Stem', (iii) 'Leaf', (iv) 'Cotyledon', (v) 'True leaf', (vi) 'Connected cotyledons', (vii) 'Tray', (viii) 'Top', and (ix) 'Plant'. Each instance of these concepts (except 'Plant' and 'Tray') has as attribute a set of 'points' connected to it. These points are equal to points in the point application ontology. The stem, and each of the leaves has a set of parameters to describe them, like 'stem length', 'leaf area', *et cetera*. These parameters are important for the classification of the plants. Each instance of 'Plant' consists of instances of 'Plug', 'Stem', 'Cotyledon', 'Connected cotyledons' and/or 'True leaf'. It has a number of properties, among which an attribute indicating the quality class indicator. A graphical representation of the plant application ontology is displayed in Figure 5.3.

Note that in the implementation of the computer vision application, additional concepts were added to the plant application ontology: 'Leaf or Stem', 'Plug body', 'Plug head'. These concepts are auxiliary concepts prompted by the transformation function from geometrical concepts to the plant application ontology. They have been obtained from discussions with the computer vision expert with knowledge of the geometry application ontology and the plant domain expert. The concept 'Plant' has several properties corresponding to fea-

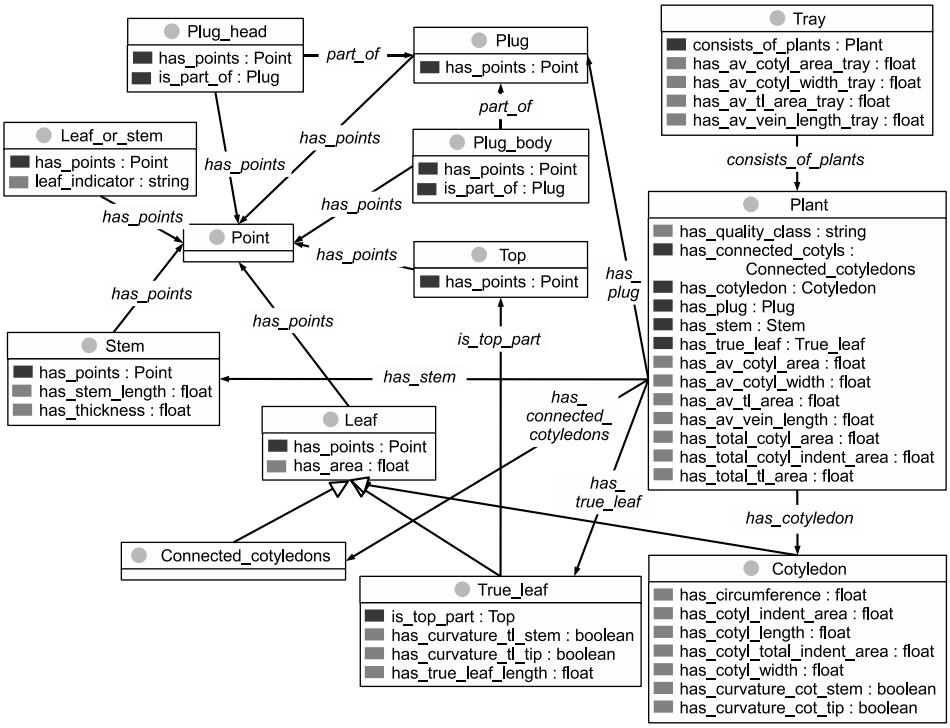


FIGURE 5.3: *A graphical representation of the Plant Application Ontology.*

tures that are relevant for the quality assessment task. It also has a property that is used to indicate the 'quality' of the inspected plant.

5.2 Implementation of the seedling inspection prototype

To facilitate testing whether the proposed method can result in a working knowledge-based computer vision application, we have implemented a prototype application for the seedling inspection case that complies with the given requirements. In this part, we describe our experimental setup, introduce the concept of sanity check rules, and go step by step through the test application. We evaluate the results and thereby show that it is indeed feasible to implement a computer-vision application using the ontology-based design method. Moreover, we show that the white-box induced sanity-check rules lead to a 'learning effect' that attunes the application more and more to its environment.

5.2.1 Experimental setup

The prototype software application has been created in the programming language LabVIEW 8.6 and the corresponding IMAQ Vision software. It has been developed for inspecting tomato seedlings that are approximately 12 days of

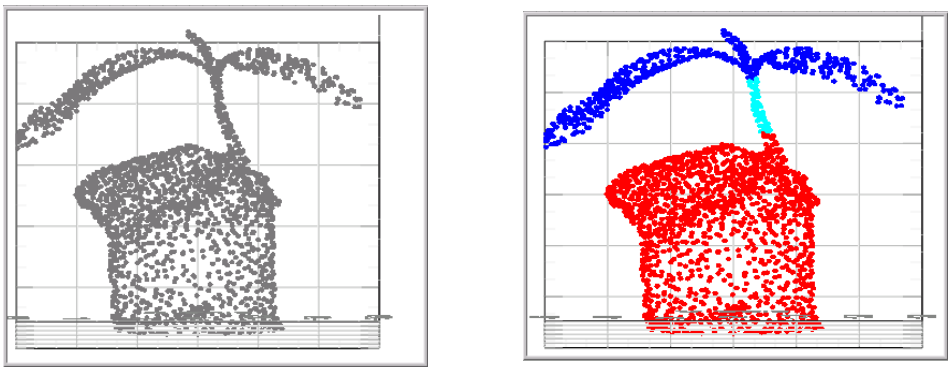


FIGURE 5.4: *The input point cloud (ID = 11) and its corresponding geometrical model. The geometrical shapes are correctly identified. The red points are part of the Thick cylinder, the turquoise points are part of the Thin cylinder, and the blue points are part of the Surface.*

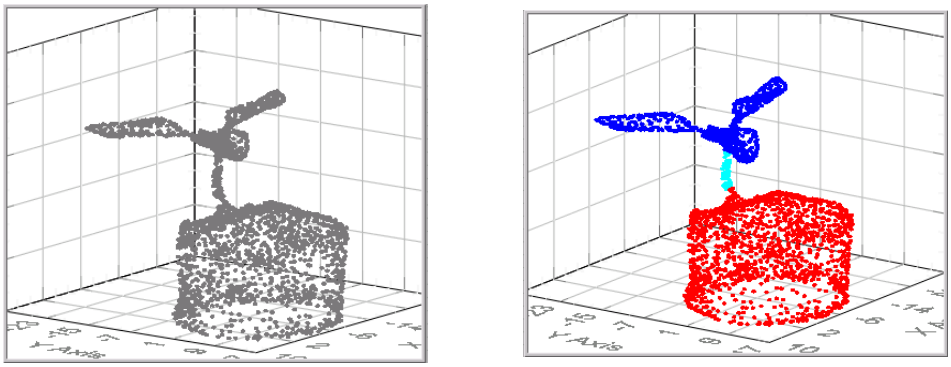


FIGURE 5.5: *An input point cloud (ID = 22) and its corresponding geometrical model. The geometrical shapes are correctly identified.*

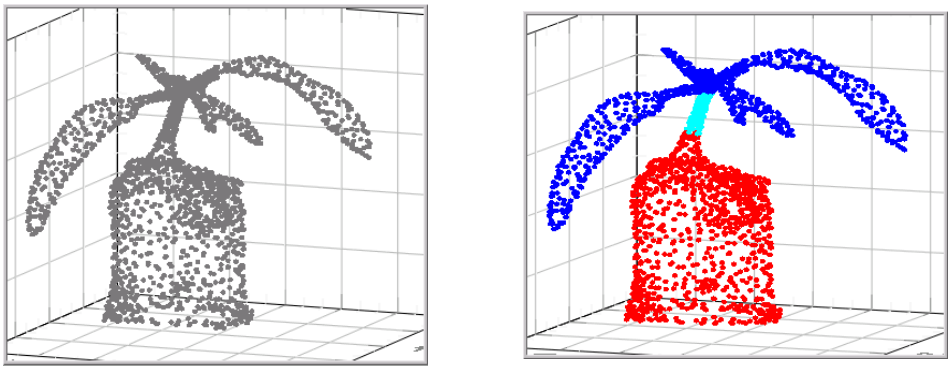


FIGURE 5.6: *The input point cloud (ID = 26) and its corresponding geometrical model. The geometrical shapes are correctly identified.*

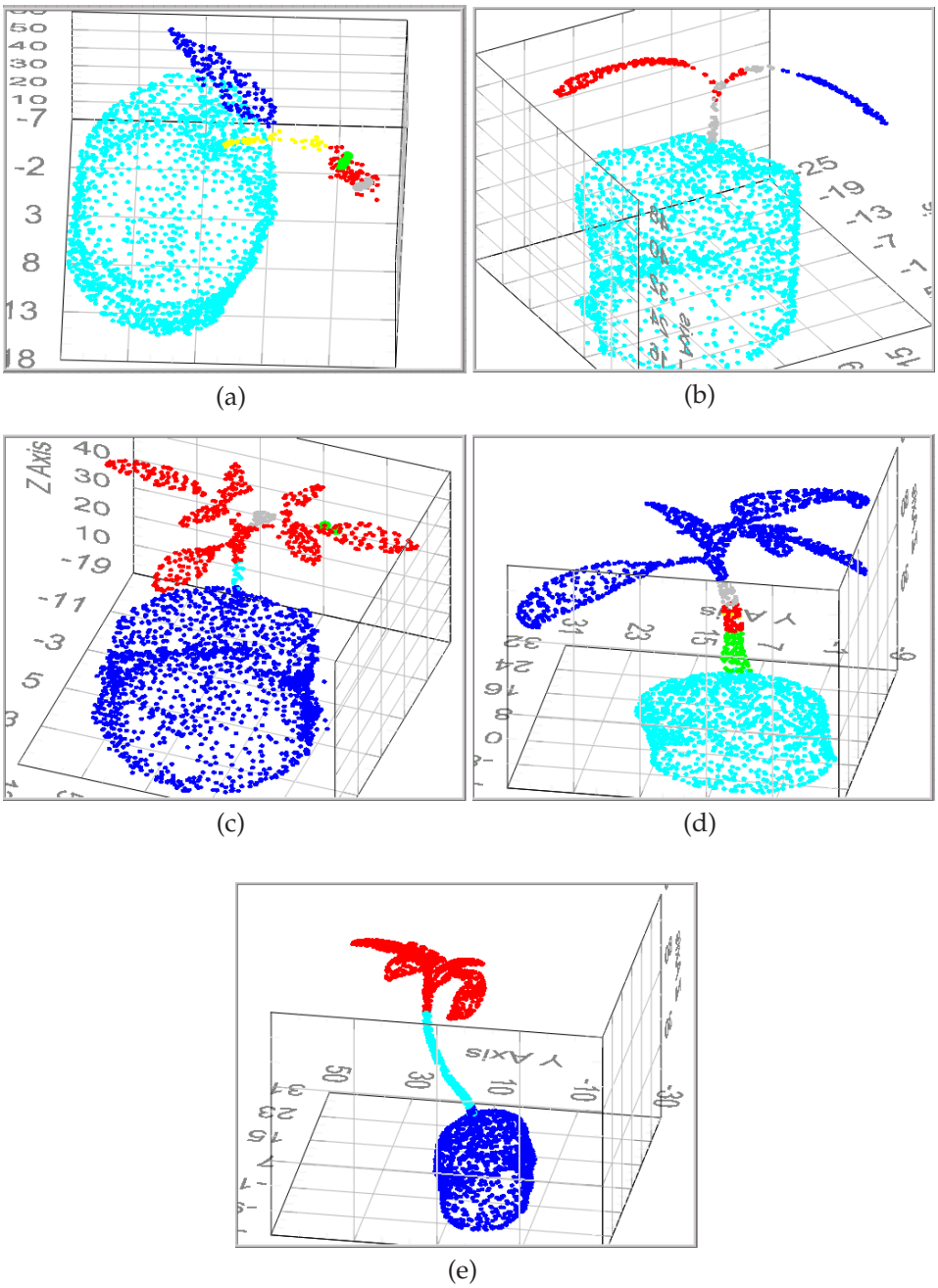


FIGURE 5.7: *The incorrectly processed point clouds. Figure (a) represents a geometric model that leads to an incorrect plant model later in the application. Figures (b) to (e) are incorrect due to bugs in the algorithms associated with the point-geometry processing steps.*

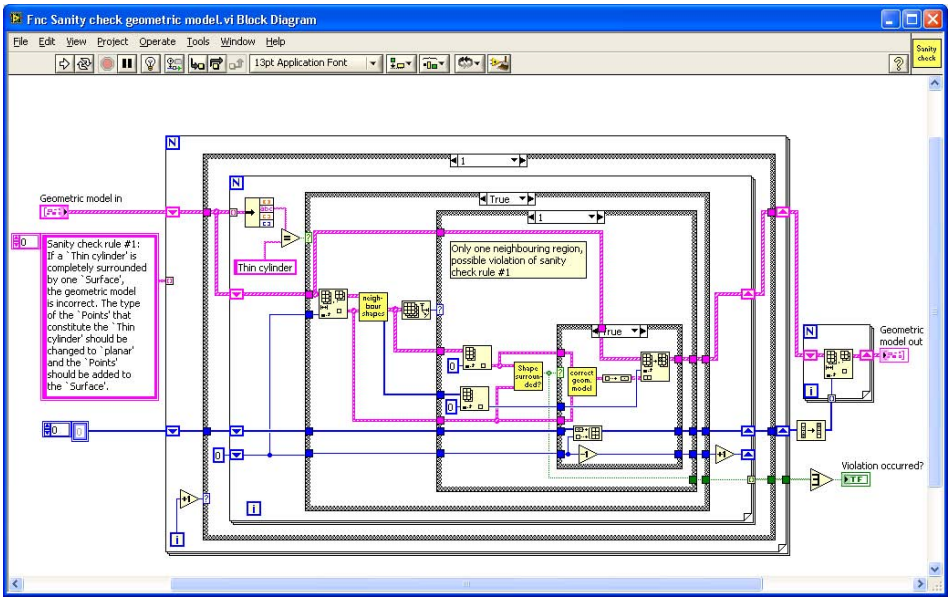


FIGURE 5.8: *The sanity check code for correcting the geometric model as implemented in LabVIEW. For each of the shapes, we check whether it is a thin cylinder. If so, we check how many neighbouring shapes the thin cylinder has. When only one region is a neighbour of the thin cylinder, we check whether this region surrounds the thin cylinder. When this check is positive as well, the points of the thin cylinder are added to the surrounding shape and the thin cylinder is removed from the list of shapes.*

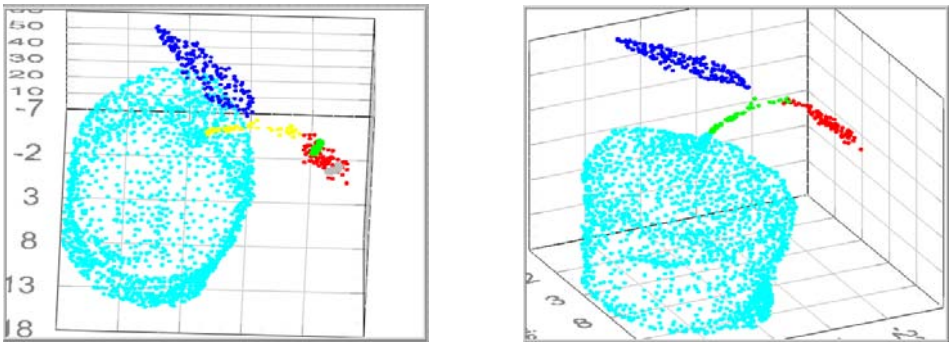


FIGURE 5.9: *The original geometric model, and the corrected geometric model after the sanity check for the point cloud displayed in Figure 5.7 (a). The new geometric model correctly consists of two surfaces (red, blue), a thick cylinder (turquoise) and a thin cylinder (green).*

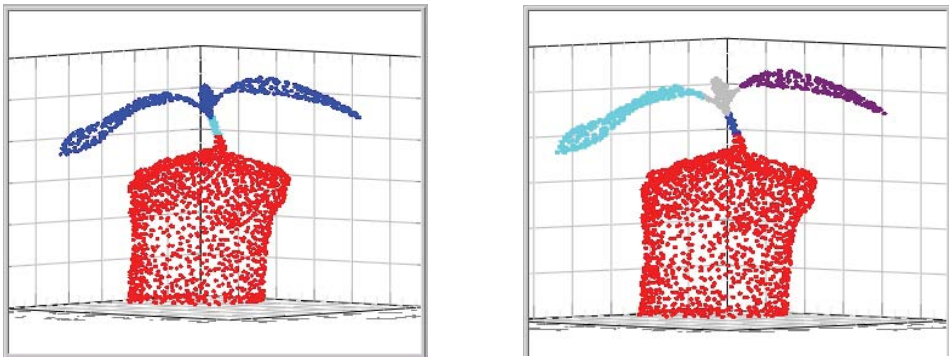


FIGURE 5.10: *The geometric model (plant ID = 10) and its corresponding plant model. The red points in the plant model correspond to a 'Plug', the blue points to a 'Stem', the turquoise and purple points to 'Cotyledons', and the grey points to a 'True leaf'. The plant parts have been identified correctly.*

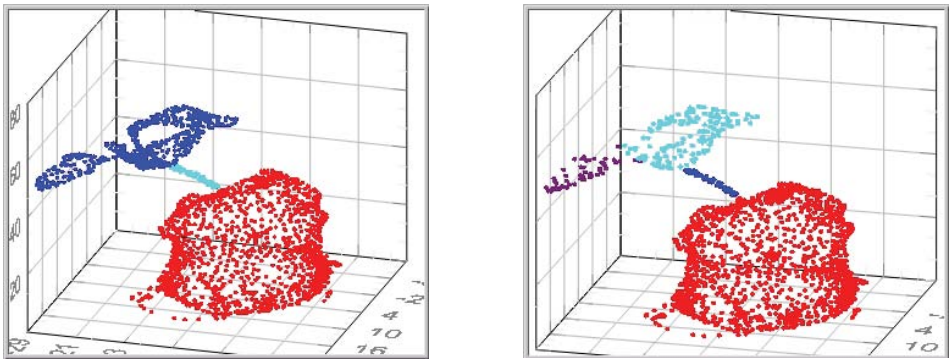


FIGURE 5.11: *The geometric model (plant ID = 17) and its corresponding plant model. The plant parts have been identified correctly.*

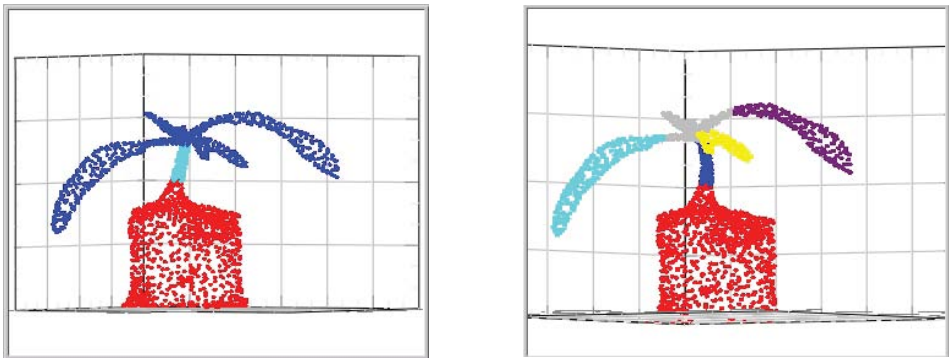
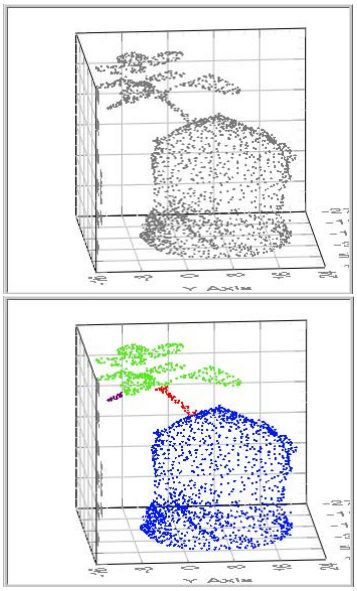
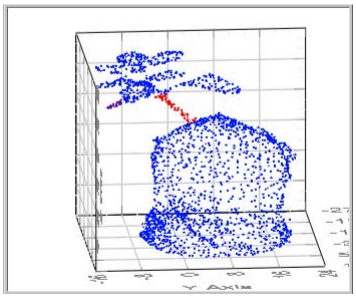


FIGURE 5.12: *The geometric model (plant ID = 26) and its corresponding plant model. The plant parts have been identified correctly.*

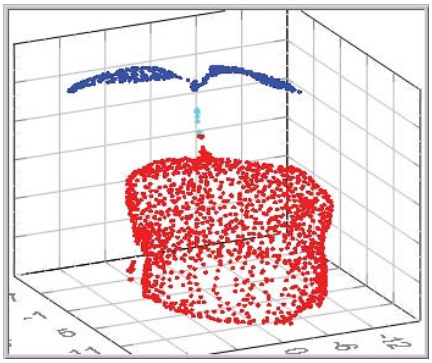




Identification of geometric regions:

red points:	Thin cylinder
blue points:	Thick cylinder
green points:	Surface
purple points:	Thin cylinder

FIGURE 5.13: *The manual processing of the point cloud to form the geometrical model. The identification of a line-like appendix in the plant that is part of the first true leaf is valid in terms of the geometric model.*



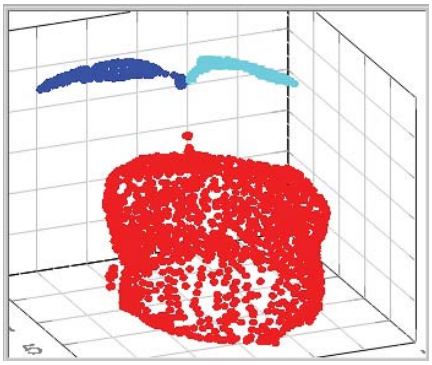


FIGURE 5.14: *The geometric model (ID = 4) and its corresponding plant model. The plant parts have not been identified correctly; the stem of the plant is not recognised as a plant part.*

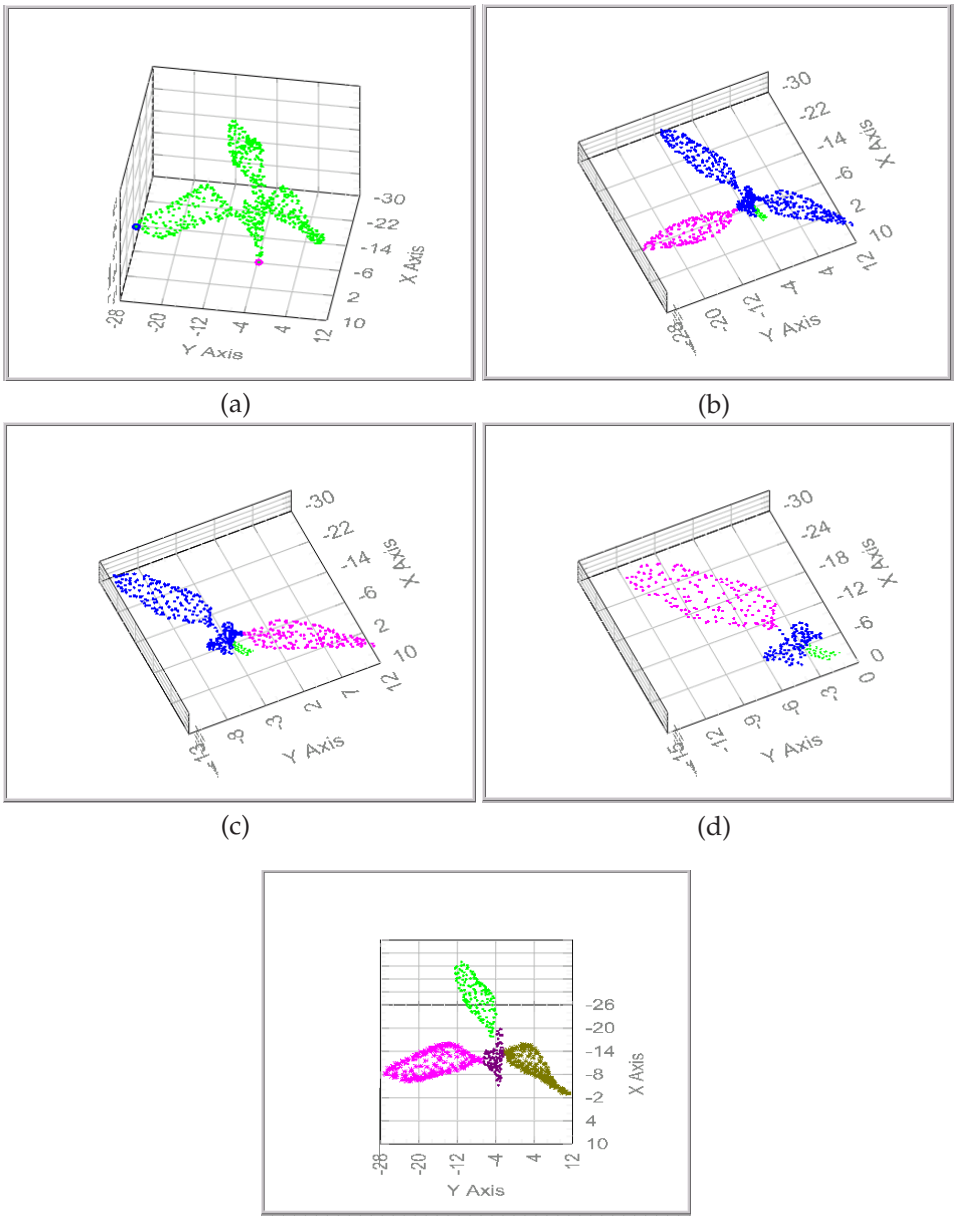
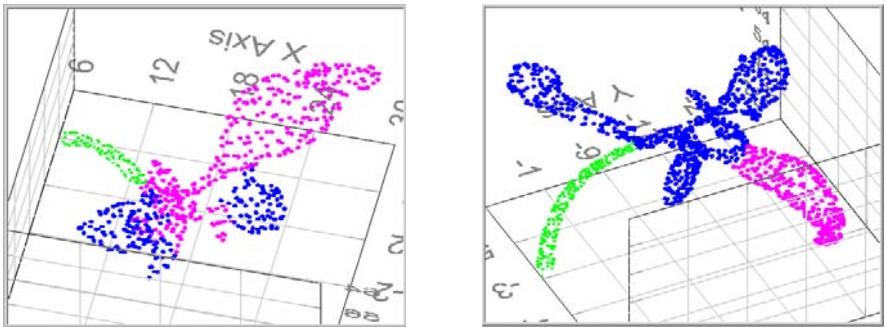
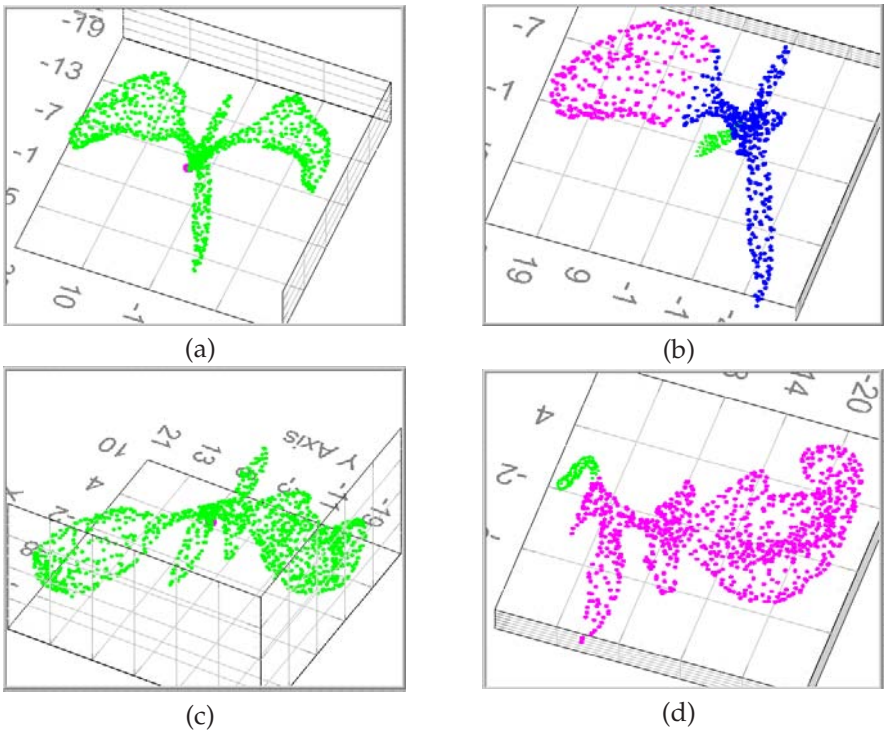


FIGURE 5.15: *The intermediate results of a correct leaf identification for a point cloud. The largest leaf is looked for first, then the points corresponding to this leaf are identified and the leaf is removed from the remaining unidentified surface points. This procedure is repeated for all leaves. It leads to the final leaf segmentation as depicted in (h).*



(a) (b)

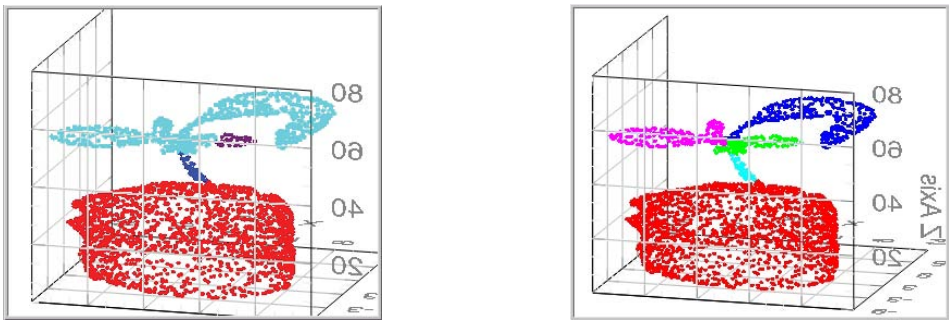
FIGURE 5.16: *In these images, the identification of the first leaf (pink) in the 'Surface' is incorrect. In figure (a), the leaf contains too many points; in figure (b), the leaf contains too few points.*



(a) (b)

(c) (d)

FIGURE 5.17: *In these images, the identification of the second leaf (pink) in the 'Surface' is incorrect. In figures (a) and (c), the complete surface is shown before the first leaf points are removed. In figures (b) and (d), the identification of the second leaf is displayed. The leaf in figure (b) contains too few points, the leaf in figure (d) contains too many points.*



(a) (b)

FIGURE 5.18: *When the example geometric model is processed automatically, it results in the plant model depicted in figure (a). This incorrect plant model consists of a plug (red points), a stem (blue points), a cotyledon (turquoise points), and a true leaf (purple points). When this plant is manually processed using the protocol defined, it results in the correct plant model displayed in figure (b). It has a plug (red points), a stem (turquoise points), a cotyledon (blue points), and two true leaves (green and pink points).*

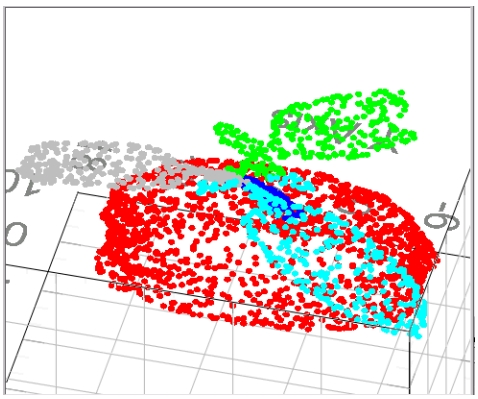


FIGURE 5.19: *The plant model in this figure consists of a plug (red points), a stem (blue), two cotyledons (turquoise and green), and a true leaf (grey). The leaf identification is incorrect: the green points should have been assigned as true leaf, and the grey points as cotyledon.*

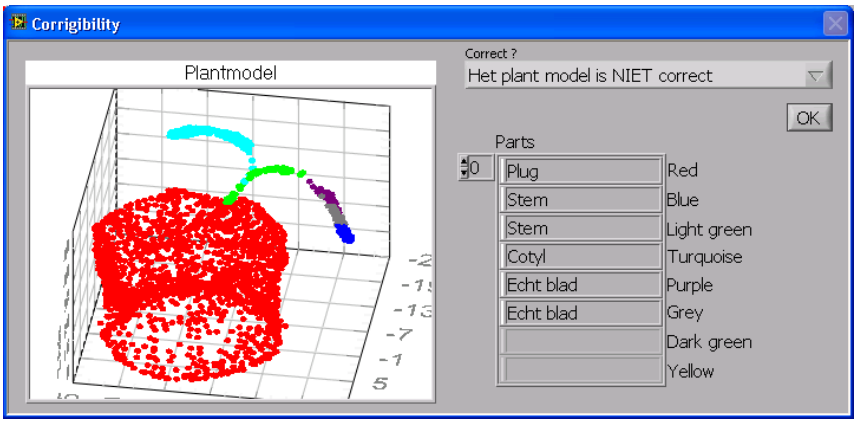


FIGURE 6.1: *The corrigibility module starts with checking whether the plant model is correct. In the screenshot, we see a 3D display of the plant model, an overview of the identified plant parts and the option for the expert to indicate whether the plant model is correct or not.*

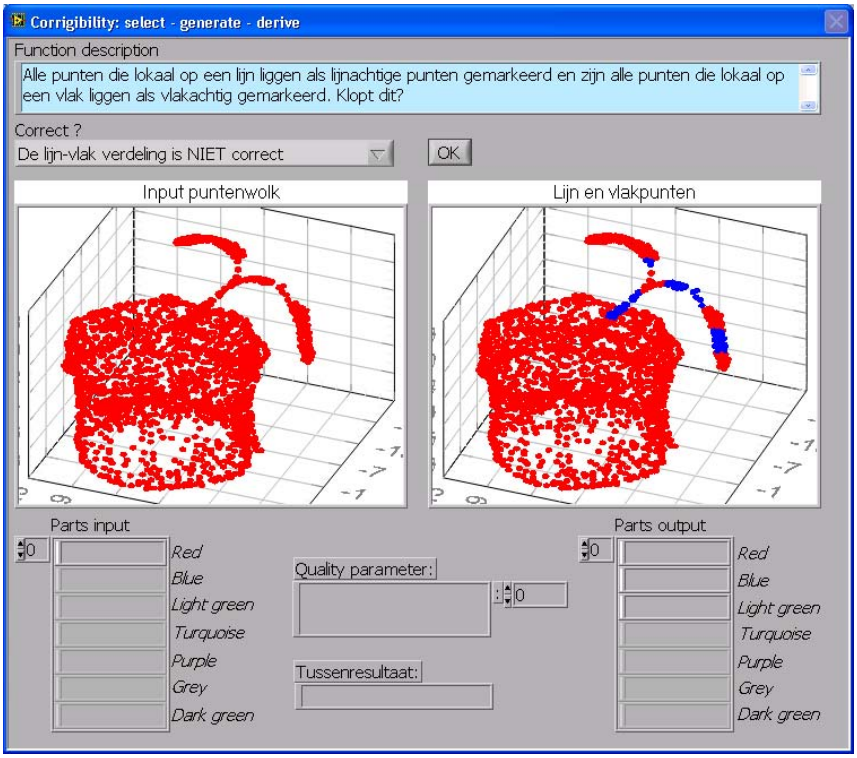


FIGURE 6.2: *In this screenshot of the corrigibility module, we see the input point cloud (for reference) and the line-plane model created by the application. In the drop-down box, the expert can indicate whether the line-plane model is correct or not.*

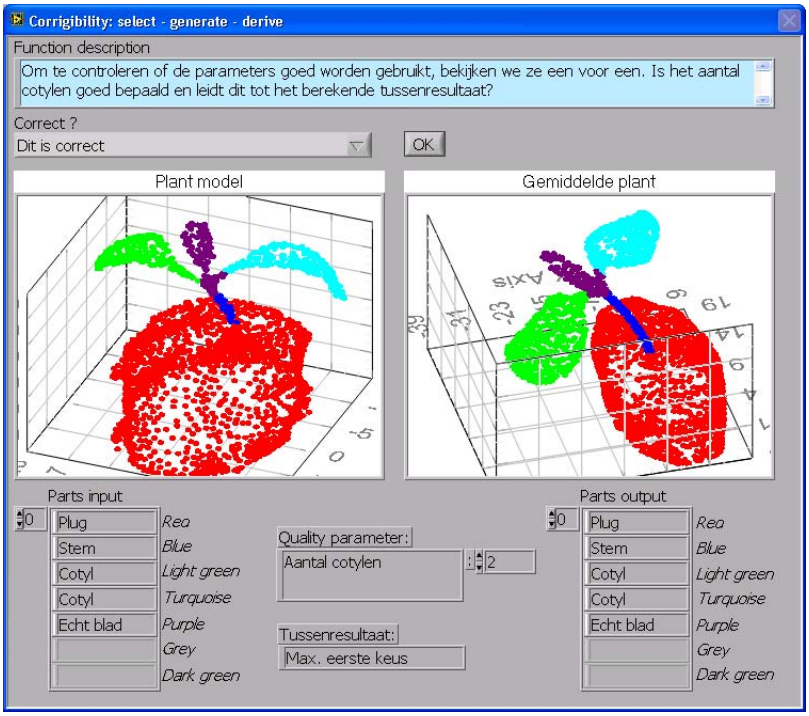


FIGURE 6.3: *As part of the quality assessment check, the Diagnosis Module asks the expert whether the plant has been classified correctly based on the number of cotyledons.*

age (see Figure 5.20). We have created a testset consisting of 96 tomato plants. These seedlings have been recorded with the 3D Scan Station (Scanbull Software GmbH, Nürnberg), using volumetric intersection [79]. Each seedling was put on a rotation table in a light cabinet. We set the software to make 24 frames for one table revolution spaced at 15 degrees. For each frame, the table was halted for 10 seconds to allow the plant to stop shaking before making a digital image. A picture was made using a digital camera mounted on a tripod in front of the cabinet (see Figure 5.21). The camera made an angle of approximately 30 degrees with the horizontal plane and was positioned at approximately 80 cm from the plant, thus allowing the camera to get a good view of the leaves of the seedling.

The recorded images were semi-automatically processed by the Scan Station software to form 3D point clouds with points that represent the surface of the recorded plants. The created point cloud models consists on average of approximately 2500 points, obtained from an automated triangulation of the object's surface.

For designing the prototype application, we have followed the framework design proposed in Chapter 2. We started with setting the scope, decomposed the task into subtasks, interviewed experts for their expert knowledge, and de-

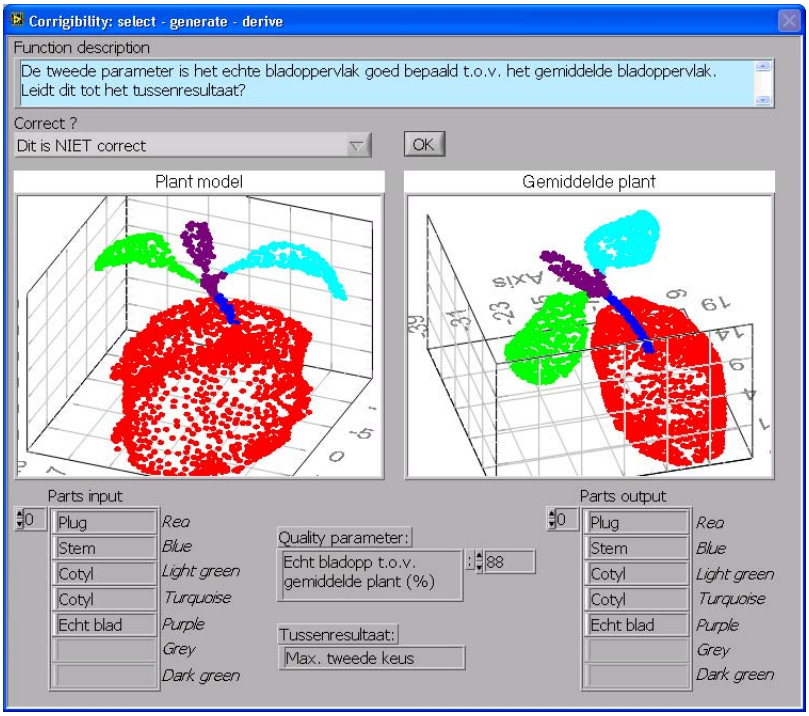


FIGURE 6.4: *As part of the quality assessment check, the Diagnosis Module asks the expert whether the plant has been classified correctly based on the true leaf area.*

fined the processing components. For showing that a computer vision application setup according to the proposed framework results in a workable computer vision application, we opted for LabVIEW, rather than Jess/Java (as proposed in Chapter 4). This implementation choice is based on our practical expertise with LabVIEW and has no consequences for testing the viability of our design method. By showing that the prototype is suited for inspecting the seedlings, we show that an application designed using the proposed white-box design method is capable of performing the task satisfactorily.

5.2.2 Sanity check rules

In any computer vision application, flaws may occur. We distinguish two kinds of errors. The first kind are genuine bugs: the algorithm is designed with a flaw and does not perform according to its specifications. This type of errors should be solved to obtain a well-functioning application. However, for our evaluation, we are interested to test the design method, not the implementation of the computer vision algorithms. Not all algorithms in the prototype application function optimally yet, since the computer vision algorithms required for processing the seedlings have to deal with complex implementation details beyond the scope of this thesis. To be able to test the design method despite

FIGURE 5.20: *This figure shows a seedling, 12 days of age, that is considered to be a first quality plant.*

the flaws, we have decided to separate the implementation of the knowledge-intensive components from the actual knowledge in the components. By using the explicit knowledge of a component to guide the processing of the object, we can mimic the performance of the algorithm had it been correctly implemented. With the internal working of the algorithms correct or mimicked, we can evaluate whether the created workflow suffices to process the plants in the test set correctly. This suffices to evaluate that the white-box design method can function correctly. The criteria that are more specifically linked to transparency are evaluated in Chapter 6.

The second kind of errors are errors that result in an incorrect model in later steps of the decision process. Such judgements of the object under inspection are in principle correct with respect to the local domain knowledge available for the decision. Only in a later step in the task when more task-specific knowledge is available, the earlier decision turns out to be incorrect. This second type of errors is of real interest for our application. Moreover, we can show added value of our design by solving this type of errors in a structured way. Due to the knowledge-based approach, we can detect the error in the advanced model, understand why the earlier decision was made, and – more importantly – know

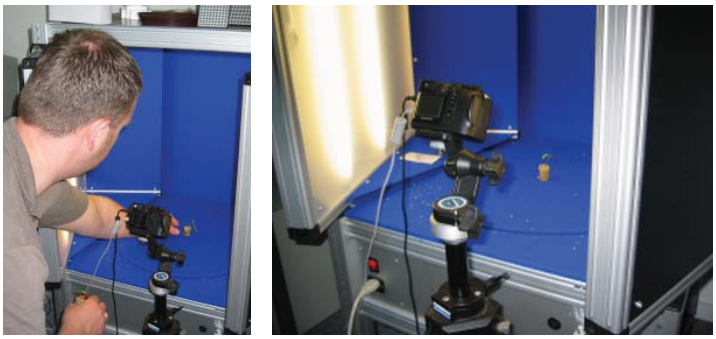


FIGURE 5.21: *The test set has been created using the 3D Scan Station.*

how to correct for it. With hindsight, the algorithm can conclude that the earlier decision is inconsistent with all possible more advanced models. When such inconsistencies occur, a corresponding *sanity check rule* can be added to the early stage of the application to automatically correct the unintended error. We will discuss this second type or error and the corresponding sanity check rules in more detail in the next sections.

We continue this chapter with evaluating the implemented application step by step. Whenever opportune, we introduce sanity check rules to the application.

5.2.3 Test: creation of the geometric model

We first test the function in the process that uses knowledge about the input 3D point cloud to obtain a model in terms of geometrical shapes. We have implemented a set of LabVIEW functions that automatically perform this model transition. By processing the test set, 91 of 96 input point clouds resulted in a correct geometric model with geometric shapes consisting of points from the input point cloud. In Figures 5.4, 5.5, and 5.6 examples are displayed of point clouds that have successfully been transformed to geometrical models.

From the remaining 5 point clouds (see Figure 5.7), 1 leads to an incorrect model in later steps of the application (an error of type 2) and 4 are incorrectly processed because of genuine bugs (errors of type 1) in the algorithms. We consider the incorrectly processed point clouds individually. As described in Section 2.5, the following processing steps have to be taken to generate the geometrical model:

- Create point groups and determine point types. This results in a classification of all points in the point cloud as linelike or planar points.
- Determine regions of related points. Points that are close enough and that have the same type are assigned to the same region. This results in a division of the point cloud into linelike and planar regions.
- Identify the geometric regions. The identified regions are recognised as specific geometric shapes.

When we consider the point cloud in Figure 5.7 (a), the creation of point groups and determining the point types of the points is successfully done. Based on the initial knowledge in the geometric ontology, there is no reason to frown upon the linelike points (green and grey) surrounded by a set of planar points (red). The division of the point cloud into linelike and planar regions based on calculating the planarity measure is done without any problems as well. Next, the geometric model is constructed. It consists of thick and thin cylinders and some surfaces. Each of these concepts exists in the geometric ontology and the identified model is initially considered to be a valid geometric model. In the next steps of the application though, the identified geometric model cannot result in a valid plant model. More specifically, we conclude that a 'Leaf or stem' part in the middle of a 'Leaf' cannot occur as separate plant part. The

geometric model that leads to this plant model is therefore incorrect and should be corrected using a sanity check rule.

Based on the information that a thin cylinder enclosed by a surface cannot occur, we conclude that linelike points in the middle of planar points cannot occur either. This leads to the following sanity check rule:

> **Sanity check rule geometric model:** If a 'Thin cylinder' is completely surrounded by one 'Surface', the geometric model is incorrect. The type of the 'Points' that constitute the 'Thin cylinder' should be changed to 'planar' and the 'Points' should be added to the 'Surface'.

The appropriate sanity check rule is implemented in LabVIEW (see Figure 5.8) and the incorrect geometric model from Figure 5.7 (a) is identified in an early stage. It is corrected by the sanity check rule. In Figure 5.9, the originally identified geometric model and the corrected geometric model are shown for the point cloud in Figure 5.7 (a). This example shows how non-local knowledge can be used to prevent errors of the second type.

Next, we consider the errors of type 1, *i.e.* the genuine bugs in the algorithms. We first consider the point cloud in Figure 5.7 (b). For this model, the creation of the point groups and the identification of the point types is performed correctly. The assignment of the regions, though, is incorrect: the linelike region is intertwined with the planar region. This is a situation that does not fulfil our expectations. The algorithms that assign the points to regions should prevent this behaviour. The processing of the point cloud in Figure 5.7 (c) has a flaw in the algorithm that determines the point type. The green points in the figure are assigned as linelike points, although the computer vision expert expects them to be planar. The grey points, though, are correctly identified as linelike points. The point cloud in Figure 5.7 (d) also has a flaw in the algorithm that determines the point type of the points. The red points in the figure are assigned as planar points, whereas the grey and green points are correctly identified as linelike points. This is unexpected behaviour. The red points should have been identified as linelike points. Figure 5.7 (e) depicts a point cloud that is correctly divided into linelike and planar points. However, a flaw occurs in the algorithm that identifies geometric regions. The model consists of the shapes 'Thick cylinder', 'Thin cylinder' and 'Surface', but the 'Thick cylinder' is not assigned to the shape that is most cylinder like, but to the red points in the figure.

Since we are interested in showing that the proposed design method results in a useful computer vision application and not in assessing the quality of the implemented algorithms, we allow a manual correction of the occurring bugs in the algorithm. When the point clouds in Figures 5.7 (b) to (e) are processed by hand according to the protocol described by the expert, a valid geometric model results. In Figure 5.13, this manual correction is shown for the point cloud in Figure 5.7 (c). The points that were incorrectly assigned as linelike points are after manual processing of the point cloud identified as planar points. The resulting geometric model corresponds with the expectation of the geometric model for this point cloud.

We conclude that the creation of the geometric model fulfills our expectations: the decomposition of the task into subtasks, the use of the concepts from the point and geometric ontology, and the application of the task descriptions defined by the expert result in a part of the computer vision application that successfully transforms the input point clouds to a geometric model. Moreover, we have shown that a sanity check rule based on the explicit domain knowledge can be implemented to prevent the detected error of the second type in the geometric model to reoccur in the future.

5.2.4 Test: creation of the plant model

The processing knowledge that has the geometrical model as input and generates a plant model is considered next. Of the remaining 92 test plants[1], 55 are transformed to a correct plant model. In Figures 5.10, 5.11 and 5.12, three examples of correctly transformed plants are shown. The geometric model in the left-hand side of Figure 5.10 is processed and a plug, a stem, two cotyledons and a true leaf are correctly identified. The plant in Figure 5.11 has two cotyledons that are stuck in the seed coat. The subtask that creates a plant model from a geometric model recognises the structure as a set of connected cotyledons. Also the remaining true leaf, the stem, and the plug are correctly identified for this plant. The geometric model in Figure 5.12 is processed and results in a plant model with two cotyledons, two true leafs, a stem and a plug.

The 37 incorrectly processed plants suffer from errors of the first or second type. The errors in the creation of a plant model from a geometric model are concentrated in the components 'stem recognition', 'individual leaf identification' and 'leaf type assignment'. We describe the results of these components in some detail.

Stem recognition For one of the plants, the geometric model is incorrectly processed with respect to the stem recognition (see Figure 5.14). Due to the few points in the thin cylinder, the plant recognition step discards these points and decides that the thin cylinder does not correspond to a leaf or stem in the model. As a consequence, the resulting plant model has no stem. The domain expert considers such a model as invalid. The plant ontology has to be equipped with a sanity check rule to correct this kind of erroneous decision:

> **Sanity check rule plant model:** A plant model with 'Leaves' must have a 'Stem' that connects the 'Leaves' to the 'Plug'.

This sanity check rule finds that the 'Thin cylinder' shape in the geometric model is located between the 'Thick cylinder' and the 'Surface'. It assigns the points corresponding to the 'Thin cylinder' to a new 'Stem' shape.

[1] The 91 plants that were initially processed correctly and the 1 plant that was corrected using the implemented sanity check rule.

Individual leaf identification The geometric model consists of one or more surfaces that are analysed and in which one or more leaves are identified. The leaf identification of individual leaves is one of the more complex processes in the system due to the natural variation in the plants. For our analysis, we distinguish between several parts of the leaf segmentation step. In each of these steps, errors of type 1 occur.

First, the algorithms check whether the plant has connected cotyledons. For an example of a plant with connected cotyledons, we refer to Figure 1.4 (f) and (g). In the test set, six plants occur with connected cotyledons. Two of these plants are incorrectly processed due to an error of the first type.

Second, the surface shapes are analysed and the individual leaves are identified. In 14 cases, the first leaf is not correctly separated from the remaining leaf mass. In 5 cases, the same problem occurs in separating the second leaf. In 10 cases, separating the third leaf is incorrect. In one case, the last leaf is incorrectly divided into two leaves. The leaf segmentation algorithm determines the individual leaves one by one within the 'Surface'. First, the largest leaf is detected, its points are identified and removed from the remaining points in the surface (see Figure 5.15 (a) and (b) for an example of a correctly processed plant model). This step is repeated for each remaining leaf, resulting in a fully segmented 'Surface' (see Figure 5.15 (c) to (h)). Identifying which points correspond to a leaf is not trivial. In Figures 5.16 and 5.17, examples are shown of incorrectly identified leaves. In most cases too many or too few leaf points are removed. This is caused by the algorithm that looks at a thinner part of the leaf closer to the stem. When other leaves are obscuring the view for the algorithm, an incorrect decision is made.

The errors in this algorithm must be solved to enable correct identification of the plant model. For evaluating the method, though, it suffices to manually process the geometric models that lead to incorrect plant models due to bugs in the algorithms according to the protocols defined. An example of such a manual processing of a point cloud is displayed in Figure 5.18.

Leaf type assignment Finally, the individual leaves are classified as cotyledon or true leaf. For most plants, this classification functions well. For 2 plants, an error of type 1 occurs in the identification of which leaf is a cotyledon and which is a true leaf. In the first of these plants, the two cotyledons are identified as true leaves and vice versa. In the second plant, the second cotyledon is seen as a true leaf and vice versa (see Figure 5.19). The leaf classification algorithm should be further improved to overcome the incorrect assignments. Therefore, once again, we process the point clouds manually according to the corresponding procedural knowledge.

We conclude that the processing of the geometric models to create the plant models functions as expected. By following the protocols outlined for the individual processing steps in terms of the geometric and the plant ontology, we have shown that the geometric models can correctly be transformed into plant

models. Moreover, a sanity check rule has been identified and implemented, ensuring that plant models with leaves also have a stem as plant part.

5.2.5 Test: classification of the plant model

Next, the classification of the plant model has to be tested. This involves the comparison of plant assessment according to the formalised classification rules with plant assessment according to a domain expert. To test this classification, we have used the test set of 96 plants. Whenever the automated creation of the plant model was incorrect, we have manually segmented the point cloud into a correct plant model.

In 80 cases, the automated assessment and the expert assessment agree. For the 16 cases in which the knowledge rules and the expert did not agree, we consulted the expert and adapted the classification rules according to his indications. We asked the other domain experts involved to study the explicit form of their classification rules as well. To do this, trays of seedlings were grown and tested with the experts' own sets of knowledge rules. In some cases this led to a company-specific refinement of the rules. With these refinements included each of the experts indicated that the individual set of knowledge rules was suitable for assessing the quality of tomato seedlings.

We note that most experts require a training of several months[2]. Even with this training phase completed, most companies instituted a weekly consultation between the seedling experts to ensure that the experts assess the plants in the same way. Even with such meetings, inter-expert variation in a company is still 5 to 10 percent [83]. The difference between the assessment according to the formalised classification rules and the expert assessment are within the inter-expert difference.

The classification of the plant model is the final step in the computer vision application. We conclude that the use of the explicitly specified quality rules leads to a successful assessment of the quality of the plants under inspection that mimics the assessment of the domain experts.

5.3 Conclusion

In this chapter, we have shown that a white-box computer vision system set up according to the method proposed in this thesis leads to an application that can successfully perform the task for which it has been developed. The input cloud can be transformed into a geometrical model, which is transformed into a plant model. This plant model can be used to determine the quality class to which the recorded plant should be assigned. The point cloud, the geometrical model and the plant model are formally defined in the ontologies shown in Section 5.1.

The problems that we have encountered in the prototype were implementation issues at the level of individual algorithms. For the incorrectly processed

[2]This information is obtained from the interviews with the companies.

point clouds, we manually ensured that the algorithms performed according to the experts' expectations – something that is only possible because of the explicitly defined procedural knowledge. This enabled us to show that the design method can lead to a viable computer vision application. For future use of the computer vision application in an industrial environment, the flawed algorithms will have to be further corrected.

Chapter 6

Evaluation of the Proposed Method

In this chapter, we evaluate the proposed method for designing white-box, knowledge-intensive computer vision applications. We show that the developed white-box application for the case study has four highly desired properties mentioned in Chapter 2: (i) corrigibility, (ii) adaptability, (iii) robustness, and (iv) reliability. Moreover, we go back to the discussion in Chapter 4 for the aspects of (v) understandability and (vi) trust. Additionally, for corrigibility and adaptability, we show how a tool can be implemented to allow the user of the computer vision application to optimally benefit from this property.

6.1 Corrigibility

In the previous chapter we have shown that the proposed white-box design method leads to a viable method and implementation for assessing seedlings. We claim that the white-box setup of the method has an inherent set of additional favourable properties. The first of these properties is *corrigibility*. We define corrigibility as the property that flaws in the computer vision application can be tracked down easily. We stress that we restrict our definition to supporting easy and accurate identification of flaws in the implementation. Actually removing the flaws and making the system perform well may still be a difficult task. To show that the proposed method results in corrigibility, we first describe corrigibility in the case study and show its functioning for several test plants. Next, we discuss what properties of the proposed method ensure corrigibility.

6.1.1 Corrigibility for the case study

Corrigibility is a property that is useful in two situations: (i) when the computer vision application is in development and needs to be tested, and (ii) when it has been implemented, is in use, and assigns a small percentage of seedlings

incorrectly to a quality class. For both cases, it is relevant that the flaw is detected accurately and the application is mended as soon as possible. During the design of the computer vision application, the knowledge models and rules of the experts from different disciplines – for the case study, the disciplines 'computer vision' and 'plant' – have been specified.

By considering the consecutive set of models – point model, point type model, geometric model, pseudo plant part model, plant model – and by listing the knowledge rules used to obtain one model from the previous model, the expert can get insight in the steps performed by the computer vision application. The corrigibility property ensures that these experts can diagnose their part of the system without understanding the black-box algorithms employed. The only requirement is that they understand the intermediate models and explicit knowledge rules. Since these are stated in the domain expert's terms, this condition is met by default. Based on their diagnosis they can inform the software team on the origin and type of the incorrect action.

6.1.2 Tool support: the Diagnosis Module

With the application set up in the proposed way, we can implement a tool – a Diagnosis Module – that makes it possible to detect errors in conceptual terms. In this section, we implement a Diagnosis Module for the case study. The Diagnosis Module is a software module that visualises and explains the subsequent steps that are taken by the application for a specific plant in such a way that the involved experts can autonomously pinpoint the location and nature of the error in the application. As soon as the flaw has been tracked down, the software team has sufficient information to start correcting the application.

Example 1: Incorrect line-plane model In Figures 6.1 and 6.2, two screenshots of the implemented Diagnosis Module are shown for plant 06. This plant is incorrectly assigned to quality class 'abnormal' instead of 'too small'. We have seen that the error originates from an incorrect transition from input model to geometric model in the currently implemented version of the computer vision system.

The plant expert noticed during testing that plant 06 was incorrectly classified. Using the Diagnosis Module, he started to track down the flaw in the system. The Diagnosis Module starts with showing the plant expert the created plant model and asking whether this model is correct or not (see Figure 6.1). The plant model is displayed as a point cloud with coloured points that represent plant parts. A list of identified plant parts is shown as well. For plant 06, the plant expert indicates that the displayed plant model is incorrect, since the parts indicated as leaves do not correspond to the real leaves of the plant. Based on this answer, the Diagnosis Module next offers a visualisation of the preceding, geometric model to the computer vision expert. This model is displayed as a point cloud with coloured points that represent geometric parts. A list of corresponding geometric parts is offered as well. Based on the information shown, the expert indicates that the geometric model is incorrect as well. This answer

prompts the Diagnosis Module to focus on the processing steps that convert the point model into the geometrical model.

Next, the computer vision expert is asked whether the input model is correctly divided into linelike points and planar points. The module offers an image of the original input cloud (for reference) and an image of the line-plane model (see Figure 6.2). An explanation of what the computer vision expert could expect is offered as well: "all points that are roughly on a line should be indicated as 'linelike point', all other points as 'planar points'.". For plant 06, the computer vision expert indicates that the line-plane model is incorrect, since he thinks that the blue points in the middle of the cotyledon should have been indicated as planar points. The diagnosis that the Diagnosis Module offers is: "The error can be found in the transition between input model and line-plane model".

The computer vision expert uses the conclusion of the Diagnosis Module to correct the errors. If these are errors of the first type, the involved algorithm is corrected. If these are errors of the second type, a sanity check rule can be implemented. For the model in Figure 6.1, the incorrect line-plane assessment leads to a geometrical model where a 'Thin cylinder' is enclosed between a 'Surface' and a 'Thin cylinder'. In plant terms, this implies that a 'Leaf or stem' part is attached to a 'Leaf' on one side and to a 'Leaf or stem' part on the other side. The plant domain expert indicates that this situation can never occur in the plant domain. Therefore, a sanity check rule is added to the point domain indicating that a 'Linelike region' that is enclosed by a 'Planar region' should be corrected. We have seen in Section 5.2.3 that implementation of such a sanity check rule leads to improved performance of the system.

We conclude that the corrigibility property of the computer vision application can lead to the identification of task specific sanity checks that can be embedded in the application. Such sanity checks are phrased in terms of the domain experts. In Section 6.3.3, we argue why sanity check rules cannot be defined on beforehand, but need to be identified and formulated when plants are processed for which early decisions lead to incorrect models later in the process.

Example 2: Incorrect classification rules For another plant, the Diagnosis Module shows a different set of models to the expert, since the first model – the plant model – is assessed as correct; the found plant parts correspond to the real plant parts. Next, the Diagnosis Module leads the expert through the components of the classification knowledge rule, indicating for each component the calculated parameters and the intermediate conclusion based on these parameters. The expert can indicate for each step whether the application makes the right decision or not. For plant 18, the number of cotyledons is calculated as 2, and based on this parameter, the plant is maximum first choice (see Figure 6.3). The expert indicates that this is correct. Next, the average area of the true leaves is determined at 88% of the average true leaf area of all plants in the tray, and based on this percentage, the plant is classified as maximum second choice (see Figure 6.4). The expert concludes that the plant should not

have been classified as second choice, but as first choice. The conclusion for the software team is that either the calculation of the true leaf area is incorrect or the true-leaf-area-rule is incorrect. In this way, the Diagnosis Module not only helps to check the implementation of the algorithms and the descriptive knowledge, but also the procedural knowledge that is used in the application.

6.1.3 Corrigibility setup in general

Corrigibility is a property that is inherent to white-box applications set up according to the method proposed in this thesis. Due to the setup of the application in terms of explicit models and processing steps, both the models and the processing knowledge is available in the domain experts' terminology. Therefore, each step in the application can be studied by looking at the input and output models and the corresponding knowledge rules. This gives us exactly the framework required to accurately identify the location and nature of any flaws in the implementation.

The case study shows us how corrigibility is obtained and even how to set up a dedicated Diagnosis Module to provide user-friendly support to domain experts. Such a Diagnosis Module can be added to the application to allow the domain expert to easily make a diagnosis of the system. The corrigibility check of a program is strongly linked to the architecture of the white-box application itself. If the computer vision application is changed in terms of application ontologies or transition functions, the Diagnosis Module can automatically be changed accordingly.

Setting up a Diagnosis Module for an arbitrary white-box application as proposed in this thesis can be done by following the same procedure as used for the case study. We have to create a software program that can show the model identifying the instance that was incorrectly assessed in a backward-chaining fashion. We start by showing the 'last' model – the model that corresponds to the input of the last set of procedural knowledge – we continue with the preceding model, until the domain expert indicates that a model is not flawed. In that case, the program has identified the component for which the input model is correct and the output model is incorrect. Next, the set of knowledge rules corresponding to the creation of the incorrect output model is to be checked. Again, the program should start with the model that is the input for the 'last' knowledge rule of the component. The information corresponding to this step can be shown to the expert and the diagnosis can be communicated to the software team. Note that a Diagnosis Module can be set up in a similar fashion when the program contains parallel instead of linear programming steps.

In this section we have shown that both the domain expert and the computer vision expert play a role in identifying errors. More generally speaking, each expert that has been involved in the development of a white-box application is involved in the corrigibility procedure. The involved individuals are experts only within their own domain. In the corrigibility diagnosis, they are each offered information from an intermediate ontology in their own terms. They can therefore each check the performance of their own part of the application.

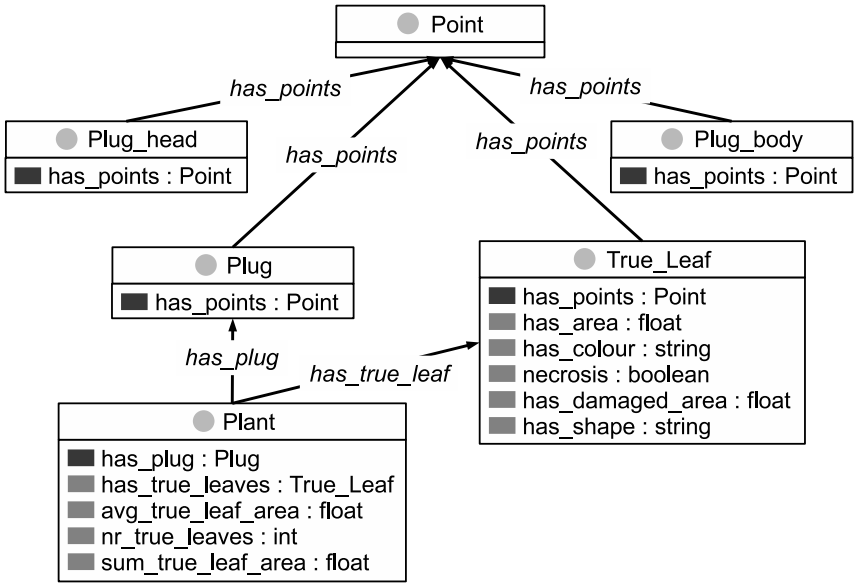


FIGURE 6.5: *A schematic representation of the Brassica plant ontology.*

In conclusion, we have shown that corrigibility is realised in white-box applications set up according to the framework proposed in this thesis. Moreover, we can identify sanity check rules that can be added to the application to enhance its performance. We have also shown that the users of a white-box application can be supported with respect to corrigibility by using a dedicated Diagnosis Module.

6.2 Adaptability

The second of the claimed properties of white-box systems is adaptability. Adaptability in the context of computer vision systems is defined as the possibility to (easily) change the software application with respect to the used quality rules for assessing the objects under inspection, or to adapt the software to assess objects that are different from but morphologically similar enough to the objects for which the software application has originally been developed. In the next sections, we show that the case application is adaptable, show a tool to support adaptation by the end users of the application, and discuss adaptability for general white-box computer vision systems set up using the method proposed in this thesis.

6.2.1 Adaptability criterion for the case study

Adaptability is, like corrigibility, a property that is of importance when the computer vision application has been implemented and is in use. In this section,

we focus on the desire of a company to adapt the computer vision application developed in the case study in such a way that it can be used to assess different but similar cultivars. For this process, both a domain expert and a knowledge engineer are involved. The task of the domain expert is to indicate the differences between the cultivars and their assessment; the task of the knowledge engineer is to formalise these differences and embed them in the existing application ontologies and knowledge rules. The consequences of our approach are that these modifications have minimal impact on the actual implementation level. They introduce a minimal amount of additional work for the software developer.

To illustrate the adaptability property, we elaborate on an adaptation of the horticultural case study in which the cultivar of interest is no longer tomato but brassica (cabbage). Brassica is, like tomato, a dicotyl plant, and the quality assessment process of brassica seedlings is similar to that of tomato seedlings. To adapt the existing software, we take the following three steps:

- First, we identify the *plant application ontology* corresponding to brassica seedlings.
- Next, we formulate the *quality assessment rules* that are applicable; these are part of the formal assessment knowledge.
- Finally, we *backpropagate* the required additional plant knowledge to the geometric ontology and the point ontology and to the corresponding procedural knowledge to ensure that the computer vision system can deal with the adapted requirements in all steps.

For the case study we have chosen to create the brassica plant ontology using an interview-only approach with the existing tomato knowledge as input information. We have interviewed two experts from a seed company that is interested in assessing brassica seedlings. In Figure 6.5, the created ontology is shown and its reusability is considered. Note that the brassica ontology does not contain the concept 'Cotyledon', which is important for tomato seedlings. Moreover, different plant features are defined for brassica seedlings, such as 'damaged area' and 'necrosis'. The other concepts from the tomato plant ontology are valid for the brassica plant ontology as well and are hence added to the brassica plant ontology.

Next, we have asked the domain experts which tomato rules are still applicable in the brassica domain, which should be adapted, which are to be discarded, and which new rules should be added to the set of brassica assessment rules. We have used the set of all identified tomato seedling assessment rules as the basis for this step. These rules were broken up into sections corresponding to tomato seedling plant parts to facilitate the discussion. We have asked the domain experts to identify for each relevant plant part whether the specified tomato rules were relevant for the brassica seedlings, or whether they should be adapted, discarded or replaced. Part of the resulting changed document is displayed in Figure 6.6. We see that in the first rule, the value that distinguishes the good plants from the abnormal plants has been changed from 50% to 25%. The second rule is adapted in such a way that it gives the same information as

1. If the true leaves are well developed (*i.e.* larger than ~~50%~~ **25%** of the average true leaf area of the plants in the tray), then the plant is a first choice plant.
2. If the true leaf area is small (~~between 25% and 50%~~ **less than 25%** of the average plant), then the plant is ~~a second choice plant~~ **an abnormal plant**.
3. ~~If the first true leaf is well developed, but the second true leaf is relatively small with respect to the average second true leaf, then the plant is a second choice plant.~~
4. ~~If the plant is smaller than the average plant, but the build of the plant is well, then the plant is a second choice plant.~~
5. If the ~~true leaf is damaged~~ **damaged area of the true leaf is larger than 50% of the true leaf area**, then the plant is a ~~second choice plant~~ **an abnormal plant**.

FIGURE 6.6: *Part of the changes required for the brassica knowledge rules.*

the first rule. The third and fourth rule of the tomato rule set are not relevant for brassica plants. The fifth displayed rule introduces a new concept: the damaged area of the true leaves relative to the total true leaf area. The whole set of brassica rules was then formulated and presented to the domain experts for verification[1].

The final step required a study of the impact on the other parts of the system of the newly defined plant application ontology. All changes with respect to the tomato ontology were reviewed, to see if they affected the geometrical ontology or even the point ontology. The procedural knowledge affected by the changes in the ontologies were reviewed as well. Contrary to the tomato ontology, the brassica ontology does not contain the concept 'Stem'. The geometrical ontology corresponding to the tomato-based computer vision application contains the concepts 'Thick cylinder', 'Thin cylinder', 'Paraboloid' and 'Surface'. We know from the explicit procedural knowledge that the geometrical ontology does not need the concept 'Thin cylinder'. Similarly, the point ontology has no need for the concept 'Linelike region'. The corresponding procedural knowledge was removed from the ontologies.

6.2.2 Tool support: adapting knowledge rules concerning quality assessment

The adaptability criterion requires that the computer vision application can easily be adapted to changing situations. To allow end users of the application to benefit optimally from the adaptability property, we propose the implementation of an Adaptation Module. Such an Adaptation Module assists the domain expert to *autonomously* adapt quality assessment rules according to his needs. This is especially useful when the application has to assess the same objects – tomato seedlings – but with different quality criteria. A change like this can be

[1]For confidentiality reasons, the presented set of rules is not the actual set of rules.

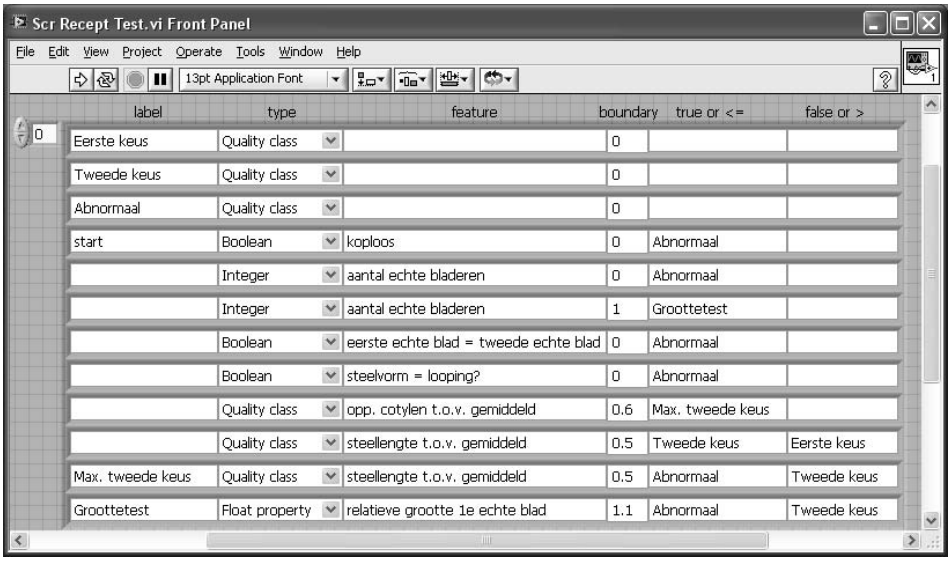
desired by a company because of strategic or commercial reasons. It is important that the computer vision application is flexible and can adapt easily when the quality definition used by the company changes.

One possible adaptation is to *change the decision value of a feature*. As an example, a company can change the rule "if the feature 'leaf area' is smaller than 50% of the average leaf area, then the plant is second choice" to "if the leaf area is smaller than 30% of the average leaf area, then the plant is second choice". Another possibility is to *add or remove an entire quality class*. One can for example indicate that the distinction between 'first', 'second', 'third' choice and 'abnormal' plants is too detailed; the quality classes 'useable plants' and 'rejected plants' are to be used instead. In other cases, a company may decide to add even more detailed quality classes to better study the seedlings under consideration. As a third possibility, a company may *introduce new features or reject features that are already in use*.

When new features are introduced, the software team obviously has to play a part in implementing the algorithms to calculate the new features. In all other cases, the domain expert should be able to make the changes on his own. A custom-made Adaptation Module can be used to change the quality assessment rules. The developed Adaptation Module for the case study is depicted in Figure 6.7. It contains a text-based representation of the set of knowledge rules that is the basis for the quality assessment. The first few lines in the module indicate the quality classes that are available in the application[2]. In the example shown in the figure, the quality classes are 'Eerste keus' (first choice), 'Tweede keus' (second choice), and 'Abnormaal' (abnormal). In the knowledge rule overview, the second column indicates that these lines correspond to quality classes. To change the quality classes that are allowed in the system, new classes can be added, existing classes can be renamed or removed in this part of the Adaptation Module.

The next lines contain individual knowledge rules, each corresponding to a component of the procedural knowledge. The first knowledge rule is identified by the label 'Start'. The underlying software executes the knowledge rules starting with this line. Each line consists of a label, the type of relation, the feature for which the plant is checked, the assessment value, the decision if the feature is below the assessment value, and the decision if the feature is above the assessment value. In the example, the 'start' line checks whether the plant is budless. If this is the case, the plant is abnormal, otherwise, the next knowledge rule is to be considered (the default action if no value is specified). The next line indicates that if the number of true leaves is less than one, the plant is abnormal. The third line indicates that if the number of true leaves is less than two, then the line labeled as 'groottetest' (check size) should be executed next, otherwise the fourth line is next. If the 'groottetest' (check size) line is executed, the relative size of the first true leaf with respect to the average size is checked. If this is smaller than 1.1, the plant has only one true leaf (otherwise 'groottetest' (check size) would not have been executed) and this is smaller than

[2]In a later version of the Adaptation Module, this information could be encoded separately from the knowledge rules to make the module more user-friendly.



	label	type	feature	boundary	true or <=	false or >
0	Eerste keus	Quality class		0		
	Tweede keus	Quality class		0		
	Abnormaal	Quality class		0		
	start	Boolean	koploos	0	Abnormaal	
		Integer	aantal echte bladeren	0	Abnormaal	
		Integer	aantal echte bladeren	1	Groottetest	
		Boolean	eerste echte blad = tweede echte blad	0	Abnormaal	
		Boolean	steelvorm = looping?	0	Abnormaal	
		Quality class	opp. cotylen t.o.v. gemiddeld	0.6	Max. tweede keus	
		Quality class	steellengte t.o.v. gemiddeld	0.5	Tweede keus	Eerste keus
	Max. tweede keus	Quality class	steellengte t.o.v. gemiddeld	0.5	Abnormaal	Tweede keus
	Groottetest	Float property	relatieve grootte 1e echte blad	1.1	Abnormaal	Tweede keus

FIGURE 6.7: *A screenshot of the user interface in which the domain expert can change the used knowledge rules. In this screenshot, seedling assessment rules are displayed.*

110% of the average true leaf size. The plant is assessed as 'abnormal'. If it is more than 110%, the plant is assessed as 'second choice'.

Example 1: change the decision value of a feature To change the decision value of a feature, the domain expert simply has to adapt its value in the Adaptation Module. For example, the rule that checks whether the number of true leaves is greater than 1 can be changed to a rule that checks whether the number of true leaves is greater than 2 by changing the value in the fourth column.

Example 2: removing and adding knowledge rules If an expert wishes to remove a rule, he can simply delete it from the list of rules. If he is interested in adding an additional rule, he can either use existing features and quality classes or indicate that a new feature is called for. In that last case, the expert can define the quality feature, but has to contact the software team to implement the desired feature.

The Adaptation Module allows the expert to compose new rules based on existing building blocks. Features that have already been implemented can be combined with other features and desired decision criteria. This allows the experts to adapt the quality criteria to meet changing quality standards.

6.2.3 Adaptability setup in general

In the previous sections, we have looked at the adaptability property for the case study. Adaptability is, like corrigibility, a property that is possible because

of the white-box approach. It allows the users of the system the flexibility of using it in different application fields. More specifically, users are enabled to autonomously make changes to the system or to instigate the development of new functionality by computer vision specialists. This is a very valuable property for industrial use of such a computer vision system.

In this section, we argue why any computer vision system that has been developed using the method discussed in this thesis has adaptability as an inherent property. Moreover, we show how a user of such a computer vision system can in fact adapt the application to meet changing requirements.

We have seen that in the case study, adaptability with respect to the classification features is enabled by the explicit and transparent setup of procedural and descriptive knowledge. Due to the explicit definition of the knowledge rules, the expert can understand the steps that the computer vision system takes *in his own vocabulary*. For any similarly developed computer vision system, the procedural knowledge is defined in such a way that it is understandable for the expert. Adaptability is based upon the expert's understanding of the knowledge rules. Only in that way, the expert is capable of modifying the classification function to suit a new purpose.

Moreover, adaptation of the system to similar but different objects is made possible by the transparent setup of the whole system. Explicit definition of the procedural knowledge and explicit specification of the application ontologies are requirements for the easy adaptation of the computer vision system. This holds true for any similarly developed application. To easily identify the changes needed in the system caused by a change of objects under inspection, a clear understanding of the transition functions and models that are in play is required.

However, it is not sufficient that the white-box computer vision system has transparency as an inherent property. If the system cannot be easily adapted *by the experts*, we cannot truly say that the system is adaptable. We have to show how in a general application, the user can be enabled to make the required changes.

If the desired adaptation involves adapting the computer vision system for different objects, the specification of the current application domain ontology will be adapted. To do this, the ROC method with the current application domain ontology as one of the basic sources can be used. Alternatively, an interview-based knowledge acquisition method can be applied, with the current knowledge model as starting point. Since the existing knowledge is used as a basis, and the new situation is similar to the existing situation, it is to be expected that changes are small and hence easy to carry through.

For changes in the classification knowledge, a custom-made Adaptation Module can be implemented to support the domain expert in his efforts. The information needed to set up such a module is defined in the explicit description of the imperative and declarative processing knowledge of which the transition function consists. Since the domain expert's assessment is available in the form of decision rules, a rule editor can be implemented as in the case study. Such a rule editor can be used by a domain expert to autonomously make the desired changes and thereby adapt the system.

6.3 Robustness and reliability

In this section, we consider the two criteria robustness and reliability together, since these two criteria are closely related. *Robustness* is defined as the property that any object that is in the scope of the task and domain for which the application has been built can be correctly assessed by the application. Moreover, robustness ensures that the application is insensitive for irrelevant defects and variations of the inspected objects. *Reliability* is defined as the ability of the application to indicate when an object lies outside its area of expertise. In some cases objects may be offered that are out of scope. Instead of blindly assessing such an object, it would be preferable if the application would indicate that the object lies outside its area of expertise.

Robustness and reliability are two sides of the same coin. They both are connected to the scope and domain of the application. Together they promise that all objects within the scope are recognised properly, and all objects outside the scope are identified as outliers.

6.3.1 Robustness

Robustness is a property that is intrinsically pursued due to the white-box setup of the application as discussed in this thesis. The domain experts have explicitly defined which features of the objects under inspection are relevant in the assessment of the object. They have indicated which seedling variations can occur and what should be the response of the computer vision application to these variations. Note, that properties that are irrelevant for the assessment are not mentioned in the descriptive or procedural knowledge. Irrelevant defects – *i.e.* defects in properties that are not used for the assessment of the object – do not disturb the assessment process, since they are not considered by the computer vision application.

Robustness is ensured by explicitly formulating the type of objects that can occur. Hence, robustness only holds when the expressed knowledge covers the complete domain on which the task needs to be executed. The setting of the scope is a crucial process to ensure robustness. It should be carried out accurately.

6.3.2 Specificity and selectivity

We consider in this section a practical aspect of robustness. When we make a decision inside the scope of the application, we would like to be certain that when a decision is made, it is indeed the correct decision; we are interested in a high *specificity* of the application. However, many algorithms are not very *selective*. In this section, we show how the availability of explicit domain knowledge can help to improve selectivity and hence specificity.

Let us, for example, look at the planarity measure that is calculated in the point ontology to determine whether a point is located in a linelike or a planar environment. To this end, a principal component analysis is performed on the point and its environment, and based on the length of the second eigenvector

with respect to the sum of the lengths of all eigenvectors, a decision is made on the planarity of the point. A threshold value determines the boundary between linelike points and planar points. When the planarity measure is very different from the threshold value, the point is very clearly linelike or planar. However, when the planarity measure is close to the threshold value, there is an uncertainty in the point type assignment.

Due to the explicit availability of domain knowledge we have additional knowledge to make the decision more certain. Suppose that some of the points with an uncertain assignment to the point type 'linelike' are later on found to be in the middle of a 'Surface' (and hence a 'Leaf'). Then, we can derive that the assignment was incorrect, since from plant knowledge, we know that a leaf only contains 'planar' points. By maintaining the information that the point assignment was uncertain, a correction based on domain knowledge can be incorporated. This ensures a *de facto* increase of selectivity and hence a more reliable application.

6.3.3 Reliability using knowledge rules

Reliability is a property that is more difficult to guarantee than robustness. The computer vision system is based on explicit knowledge defining which plants are inside its scope. These plants will be assessed as first, second, third choice plants or as abnormal plants. However, plants that lie outside the scope of the application, *e.g.* parsley or tulips, will probably be rejected, but they are not explicitly recognised as out-of-scope plants, but simply as abnormal tomato plants. This is undesirable, since for a company it is important to (i) sell seeds with preferably no pollution from other species, and (ii) to assess only the quality of the intended seeds, not lower the quality indication due to chance pollutions present in the planted seeds.

For an application to identify which objects lie outside its area of expertise, exclusion criteria can be defined and encoded using knowledge rules. There are some nuances to the use of such excluding knowledge rules. Some of these rules relate to criteria that are natural to the expert; an example in the case study is to state that a plant can only have one stem, only one plug and maximum four cotyledons. Plants with more cotyledons (*e.g.* parsley) or more stems (*e.g.* onions) do not meet the criterion. They may occur, but must be recognised as 'strange plants'. Other rules can be expressed explicitly, but are not naturally a part of the domain expert's description of the domain. An example is "a plant with leaves connected to the stem where it exits the plug is not a tomato seedling" (but for example a tulip). The occurrence of such objects in the task domain is so unlikely that it seems unnatural to add the corresponding rules to the ontology.

In short, full reliability would require an exhaustive list of rules to describe all objects outside the scope, and is therefore not feasible. Apart from it being impossible to obtain a complete list, it is not the most practical solution from a performance point of view. Users of a computer vision system are interested in recognising and rejecting objects that lie outside the scope. It makes sense to only add knowledge rules to recognise foreign objects that may actually occur

in the inspection environment. Bell peppers and other vegetable seedlings are grown in the same environment as tomato seedlings; tulips are not.

Both sanity check rules and for reliability rules are applied on a 'learning by seeing' basis. By adding rules only when triggered by an unexpected model respectively object, the rules are set to deal with naturally occurring cases. This allows the system to gradually become more and more attuned to its setting.

6.3.4 Proposed reliability rules for the case study

To ensure reliability in the case study, the knowledge engineer and domain expert have to explore the scope boundaries and the domain environment. It makes sense to start with the plant domain. Another source of input for reliability rules are the sanity checks that are identified by the corrigibility procedure. The domain expert can *e.g.* indicate that a tomato plant can only have one stem, only one plug and maximum four cotyledons. These restrictions can be expressed by the knowledge engineer in the ontology modelling language OWL using *owl:Cardinality* respectively *owl:maxCardinality* constructs.

The expected shape of the cotyledons and true leaves can be added to the plant ontology to ensure that *e.g.* a bell pepper seedling is not allowed to pass as a tomato seedling. The corresponding rule would be 'the true leaf has to contain indents, otherwise the plant under inspection is not a tomato seedling'. This rule can be expressed using Jess as a rule that calls an algorithm to determine whether such indents are present or not.

For the geometric ontology, some reliability rules in the plant ontology translate to reliability rules on the number of thin cylinders that could be present in the object (namely one) and the number of thick cylinders (maximum one). The number of surfaces can still vary, since the transition function between the geometric ontology and the plant ontology does not translate each surface to exactly one leaf. The geometric ontology will also contain rules that have no counterpart in the plant ontology. An object can *e.g.* lie outside the scope of the ontology when it consists of other geometrical shapes (such as cubes, spheres, tetrahedra). It is up to the domain expert to decide which rules correspond to non-exotic occurrences of abnormal objects.

Finally, we consider the point ontology. We have designed this ontology based on the fact that the representation of the recorded tomato seedling consists of points that form linelike or planar regions. To implement the reliability criterion, we state that if a region occurs that is not linelike nor planar, then the object under inspection lies outside the scope of the application. To fully cover this statement, we should enrich the ontology with a measure that indicates the linelikeness and planarity of the present regions. If a region is not sufficiently linelike or planar, the computer vision application should indicate that the object under inspection lies outside its scope. The value of the indicated measure can be determined empirically. This rule can be implemented using a Jess rule that calls upon an algorithm to determine whether the planarity measure is within the correct range for the plant to be a tomato plant.

6.3.5 Reliability in transitional situations

So far, we have discussed reliability in cases where on rare occasions an unexpected object is offered to the computer vision application. However, in practice a different situation may occur in which reliability is relevant. Suppose that the computer vision application is used to alternately assess tomato seedling batches and bell pepper batches. In such cases, it would be necessary that the application can assess both tomato and bell pepper seedlings. We can imagine a scenario in which the application can automatically indicate whether the new tray of seedlings is a bell pepper or a tomato tray and what the quality assessment of the individual seedlings is. This would be convenient for the operators of the machine, since they do not have to enter this information manually on beforehand.

We might be able to equip the computer vision application with the possibility to use both task models at the same time. The result of an assessment can be that based on the tomato knowledge, the seedlings are mostly abnormal and third choice, whereas the bell pepper knowledge indicates that most seedlings are first choice bell peppers with a small number of second choice seedlings. As a conclusion, the application could decide that the inspected batch is most likely a bell pepper batch, but has a small chance of being a low quality tomato batch.

The envisaged application could indicate whether the inspected batch is a batch of tomato seedlings or of bell peppers and provide a probability measure for this decision. By allowing the two knowledge models to be used simultaneously, the application has the possibility to look at a batch of input objects from two different but frequently occurring view points and determine the validity of each of these views. It thereby gains the possibility to assess whether an object lies outside one domain but inside another domain. This is a valuable property of the computer vision application in practical situations.

6.4 Conclusions

In this chapter, we have shown that the application has corrigibility, adaptability, and robustness as inherent properties and can be equipped with axioms to ensure reliability as well. These properties offer a significant advantage to the user of the computer vision system over black-box systems.

Due to adaptability, the user is allowed a certain *degree of freedom* in applying the system to (slightly) different tasks and domains. Corrigibility allows the user to get a *clear insight* in the decisions taken by the system and in pinpointing flaws that the system makes. Robustness and reliability deal with the property of the system that all objects within the specified scope are properly assessed and objects that are outside the scope can be identified as such. The level of reliability required by the user is dependent on the environment in which the application is used and its desired performance. Robustness and reliability *ensure appropriate behaviour* of the computer vision system, even when something unusual occurs.

In Chapter 2, two additional properties have been mentioned: *expert acceptance* and *speed*. These properties have not been discussed in this chapter. Expert acceptance was discussed in Chapter 4 in two forms: understandability and trust. Understandability, required for organisational learning, can be supported due to the direct pairing of transparency in the application and the underlying reasoning behind the application steps. We have shown that training material can easily be obtained when a transparent application has been created (see Figure 4.8). Trust in an application was achieved by designing the application on the proper level of transparency. By adding transparency in such a way that the most detailed level of the application contains either explicit domain knowledge, trusted third-party components, or trivial components, the expert's trust is built.

Speed, the last mentioned objective, is not applicable to the prototype system that we have created so far. However, the industrial parties that have been involved in this project have decided to continue with a commercial follow-up of this project. In this new project, the approach proposed in this thesis and the pilot software will be further developed. The goal of this follow-up project is to develop an automated seedling inspection machine that is commercially available within the next two years. The *speed* requirement for this machine is that it can handle between 10,000 and 20,000 plants per hour. One of the major bottlenecks, fast image acquisition, has recently been solved by us [61]. From first experiments, it is our expectation that the required speed can be met for the processing of the plants as well.

Chapter 7

Discussion & Conclusion

This thesis is built on three paradigms: (i) the choice to develop a white-box instead of a black-box computer vision system, (ii) the choice to consider the domain expert as key person in the design process, and (iii) the choice to develop a design framework that combines techniques from software engineering with knowledge engineering. In this concluding chapter, we critically assess how these paradigms should be approached in practice.

The last section in this chapter provides the answers to the research questions posed in Section 1.7 and the overall conclusion of this thesis.

7.1 White-box systems complemented with opaque elements

The first paradigm that we review is our choice to design *white-box* computer vision applications. In Chapters 4 and 6 of this thesis, we have shown that white-box systems based on explicit expert knowledge result in the important benefits of transparency, adaptability, corrigibility, reliability, robustness, maintainability and expert acceptance. In addition to these advantages, the white-box design approach has some useful side effects. Due to the explicit formulation of the expert's domain and task knowledge, the explicit knowledge can easily be adapted *e.g.* for instruction of new employees.

In the case study, experts have expressed their task knowledge: they have indicated which plant features are relevant for assessing the quality of the plants and for which values of these features the plant should be assessed as a first class, second class or abnormal plant. Such explicit task knowledge can easily be represented in a flow diagram that is interpretable by the experts. Such a diagram gives a concise overview of the task knowledge and is useful in training new employees.

Implementing a computer-vision system in a white-box fashion may have potential disadvantages, though. First, the process of explicating the expert knowledge is time-consuming. For each inspection task, a careful consideration must be made on whether a white-box approach is the best choice based on the actual

amount of knowledge required for executing the task versus the ease with which can be learned from example cases. Another factor is how often the inspection process will have to be adapted for similar objects or different quality criteria. A third factor is the expected need of the experts to communicate about the actual steps taken in the inspection process to colleagues or third parties. We recommend that a project for designing any computer vision application starts with interviewing the problem owner and one or more experts to learn about the complexity of the task and taking inventory of the availability of training data. Based on these interviews, a decision can be made for the type of computer vision system that is required.

At the beginning of the horticultural project, the companies involved explained how the different experts needed a few months of training before they were able to assess the seedlings properly. This information was used by us to conclude that the seedling inspection task indeed is a knowledge-intensive task and that a white-box design is applicable.

The second possible disadvantage of using explicit knowledge can be the negative effect on the required speed of the application. Especially when the computer vision application has to perform in a real-time high-throughput setting, speed-optimised software is required. It is, however, difficult to decide beforehand how an implementation based on explicit knowledge will perform in terms of speed relative to an implementation using embedded implicit knowledge. Subjective factors such as the programming skill of the individual programmer, the chosen encoding of the algorithms, memory requirements, all influence the speed of a computer program.

A distinct advantage of white-box design over pure black-box design is that the task and domain decomposition of the task in the white-box design allows the subcomponents to be profiled or benchmarked [51] to determine which process step is slowest. The optimisation effort can then be targeted to this process step.

The success of designing a white-box system based on expert-knowledge depends on the possibility to correctly explicate expert knowledge. We submit that if only part of the expert knowledge can be explicated, the proposed design method is still of value. For the parts of the task where tacit knowledge cannot be expressed, the software engineers can implement a grey-box or black-box component in the white-box setup. This combination of white-box and grey-box or black-box components occurs on several scales.

At the lowest level of the application, processing knowledge is specified in terms of complex calculations. Such calculation-based knowledge rules (see Chapter 4) contain grey-box or black-box components that determine the value of the property that is to be calculated. In most cases, the algorithm is known in the form of interpretable equations, and hence a grey-box component is used. In some cases though, *e.g.* in the case of neural networks, the inner workings of

the algorithm are unknown in interpretable terms. In such cases, a black-box component is used to automate the calculation of the property.

The implementation of grey-box and black-box components in a white-box system to represent tacit expert knowledge is a good solution to designing systems with as much explicit expert knowledge as possible. When explicit knowledge becomes available, *e.g.* from a different domain expert, the black-box component can be replaced by a grey-box or even a white-box component.

The procedural knowledge in the case study indicates how to interpret raw vision data by grouping it into larger, more meaningful units (from single points to point groups to planes and cylinders and so on). When we focus on determining the point type of the central point in a point group, we use a grey-box method to calculate the principal components of the point group. By looking at the length of the first, second and third component, we interpret the shape of the point group and decide whether the point group is linelike or planar.

In the case study, several trusted third party components are used. Examples are the PCA-analysis and the Levenberg-Marquardt fitting algorithm.

7.2 The expert as the key person in the design process

We now discuss the central role of the expert in the design of a computer vision application. Since a white-box application is based on explicit expert knowledge, a precise encoding of this knowledge is required. The focus on expert knowledge puts the domain expert in a central position in the design phase.

For a successful involvement of experts in the design process, we require a (i) motivated expert to cooperate in designing the computer vision application. Moreover, the expert has to (ii) have the opportunity and (iii) the ability to express his knowledge [99]. In knowledge sharing research, the reluctance of experts to share their specialistic knowledge is widely recognised as an inhibition to knowledge elicitation [5, 9]. Motivation is potentially adversely influenced by several factors [92]. For our discussion, we highlight job security, expert recognition, and organisational climate. *Job security* is a difficult aspect when automating tasks. The expert may (rightly) fear that the application that is developed may take over his job. This may in a negative way influence the expert's willingness to express his knowledge. *Recognition* of the practitioner as the company's expert on the subject may improve his motivation. An *organisational climate* in which discussing the task and sharing each other's views is common practice will also have a positive contribution to the expert's motivation. Below, we discuss these points for the case study.

In the horticultural case study, the experts did not show any fear for losing their jobs. In none of the interviews, this point was raised. In each interview, the respective expert was very willing and open in sharing his knowledge. We suspect that fear did not play a role, because only part of the expert's task is assessing the quality of seedlings, and the

remainder of the work is enough to provide a full-time, less repetitive, job. Moreover, the experts are used to automation in their work environment. Dutch horticulture is a very innovative sector in which new technologies and automated processes are adapted on a regular basis [108].

Recognition is a factor that works in favour of knowledge sharing in the horticultural case study. Experts are the carriers and guardians of the company's task knowledge. With the computer vision application in place, the experts will have an important role in ensuring that the process is performed correctly. When changes in the process are required, these experts will be responsible for ensuring that the new requirements are correctly communicated to the computer vision application designers. Moreover, the experts will be asked to communicate with other companies' experts on maintaining and improving the high quality standard for vegetable seedlings. In short, they get credited for their expertise.

Finally, the organisations involved in the case study already have a knowledge sharing procedure in place. In most organisations, the quality assessment experts come together once a week to jointly assess a tray of seedlings and to discuss why not all plants have received the same quality assessment. Therefore, discussing their knowledge with others was not a new task for the experts.

With respect to the expert getting his opportunity to express his knowledge, the second prerequisite, it is important that experts who are to participate in designing a computer vision application can indeed do so. The management has to grant them time to discuss with knowledge engineers and software engineers, and to explicate their knowledge as required.

For the case study, opportunity was no point of concern. Each company allowed experts to participate in the process. Whether we interviewed the experts in their work environment, or whether we asked the experts to meet with other company's experts on a central location, the experts could devote the required time to the process.

The ability of experts to express their knowledge is the third factor to characterise the effectiveness of knowledge acquisition. When an expert is motivated and available, but lacks the ability to communicate his knowledge, the knowledge acquisition process will not be successful. Dimensions that are of interest for the expert's ability are (i) the stage of development of the expert [64], and (ii) the expert's ability to analyse and explain his knowledge to others [21]. In general, practitioners that are in the transition process from novice to expert are considered to be best capable to analyse and explain their knowledge [21] for knowledge elicitation using the interviewing method. The ROC-method also requires experts that can explicate their knowledge. However, due to the prompting of possibly relevant concepts from reconstructed sources, the ROC-method enables both intermediate and full experts to make their knowledge explicit.

In the case study, we interviewed both intermediate and full experts to obtain explicit domain knowledge. In one interview, we found a strong indication that the full expert that we talked to had trouble expressing his knowledge. He indicated that a plant is only assigned to the highest quality class, if this plant has 'body'. We asked him to explain this term, and he indicated that the plant should have 'spirit' and 'mass', and that you could 'just see' those properties when looking at the plant. No further explanation on these descriptions could be elicited from this expert. In general though, the interviewed experts were capable of explaining their knowledge.

To ensure that the motivated expert can indeed significantly contribute to the design of a knowledge-intensive computer vision application, sufficient support for the expert has to be available. In Chapter 3, we have covered this aspect by introducing the ROC component. The ROC component allows the domain expert to *autonomously* define descriptive knowledge relevant for the task, thereby empowering him to fully participate in the computer vision design process. In the next paragraphs, we assess future developments of ROC required to provide even better support to the expert.

For an efficient use of the ROC component, it is vital that suitable knowledge sources are available. Moreover, the existing knowledge sources must be available in a format that is usable by the ROC component. At present, we mainly use SKOS, RDF and OWL models to populate the repository of reusable sources. In the future, we expect that more and more additional structured resources will become available via *e.g.* SPARQL endpoints. Moreover, we expect that natural language processing (NLP) techniques will enable harvesting knowledge from texts. These developments will make more sources available for the ROC component.

A function that is not available in the ROC-tool yet, is the possibility to explicitly document the used concepts and relations. Adding definitions, perhaps even discussions that led to these definitions, will give a clearer understanding of the created knowledge model.

At present, the ROC component supports the domain expert in expressing concepts and relations between these concepts. Automated support for expressing domain statements via a 'ROC for Rules' component is not available yet. Such a tool would be valuable in the design of knowledge-intensive applications. The purpose of this 'ROC for Rules' component would be to allow experts to autonomously express rules that make inferring of new instances possible. Moreover, the expressed rules can be (partly) reused for similar tasks or domains.

The ROC component and a future 'ROC for Rules' tool change the role of the knowledge engineer even further to a person who coaches the domain expert in expressing his knowledge instead of a person who must obtain full knowledge of the expert's domain. The knowledge engineer will always need some knowledge of the domain, but with the domain expert creating the proto-ontology, the necessity for having a detailed active understanding of the domain is reduced.

The design method presented in this thesis is suited for single-expert and multi-expert knowledge models (see Figure 3.1). When asking multiple experts to express their domain knowledge, there is a risk that the experts contradict each other. In those cases, it may be difficult to coach the experts in changing their conflicting models and discussing with each other to obtain common ground. When multiple experts are able to elaborate on each other's models and together express a richer model than each individual model, this common model represents a better conceptualisation of the domain.

7.3 Combining knowledge engineering and software engineering

The third aspect that we single out is the proposed combination of software engineering with knowledge engineering (see Figure 2.2).

Without this combination, the computer vision application would be implemented solely by expertise from the software engineering field. The application would be divided into image acquisition, segmentation and classification and for each of these steps the computational steps and algorithms required to assess the recorded objects would be specified (see Figure 7.1, right-hand side). In this way, a grey-box or black-box computer vision application would be created.

Knowledge engineers would traditionally not be asked to assist in the automation of processes, but would more likely be asked to assist in explicating and transferring the knowledge of task experts within the company. Hereto, they would define the scope, interview experts to obtain an overview of their descriptive knowledge, combine the descriptive knowledge into a company-wide model, and enrich these models with procedural knowledge rules (see Figure 7.1, left-hand side). In this way, expert knowledge is made explicit and can be used to *e.g.* train new experts or assist in domain-specific research.

The proposed design method combines the fields of knowledge engineering and software engineering and has a white-box computer vision system as a result. Useful byproducts are an explicit model of the descriptive knowledge that can be used in research, and an overview of the know-how knowledge that can be used to train experts and to adapt the process to new requirements.

The co-operation between knowledge engineer and software engineer follows the workflow defined in Figure 2.2. In this paragraph, we briefly describe the products that they exchange during their cooperation.

Both types of engineers and the problem owner are involved in defining the task and setting the scope of the application. The software engineer creates a document in which the task is decomposed into subtasks interspersed with intermediate interfaces; the global software architecture. The knowledge engineer uses the descriptions of the subtasks to identify corresponding domain experts. These experts are asked to explicate the relevant domain knowledge for their subtask and domain. This results in ontologies that are refined by using one of the methods discussed in Chapter 3. The ontologies are offered to the software engineer in the form of diagrams or class objects. The software engineer can use

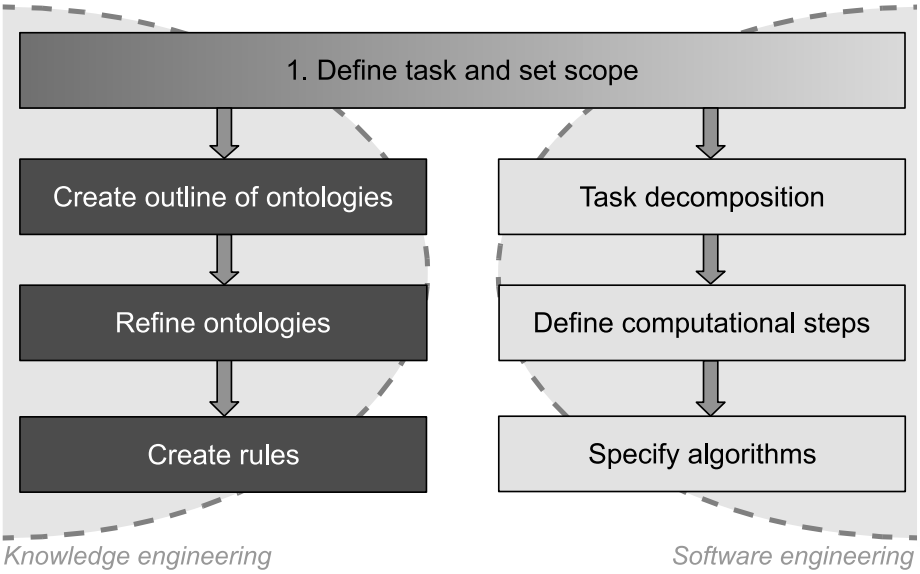


FIGURE 7.1: *Software engineering can lead to a black-box application, knowledge engineering to a knowledge model.*

these diagrams and his earlier decomposition of the task to define the computational steps required in domain-specific terms. The domain expert is consulted to offer his procedural knowledge, and a process flow on paper results. The knowledge engineer uses the process flow to identify fact-inferring statements. These statements can then be implemented in a declarative programming language. The software engineer uses the list of required calculations to design and implement the corresponding algorithms.

At present, one of the main obstacles in using ontologies in computer-vision applications is the practical unfamiliarity of most computer vision engineers with ontology languages and – to a lesser degree – with declarative programming languages. In our opinion, stimulation of knowledge exchange between the fields of ontology engineering and computer vision engineering allows a major step forward in the application of full knowledge-intensive computer vision tasks in practice.

It is interesting to see which white-box benefits can be obtained when only part of the knowledge engineering work is integrated in the software design. Suppose that the domain expert is only asked to give an informal description of the computer vision task but no formal encoding of descriptive and procedural knowledge is made. In that case, the consecutive steps taken in the application would be provided as informal descriptions stated in domain expert's terms. The benefit of such a setup over a complete black-box application is that the consecutive steps can be explained to the domain experts. As a result, it will (slightly) contribute to expert acceptance.

Suppose that only descriptive knowledge is formally encoded and used in the application, but the 'how to'-knowledge is implemented in a black-box fashion. This would be more transparent than a mere black-box method, since it allows *e.g.* an easy visualisation of consecutive intermediate results to the domain experts. Such set of visualisations in recognisable subsequent models would allow for corrigibility, since the specific black-box component containing the error can be identified. Moreover, with the descriptive knowledge formally expressed, all objects in the scope of the application are defined and hence the application has robustness as property (see Section 6.3.1).

7.4 Conclusion

The main research question of this thesis is "*how can we develop a method for designing image analysis applications to automate knowledge-intensive tasks?*". This question has been divided into a number of sub-questions. In the next sections, we give the answers to these sub-questions.

1. What are the characteristics of knowledge-intensive tasks? In which cases is automation opportune? What is special about automating knowledge-intensive tasks?

The first step in our research was to define the notion of *knowledge-intensive task* in more detail. knowledge-intensive tasks are tasks that need a high level of specialist knowledge for a correct performance of the task. When such a task is performed by a human, the task knowledge can be partially explicit and partially tacit.

Several motives for automating knowledge-intensive tasks are known. First, task experts may not always be available. It takes time and money to train employees to become experts, and due to job transfers and retirements experts may leave the company. Second, there may not be enough experts to perform the task in a satisfactory manner at the desired throughput. Third, experts are humans; they may show less objectivity and consistency in their functioning than desired. Fourth, experts may not be fast enough to keep up with increasing processing speed. To overcome such difficulties, the automation of a knowledge-intensive task is opportune.

Domain experts are a valuable source of information for the process of automating knowledge-intensive tasks. They have a lot of experience in performing their task, however a large part of their knowledge is tacit knowledge. To have a machine perform the task in the same way as a domain expert, the implicit knowledge of the expert should be made explicit. The explicit knowledge should be encoded in such a way that it is usable for a machine. In other words, the machine has to be able to 'reason' with the explicit knowledge.

The answers to these questions gave us a demarcation of our research topic. They set the scope and defined our task.

2. What are possible approaches for automating knowledge-intensive tasks? What are the benefits of the chosen approach?

For the automation of knowledge-intensive tasks, we have used an ontology-based white-box approach. This approach has some inherent properties that highly improve the usability of the designed computer vision application. These properties are transparency, expert acceptance, adaptability, corrigibility, robustness and reliability.

Transparency influences expert acceptance in a positive way. Due to adaptability, the user is allowed a certain *degree of freedom* in applying the system to slightly different tasks and domains. We have shown how the tomato inspection system can be adapted for the inspection of tomato seedlings for grafting suitability (changed task) and for inspecting brassica seedlings (changed domain).

Corrigibility allows the user to get a *clear insight* in the decisions taken by the system and in pinpointing flaws that the system makes. We have discussed how a corrigibility module can help to identify errors and to find sanity check rules for the involved domains.

Robustness and reliability deal with the property of the system that all objects within the specified scope are properly assessed and objects that are outside the scope can be identified as such. We have argued that the level of reliability required by the user is dependent on the environment in which the application is used and its desired performance. Robustness and reliability *ensure appropriate behaviour* of the computer vision system, even when something unusual occurs.

3. How can we obtain the knowledge that is relevant for successfully automating the task?

Knowledge acquisition is a process that takes place in three dimensions: the application dimension, the domain dimension, and the discipline dimension (see Section 3.3.3). For each discipline, the problem owner and domain expert ensure that the knowledge acquisition is relevant for the task.

Descriptive knowledge is acquired by applying a combination of interview-observation-based methods and the ROC-component. The ROC-component supports the domain expert in actively participating in the expression of his knowledge. It uses associations obtained from existing knowledge sources to prompt the domain expert in creating a semi-formal knowledge model.

The resulting model of the ROC-method can be used as input for the interviewing process and vice versa. Either or both methods can be used to obtain an optimal definition of the descriptive knowledge required for the computer vision application.

Inferential knowledge can be elicited as well. Such knowledge indicates how the quality inspection task is performed. The end result of the knowledge modelling process is a set of application ontologies, each corresponding to a discipline, and a set of knowledge rules used to infer new facts in the application.

The knowledge engineering effort in the case study has lead to a point ontology, a geometric ontology, and a plant ontology consisting of concepts and relations. Moreover, sanity check rules for these domains have been expressed. 'How to'-knowledge has been identified as well to allow descriptions of the recorded seedling in one vocabulary to be expressed in another vocabulary.

4. How can procedural knowledge be encoded in a semantically meaningful way? How can we systematically embed the definition of procedural knowledge in the proposed framework for knowledge-intensive computer vision applications?

Expert knowledge concerning a task not only consists of descriptive knowledge, but also of procedural knowledge. We argue that a white-box approach in which the procedural knowledge is expressed explicitly is in principle preferred over systems in which the applied expertise is hidden in the system code. After all, internal transparency makes it easier to adapt the system to new conditions and to diagnose faulty behaviour. At the same time, explicitness comes at a price and is always bounded by practical considerations. Therefore we have introduced in this thesis a method to make a balanced decision between transparency and opaqueness.

To decide on transparency, we look further than the obvious objectives of an application such as accurateness, reliability, robustness, and speed, and focus on underlying design objectives and criteria, such as *trust*, *system modification* and *understandability*. Depending on the design objectives for a specific application, a choice for (i) a further task and domain decomposition, (ii) a refinement of the descriptive knowledge, (iii) the use of logical inferences, or (iv) the introduction of a black-box component created by a trusted third party is made.

The set of decision criteria concern the availability of explicit domain expertise, the application range of a component, the level of common sense and triviality of a subtask, the need for explanation of a subtask, and the availability of trusted third party software. These criteria offer structure to the application designer in making decisions concerning transparency. The proposed method ensures a careful weighing of costs and benefits in the implementation process.

5. How well does the proposed method perform in terms of a number of predefined criteria?

We have shown that the proposed design method leads to a white-box computer vision application that can properly perform the task for which it has been developed; in other words, correctness can be attained. Moreover, we have shown that the proposed white-box applications are inherently equipped with the possibilities to support corrigibility, adaptability, robustness and reliability. The remaining two properties – speed and expert acceptance – have not been tested, since the computer vision application for the case study developed using the proposed method is in prototype stage and cannot be evaluated for these criteria yet.

6. How applicable is the developed method to the automation of other knowledge-intensive computer vision tasks?

The presented paradigm to systematically design ontology-based computer vision applications can be used for any knowledge-intensive object inspection task for which shape features determine the quality of the object. For each task, a task description and a definition of the domain can be made. The division of the computer vision application into image acquisition, image segmentation, and image classification is generic. The use of explicit procedural knowledge to propagate knowledge about the recorded object through the application can also be used for arbitrary knowledge-intensive computer vision tasks.

The design method can be applied to inspection tasks in any domain. Raw camera data, in the form of *e.g.* point clouds, triangulated laser range data, or voxels, can be mapped to a carefully defined set of structures. Such structures can be geometrical, or areas with similar texture or colour, edges, clusters *et cetera*. The identified structures then transfer into a model of the object under inspection, *e.g.* a plant, a person, or a car, but again only into objects that can possibly exist in this specific task and domain context. The segmented object in turn is mapped to an assessment class, such as quality, price level, style, again specifically selected for this task.

The main research question: Is the ontology-based white-box design method presented in this thesis suitable for designing knowledge-intensive computer vision systems for biological objects?

With the subquestions answered, we have gathered sufficient knowledge to answer the main question of our thesis. We have shown in this thesis that it is indeed feasible to design a white-box computer vision application that successfully performs knowledge-intensive visual inspection tasks. We have presented a design method, involving the fields of knowledge engineering and software engineering. Due to the white-box approach, the scope of the application, the division into subtasks, and the explicit descriptive and inferential knowledge play an important role. The domain expert is considered as the key person in the design process; his knowledge is leading for the computer vision application.

Not only have we shown how we can design knowledge-intensive computer vision applications, we have also shown that applications developed according to the proposed paradigm have the highly desirable properties adaptability, consistency, corrigibility, expert acceptance, maintainability, reliability, robustness, and transparency. We have proposed a way to create tools that support the users and developers of the application in benefiting from these properties.

In our opinion, these white-box properties are extremely valuable for the users of the application and give the design of white-box computer vision applications a definite edge over black-box applications.

Bibliography

[1] A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system appraoches. *Artificial Intelligence Communications*, 7:39–52, 1994.

[2] J. Albusac, J. J. Castro-Schez, L. M. Lopez-Lopez, D. Vallejo, and L. Jimenez-Linares. A supervised learning approach to automate the acquisition of knowledge in surveillance systems. *Signal Processing*, In Press, 2009.

[3] J. Annett, K.D. Duncan, R.B. Stammers, and M.J. Gray. Task analysis. Technical Report Her Majesty's Stationary Office (HMSO), Department of Employment Training Information Paper No. 6., 1971.

[4] G. Antoniou and F. van Harmelen. *A Semantic Web Primer*. MIT Press, 2004.

[5] L. Argote, B. McEvily, and R. Reagans. Managing knowledge in organizations: An integrative framework and review of emerging themes. *Manage. Sci.*, 49(4):571–582, 2003. 970431.

[6] A. Asgharzadeh. Image analysis and enhancement using fuzzy rule based expert system. In *ACM Symposium on Applied Computing*, pages 529–531, 1996.

[7] T. Berners-Lee. Realising the full potential of the web, 1997.

[8] S.M. Bhandarkar, A.S. Chowdhury, Y. Tang, J.C. Yu, and E.W. Tollner. Computer vision guided virtual craniofacial reconstruction. *Computerized Medical Imaging and Graphics*, 31(6):418–427, 2007.

[9] G.-W. Bock and Y.-G. Kim. Breaking myths of rewards: a study of attitudes about knowledge sharing. In *Informs & Korms, Seoul, Korea*, pages 1042 – 1049, 2000.

[10] Paul Boissard, Vincent Martin, and Sabine Moisan. A cognitive vision approach to early pest detection in greenhouse crops. *Computers and Electronics in Agriculture*, 62(2):81–93, 2008.

[11] A. Bosch, X. Muñoz, and J. Freixenet. Segmentation and description of natural outdoor scenes. *Image and Vision Computing*, 25(5):727–740, 2007.

[12] M.R. Chandrarartne, D. Kulasiri, and S. Samarasinghe. Classification of lamb carcass using machine vision: comparison of statistical and neural network analyses. *Journal of Food Engineering*, 82:26–34, 2007.

[13] S.L. Chang, L.S. Chen, Y.C. Chung, and S.W. Chen. Automatic license plate recognition. *IEEE Transactions on Intelligent Transportation Systems*, 5(11):42–53, 2004.

[14] Y. Chtioui, D. Bertrand, and D. Barba. Feature selection by a genetic algorithm, application to seed discrimination by artificial vision. *Journal of the Science of Food and Agriculture*, 76(1):77–86, 1998.

[15] D.A. Cirovic. Feed-forward artificial neural networks: applications to spectroscopy. *Trends in Analytical Chemistry*, 16(3):148–155, 1997.

[16] R. Clouard, A. Elmoataz, Ch. Porquet, and M. Revenu. Borg: A knowledge-based system for automatic generation of imge processing programs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(2):128–144, 1999.

[17] D. Crevier and R. Lepage. Knowledge-based image understanding systems: A survey. *Computer Vision and Image Understanding*, 67(2):161 – 185, 1997.

[18] James L. Crowley, Daniela Hall, and Rémi Emonet. Autonomic computer vision systems. In *2007 International Conference on Computer Vision Systems, ICVS'07*. Springer Verlag, 2007.

[19] M. d'Aquin, M. Sabou, and E. Motta. Reusing knowledge from the semantic web with the watson plugin. In *Demo at the International Semantic Web Conference, ISWC 2008*, 2008.

[20] E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, *in press*, 2008.

[21] J.S. Dhaliwal and I. Benbasat. A framework for the comparative evaluation of knowledge acquisition tools and techniques. *Knowledge Acquisition*, 2:145–166, 1990.

[22] R. Diaz, L. Gil, C. Serrano, M. Blasco, E. Moltó, and J. Blasco. Comparison of three algorithms in the classification of table olives by means of computer vision. *Journal of Food Engineering*, 61:101–107, 2004.

[23] A. Dipanda, S. Woo, F. Marzani, and J.M. Bibault. 3-d shape reconstruction in an active stereo vision system using genetic algorithms. *Pattern Recognition*, 36:2143–2159, 2003.

[24] Ch-J. Du and D-W. Sun. Learning techniques used in computer vision for food quality evaluation: a review. *Journal of Food Engineering*, 72:39–55, 2006.

[25] J.A. Edosomwan. Ten design rules for knowledge based expert systems. *Industrial Engineering*, August:78–80, 1987.

[26] M. Egmont-Petersen, D. de Ridder, and H. Handels. Image processing with neural networks – a review. *Pattern Recognition*, 35:2279–2301, 2002.

[27] M. Egmont-Petersen and E. Pelikan. Detection of bone tumours in radiographic images using neural networks. *Pattern Analysis & Applications*, 2:172–183, 1999.

[28] K-P. Fähnrich, G. Groh, and M. Thines. Knowledge-based systems in computer-assisted production – a review. *Knowledge-Based Systems*, 2(4):249–256, 1989.

[29] D. Fensel and Ch. Bussler. The web service modeling framework wsmf. *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.

[30] C. Fernández, P. Baiget, X. Roca, and J. Gonzàlez. Interpretation of complex situations in a semantic-based surveillance framework. *Signal Processing: Image Communication*, 23(7):554–569, 2008.

[31] M. Fernández-López, A. Gómez-Pérez, and N. Juristo. Methontology: From ontological art towards ontological engineering. In *AAAI Ontological Engineering: Papers from the 1997 Spring Symposium*, 1997.

[32] S.Y. Foo. A rule-based machine vision system for fire detection in aircraft dry bays and engine compartments. *Knowledge-Based Systems*, 9:531–540, 1996.

[33] E. Friedman-Hill. *Jess in Action: Java Rule-Based Systems*. Hanning, 2003.

[34] A. Funk, V. Tablan, K. Bontcheva, H. Cunningham, B. Davis, and S. Hadschuh. Clone: Controlled language for ontology editing. In *IWSC/ASWC 2007*, LNCS 4825, pages 142–155, 2007.

[35] M.R. Genesereth and N.J. Nilsson. *Logical Foundations of Artifical Intelligence*. Morgan Kaufmann Publishers, Inc., Palo Alto, USA, 1987.

[36] E.H. Gombrich. *Een kleine geschiedenis van de wereld*. Uitgeverij Bert Bakker, 2006.

[37] A. Gómez-Pérez, M. Fernández-López, and O. Corcho. *Ontological Engineering*. Advanced Information and Knowledge Processing. Springer-Verlag New York, Inc., 2003.

[38] G. Granlund. Cognitive vision – background and research issues. In *Summer School on Cognitive Vision*, 2003.

[39] Th.R. Gruber. Ontology. In *Ecyclopedia of Database Systems*. Springer-Verlag, 2008.

[40] D. Guyer and X. Yang. Use of genetic artificial neural networks and spectral imaging for defect detection on cherries. *Computers and Electronics in Agriculture*, 29:179–194, 2000.

[41] H. Handels, Th. Roß, J. Kreusch, H.H. Wolff, and S.J. Pöppl. Feature selection for optimized skin tumor recognition using genetic algorithms. *Artificial Intelligence in Medicine*, 16:283–297, 1999.

[42] G. Hart, M. Johnson, and C. Dolbear. Rabbit: Developing a control natural language for authoring ontologies. In *5th European Semantic Web Conference*, pages 348–360, 2008.

[43] A. Haverkort, J.L. Top, and F. Verdenius. Organizing data in arable farming: towards an ontology of processing potato. *Potato Research*, 49:177–201, 2006.

[44] S. Hélie, S. Chartier, and R. Proulx. Are unsupervised neural networks ignorant? sizing the effect of environmental distributions on unsupervised learning. *Cognitive Systems Research*, 7:357–371, 2006.

[45] Joachim Hertzberg and Alessandro Saffiotti. Using semantic knowledge in robotics. *Robotics and Autonomous Systems*, 56(11):875–877, 2008.

[46] M. Hintz-madsen, L.K. Hansen, J. Larsen, E. Olesen, and K.T. Drzewiecki. Design and evaluation of neural classifiers – application to skin lesion classification. In *IEEE Workshop on Neural Networks for Signal Processing V*, pages 484–493, 1995.

[47] S-Y. Ho and H-L. Huang. Facial modeling from an uncalibrated face image using a coarse-to-fine genetic algorithm. *Pattern Recognition*, 34:1015–1031, 2001.

[48] J.H. Holland. *Adaption in Natural and Artificial Systems*. MIT Press, 1975.

[49] W. Hu and Y. Qu. Discovering simple mappings between relational database schemas and ontologies. In *International Semantic Web Conference*, pages 225–238, 2007.

[50] C. Hudelot and M. Thonnat. A cognitive vision platform for automatic recognition of natural complex objects. In *15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'03)*, pages 398–405, 2003.

[51] R. Jain. *The art of computer systems performance analysis – techniques for experimental design, measurement, simulation, and modeling*. John Wiley, 1991.

[52] J. Jarmulak, E.J.H. Kerckhoffs, and P-P. Van 't Veen. Case-based reasoning for interpretation of data from non-destructive testing. *Engineering Applications of Artificial intelligence*, 14:401–417, 2001.

[53] T. Jeliński, Ch. Du, D. Sun, and J. Fornal. Inspection of the distribution and amount of ingredients in pasteurized cheese by computer vision. *Journal of Food Engineering*, 83(1):3–9, 2007.

[54] K. Kaljurand and N.E. Fuchs. Bidirectional mapping between owl dl and attemp to controlled english. In *4th Int. Workshop on Principles and Practice of Semantic Web Reasoning*, LNCS 4187, pages 179–189. Springer, 2006.

[55] D. Khemani, M.M.J. Joseph, and S. Variganti. Case based interpretation of soil chromatograms. In *9th European Conference on Case Based Reasoning*, LNCS 5239, pages 587–599, 2008.

[56] H. Knublauch, R.W. Fergerson, N.F. Noy, and M.A. Musen. The protégé owl plugin: An open development environment for semantic web applications. In *Third International Semantic Web Conference*, pages 229–243, 2004.

[57] N.J.J.P. Koenderink, M. van Assem, J.L. Hulzebos, J. Broekstra, and J.L. Top. Roc: a method for proto-ontology construction by domain experts. In *Asian Semantic Web Conference*, Bangkok, Thailand, 2008.

[58] N.J.J.P. Koenderink, J.L. Hulzebos, H. Rijgersberg, and J.L. Top. Food informatics: Sharing food knowledge for research & development. In *EFITA/WCCA 2005*, pages 1161–1168, 2005.

[59] N.J.J.P. Koenderink, J.L. Top, and L.J. van Vliet. Expert-based ontology construction: a case study in horticulture. In *TAKMA workshop at the DEXA conference*, Copenhagen, 2005.

[60] N.J.J.P. Koenderink, J.L. Top, and L.J. van Vliet. Supporting knowledge-intensive inspection tasks with application ontologies. *International Journal of Human-Computer Studies*, 64:974–983, 2006.

[61] N.J.J.P. Koenderink, M. Wigham, F. Golbach, G. Otten, R. Gerlich, and H.J. van de Zedde. Marvin: High speed 3d-imaging for seedling classification. In *Joint International Agricultural Conference, JIAC 2009*, Wageningen, the Netherlands, 2009.

[62] Danica Kragic, Mårten Björkman, Henrik I. Christensen, and Jan-Olof Eklundh. Vision for robotic object manipulation in domestic settings. *Robotics and Autonomous Systems*, 52(1):85–100, 2005.

[63] S. Krishnan and K. Bhatia. Soas for scientific applications: Experiences and challenges. *Future Generation Computer Systems, in press*, 2008.

[64] M. LaFrance. The quality of expertise: Implications of expert-novice differences for knowledge acquisition. *SIGART Newletter*, April 1989(No. 108):6–14, 1989.

[65] Ch.W. Lapp and M.W. Golay. Modular design and construction techniques for nuclear power plants. *Nuclear Engineering and Design*, 172:327–349, 1997.

[66] N. Lavrac and S. Dzeroski. *Inductive Logic Programming, Techniques and Applications*. Ellis Horwood, New York, 1994.

[67] Bastian Leibe, Alan Ettlin, and Bernt Schiele. Learning semantic object parts for object categorization. *Image and Vision Computing*, 26(1):15–26, 2008.

[68] Sh-H. Liao. Expert system methodologies and applications – a decade review from 1995 to 2004. *Expert Systems with Applications*, 28:93–103, 2005.

[69] H Lin, S.L. Ong, W.J. Ng, and E. Khan. Monitoring of bacterial morphology for controlling filamentous overgrowth in an ultracompact biofilm reactor. *Water Environment Research*, 76(5), 2004.

[70] H-D. Lin and W-T. Lin. Automated process adjustments of chip cutting operations using neural network and statistical approaches. *Expert Systems with Applications, in press*, 2008.

[71] R.P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, April:4–22, 1987.

[72] E.G. Llano, F.S. Mata, I.T. Bustamante, N.H. González, and R.G. Gazapo. Improvement deoxyribo nucleic acid spots classification in polyacrilamide gel images using photometric normalization algorithms. *Analytica Chimica Acta*, 595(1–2):145–151, 2007.

[73] N. Maillot, M. Thonnat, and A. Boucher. Towards ontology based cognitive vision. In *Third International Conference On Computer Vision Systems*, pages 44–53. Springer-Verlag, Berlin, 2003.

[74] Nicolas Maillot and Monique Thonnat. Ontology based complex object recognition. *Image and Vision Computing*, 26(1):102–113, 2008.

[75] J. Main, S. Dillon, and S.C.K. Shiu. A tutorial on case-based reasoning. In S.K. Pal, T.S. Dillon, and D.S. Yeung, editors, *Soft Computing in Case Based Reasoning*. Springer, London, 2000.

[76] L. Main. Decision support with decision-modeling software. *Library Software Review*, May-June:128–133, 1987.

[77] A.S. Martyn. Some problems in managing complex development projects. *Long Range Planning*, 8(2):13–26, 1975.

[78] T. Matsuyama. Expert systems for image processing, analysis, and recognition: Declarative knowledge representation for computer vision. *Advances in Electronics and Electron Physics*, 86:81–171, 1993.

[79] W. Matusik, Ch. Buehler, R. Raskar, S.J. Gortler, and L. McMillan. Image-based visual hulls. In *27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 369–374, 2000.

[80] D.L. McGuinness and F. Van Harmelen. Owl web ontology language overview - w3c recommendation 10 februari 2004, 2004.

[81] S. Muggleton. Inductive logic programming. *New Generation Computing*, 8:295–318, 1991.

[82] Jiro Nakayama, Hiroyuki Hoshiko, Mizuki Fukuda, Hidetoshi Tanaka, Naoshige Sakamoto, Shigemitsu Tanaka, Kazutoshi Ohue, Kenji Sakai, and Kenji Sonomoto. Molecular monitoring of bacterial community structure in long-aged nukadoko: Pickling bed of fermented rice bran dominated by slow-growing lactobacilli. *Journal of Bioscience and Bioengineering*, 104(6):481 – 489, 2007.

[83] Naktuinbouw. Verslag ringtoets bruikbare planten. Technical report, Project 133, mar 1999. NAK1999.

[84] S Narkilahti, K Rajala, H Pihlajamäki, R Suuronen, O Hovatta, and H Skottman. Monitoring and analysis of dynamic growth of human embryonic stem cells: comparison of automated instrumentation and conventional culturing methods. *Biomedical Engineering Online*, 6(11), 2007.

[85] I. Nonaka and R. Toyama. The knowledge-creating theory revisited: knowledge creation as a synthesizing process. *Knowledge Management Research & Practice*, 1:2–10, 2003.

[86] N.F. Noy and D.L. McGuinness. Ontology development 101: A guide to creating your first ontology, 2001.

[87] Andreas Nüchter and Joachim Hertzberg. Towards semantic maps for mobile robots. *Robotics and Autonomous Systems*, 56(11):915–926, 2008.

[88] P. Perner. An architecture for a cbr image segmentation system. *Engineering Applications of Artificial intelligence*, 12:749–759, 1999.

[89] P. Perner, A. Holt, and M. Richter. Image processing in case-based reasoning. *The Knowledge Engineering Review*, 20(3):311–314, 2006.

[90] Wolfgang Ponweiser, Markus Vincze, and Michael Zillich. A software framework to integrate vision and reasoning components for cognitive vision systems. *Robotics and Autonomous Systems*, 52(1):101–114, 2005.

[91] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing, 2nd edition*. Cambridge University Press, 1992.

[92] A. Riege. Actions to overcome knowledge transfer barriers in mncs. *Journal of Knowledge Management*, 11(1s):48–67, 2007.

[93] S.M. Roberts and A.J. Willetts. *Introduction to biocatalysis using enzymes and microorganisms*. Cambridge University Press, 1995.

[94] J.M. Schooling, M. Brown, and P.A.S. Reed. An example of the use of neural computing techniques in material science – the modelling of fatigue thresholds in ni-base superalloys. *Materials Science and Engineering A*, A260:222–239, 1999.

[95] G. Schreiber, H. Akkermans, A. Anjewierden, R. De Hoog, N. Shadbolt, W. Van de Velde, and B. Wielinga. *Knowledge Engineering and Management – The CommonKADS Methodology*. MIT Press, 2000.

[96] S.A. Scott, J.E. Clayton, and E.L. Gibson. *A practical Guide to Knowledge Acquisition*. Addison-Wesley Publishing Company, 1991.

[97] H.I. Shafeek, E.S. Gadelmawla, A.A. Abdel-Shafy, and I.M. Elewa. Automatic inspection of gas pipeline welding defects using an expert vision system. *NDT & E International*, 37:301–307, 2004.

[98] G. Shakhnarovish, T. Darrell, and D. Indyk. *Nearest-Neighbor Methods in Learning and Vision*. The MIT Press, 2005.

[99] Enno Siemsen, Aleda V. Roth, and Sridhar Balasubramanian. How motivation, opportunity, and ability drive knowledge sharing: The constraining-factor model. *Journal of Operations Management*, 26(3):426–445, 2008.

[100] M.W. van Someren, Y.F. Barnard, and J.A.C. Sandberg. *The Think Aloud Method: a Practical Guide to Modelling Cognitive Processes*. Academic Press, London, 1994.

[101] H. Stern, U. Kartoun, and A. Shmilovici. A prototype fuzzy system for surveillance picture understanding. In *IASTED International ConferenceVisualization, Imaging, and Image Processing (VIIP 2001)*, pages 624–629, Marbella, Spain, 2001.

[102] M. Thonnat. Knowledge-based techniques for image processing and for image understanding. *Journal de Physique IV*, 12:189–236, 2002.

[103] A. Tiwari and A.K.T. Sekhar. Workflow based framework for life science informatics. *Computational Biology and Chemistry*, 31:305–319, 2007.

[104] U.S. Department of Transportation. The national shipbuilding research program – product work breakdown structure. Technical Report NSRP-0164, 1982.

[105] O. M. F. De Troyer and C. J. Leune. Wsdm: a user centered design method for web sites. *Computer Networks and ISDN Systems*, 30(1–7):85–94, 1998.

[106] E. Turban and J.E. Aronson. *Decision support systems and intelligent systems*. Prentice International Hall, Hong Kong, 2001.

[107] D. Valentin, H. Ardi, A.J. O'Toole, and G.W. Cottrell. Connectionist models of face processing: A survey. *Pattern Recognition*, 27(9):1209–1230, 1994.

[108] M. van Galen and J. Verstegen. Innovatie in de agrarische sector: we kunnen er niet genoeg van krijgen! Technical report, Den Haag, 2008.

[109] H. van Vliet. *Software engineering – principles and practice*. John Wiley & Sons, 1993.

[110] David Vernon. Image and vision computing special issue on cognitive vision. *Image and Vision Computing*, 26(1):1–4, 2008.

[111] Markus Vincze, Michael Zillich, Wolfgang Ponweiser, Vaclav Hlavac, Jiri Matas, Stepan Obdrzalek, Hilary Buxton, Jonathan Howell, Kingsley Sage, Antonis Argyros, Christoph Eberst, and Gerald Umgeher. Integrated vision system for the semantic interpretation of activities where a person handles objects. *Computer Vision and Image Understanding*, 113(6):682–692, 2009.

[112] Karel Vredenburg, Ji-Ye Mao, Paul W. Smith, and Tom Carey. A survey of user-centered design practice. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 471–478, 2002.

[113] I. Watson and F. Marir. Case-based reasoning: a review. *The Knowledge Engineering Review*, 9(4):355–381, 1994.

[114] A. Weber. Flexible factory reveals future of gm. *Assembly Magazine*, (May), 2003.

[115] B.J. Wythoff. Backpropagation neural networks – a tutorial. *Chemometrics and Intelligent Laboratory Systems*, 18:115–155, 1993.

List of Figures

List of Tables

Summary

This thesis focusses on the modelling of knowledge-intensive computer vision tasks. Knowledge-intensive tasks are tasks that require a high level of expert knowledge to be performed successfully. Such tasks are generally performed by a task expert. Task experts have a lot of experience in performing their task and can be a valuable source of information for the automation of the task. We propose a framework for creating a white-box ontology-based computer vision application.

White-box methods have the property that the internal workings of the system are known and transparent. They can be understood in terms of the task domain. An application that is based on explicit expert knowledge has a number of inherent advantages, among which corrigibility, adaptability, robustness, and reliability. We propose a design method for developing white-box computer vision applications that consists of the following steps: (i) define the scope of the task and the purpose of the application, (ii) decompose the task into subtasks, (iii) define and refine application ontologies that contain the descriptive knowledge of the expert, (iv) identify computational components, (v) specify explicit procedural knowledge rules, and (vi) implement algorithms required by the procedural knowledge.

The scope is one of the cornerstones of the application, since it sets the boundaries of the task. The problem owner and the domain experts are together responsible for setting the scope and defining the purpose. Scope and purpose are important for the task decomposition and for the specification of the application ontologies. The scope and purpose help the domain engineer to keep focus in creating dedicated ontologies for the application.

The decomposition of the task into subtasks models the domain expert's "observe – interpret – assess" way of performing a visual inspection task. This decomposition leads to a generic framework of subtasks alternated with application ontologies. The list of consecutive subtasks – record object, find structures, identify object parts, determine parameters, determine quality – can be reused for any visual inspection task.

Application ontologies are task-specific ontologies containing the descriptive knowledge relevant for the task. We have described an interview-based knowledge acquisition method that is suited for modelling multi-domain, multi-expert task-specific ontologies. Using the knowledge of multiple experts leads to a rich application ontology; adding an outsider's perspective from domain

experts from other involved domains, leads to an expression of knowledge that may be too trivial for task experts to mention or may not be part of the usual perspective of the task experts.

Knowledge acquisition based on interviews and observations only has some disadvantages. It takes a lot of modelling time for domain expert and knowledge engineer, it is difficult for the knowledge engineer to give a structured and full overview of his knowledge, and a model is created from scratch, even though reusable sources may exist. We have therefore introduced a reuse-based ontology construction component that gives domain expert a more prominent and active role in the knowledge acquisition process. This component prompts the domain expert with terms from existing knowledge sources to help him create a full overview of his knowledge. We show that this method is an efficient way to obtain a semi-formal description of the domain knowledge.

With the decomposition of the knowledge-intensive task into subtasks interspersed with descriptive knowledge models completed, we focus on the subtasks. Each of these subtasks can be represented by a sequence of components that perform a clearly defined part of a task. To specify these components, we explicitly identify for each service in the computational workflow (i) the input concepts, (ii) the output concepts, and (iii) a human readable (high level) description of the service. This information is used as documentation for the procedural knowledge.

Besides transparency of descriptive knowledge, transparency of processing knowledge is a desirable feature of a knowledge-intensive computer vision system. We show that blindly embedding software components in a transparent way may have an adverse effect. In some cases, transparency is not useful or desired. To support the software developer to make a balanced decision on whether transparency is called for, we have proposed a set of decision criteria – availability of expertise, application range of a component, triviality, explanation, and availability of third-party expertise. These decision criteria are paired to means of adding transparency to an application. We have elaborated several examples from the horticultural case study to show which transparency decisions are made for which reasons.

Using the framework for designing knowledge-intensive computer vision applications, we have implemented a prototype system to automatically assess the quality of tomato seedlings. We have shown that the proposed design method indeed results in a white-box system that has adaptability, corrigibility, reliability and robustness as properties. We provide guidelines on how to implement tool support for the adaptability and corrigibility properties of the system, to better assist the end users of the application. Moreover, we show how organisational learning and building trust in the application are supported by the white-box setup of the computer vision application.

Samenvatting voor iedereen

De visuele inspectie van producten is een belangrijk onderdeel van veel industriële processen. Denk bijvoorbeeld aan de kwaliteitscontrole van grondstoffen en halffabricaten, of aan het classificeren en sorteren van producten op uiterlijke kenmerken. Soms vindt er een eenvoudige controle plaats op de aanwezigheid of afwezigheid van een onderdeel, maar in andere gevallen is er sprake van een complexere beoordeling. Een voorbeeld van een complexere taak is het beoordelen van *natuurlijke producten* op hun kwaliteit, zoals kiemplanten, snijbloemen, fruit, champignons, et cetera.

Natuurlijke objecten ontstaan vanuit een groeiproces en komen daardoor in allerlei soorten en maten voor. Twee bloemen van dezelfde soort die allebei als eerste klas bloemen zijn beoordeeld hoeven helemaal niet op elkaar te lijken. Dit is een duidelijk verschil met industriële producten, zoals schroeven en bouten. Alle schroeven van dezelfde soort zien er exact hetzelfde uit. Het is voor industriële producten dan ook gemakkelijker om afwijkende producten te vinden dan voor natuurlijke producten.

De beoordeling van natuurlijke producten wordt veelal gedaan op basis van onderliggende kwaliteitskenmerken. De inspectietaak wordt uitgevoerd door goed getrainde experts, die door hun ervaring precies weten welke producten eerste keus of tweede keus producten zijn en welke producten moeten worden afgekeurd. In dit proefschrift beargumenteren we dat het voor het automatiseren van dergelijke kennisintensieve inspectietaken noodzakelijk is dat de kennis en ervaring van de experts wordt gebruikt als basis van het beeldverwerkingsalgoritme.

Het gebruiken van de expertkennis op zo'n manier dat de kennis terug te vinden is in het systeem – dit noemen we een *white-box systeem* – resulteert niet alleen in een correcte beoordeling van de natuurlijke objecten, maar het heeft nog een aantal andere voordelen. Deze voordelen hebben te maken met de *transparantie* van de applicatie. Hierbij denken we aan

- *Uitlegbaarheid:* door de expertkennis expliciet te maken, wordt het mogelijk om de kennis ook te gebruiken om communicatie tussen verschillende experts te ondersteunen en om nieuwe mensen op te leiden tot expert.
- *Vertrouwen in de applicatie:* doordat het beeldverwerkingssysteem gebaseerd is op expliciete kennis, kunnen de beslissingen van het systeem wor-

den verklaard in termen die voor experts te begrijpen zijn. Dit bevordert het vertrouwen van de experts in het systeem.

- *Correcties en aanpassingen:* door het gebruik van expliciete expertkennis, kunnen fouten worden opgespoord en opgelost. Het systeem kan ook worden aangepast om andere maar vergelijkbare producten te beoordelen, door precies dat stuk van de software te vervangen dat anders is ten opzichte van de oorspronkelijke taak.
- *Robuustheid en betrouwbaarheid:* doordat het systeem gebaseerd is op de volledige beschrijving van de taak en de mogelijke verschijningsvormen van de producten, zal het goed functioneren voor alle producten die binnen het taakdomein vallen.

In de rest van deze samenvatting bespreken we onze voorbeeldapplicatie en de drie belangrijkste onderwerpen uit dit proefschrift. Eerst richten we ons op het modelleren van expertkennis. Daarna beschrijven we hoe het opdelen van een taak in deeltaken bijdraagt aan de transparantie van de applicatie. Tot slot kijken we in meer detail naar transparantie en beschrijven we wanneer meer transparantie wel of niet gewenst is.

De voorbeeldapplicatie: tomatenkiemplanten

In het proefschrift gebruiken we een case study om de ontwikkelde kennisintensieve beeldverwerkingsmethode te illustreren: het classificeren van tomatenkiemplanten. In de Nederlandse tuinbouw is kiemplantinspectie een belangrijk proces. Voor zaadveredelaars en voor plantenkwekers is het belangrijk dat in een vroeg stadium kan worden voorspeld hoeveel groente of fruit een volwassen plant zal opleveren. Het doel van het inspectieproces is om de kans op een hoge opbrengst van het volwassen gewas te maximaliseren.

Kwaliteitscontrole van kiemplanten is een complexe taak, aangezien ze veel verschillende verschijningsvormen hebben (zie figuur 1.4). Om het rendement van een volwassen plant te voorspellen wordt door experts een set van proefondervindelijk gevalideerde kwaliteitscriteria gebruikt. Enkele eenvoudige criteria zijn bijvoorbeeld bladoppervlak, steellengte, en bladkromming. Voorbeelden van complexere criteria zijn de kans dat een plant een kroeskop is, of de onregelmatigheid van de bladvorm. Op dit moment wordt de kwaliteitsbepaling uitgevoerd door goed getrainde experts. Hoewel elke deskundige intensief wordt getraind door het bedrijf waar hij werkzaam is, verschilt de beoordeling van de experts in hetzelfde bedrijf doorgaans tot wel tien procent. De variatie tussen experts van verschillende bedrijven is nog hoger. De verschillende beoordeling wordt deels veroorzaakt doordat mensen subjectief zijn: als een expert vermoeid raakt, gaat hij fouten maken. Deels speelt ook mee dat de opleiding van de experts is gebaseerd op acht officieel omschreven classificatieregels van Naktuinbouw, terwijl uit interviews met experts bleek dat in de praktijk meer dan 60 verschillende classificatieregels worden gebruikt. Bovendien zitten er verschillen in de kwaliteitsbeoordeling tussen bedrijven, omdat niet alle

planten voor dezelfde markt bedoeld zijn. De Noord-Europese markt en de Zuid-Europese markt stellen andere kwaliteitseisen aan kiemplanten.

Om een succesvolle beeldverwerkingsapplicatie te ontwikkelen is het enerzijds nodig om driedimensionale opnames van de kiemplanten te maken, zodat alle vormvariaties kunnen worden opgemerkt. Anderzijds moet er per bedrijf een bedrijfseigen set van beoordelingsregels worden opgesteld die door de applicatie wordt gebruikt.

Het modelleren van expertkennis

We hebben al opgemerkt dat experts veel kennis en kunde hebben om een taak succesvol uit te voeren. Deze kennis is in het algemeen niet of niet volledig opgeschreven, maar bevindt zich in het hoofd van de experts; de kennis is *impliciet* aanwezig. Als we een white-box beeldverwerkingsapplicatie willen maken, is het echter belangrijk dat deze kennis *expliciet* wordt gemaakt. Pas als de kennis zo is opgeschreven dat hij interpreteerbaar is voor een computer, dan kan hij gebruikt worden voor de applicatie.

In het vakgebied *knowledge engineering* wordt veel aandacht geschonken aan het achterhalen en expliciet maken van expertkennis. Een veelgebruikte methode daarvoor is interviewen en observeren. Voor de kiemplantenapplicatie hebben we gebruik gemaakt van een interviewgebaseerde aanpak om de kennis van de experts boven water te krijgen (zie figuur 3.1). Nadat experts geïnterviewd zijn, is de expertkennis opgeschreven in de vorm van een gespreksverslag. Dit gespreksverslag is nog niet leesbaar voor een computer. De knowledge engineer moet vanuit het gespreksverslag een formeel kennismodel – een zogenaamde *ontologie* – maken.

Ontologieën zijn formele beschrijvingen van een kennisdomein. Ze bestaan uit *concepten*, die de belangrijke begrippen in het domein aangeven. Voor het kiemplantdomein kun je bijvoorbeeld denken aan ‘plant’, ‘steel’, ‘zaadlob’ of ‘echt blad’. De concepten kunnen *attributen* bevatten, die eigenschappen aanduiden, zoals ‘steellengte’ of ‘bladoppervlak’. De concepten kunnen ook met *relaties* met elkaar verbonden zijn. In een ontologie wordt gebruik gemaakt van eenvoudige zinnen, grofweg in de vorm van ‘onderwerp – werkwoord – lijdend voorwerp’. Voorbeelden zijn zinnen als ‘plant – heeft – steel’ of ‘echt blad – is onderdeel van – kop’ (zie figuur 5.3). Doordat de zinnen zo eenvoudig zijn en altijd dezelfde vorm hebben, is het mogelijk dat een computer ze interpreteert. Op deze manier kan de gemodelleerde expertkennis gebruikt worden door de beeldverwerkingsapplicatie.

Het interviewen van experts om hun kennis te verkrijgen heeft een aantal voordelen. De experts zijn betrokken bij de ontwikkeling van het kennismodel, ze vullen elkaar aan en komen samen tot een model waar alle experts achter staan. De ontologie die zo ontstaat, is precies afgestemd op de inspectietaak. Er zijn echter ook wat nadelen. De knowledge engineer moet geschoold worden in het domein van de experts om een zinvolle ontologie te kunnen maken. Voor de expert is het formuleren van zijn kennis een nieuw proces, waardoor het

risico bestaat dat gedeeltes van het domein vergeten worden. Tot slot wordt met de interviewaanpak van voor af aan begonnen met het bouwen van een kennismodel. Als er thesauri, ontologieën of andere gestructureerde bronnen bestaan die deels kunnen worden hergebruikt, wordt daar geen gebruik van gemaakt.

In dit proefschrift hebben we daarom niet alleen interviewgebaseerde ontologieën besproken, we hebben ook een methode geïntroduceerd waarmee de expert zelfstandig alle kennis voor een ontologie kan verzamelen. Dit is de ROC-methode, waarbij ROC staat voor *reuse-based ontology construction*. De manier van ontologieën bouwen die door ROC wordt voorgesteld maakt gebruik van bestaande bronnen. De expert noemt om te beginnen een aantal belangrijke begrippen in zijn domein. De ROC-software zoekt vervolgens in bestaande ontologieën en thesauri of deze begrippen daarin voorkomen. Als dat zo is, dan worden alle concepten die met een relatie gekoppeld zijn aan de beginbegrippen opgehaald en ter beoordeling voorgelegd aan de expert. De expert kan door de nuttige concepten goed te keuren en de applicatie opnieuw te laten zoeken een steeds betere verzameling van relevante begrippen verkrijgen. Hiermee wordt het proces om ontologieën te bouwen versneld.

In hoofdstuk 3 hebben we laten zien hoe interviewgebaseerde en ROC-gebaseerde kennismodellering werkt. We hebben ook laten zien dat de methodes met elkaar gecombineerd kunnen worden om op efficiënte wijze een ontologie te ontwikkelen.

Het opdelen van een taak in deeltaken

Nu we een methode hebben om expertkennis te formaliseren, kunnen we ons richten op het maken van een transparante beeldverwerkingsapplicatie. Dit gedeelte staat beschreven in hoofdstuk 2 van het proefschrift.

Elke beeldverwerkingsapplicatie bestaat uit drie achtereenvolgende processen: *kijk – interpreteer – beslis*, of meer formeel *beeldacquisitie – segmentatie en analyse – classificatie*. Het doel van de beeldacquisitie is om een opname van het te beoordelen object te maken, waardoor er een interne representatie van het object bestaat. Zo'n interne representatie bestaat uit een stel punten die wat de computer betreft een willekeurig object zouden kunnen beschrijven. De interpretatiestap is bedoeld om het opgenomen object te verdelen in samenhangende gebieden, die gekoppeld worden aan objectonderdelen. Voor het kiemplantvoordeel houdt dit in dat in de opgenomen puntenwolk lijnachtige en vlakachtige gebieden worden gezocht. Voor deze gebieden worden geometrische vormen gezocht, zoals cilinders met een grote of kleine straal, en gekromde vlakken. Later zal dan blijken dat de punten die horen bij de cilinder met een grote straal precies die punten zijn die bij de plug van het kiemplantje horen. De punten in de dunne cilinder vormen de steel, en elk van de gekromde vlakken komt overeen met een blad. In de classificatiefase worden tot slot de attribuutwaarden van de plantonderdelen berekend, zodat bladoppervlak, steellengte et

cetera bekend zijn. Deze waarden kunnen gebruikt worden om de kwaliteitsregels van de experts aan te roepen en het plantje te classificeren.

De opdeling van de beeldverwerkingstaak in deze stappen kan gecombineerd worden met verschillende ontologieën met domeinkennis. Uit bovenstaande beschrijving blijkt dat we een puntenontologie, een geometrische ontologie, en een plantontologie nodig hebben (zie figuur 2.6). De plantontologie hebben we met de experts opgesteld. Hierdoor weten we welke plantonderdelen voorkomen bij kiemplantjes. Dit stelt ons in staat om de bijbehorende geometrische vormen op te stellen. Kubussen en piramides zijn niet nodig in het plantdomein, maar dikke en dunne cilinders en gekromde vlakken juist wel. Vanuit de geometrische vormen kunnen we ook beslissen wat we voor gegevens nodig hebben in de puntontologie. Door te bepalen welke punten in een lijnachtige omgeving liggen en welke in een vlakachtige, kunnen we een vroege groepering maken van de punten. Deze groepering helpt om de geometrische vormen te vinden.

De opdeling van de inspectietaak volgens het 'kijk – interpreteer – beslis'-principe leidt tot een generiek toepasbare ontwerpstructuur. De lijst van opeenvolgende deeltaken van (i) opnemen, (ii) structuren vinden, (iii) deelobjecten vinden, (iv) parameters bepalen en (v) kwaliteit bepalen kan worden gebruikt voor elke visuele inspectietaak. Door de tussenliggende ontologieën te specificeren worden opeenvolgende modellen van het object getoond, en wordt met name bijgedragen aan de transparantie-eigenschappen 'correcties' en – indirect aan – 'vertrouwen'. Als, immers, een plant verkeerd beoordeeld wordt, dan kan model voor model worden teruggekeken tot een correct tussenmodel wordt gevonden (zie figuur 6.1). We weten dan dat de beoordelingsfout te vinden is in de deeltaak die dit correcte model als input en een foutief model als output heeft. Als een beoordeling betwijfeld wordt door een expert, kan het systeem laten zien welke opeenvolgende modellen zijn gebruikt. Dit biedt de expert inzicht in de werking van de applicatie.

Transparantie

Zoals eerder gemeld, zijn correcties en vertrouwen niet de enige gewenste eigenschappen van de beeldverwerkingsapplicatie. In hoofdstuk 4 van dit proefschrift laten we zien dat een applicatie op drie manieren transparanter kan worden gemaakt (zie figuur 4.1). Ten eerste kunnen we de methode uit de vorige sectie gebruiken: het opdelen van taak en domein in deeltaken en bijbehorende tussenmodellen. Ten tweede is het mogelijk om inputfeiten van een taak te verfijnen. Ten derde kunnen we de gebruikte procedurele kennis – de kennis die beschrijft *hoe* een taak wordt uitgevoerd – in de vorm van expliciete kennisregels geven.

Deze laatste manier van kennis specificeren hadden we nog niet besproken. Tot nu toe zijn we uitgegaan van expertkennis die in de vorm van ontologieën wordt vastgelegd. Dat soort kennis is geschikt om 'statische' kennis te beschrijven. Expertregels zijn een andere vorm van expertkennis. Deze regels

hebben de vorm van een 'als ... dan'-uitspraak en beschrijven hoe een expert een bepaalde beslissing maakt. Voorbeelden in het kiemplantdomein zijn: 'als een plant een steel heeft die korter is dan 80% van de gemiddelde steellengte van de planten in de tray, dan is de plant tweede keus of afkeur'. Dit soort 'dynamische' kennis kan worden vastgelegd in de vorm van formele kennisregels. Ze kunnen door een applicatie worden geïnterpreteerd.

We hebben drie manieren gedefinieerd om transparantie toe te voegen aan een applicatie, maar hiermee zijn we er nog niet. De gewenste mate van transparantie is namelijk lastig te bepalen. Alles volledig transparant maken is overbodig en kan verwarrend werken. Stel bijvoorbeeld dat je een programma gebruikt om een ingewikkelde analyse uit te voeren. Het is dan nuttig om te weten dat er eerst een gemiddelde wordt genomen, waarna een andere berekening wordt toegepast. Het is niet zo nuttig om precies uitgespeld te zien hoe het nemen van een gemiddelde werkt, omdat het een alom bekende berekening betreft. Details over het gemiddelde berekenen voegen niet meer begrip toe over de ingewikkelde analyse. De aanwezigheid van triviale kennis kan een reden zijn om een niet-transparante component – een *black-box component* – toe te laten in een white-box applicatie.

In hoofdstuk 4 geven we naast het trivialiteitcriterium nog een aantal andere criteria die de softwareontwikkelaar moeten helpen bij het beslisproces op welke punten transparantie gewenst is. Deze criteria zijn: (i) beschikbaarheid van expertkennis, (ii) applicatiedomein, (iii) uitlegbaarheid en (iv) vertrouwde software van anderen. Als er geen expertkennis beschikbaar is, dan is er geen keus; een *black-box* component is dan de enige oplossing. Als het applicatiedomein van een deeltaak alleen van toepassing is op bijvoorbeeld tomatenkiemplanten, maar niet op paprikakiemplanten, dan is het nodig om bij de inputfeiten van de taak aan te geven voor welk type kiemplanten de taak geschikt is. Als de kennis die gebruikt wordt in de applicatie ook gebruikt wordt voor opleidingsdoeleinden, dan speelt uitlegbaarheid van deeltaken een belangrijke rol. Er kan dan gekozen worden om expliciet extra kennis in het systeem te specificeren, die misschien ook als black-box component beschikbaar was geweest, maar die nuttige achtergrondinformatie geeft over het proces. Tot slot kan de applicatieontwikkelaar kiezen voor het gebruik van software van anderen, als deze software volledig vertrouwd wordt.

Door de transparantiecriteria op elk niveau van de applicatie toe te passen (zie figuur 4.2), kan een uitgebalanceerde, transparante beeldverwerkingsapplicatie verkregen worden, waarmee alle genoemde voordelen – correctheid, corrigeerbaarheid en aanpasbaarheid, robuustheid en betrouwbaarheid, vertrouwen in de applicatie, en uitlegbaarheid – kunnen worden ondersteund.

In hoofdstuk 5 van dit proefschrift laten we de ontologieën en de software voor de voorbeeldapplicatie zien, die ontwikkeld zijn volgens de hierboven beschreven methoden. We laten zien dat de voorgestelde methode inderdaad succesvol is in het implementeren van een kwaliteitsinspectiesysteem voor tomatenkiemplanten. Hoofdstuk 6 evalueert de beschreven theorie met betrekking tot de eigenschappen corrigeerbaarheid, aanpasbaarheid, robuustheid en be-

trouwbaarheid. We laten niet alleen zien dat de voorbeeldapplicatie aan deze eigenschappen voldoet, maar we geven ook aanwijzingen over hoe je tools zou kunnen ontwikkelen om daadwerkelijk te profiteren van deze eigenschappen. Hoofdstuk 7, tot slot, bevat de discussie, waarin de belangrijkste paradigma's – het ontwikkelen van een white-box applicatie, het beschouwen van de expert als hoofdrolspeler en het combineren van technieken uit de software engineering en knowledge engineering vakgebieden – besproken worden.

Dankwoord

Vrij naar het gezegde “It takes a village to raise a child”, zou ik willen zeggen “It takes a village to complete a thesis”. In mijn eentje had ik dit proefschrift nooit kunnen schrijven en ik ben dan ook veel mensen dankbaar voor hun bijdrage aan mijn promotietraject.

Jan Top en Lucas van Vliet, mijn beide promotoren, hebben me steeds begeleid op mijn promotiezoektocht. Met Jan heb ik vele uurtjes, vaak aan het eind van de middag of vroeg op de avond, gediscussieerd over modelleerkeuzes en mogelijke oplossingsrichtingen voor problemen waar ik tegenaan liep. Zowel als promotor als als themaleider Informatie Management binnen A&F ben je wat mij betreft ‘top’: doordat je goed luistert en altijd de juiste vragen weet te stellen, hielp je me steeds weer een stap vooruit. Ik heb de discussiesessies altijd als zeer inspirerend ervaren en zie ernaar uit om ze in de toekomst voort te blijven zetten. Lucas was altijd bereid om mee te denken over de beeldverwerkingskant van mijn werk. Ik ben erg blij dat je me vol enthousiasme hebt gesteund om informatiemanagement en beeldverwerking met elkaar te combineren. Bedankt dat ik zoveel ik wilde in je groep kon werken. Hierdoor kon ik mijn aandacht op een Delftse dag volledig richten op mijn promotiewerk, zonder afgeleid te worden door Wageningse projecten.

Zonder Toine Timmermans zou ik nooit aan dit boekje zijn begonnen: hij vroeg me op een dag of ik niet een voorstel wilde schrijven om op basis daarvan te promoveren. Toine, bedankt voor het vertrouwen. Dankzij hem en Hans Maas kon ik zelfs een jaar full-time aan mijn promotieonderzoek besteden in Zürich, terwijl ik gewoon bij A&F in dienst bleef. Die tijd in Zürich was erg nuttig en is me nog steeds dierbaar. In the group of Luc van Gool at ETH, I could work alongside other computer vision experts. Luc, thank you for having me in your group! And even though the BIWI-group was more involved in tracking, gesture recognition and medical imaging, I could always talk with interested colleagues. Andreas Griesser, Matthieu Bray, were my roommates. I remember their surprise when I was enthusiastic about the first snow! Little did I know that the snow flakes would continue to fall throughout the season .... Bryn Lloyd, you were a student in my project and have helped to create the basis for the implementation of the case study. Peter Czech (Monthy Python) and Herbert Bay (gestampte muisjes) were always ready for some small talk. In Zürich, we were part of a group of international friends: Rob, Heather, Ulf, Robert und Sabine, Mario and Martha, Michelle and Ben. I enjoyed your company and

friendship, and will always remember the Sinterklaas-evening, the ExpoVina-visits, the sightseeing, the hikes in the Swiss mountains, the cheese fondues. I am glad that we are still in touch, even though most of us are living elsewhere nowadays.

Wageningen is gedurende mijn promotietijd de belangrijkste uitvalsbasis geweest. Toen ik begon met promoveren, was Anneke mijn kamergenote: gezelligheid en nu en dan tropisch fruit. Nadat Anneke A&F verliet, werden Martijntje en ik kamergenoten. Martijntje, we hadden al ervaring met kamerdelen in Eindhoven en Kopenhagen, en ik vond het erg prettig om die traditie een vervolg te geven in Wageningen. De discussies op het white-board, de verschillende smaken sinaasappelthee en het delen van wel en wee karakteriseren voor mij deze tijd. Ik hoop dat je in je eigen promotietraject net zo veel hebt aan je kamergenoot als ik. In de eindfase van mijn promotiewerk deelde ik de kamer met Mariëlle. Mariëlle, onze gesprekken bij de ochtendthee geven altijd een goed begin aan een nieuwe dag. Dank voor het meeleven met de afrondende promotiestappen. Jacco en Rick zijn achtereenvolgens als themaleider Vision betrokken geweest bij mijn promotie en hebben het werk steeds met grote belangstelling gevolgd. Met Franck heb ik vergadering na vergadering van de IOP-projecten bijgewoond. Gerwoud en André hebben me meerdere keren uit de brand geholpen bij het programmeren in Labview en C en het bedenken van slimme algoritmes. Later, toen het MARVIN-project gestart was, zijn behalve Gerwoud ook Rick, Remco, Franck en Mari inhoudelijk betrokken geraakt bij het project. Binnen Food Informatics is een gedeelte van het ROC-werk ondergebracht, waaraan ik met Lars, Hajo, Jeen, Remko en Jan heb gewerkt. Ook de andere collega's (en oud-collega's) bij LIVE hebben ervoor gezorgd dat ik de afgelopen jaren plezier in mijn werk heb gehad en gehouden. Behalve inhoudelijke contacten gingen we samen lunchen, in de kantine of in de stad, en werden er regelmatig film- en spelletjesavonden georganiseerd. Ik kan me geen betere collega's wensen! Qing Gu en Elwin Staal, jullie hebben als afstudeerder bijgedragen aan de uitwerking van de theorie respectievelijk de case study. Bedankt voor jullie inzet. Steven Groot heeft vanuit biologische hoek steeds meegekeken naar de bepaling van zaailingkwaliteit en heeft me met zeer waardevolle achtergrondinformatie en een altijd open blik geholpen om snel in de plantenwereld thuis te geraken. Van Pieter de Visser mocht ik de Scanstation lenen om mijn eerste (en ook vele latere) testopnames te maken. Robert, bedankt voor de hulp bij het meten en opknippen van testplantjes om een representatieve dataset te krijgen. Mari, bedankt voor de Engelse vertaling van de stellingen. Tot slot wil ik graag Rinus Seijnaeve bedanken voor de ondersteuning bij het printen van de verschillende versies van mijn proefschrift, Corry Snijder voor de talloze literatuurspeurtochten, en Mirjam van den Berg voor het uittypen van ellenlange gespreksverslagen.

Vanaf de start van het project heb ik veel te danken aan de bedrijven die me in staat hebben gesteld om de kennis-intensieve beeldverwerkingsmethode toe te passen op het kiemplant-inspectie probleem. Specifiek wil ik Kees van den Berg, Wim Hazeu, John Ammerlaan (Leo Ammerlaan); Marc Balemans, Jos Dukker (Beekenkamp); Corine de Groot, Arnaud Hulskes, Marion Bruin, Johan

de Grauw (Bejo); Edith van Dijk-Nas, Frank de Rooij, Shankara Naika (De Ruiter Seeds); Meindert Klooster, Merel Langens (Enza); Bram Voogd (Grow Group); Miriam Zandstra, Jan Hauwert, Henry Bruggink, Gerrit Lakeman (Incotec); Jan Innemee, Maaike van Kilsdonk, Gerard van Bentum (Nickerson-Zwaan); Marjo Smeets, Jan Bergs, Ruud Nabben (Nunhems); Marc Rijneveen, René Janssen (RijkZwaan); Emile Clerkx (Seminis); Edward Blank, Tonko Bruggink, Agnes Chardonnens, Barbara Westland, Maarten Homan, Paul van den Wijngaard (Syngenta); Arko van der Lugt, Jilles Koornneef, Ron van Etten (van der Lugt); Jan Tamerus (Vreugdenhil); en Alie Sahin, Erik van den Arend (WPK) noemen. Zonder de interviews met jullie, jullie feedback, en het blijvende enthousiasme zou hoofdstuk 5 er niet zijn. Clemens Stolk heeft vanuit Plantum het grootste gedeelte van mijn promotietijd meegemaakt. Hartelijk dank, Clemens, voor de goede contacten de afgelopen jaren. Thijs Simons heeft het stokje van Clemens overgenomen en ook hem wil ik bedanken voor zijn niet aflatende enthousiasme voor het project. HenkJan Reus, als leider van de IOP-begeleidingscommissie zorgde je ervoor dat we als promovendi vooruitgang bleven boeken en van elkaars werk konden leren.

Promotietijd bestaat gelukkig niet alleen uit werken, maar ook uit sociale activiteiten in de soms spaarzame vrije tijd. Ik prijs me gelukkig met een vriendengroep vol lieve vrienden in Utrecht en omgeving en in Brabant. De vele gezellige etentjes, operavoorstellingen en thee-leuterijen hebben we voldoende energie gegeven om door te blijven gaan. Michiel en Marijne, onze avondjes samen koken en eindloos natafelen zijn hoogtepunten geweest. Ingrid en Dorian, laten we de traditie om regelmatig samen op pad te gaan vasthouden. Guido en Simon, dank voor de wiskundige hulp op cruciale punten en dank ook voor de niet afnemende belangstelling. Gerard en Tineke, Gert-Jan en Hennie, Lonneke, Mariëlle, Hans, Paul, Marger, Simone, Ester, dank voor jullie vriendschap. Eindelijk kan ik jullie vragen bevestigend beantwoorden: ja, het is af! Michiel le Comte and Rob Pfab have both been so kind to read my manuscript to provide it with very useful feedback. Thanks a lot! Ook mijn (schoon)broertjes en (schoon)zusjes wil ik hier noemen: Michael en Daniëlla, Daniëlle and Jiorgos, Gijsje en Ramon, ook jullie zijn steeds vol interesse het promotiewerk blijven volgen. Daniëlle, ik vond het geweldig dat je via Skype alles over mijn onderzoek wilde weten. Michael, het lopen van de Strand6Daagse gaf me een mooie ontspannende week na het inleveren van mijn proefschrift. Heerlijk om samen uit te waaien en bij te kletsen! Jan en Ans, bedankt voor jullie interesse en uiteraard voor de creatieve fotosessie voor de kaft. Mam, pap, van kleins af aan hebben jullie me gestimuleerd om het beste uit me zelf te halen. Nog steeds vragen jullie vol interesse naar wat mij bezig houdt. Dankzij jullie aandacht, liefde en zorg ben ik geworden wie ik ben, zonder jullie aandacht, liefde en zorg zou ik hier niet hebben gestaan. Tot slot wil ik Femius bedanken. In Utrecht, Enschede, Wageningen, Zürich en weer Utrecht was je er altijd voor me. Je hielp me met de technische LaTeXdetails, je dacht mee over algoritmische vraagstukken. Maar belangrijker nog: in de stressvolle periodes van mijn promotie, zorgde je voor ontspanning. Je reed me bijvoorbeeld in onze bolide naar Middelburg of Deventer, terwijl ik als bijrijder op de laptop hoofdstuk na hoofd-

stuk bijschaafde. Zo zorgde je ervoor dat in die drukke weekenden de balans tussen werk en ontspanning goed bewaakt werd. Bedankt voor je voortdurende steun en liefde.

Curriculum Vitae

Nicole was born on November 24, 1975 in Helmond. She attended the Carolus Borromeus gymnasium in Helmond from 1989 to 1994. Next, she started a study mathematics at Utrecht University. This study was completed in 1999 with a thesis entitled "p-Adic Numbers and Hilbert Symbols". After her mathematics study, she continued her education as a Master of Technological Design student at the Eindhoven University of Technology with the post-master study Mathematics for Industry. This two-year study was finished in 2001 with a thesis "Automated Registration of License Plates".

In 2001, she started as a researcher in computer vision and in information management at the research institute Wageningen UR – Food & Biobased Research where she is still employed. At this institute, she started in 2003 in part-time with her Ph.D. project "A Knowledge-Intensive Approach to Computer Vision Systems". She had the opportunity to attend the ETH Zürich as a guest researcher from January 2004 to March 2005 as part of her Ph.D. work.

List of Publications

- Harmsen, S.R., Koenderink, N.J.J.P., 2009, Multi-target tracking for flower counting using adaptive motion models, Computers and Electronics in Agriculture, 65 (1), pp. 7-18.
- Koenderink, N.J.J.P., Wigham, M., Golbach, F., Otten, G., Gerlich, R., Zedde, H.J. van de, MARVIN: High speed 3D imaging for seedling classification, Seventh European Conference on Precision Agriculture, pp. 279 – 286.
- Koenderink, N.J.J.P., van Assem, M., Hulzebos, J.L., Broekstra, J., Top, J.L., 2009, ROC: a method for proto-ontology construction by domain experts, Third Asian Semantic Web Conference 2008.
- Tromp, R.H., Primo-Martín Ch., Van de Zedde, R., and Koenderink, N.J.J.P., 2008, Quantifying the Morphology of Bread Crusts, in 'Bubbles in Food

2: Novelty, Health and Luxury', AACC International, St. Paul, MN 55121 USA.

- Koenderink, N.J.J.P., Hulzebos, L., van Assem, M., Broekstra, J., van Brakel, R., Top, J.L., 2008, Experts aan het stuur - modelleren met ROC (Dutch), Agro Informatica 21 (4), pp. 19-21.
- Broekstra, J., Koenderink, N.J.J.P., Top, J.L., 2008, Ontologieën voor voeding, <!Element, 14 (2), pp. 14-16.
- Koenderink, N.J.J.P., Top, J.L., and van Vliet, L.J., 2006, Supporting Knowledge-Intensive Inspection Tasks with Application Ontologies, International Journal of Human-Computer Studies, 64 (10), pp. 974-983.
- Koenderink, N.J.J.P. and Hulzebos L.J., 2006, Dealing with Bottlenecks in Traceability Systems , Invited chapter for 'Improving Traceability in Food Processing and Distribution', Woodhead Publishing, London, pp. 88-106.
- Hulzebos, L.J. and Koenderink, N.J.J.P., 2006, Modelling Food Supply Chains for Tracking and Tracing, Invited chapter for 'Improving Traceability in Food Processing and Distribution', Woodhead Publishing, London, pp. 67-87.
- Koenderink, N.J.J.P., 2006, Ontologieën in de agribusiness: theorie en praktijk, Agro Informatica 19 (2), pp. 45-46.
- Koenderink, N.J.J.P., Hulzebos, L.J., Rijgersberg, H. and Top, J.L., 2005, FoodInformatics: Sharing Food Knowledge for Research and Development, Sixth Agricultural Ontology Service Workshop at the joint EFITA/WCCA conference.
- Koenderink, N.J.J.P., Top, J.L., and van Vliet, L.J., 2005, Expert-Based Ontology Construction: a Case-Study in Horticulture, Fifth TAKMA Workshop at the DEXA Conference, pp. 383-387.
- Verdenius, F. and Koenderink, N.J.J.P., 2004, Focus on the traceability in the organic chain and traceability oriented on the quality in horticulture, Invited paper, Proceedings of the International Conference on Food Traceability, Sao Paolo, Brazil, p. 144-146.
- Top, J.L., Koenderink, N.J.J.P., and Hulzebos, L., 2004, Conserveermiddelen on-line, Voedingsmiddelentechnologie, issue 4, p. 27.
- Verdenius, F. and Koenderink, N.J.J.P., 2003, Designing traceability systems, Proceedings of the Second International Workshop Information Technologies and Computing Techniques for the Agro-Food Sector (AfoT2003), Barcelona, Spain.
- Koenderink, N.J.J.P., Hulzebos, J.L., Roller, S., Egan, B., and Top, J.L., 2003, Antimicrobials On-line: Concept and Application for Multidisciplinary

Knowledge Exchange in the Food Domain, Proceedings of the Second International Workshop on Information Technologies and Computing Techniques for the Agro-Food Sector (AfoT2003), Barcelona, Spain.

- Ketelaars, N.J.J.P., Verdenius, F. and Top, J.L., 2002, Benut de meerwaarde van tracking & tracing, Voedingsmiddelentechnologie, issue 13, pp. 30-33.