Model-Based Development of a Multi-algorithm Harvest Planning System

Luis Diogo Couto[1], Peter W.V. Tran-Jørgensen[1(✉)], and Gareth T.C. Edwards[2]

[1] Department of Engineering, Aarhus University, Aarhus, Denmark
pvj@eng.au.dk
[2] Agro Inteligence ApS, Aarhus, Denmark

Abstract. Planning systems for harvest operations need to employ complex algorithms to calculate various aspects of the harvest plan such as the order in which to harvest field rows or when and where to unload harvesters. In traditional modelling and simulation approaches, it is not easy to vary the algorithm as a simulation parameter. This either limits the solution space for a system or it forces significant duplication to set up various models with the necessary algorithms. In this paper, we present the Model-Based Development of a planning system that leverages the strategy pattern to enable efficient variation of the optimisation algorithms at various stages of the planning process. We illustrate the system by applying it to a real field and discuss issues such as coping with large fields and how to carry out a real harvest operation according to the plan.

1 Introduction

There are various steps to calculating optimised solutions for harvest operations. These steps include partitioning of the field and calculating optimised coverage plans for harvesters and route plans for other vehicles. One approach to the problem involves the use of various optimisation algorithms that produce coverage plans for the harvesters [1,2]. However, planning of harvester routes is just one part of the harvest operation planning. Path planning for grain wagons (or similar) that service the harvesters must often also be developed. Algorithms exist for optimising service plans [3] but they are independent from those of harvesters. This independence makes it difficult to explore in detail how the various types of algorithms interact and combine to produce a complete solution for the harvest operation.

As an example, little research has previously been conducted into how harvesting and loading algorithms can affect operational execution times of harvest operations. Examples of planning tools for operations often employ a single algorithm; such as in-field unloading [4] or single point unloading [5]. Farmers will generally choose a plan with which they are familiar without considering alternatives.

M.S. Obaidat et al. (eds.), *Simulation and Modeling Methodologies, Technologies and Applications*, Advances in Intelligent Systems and Computing 676,
https://doi.org/10.1007/978-3-319-69832-8_2

In this paper, we seek to explore how different optimisation algorithms can be combined. We will explore this using a formal[1] model in combination with the strategy pattern from software engineering. The strategy pattern is used in the model to encode different optimisation algorithms. A novel aspect here is that the strategies representing the different kinds of algorithms (harvest routing and grain wagon path planning) co-exist and collaborate to produce the final solution.

From an operational research perspective, the harvest operation is an example of an output material flow (OMF) operation where material is removed from the field and transported to another location [6]. The machinery utilised within the OMF operation can be divided into two groups; Primary Units (PUs) which perform the main task i.e. harvesting the crop, and Service Units (SUs) which service the PUs by receiving harvested material and transporting it away. The capacity of the PU is many times smaller than the expected yield of the field, and therefore a PU unloads either to a nearby SU or directly to an out of field storage point.

The planning of the tasks of the PUs and SUs are often considered separately [7], with coverage plans being developed for PUs [1,2] and path plans being developed for grain wagons [3]. However, the tasks are spatially and temporally dependant on one another, so in order for efficient plans to be produced the plans must be developed concurrently [8].

To assist with the planning of in-field operations, fields can be decomposed into a number of tracks or rows. Many methods have been proposed for the decomposition of fields [4,9–11]. Fields are typically divided into headlands which encircle the field and can be used for turning, and working rows which transect the main area of the field. By confining all field traffic to drive along these predefined rows, the trafficked area of the field can be limited which has been shown to produce benefits on increased yield and better soil structure [12].

In the above mentioned approaches, the planning for the various kinds of vehicles is performed independently, as is the decomposition of the field. In our work, we consider all vehicles simultaneously when planning, although field decomposition is still done separately.

A different approach to optimisation was carried out in a EU project called DESTECS. In this project design space exploration is performed by sweeping parameters of models of cyber-physical systems [13]. Among other things, the DESTECS project proposes methodological guidelines for modelling fault-tolerant cyber-physical systems, which also involve the use of the strategy pattern to model faulty behaviour as well guarding against it [14]. This is similar to the presented approach, in that the strategy pattern is used in the DESTECS project to explore different behaviours of a system. However, while the DESTECS project used the strategy pattern to make a system more fault-tolerant, in this work the strategy pattern is used to help find optimised solutions to use in a harvest operation.

[1] *Formal* in this context means that the model is developed in a notation that is given semantics in a formal logic.

The strategy pattern is a design pattern [15] with two key features. First, the strategy pattern allows selection of different algorithms to be done at execution time and; secondly, it defines a family of interchangeable algorithms. Essentially, this allows the same functionality to be executed in different ways. Broadly speaking, the strategy pattern consists of a contract that defines the functions of a strategy in terms of their inputs and outputs including the properties that these functions may have. Given this contract, a specific strategy must provide an implementation of the functions that obeys the input and output properties of the contract, but which is free to use whatever algorithms are desired.

The remainder of this paper is structured as follows: in Sect. 2 we present the architecture of the formal model of the harvest operation based on the strategy pattern. The technologies that have been used to implement the model are described in Sect. 3. Next, the execution of the model is demonstrated in Sect. 4. Following that, in Sect. 5, we report the results of applying the model to a case study of a real field. The results are then discussed in Sect. 6. Afterwards, in Sect. 7, we describe how analysis of data reported by harvesters and grain wagons can be used to continuously optimise a plan over the course of a harvest. Finally, we conclude the paper in Sect. 8.

2 Model Architecture

2.1 Model Overview

The model was developed according to the structure shown in Fig. 1. The Execution Engine is responsible for coordinating the simulation and is connected to both the State and the three Strategy classes. The State contains the physical entities involved in the harvest operation. The harvesters are the PUs of the operation. Coverage plans and coordinated service points are developed for the harvesters by the employed strategies. The grain wagons are the SUs of the harvest operation and are used to convey material from the harvesters to the out-of-field storage. The service points coordinate when and where the grain wagons must meet the harvesters in order for material to be passed between the two.

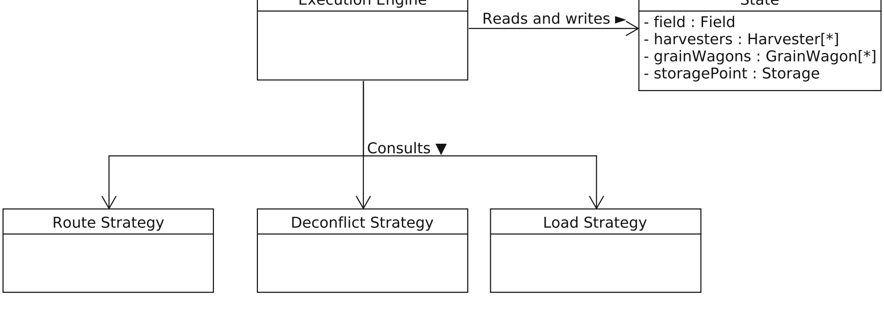

Fig. 1. Model structure realised as a UML class diagram. Originally published in [16].

Both the harvesters and the grain wagons are modelled by their physical parameters such as their working/non-working speed, storage capacity and material offload rate. These parameters are specified in the initialisation of the model. The storage point is the out-of-field storage where all material from the field must be transported to in order for the harvest operation to be completed. This too is modelled by its capacity.

The strategy classes define how certain aspects of the harvest operation are executed. In Fig. 1 these strategies are represented by the Route Strategy, Deconflict Strategy and Load Strategy classes.

Route Strategy. A route strategy is responsible for constructing the routes for harvesters. The routes direct the harvester from its location to a point where it will next require a service. A similar approach to the planning of routes for harvesters was also utilised in [4]. In this way the routes for multiple harvesters can be constructed in a consecutive manner.

As already stated, the construction of routes for the harvester and grain wagon are dependent on one another, therefore the route strategy must call functions from the loading strategy to ensure that the harvester is able to be serviced at the end of the route. The route strategies are allowed to produce more than one possible route for the harvester, these are later distinguished by the load strategy as appropriate.

Two route strategies have been implemented within the model: Predefined Route strategy and Greedy Route strategy.

The Predefined Route strategy enables the model to execute coverage plans that have been developed externally, provided they are represented as a sequence of rows to harvest. This strategy receives the assignment of a sequence of rows to a harvester as an input. A route is constructed which navigates the harvester along the sequence of rows, inserting service points where they are needed.

The Greedy Route strategy employs a search algorithm on the field to create a route for the harvester which will end with the harvester being as full as possible and in a position where it can be serviced. An extra constraint is also implemented within the strategy that every row must be harvested in its entirety and that all headland rows must be harvested before work rows.

Deconflict Strategy. A deconflict strategy is responsible for determining if a vehicle can move along its route, or calculating new routes if this is not possible. In the Simple Deconflict strategy a vehicle to reroute is chosen non-deterministically.

A deconflict strategy is responsible for the infield coordination of the vehicles. It is possible that conflicts can arise when a vehicle may block the path of another vehicle. In this case the deconflict strategy is employed to determine what course of action (such as planning a new route, or waiting for the obstruction to pass) is to be taken.

The Simple Deconflict strategy ensures that two vehicles cannot travel towards each other either along the same row or along two adjacent rows.

Load Strategy. A load strategy is responsible for assisting the route strategy to find a location where the harvester can be serviced and for constructing a route for the grain wagon from its current position to the service point and then to the out of field storage.

This is done through three functions of the load strategy that are called by the route strategy: `isDoneExtendingRoute()`, `isRouteServiceable()`, and `finaliseRoute()`.

`isDoneExtendingRoute()` checks if it is possible to extend a harvester's route. A common reason why it would not be possible to extend a harvester's route is if there are no more remaining rows in the field to be harvested, or if the harvester is full.

`finaliseRoute()` modifies a harvester's route to ensure the final position of the harvester is valid. For example if harvesting the full length of the final row of a harvester's route will cause the harvester to exceed its capacity, the route is modified so that a service point is required at some point along the length of the final row.

`isRouteServiceable()` checks that a grain wagon is able to converge on the service point that is required by the harvester's route, for example that there is a previously harvested row adjacent to the service point in which the grain wagon can move.

Four different versions of the load strategy have been developed in the model. These cover the four basic ways in which harvesters are unloaded during grain harvests.

The Single Point Unload version requires the harvester to transport material directly to the out of field storage point without using a grain wagon. It is important that the harvester must avoid the event of becoming full without a navigable path to the out of field storage. This strategy limits the amount of traffic in the field, which could offer benefits when reducing soil compaction.

The Headland Unload version limits the grain wagon to only travelling in the headland areas of the field. The harvester must avoid becoming full in the middle of the field as a grain wagon would not be able to meet it, therefore service points must be coordinated before the harvester becomes full while it is turning in the headland area.

The Infield Static Unload version allows the grain wagons to drive in the working areas of the field in order to meet the harvester. Service points are planned for the last possible moment to ensure that the harvester is full when it unloads.

The Infield Moving Unload version is similar to the Infield Static Unload strategy, however the harvester and the grain wagon are both moving when the load is being passed. As the machines remain in motion it is imperative that the grain wagon is travelling in the same direction as the harvester when they meet at the service point.

The Route, Load and Deconflict strategies are represented in Fig. 1 by their contracts. The various concrete versions of each strategy must conform to these contracts. Figure 2 shows how the various load strategies are realised based on the `ILoadStrategy` class that defines the contract. Whenever the model is executed, a concrete strategy of each kind must be provided to the Execution Engine.

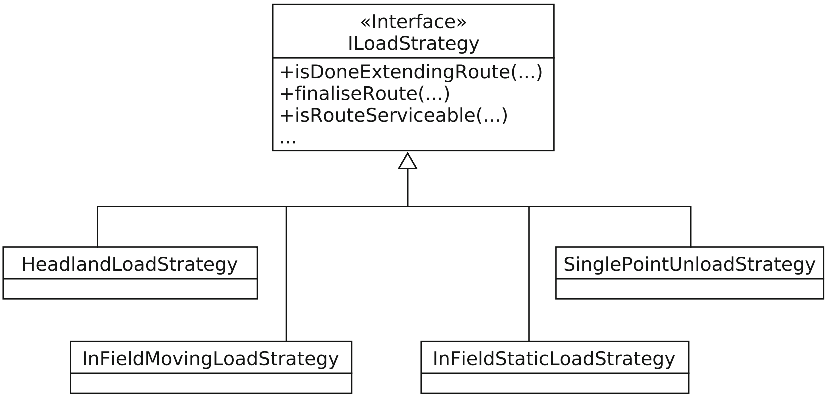

Fig. 2. Load strategy hierarchy realised as a UML class diagram.

Not all versions of a strategy can be used in all situations. In order to cope with this, a notion of *strategy feasibility* has been introduced. The strategy feasibility check is implemented as a function in each of the strategies and invoked at the beginning of model execution in order to check if the field meets the requirements of the strategy configuration. The advantage of this approach is that the feasibility of each version of a strategy is encapsulated in that version itself, so the remaining parts of the model need not be aware of its specific details.

The concrete versions of strategies can be used to model different optimisation algorithms and therefore vary in implementation detail as well as the restrictions they impose on the harvest operation.

3 Model Implementation

The model drives the development of a harvest planning system, which is developed using the Vienna Development Method (VDM) and implemented using code generation. VDM is one of the longest-established formal methods for the development of computer-based systems. This method focuses on the development and analysis of a system model expressed in a formal language.

The strategy pattern is based on object-oriented (OO) features [17], as enabled by the VDM++ formal modelling language [18]. VDM++ is the OO dialect of VDM. Broadly speaking, a VDM++ model consists of a series of definitions for types, functions, operations, etc. The OO features of VDM++ allow for structuring the model into classes and provide standard OO mechanisms such as inheritance.

In addition to allowing for an effective implementation of the strategy pattern, the OO features of VDM++ have other useful benefits, including the ability to add new versions of a strategy that reuse parts of an existing strategy and change only those parts that must be different. Additionally, object-orientation facilitates modularity and encapsulation which, while not essential to develop the model, make it easier to do so.

There are several reasons for choosing a formal language such as VDM++ over an OO implementation language such as Java or C++. The use of VDM++ promotes a high-level approach that abstracts away details that are of little importance to harvesting operations. The formal semantics underpinning the VDM language allow us to have confidence in the results and that there are no errors in the language and tool that can "contaminate" the result. Additionally, VDM has features that enable us to describe the properties of the model and its functions, and these properties are constantly checked during model execution. For example, in the model the capacity is expressed as a floating point number, which must always be positive and smaller than 1. VDM invariants allow us to attach such a property to the capacity variable in order to ensure that the model never violates this. While that is a simple example, VDM allows us to express any arbitrary property that can be described in terms of first-order logic. Many of the benefits of using VDM cannot be achieved using implementation languages, which operate at a lower level of abstraction. In particular implementation languages must take things such as the underlying hardware platform into account. Use of VDM allows us to focus solely on the development of the strategies, which is our primary concern.

4 Model Execution

In order to execute the model, it is first necessary to configure the harvest operation by loading both the field and the resources, i.e. the State, and also one of each class of strategy to guide the Execution Engine during the simulation. Once this is done, the model is executed and whenever the Execution Engine reaches a point where it needs to make a decision that depends on a strategy, it will consult whatever strategy it has loaded and the output of the strategy will be used to further progress execution of the model. As an example, in Fig. 3, the Execution Engine needs to know which vehicles are movable at a given point in time. One particular version of the strategy may allow the harvesters to move because they can offload in the work rows. Another version may not allow the harvesters to move because they can only offload in the headlands and they cannot fully harvest the next work row.[2] In this way, different versions of a strategy lead to different outcomes in the model.

One of the key features of the model is the ability to explore strategy combinations and how their interactions affect the performance of the harvest operation. One way to do this is by fixing two kinds of strategies and varying the remainder (for example, load strategies) thus investigating how a particular aspect of optimisation affects the overall harvest operation. Conversely, if external restrictions dictate the use of a particular strategy, then the other strategies may be manipulated to find the best solution within the restrictions. For a small number of

[2] In both of these examples, the route strategy consults the load strategy as part of its calculation of movable vehicles.

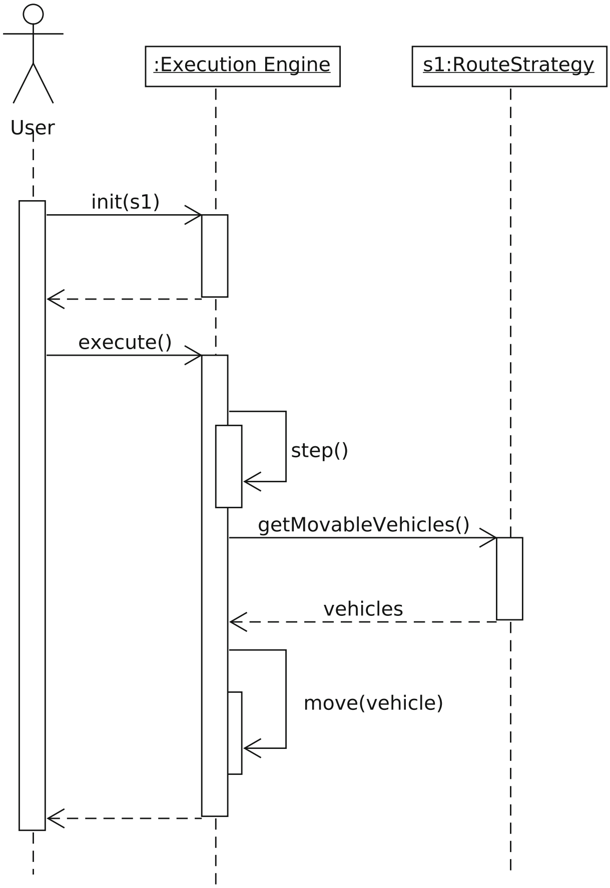

Fig. 3. Strategy dispatching realised as a UML sequence diagram. Originally published in [16].

strategies, testing the different scenarios of interest can be done with manually written tests. However, when the number of scenarios to be tested is large then an automated combinatorial testing feature for VDM can be used to concisely specify the various combinations and automatically generate and execute the corresponding tests [19].

4.1 Simulation Visualisation

As part of model execution, a log of all the important events in the harvest operation is produced. Logged events include vehicle movement, harvesting of a row, passing load between harvesters and grain wagons, etc. Once execution is completed, this log can be inspected in order to get a full understanding of the harvest operation outcome. This log can also be seen as a harvest plan since it contains detailed instructions of when and where the different vehicles must go.

In order to better understand what occurred during the simulation, the log can also be analysed. However, as manual inspection of the log is difficult, a proof-of-concept visualization tool was developed to analyse the log and replay the simulation as shown in Fig. 4. The figure shows a representation of the field partitioned into work rows and headlands. The black square represents the harvester, the circle represents the grain wagon and the square at the bottom represents the storage point. As the log is processed, the visualiser displays an animation of the vehicles moving along the field.

Fig. 4. Simulation visualisation. Originally published in [16].

5 Results

This section demonstrates the approach by reporting results of executing various simulations with the model in order to explore the interactions between all possible combinations of the strategies described in Sect. 2.1. Every execution was performed with the same resources and on the same field. The focus is not on changing the parameters of the simulation such as number of harvesters or harvester capacity but in changing the strategy versions used in each simulation.

The simulations were carried out on a representation of a real field located in the vicinity of the Research Center at Foulum, Denmark (56°29′N, 9°35′E). The yield of the field is simulated and is lower for headland rows than for working rows, as is typical in real fields (due to excess soil damage, lower nutrients, etc.). The yield is further constrained such that a complete lap of the field can be made without exceeding the harvester capacity, and no single working row can exceed the capacity of the harvester. The field, partitioned into rows, is shown in Fig. 5.

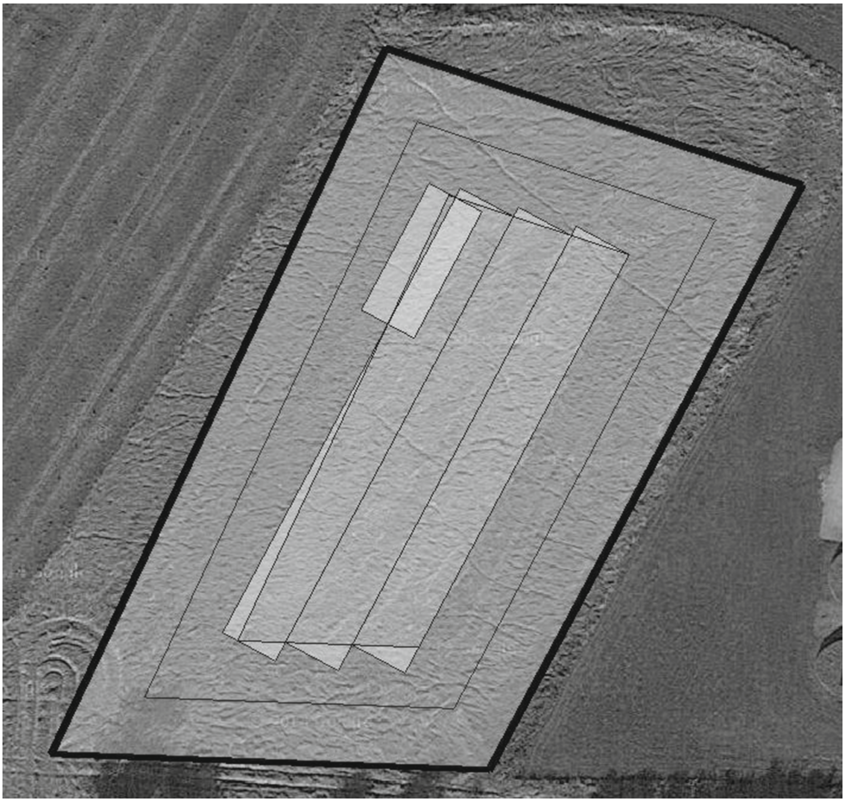

Fig. 5. Agro Park field. Originally published in [16].

The results of the simulations are summarised in Table 1. Each row in the table represents a particular simulation, indexed by the *Sim.* (Simulation) column. The *Route* and *Load* columns identify the combination of strategies used in each particular simulation (the same deconflict strategy – Simple Deconflict – is used for all simulations). The *Op. Time* (Operational Time) column reports the duration of the harvest operation in seconds and serves as an indication of how well a combination of strategies performs. Finally the *Exec. Time* (Execution Time) column reports the actual, physical time in seconds it takes to execute the simulation.

The simulation was executed using a Java 7 code generated version of the model on a Fujitsu LIFEBOOK U772 laptop with a 1.7 GHz Intel Core i5 processor and 8 GB of memory running a Windows 7 Professional Edition operating system.

Table 1. Results summary. Originally published in [16].

Sim	Route	Load	Op. time [s]	Exec. time [s]
1	Greedy	Headlands	425.558	12.619
2	Predefined	Headlands	497.38	13.417
3	Greedy	In field static	420.694	12.319
4	Predefined	In field static	463.484	13.912
5	Greedy	In field moving	410.298	7.056
6	Predefined	In field moving	446.854	7.25
7	Greedy	Single point	679.498	26.977
8	Predefined	Single point	623.347	4.421

6 Discussion

Table 1 shows that for the field subject to analysis, for most of the unloading strategies, the *Greedy Route* strategy produces a better solution, than the *Predefined Route* strategy as indicated by the operational time. This is due to the harvester's route used as an input for the *Predefined Route* strategy being developed as a coverage plan that ignores the coordination of the grain wagons. As the *Greedy Route* strategy was able to enquire the constraints of the unloading strategy while developing the harvester's route, the final solution is more integrated and allows for more efficient operations. This indicates that it may be advantageous to use optimisation approaches that consider both harvesters and grain wagons when developing routes.

The *Infield Moving Unloading* strategy offers the best operational times for both of the routing strategies. This unloading strategy is likely to offer the best solution as it allows the harvester to be completely full when it offloads and does not require the harvester to stop. It is also worth noting that the model allows this hypothesis to be further confirmed by adding additional route strategies and checking the resulting operational times.

In terms of actual execution times, most combinations yield similar results for *Greedy* and *Predefined* strategies. The exception is for the *Single Point Unload* strategy, where the *Greedy* version has a significantly higher execution time. This is mostly due to the fact that many more routes have to be computed for this particular combination, which makes it significantly slower than its *Predefined Route* counterpart.

The field used for these simulations is small, especially in terms of the numbers of rows and headland laps. Indeed, when the model was under initial development, small fields were preferred as they allowed for quick execution of model simulation, which enabled fast iterations of model development. However, as the model stabilised and we began to apply the system for the harvest planning of larger fields, we experienced significant performance issues.

For larger fields (ten rows or more), the performance of the system was unacceptably slow. The primary culprit for the poor performance was the data representation of the field and the algorithms used to implement common operations on the field (such as shortest path calculation). One of the reasons for these inefficient implementations was the use of VDM itself. As a formal modelling language, VDM and its associated tools are more concerned with semantic fidelity and validity of analyses than with simulation performance.

To address this issue, the field representation in the model was replaced with a handwritten Java implementation. The Overture VDM-Java bridge [20] was used to connect this implementation to the model, and a new mechanism called the *delegate* was introduced to ensure seamless realisation and integration of the final system. The introduction of the Java component led to performance gains of 3000% [21] that effectively addressed the performance issues of the previous version of the model and helped us achieve acceptable system performance for much larger fields.

7 Live Planning

The harvest planning system as described so far uses *offline planning* techniques to analyse harvest operations. Essentially, this means that planning is only done once, and that the system takes no measures against unforeseen scenarios that may necessitate re-planning over the course of the harvest operation. *Live planning* techniques, on the other hand, continuously analyse data reported by the vehicles (positions, bin levels etc.) and try to optimise the harvest plan by assigning more efficient routes to the vehicles. A live version of the system can therefore be seen as a tool that serves to guide the operators of the vehicles throughout the harvest operation.

A live system is currently under development, and we expect to use it in a realistic harvest setting in the near future. The live system will run on a server that receives live data from the vehicles and uses this data to further improve the harvest plan. Once new routes have been calculated, these will be assigned to the vehicles, which connect to the server via light-weight clients. A light-weight client is responsible for communicating live data to the server and provide the operator of the vehicle with information about the route that the vehicle is currently assigned. The structure of the live system is visualised in Fig. 6.

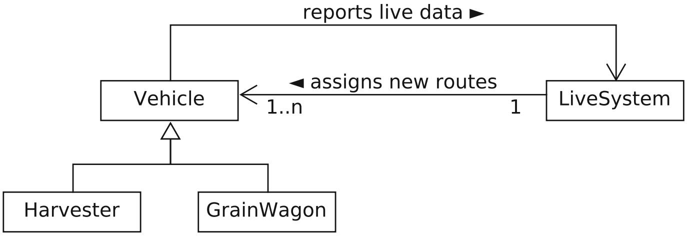

Fig. 6. The structure of the live system realised as a UML class diagram.

As a first step, the live planning system computes initial routes for each vehicle, which is similar to what the offline planning system already does. Based on the initial routes, the vehicles start to harvest the field, and as the harvest progresses, the vehicles report live data to the server. If the data reported by the vehicles indicates a need for re-planning, then the live system responds by calculating and assigning new routes to the vehicles. This step is repeated until the harvest operation has completed.

Several situations may occur which necessitate re-planning. For example, a harvester may report a bin level that is smaller than what is expected by the system. In that case the system may decide to extend the harvester's route to compensate for the smaller bin level. However, changing the harvester's route may necessitate re-calculations of unload points, hence also affecting the routes of other vehicles. As another example, a vehicle may encounter an unexpected obstacle in the field that prevents it from following the route it has been assigned. Similar to the previous example, such situations necessitate re-planning which may affect the other vehicles that participate in the harvest operation.

The harvest planning system is only a viable solution if using the system leads to better harvest results. To show this, the results obtained using the live system will be compared to those obtained using traditional harvest approaches, i.e. before the harvest planning system was used. When the live system has been used in a realistic harvest setting it will be possible to obtain quantitative evidence that shows if the system achieves better results. Demonstration of this is crucial in order to convince farmers to start using the system.

8 Conclusions and Future Work

In this paper, we have presented the model-based development of a harvest planning system. We have shown how the strategy pattern enables the application of different optimisation algorithms to different phases of the harvest planning system. Further, we have shown how such algorithms can be easily varied across multiple simulations, enabling a swift and comprehensive exploration of the design space.

We have shown an example application of our system to develop plans for a small real field in Denmark. As the size of the fields increases, the performance of the model was greatly degraded. This was dealt with by replacing a portion of the model with a handwritten Java component that provides efficient implementations of the more computationally-intensive operations in the model.

The next step in our work is to validate the plans produced by the system by applying them to a real harvest and verifying if the results are better than those obtained with traditional harvest planning approaches. Towards that end, work has begun on a live planning system that will assist vehicle operators in following a harvest plan. This system will also take advantage of live data to adapt and improve the plan on-the-fly.

Acknowledgements. A previous version of this paper was presented at the SIMULTECH 2016 conference [16]. The work described in this paper was partially carried out in the context of the Danish High Technology Foundation research project Off-line and on-line logistics planning of harvesting processes. We would like to thank all our colleagues on the project for their valuable contributions and feedback, particularly Peter Gorm Larsen, Claus Grøn Sørensen, Dionysis Bochtis and Morten Bilde. We also thank Kun Zhou for his assistance with the harvest visualisation.

References

1. Spekken, M., de Bruin, S.: Optimized routing on agricultural fields by minimizing maneuvering and servicing time. Precis. Agric. **14**, 224–244 (2013)
2. Edwards, G., Christiansen, M.P., Bochtis, D.D., Sørensen, C.G.: A test platform for planned field operations using LEGO mindstorms NXT. Robotics **2**, 203–216 (2013)
3. Jensen, M.A.F., Bochtis, D.D., Sørensen, C.G., Blas, M.R., Lykkegaard, K.L.: In-field and inter-field path planning for agricultural transport units. Comput. Indus. Eng. **63**, 1054–1061 (2012)
4. Oksanen, T., Visala, A.: Coverage path planning algorithms for agricultural field machines. J. Field Robot. **26**, 651–668 (2009)
5. Edwards, G., Jensen, M.A.F., Bochtis, D.D.: Coverage planning for capacitated field operations under spatial variability. Int. J. Sustain. Agric. Manag. Inf. **1**, 120–129 (2015)
6. Bochtis, D.D., Sørensen, C.G.: The vehicle routing problem in field logistics part I. Biosyst. Eng. **104**, 447–457 (2009)
7. Jensen, M.A.F.: Operations planning for agricultural machinery under capacity constraints. Ph.D. thesis, Aarhus University (2014)
8. Scheuren, S., Stiene, S., Hartanto, R., Hertzberg, J., Reinecke, M.: Spatio-temporally constrained planning for cooperative vehicles in a harvesting scenario. KI-Künstliche Intelligenz **27**, 341–346 (2013)
9. Jin, J., Tang, L.: Optimal coverage path planning for arable farming on 2d surfaces. Trans. ASABE **53**, 283 (2010)
10. Zandonadi, R.S.: Computational tools for improving route planning in agricultural field operations. Ph.D. thesis, University of Kentucky (2012)
11. Hameed, I., Bochtis, D.D., Sørensen, C.G., Jensen, A.L., Larsen, R.: Optimized driving direction based on a three-dimensional field representation. Comput. Electron. Agric. **91**, 145–153 (2013)
12. Tullberg, J.: Tillage, traffic and sustainability – a challenge for ISTRO. Soil Tillage Res. **111**, 26–32 (2010)

13. Fitzgerald, J., Larsen, P.G., Verhoef, M. (eds.): Collaborative Design for Embedded Systems – Co-modelling and Co-simulation. Springer, Heidelberg (2014)
14. Broenink, J.F., Fitzgerald, J., Gamble, C., Ingram, C., Mader, A., Marincic, J., Ni, Y., Pierce, K., Zhang, X.: Methodological guidelines 3. Technical report, The DESTECS Project (INFSO-ICT-248134) (2012)
15. Gamma, E., Helm, R., Johnson, R., Vlissides, R.: Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley Publishing Company, Reading (1995)
16. Couto, L.D., Tran-Jørgensen, P.W.V., Edwards, G.T.C.: Combining harvesting operation optimisations using strategy-based simulation. In: Proceedings of the 6th International Conference on Simulation and Modeling Methodologies, Technologies and Applications - Volume 1: SIMULTECH, pp. 25–32 (2016)
17. Meyer, B.: Object-Oriented Software Construction. Prentice-Hall International, Upper Saddle River (1988)
18. Fitzgerald, J., Larsen, P.G., Mukherjee, P., Plat, N., Verhoef, M.: Validated Designs for Object-Oriented Systems. Springer, New York (2005)
19. Larsen, P.G., Lausdahl, K., Battle, N.: Combinatorial testing for VDM. In: Proceedings of the 2010 8th IEEE International Conference on Software Engineering and Formal Methods, SEFM 2010, Washington, DC, USA, pp. 278–285. IEEE Computer Society (2010). ISBN 978-0-7695-4153-2
20. Nielsen, C.B., Lausdahl, K., Larsen, P.G.: Combining VDM with executable code. In: Derrick, J., Fitzgerald, J., Gnesi, S., Khurshid, S., Leuschel, M., Reeves, S., Riccobene, E. (eds.) Abstract State Machines, Alloy, B, VDM, and Z. Lecture Notes in Computer Science, vol. 7316, pp. 266–279. Springer, Heidelberg (2012)
21. Couto, L.D., Tran-Jørgensen, P.W.V.: Integrating real system components in model-based development. Submitted to the 33rd ACM/SIGAPP Symposium on Applied Computing (SAC 2018)

Springer

http://www.springer.com/978-3-319-69831-1

Simulation and Modeling Methodologies, Technologies and Applications
International Conference, SIMULTECH 2016 Lisbon, Portugal, July 29-31, 2016, Revised Selected Papers
Obaidat, M.S.; Oren, T.; Merkuryev, Y. (Eds.)
2018, XIV, 316 p. 145 illus., Softcover
ISBN: 978-3-319-69831-1