Agent-based simulation framework for virtual prototyping of advanced livestock precision feeding systems

J. Pomar a, V. Lópeza, C. Pomb,⇑

a Department of Agricultural Engineering, Universitat de Lleida, Alcalde Rovira Roure, 191, 25198 Lleida, Spain
b Dairy and Swine Research and Development Centre, Agriculture and Agri-Food Canada, P.O. Box 90, 2000 Route, 108 East Lennoxville, Que., Canada J1M 1Z3

Keywords:
Precision livestock farming
Precision feeding
Farm equipment design
Multi-agent
Virtual prototyping
Composable simulation

1. Introduction

Precision livestock farming is an innovative production system approach which is based on intensive and integrated use of advances in animal sciences and in the new technology of information and communication (Berckmans, 2004; Day, 2005). Its main objective is to optimize animal production and the management of the productive processes (Groot Koerkamp et al., 2007; Berckmans, 2008). Today, this approach offers many opportunities (Wathes et al., 2008), such as the increased efficiency in the use of productive resources which can be obtained by reducing uncertainty in decisions relating the control of the variability that exists among farm animals. A relevant contribution at present in this regard is the development of precision feeding systems (Pomar et al., 2009, 2010; Niemi et al. 2010) which lays the groundwork for addressing key issues in today intensive livestock farming which are: (1) improving production system sustainability by increasing feed efficiency, (2) reducing the environmental impact and (3) increasing food safety through traceability.

1.1. Precision feeding equipment design

Precision feeding involves the use of feeding techniques that allow the proper amount of feed with a suitable composition to be supplied in a timely manner to each animal in the herd. Practical on-farm application requires designing and developing automatic feeding systems able of supplying in real-time the feed whose composition and quantity are adapted daily to the needs of each animal of the farm (Pomar et al., 2009). The practical application of these systems can be difficult because it requires estimating in a short period the nutrient requirement of each animal after filtering and cleaning all the data collected by sensors. An automatic precision feeding device is only able to feed a limited number of animals as feeder occupation time must be less than 100%. For this reason, a farm requires a considerable number of units according to the total number of animals. To do their work, these feeders require sensors, controllers, and actuators together with robust and efficient management and control systems with the capacity for complex data processing. They must also be able to meet in real-time the demand for a high number of processes to be executed cooperatively by all feeders of the farm. The main difficulties associated with the design and development of these systems are, on the one hand, their complexity and the time and cost required for their development and, on the other, the difficulty in validating and testing them prior to their physical use on farms.
1.2. Virtual simulation and virtual prototyping

Virtual simulation tools currently play an important role in the development of new products, as in the case of precision feeding systems equipped with automatic control and advanced information management. These tools have attracted the interest of both the scientific community, which is interested in exploring their potential, and developers, who see them as a particularly interesting tool for reducing development time and cost (Becker et al., 2005; Choi and Cheung, 2008). Virtual prototyping can be viewed as a computer-aided design process based on modeling and simulation tools which easily combine different design aspects such as physical layout, logical components, operational concept, functional specifications and dynamic analysis under different operating conditions (Drews and Weyrich, 1997). To make full use of the potential of virtual prototyping when modeling systems based on the integration of physical components, Xiang et al. (2004) proposed a series of challenges that must be met, which we have adapted to the realm of precision feeding, namely:

1.2.1. Precise representation of the real system

Virtual prototyping must properly model the real system. This modeling may use the technique of incremental prototyping in which functionalities are gradually and hierarchically introduced into the prototype verifying at each step that the integrated component reliably represents the part of the real system being modeled.

1.2.2. Composability

Each virtual component must constitute an independent entity that represents a specific function or component of the actual system, thereby allowing the virtual system to be built by assembling those components. These components may be designed using the widely accepted method of decomposition by subfunctions (modularity), thereby reducing complexity in the implementation of the components that the model must represent. In this way, the system becomes a set of composable components that interact cooperatively to represent the actual system. However, although composability is of great interest in modeling and simulation, its implementation is not without difficulties (Diaz-Calderón, 2000).

1.2.3. Communication and coordination

Virtual prototyping requires a mechanism that makes it possible for the model’s set of virtual components to communicate among themselves and to act cooperatively. Another desirable feature is the ability to adapt to changing circumstances in the environment. This is the domain of distributed artificial intelligence. The current trend is to use intelligent agents to achieve distributed coordination (Tatara et al., 2007). Therefore, for virtual prototyping to have this capacity, an agent-based framework is required. An agent is defined in this context as a piece of software that contains the code and data necessary to perform well-defined functions. Agents may assume tasks of varying complexity that can go from simple functions (e.g., visualization of parameters) to more complex functions that require heavy data processing (e.g., determination of the optimal composition of feed for each animal on the farm in real time). Agents differ from objects and other types of encapsulated software components in that they are capable of exchanging information, requesting services and negotiating with other agents via a high-level agent communication language (ACL) (Labrou et al., 1999). An ACL allows a set of agents to work cooperatively to resolve a problem or perform a function that exceeds the capacity of a single agent (Parrott et al., 2003). In this process, many different software components are exchanging messages and asynchronously processing responses to solve a problem or meet a requirement. In this context, interoperability between components and coordination and communication between agents are two key aspects in the development of these systems (Labrou et al., 1999).

1.2.4. Reliability and efficiency

Lastly, but no less importantly, there are the challenges associated with reliability and efficiency in the implementation process and testing of a system based on virtual prototyping. Some of the advantages that academic literature attributes to virtual prototyping relate to significantly reduced development costs and times, which, together with virtual simulation, is frequently viewed as a means of overcoming the cost and time limitations associated with physical experimentation on the actual system (Becker et al., 2005). In fact, these not only help reduce experimentation and testing costs, but also help improve design quality (Choi and Cheung, 2008) as information on the behavior and performance of the system is available very early during the development process.

This article proposes an agent-based composable simulation framework for the virtual prototyping of on-farm precision feeding systems that takes the above-mentioned challenges into account. This simulation and prototyping framework has been built to provide a computer-aided design tool to assist in the process of designing, implementing, validating and testing automated precision feeding systems. An important and value-added contribution of the proposed approach is the ability to convert the validated virtual prototype into a real system. This is achieved using the capacity of the agents implemented in NI LabView G language of interacting with the real devices that constitute the physical layer.

2. Agent-oriented system design

For the correct functioning of a precision feeding system, all components of the system must interact with one another to achieve full functionality of the system. This principle is analogous to an agent-based framework. Each functionality of the system, either logical (monitoring, data analysis, supervision, etc.) or pertaining to the physical layer (feeders, feed mixer, distribution system, etc.) can be represented by an agent that cooperates and interacts with other agents to carry out the different functions of the actual system. In this proposed approach, each component of the precision feeding system is represented by a virtual domain agent. This section focuses on the typification and classification of the agents proposed in this study, a description of the proposed agent architecture and the procedures for coordination and communication between agents.

2.1. Agent implementation structure

Based on the functionalities to be achieved in the field of precision feeding, two types of agents are proposed for the virtual prototyping: domain agents and control agents. Domain agents are responsible for responding to the set of tasks required for normal functioning of the system. Control agents are responsible for supervising and assisting in maintaining proper functioning of the system and acting preventively in the event of an anomaly. This definition based on agents’ responsibilities is well adapted to the features of control and management processes for automatic equipment such as that presented in this paper.

There are many different means of implementing an agent architecture and agent communication infrastructure (Parrott et al., 2003). One efficient method for the implementation proposed in this paper is to create an architecture common to all agents and encapsulate the functionality associated with each agent in a specific component (domain module). This approach
helps effectively encapsulate the specific functionality of a component, while respecting interface conventions and the internal control mechanisms of the agent, thereby creating an architecture common to all agents. This design option facilitates implementation and validation, since to create new agents we only need to focus on the implementation of their specific functionalities. In our context, encapsulation refers to separation of the specific functional aspects of the agent from the internal details of the component that are hidden from components. Fig. 1 shows a conceptual model of the agent architecture proposed in this paper, which essentially consists of four components:

2.1.1. Domain process set

The domain process set contains the knowledge, algorithms and essential data that determine the specific functional behavior of each agent. The process output generates different actions according to the context as a message in response to a task requested by another agent, a message requesting a service by another agent, triggering actuators or monitorization process output in the user interface. One example of visualization of variables in real time in the user interface is that demonstrated in Fig. 2 (feeder agent), which shows whether the feeder is free or occupied by an animal, the weight of food stored in the hopper available to the animals, identification of the animal when it is at the feeder, feed consumption, etc.

2.1.2. Control unit (CU) and agent coordination

The CU controls the execution of process demands from other agents and the subsequent reaction of the agent. The set of CU for all agents behaves as control mechanism for all the system by concurrent execution of all CU and determines the asynchronous functioning of the system. This coordination mechanism facilitates the implementation of new functionalities by allowing new agents to be incorporated without modifying the existing ones. The set of agents that model and simulate the actual system, together with their interactions, can be considered a multi-agent system.

2.1.3. Agent communication interface

The communication interface captures and sends different messages between one agent and the rest of the agents through the communication network. The message reader listens for messages that are sent to the network and filters the messages sent to itself and pushed into the input message subscription queue. This mechanism prevents the loss of messages when the agent is processing a previously requested task. The message parser parses the incoming

---

![Fig. 1. Conceptual agent architecture model.](image-url)
message and pushes it to the domain process set through the control unit. Finally, if the process output is a response message, it is pushed to the message builder and output message queue and finally into the message writer, to be sent to the network.

2.1.4. Agent user interface

The agent user interface (AUI) is responsible for interaction between the user and the agent itself. The user interface proposed in this study is graphic in nature (see Fig. 2). Only the agents communicating with the user have this component activated in this study.

2.2. Agent communication

In our system, several agents must collaborate to carry out the different tasks required by the system. Under this approach, communication between agents takes on special importance. This requires the use of a common agent communication language (ACL) to help achieve that interoperability. Messages must also adopt a common domain-specific ontology in which to be expressed. There are many ways to encode and exchange messages through different definitions of ACLs. The current trend is towards standardized representation of information and expanded scope from simple messages to complete transactions (Chen et al., 2008). The use of standardized representation of the information in the ACL will definitely simplify the development and implementation of multi-agent systems. Extensible Markup Language (XML) is a standard for building tag-based documents and structured data. It is a markup language for representing and exchanging data over the Internet. A number of researchers have proposed the use of XML to encode ACL messages as this offers a number of advantages: it is easy to generate, parse and edit, and is also self-descriptive, consistent and structured (Chen et al., 2008).

An ACL has been defined for the virtual simulation environment presented in this article that uses message coding based on a simplified XML format. A sample message can be seen in Fig. 3. In this example, a message sent by a feeder agent when an animal's visit to the feeder is completed is directed at the agent responsible for recording feed consumption events. In our system, the sender and intended recipient of the message are identified by a unique identifier for each agent established using a method similar to the IP addressing used on the Internet. In this work, identifiers with two numeric groups only have been used in place of the four groups used in IP, since the addressing capacity with two groups is sufficient to meet the requirements. In the sample in Fig. 3, we can see that the identifiers for the sender and recipient are "06.08" and "21.01", respectively. This method of identification has advantages for the proposed virtual simulation environment: multiple addressing capability and grouping of agents by functional families. For example, on a farm that has installed precision feeding with a certain number of physical feeders set up in different rooms, the virtual agents that represent those feeders and that are responsible for their management and control, and that are hypothetically installed in Room 6, could be identified as "06.01", "06.02", … "06.0n". In this way, if a message on a parameter change is to be sent to all control agents for the feeders in-
stalled in Room 06, “06.” could be used as identification of the recipient, with those agents viewed as the same group. The “content type” attribute is used to give meaning to the information contained within “content” tags.

3. Agent virtual environment implementation

3.1. Problem description

Precision feeding currently constitutes a relevant contribution in the field of precision livestock farming and a new approach to improved sustainability in hog production (see Section 1). It takes into account the existence of variability (heterogeneity) in the nutritional needs of farm animals. Its objective is therefore to increase efficiency in the use of feed and reduce the production of polluting waste (mainly N and P) by automatically feeding each animal according to its needs (Pomar et al., 2009, 2010). On-farm implementation of the concept of precision feeding requires the design of new, automatic feeding equipment capable of measuring that variability and the use of mathematical models for estimating the potential for individual growth, and must be capable of automatically supplying feed whose composition is adjusted to the daily needs of individual animals or small groups of animals.

This paper has been based on what we could classify as first-generation feeding precision equipment such as that installed on an experimental farm at the AAFC Dairy and Swine Research & Development Centre, Sherbrooke, Quebec, Canada, in which animals are housed in groups of 12–15 animals. Each of these groups shares a unique feeder that dispenses feed whose composition is tailored to the daily needs of the group. To achieve this functionality, the system is basically composed of the following physical components (see Fig. 4):

3.1.1. Feeder units

The feeder unit uses a commercial automated recording system (IVOG®-station, Insentec, Marknesse, Netherlands). The function of these units is to supply a specific feed ad libitum to an individual animal that voluntarily visits the feeder. The units consist of a hopper that stores a certain amount of feed, a device that dispenses small quantities of feed that is voluntarily activated by the animal and a bowl with weighing capacity onto which the feed is poured and taken from by the animal. The functions they carry out are: electronic identification of the animal (e-ID) through the use of a transponder chip (electronic ear tags), recording of the time of entry and departure of the animal, measurement of feed consumption by the animal during the visit and, optionally, measurement of other optional variables such as weight. The number of these units that must be installed depends primarily on the farm’s capacity in number of animals. For example, a farm with a capacity for 500 hogs and a maximum of 12 animals per feeder would require 40 units. These units are connected through a communication network to transmit the data generated and exchange information with a centralized data processing and control system.

3.1.2. Feed blender unit

The function of this unit is to produce small quantities of feed (20–40 kg) whose composition is tailored to the daily needs of the animals of each group. The desired composition of the feed is obtained by means of a mechanism for precisely dosing and mixing two to four basic feeds.

3.1.3. System for transporting feed to the feeding units

The function of this unit is to convey the feed blend in the previous unit to the feeder for which it is intended. This system therefore has a tube to transport solid feed by drag chain conveyor and a

![Fig. 4. Physical components of the first-generation precision feeding system and their representation as agent-based virtual components.](image-url)
set of hatches with openings triggered by an electric actuator managed by the central electronic control unit.

3.1.4. Electronic control units

Each of the previous units has its own electronic controller connected to a central computer that supervises and monitors the system and performs data management.

3.2. Multiagent environment programing

The virtual simulation environment has been programed in the National Instruments (NI) LabView programing environment. LabView (Laboratory Virtual Instrumentation Engineering Workbench) is a development platform that has been gaining acceptance with a wide base of users in the scientific and engineering fields. There are currently open-source communities such as OpenG.org that promote and share code developed for LabView (Elliott et al., 2007). LabView is a visual programing language (VPL) designed to facilitate the development of applications that interact with the real world through the acquisition and analysis of data, monitoring and automatic control. There are various key characteristics of the language that make it particularly useful for virtual prototyping and the virtual simulation of applications that facilitate interaction with the real world through data acquisition (Whitley et al., 2006). Our experience with creating integrated automation systems has validated our choice of LabView as an appropriate tool.

LabView includes a rich set of control and analysis functions that offers functionality for a variety of communication protocols including RS232, GPIB and TCP/IP and hardware interfaces such as PCI, USB and PXI (Elliott et al., 2007). LabView implements the dataflow paradigm in which code is not written in text form but rather graphically represented in G language (see Fig. 5). Each function or routine is contained in a virtual instrument (VI) that has two components: the front panel that contains the user interface and a block diagram in which the G-code is graphically represented. LabView supports multithreaded application design and executes code in an inherently parallel environment. This capacity makes it possible to invoke multiple instances or clones of the same function by maintaining their own data space in each. For example, we can drag various clones of the feeder-agent virtual instrument (see Section 3.3) into the LabView environment and all can be run simultaneously in parallel. This feature is extremely useful in the implementation of a multi-agent virtual environment in the field of precision feeding, in which each agent requires its own independent concurrent process and data space.

To create a flexible virtual simulation environment with a good cost-effectiveness ratio, we have taken the approach of building each virtual component as an independent VI. This feature means that the virtual simulation environment can be built by integrating those components in a common execution environment. Once the virtual model has been validated and tested, LabView offers the possibility of easy connectivity and integration with devices that constitute the physical layer of the actual system. This allows expanding the multi-agent system to physical devices thus completing the entire design cycle and turning it into a true computer-aided design process.

3.3. Agent-based virtual components

Applying the principle that a precision feeding system can be implemented as a set of specialized blocks that interact with one another to achieve full functionality (see Section 2), in our virtual simulation environment, each block is represented by an agent
based virtual component. Each physical component and logical functionality will have one-to-one mapping to virtual components. This is important given that a virtual prototype can be constructed in the same way as an actual system by assembling the corresponding virtual agent components.

For purposes of investigation and evaluation of the virtual simulation environment, a set of agent-based virtual components has been implemented that simulates the physical devices and functionalities that make up the actual precision feeding system referred to in Section 3.1. Following the principles of the proposed agent architecture (see Section 2.2), the agent-based components have been implemented with a common architecture, encapsulating the functionalities proper to the agent’s domain in a specific module. This feature is quite advantageous in the implementation process for new agents as it only requires implementation of the specific domain module while the other internal elements of the agent are reused.

The functionalities that make common behaviors and interoperability possible for the set of agents include:

- **Initialization and shutdown** – by means of which procedures to initialize or shut down the agent are activated.
- **Subscription management** – by means of which each agent is discharged during the startup process through a subscription message. An updating message, according to a time parameter, subsequently allows validation of its operating status.
- **Message sending and receiving** – hiding the underlying transport medium (sockets, or a low level transport medium), this listens for, receives, parses and queues incoming messages.
- **Default message handling** – preventing lost messages. By default all messages have an incremental message number.
- **Broadcast messages** – some agents have the ability to “broadcast” messages to specific groups of agents.
- **Ontology management** – which agents are able to process which performatives.

The domain-specific agent-based virtual components implemented for virtual simulation are:

**Feeder agent.** This agent models the processes and events that occur in the feeding unit. The agent provides an interface that monitors the following events: (1) entry of an animal into a feeder (with date and time recording), (2) electronic identification of the animal; (3) recording of the weight of the trough at the time of entry; (4) departure of the animal from the feeder (with date and time recording), and (5) recording of the weight of the trough at the time of departure of the animal. The main control processes associated with these events are: (1) control of the level of feed in the hopper; (2) request for refilling of the hopper with specifically composited feed when the amount remaining is lower than a setpoint, (3) alarm management (lack of feed, failure of replacement orders) and (4) feed consumption recording.

**Feed mixer agent.** This agent models the production unit for feed whose composition is tailored to the needs of the animals described in Section 3.1. The processes modeled are: (1) receipt of the request for feed by a given feeder; (2) request to the formulator agent for the feed formula to produce; (3) receipt of the formula to be manufactured and subsequent activation of the actuators linked to each hopper and (5) order to activate the feed-to-feeder transport agent. The associated control processes are: (1) control of the basic feed level in each hopper; (2) request to feed refill when the quantity available is less than the setpoint; and (3) management of alarms (lack of food in the hoppers, failure of actuators or the transport system).

**Formulator agent.** This agent uses a mathematical model to determine the formulation of the feed requested by a feeder based on four premixes at the request of the feed mixer agent.

**Feed to feeder transport agent.** This agent models transport of the feed produced by the feed mixer agent to the feeder that has sent a request to refill the hopper.

4. Experiments and results for performance evaluation: a virtual prototyping simulation case study

The most important application of the simulation and prototyping virtual environment presented in this paper is probably its use as a tool for rapid modeling, validation and testing in the design and implementation of automatic precision feeding systems. The capacity to model actual systems helps reduce the development time and cost of new products and improve the quality of the design of automatic management and control systems, especially when complex functions are required. A key aspect during the development process is to have early information on the behavior, robustness and performance of the system under different operating conditions and workloads prior to its on-farm use.

In this section, we present the results of a set of experiments designed to evaluate the performance and verify the consistency of the design through virtual simulation of a prototype precision feeding system built using the agent-based virtual components described in the preceding section.

4.1. Composable simulation case

For this purpose, the virtual prototype of the automatic precision feeding system has been built by assembling a set of virtual components within a multiagent environment (see Section 3.2) that models the automatic precision feeding system in its entirety. The virtual components used to build the virtual model in this case are: 10 feeder agent components that simulate a set of 10 precision feeders installed in a farm with the capacity to feed approximately 200 growing pigs; a supervising agent that oversees the proper functioning of the multiagent system through an ongoing checking process, a formulator agent that determines the composition of the feed for each feeder; a feed mixer agent that simulates the process of pre-mix mixing to obtain the specific composition for each feeding unit; and a feed-to-feeder transport agent, which simulates the process of transporting food to each feeder agent.

The specific agent that simulates the occurrence of the main event in the automatic feeding system (animal entry and departure of the feeder) and that triggers the occurrence of the other events in the system has been included. Event generation frequency can be modified during the simulation process up to a maximum that depends on the processor speed of the computer that is running the simulation. With each event generated, a message is sent to the multi-agent environment in the XML format described in Section 2.3 with information on the occurrence of the event. This capability of increasing the event generation frequency enables two simultaneous effects: (1) it significantly shortens the time required to simulate processes that would have a duration of several days in real time and (2) it increases the workload of the virtual system, thereby making it possible to assess its behavior under different workload scenarios. This capability helps determine the critical points that might exist in the system, especially with a strong workload demand.

Due to their greater simplicity, the most common processing control systems on farms are the so-called centralized systems. These systems have a single controller that performs all the control actions. In precision feeding systems such as the one described in Section 3.1, the feeding units and other devices are specially distributed across different rooms and/or buildings, for which reason it may be advantageous to use an architecture design based on distributed control. Distributed control allows the integration of dif-
ferent devices connected via a data communication network to allow them to work cooperatively and concurrently to achieve harmonized functioning of the entire system. In order to assess the possible performance advantages of a distributed process control over an architecture with a centralized control process design, performance and robustness tests have been conducted under the two modes. For virtual simulation in the centralized control process mode, execution of the multiagent virtual environment has been carried out on a single PC, while two PCs with similar features have been used for the distributed control mode (AMD Athlon-3400+ processor with 64-bit architecture, 1 GB of RAM, internal clock at 2.2 GHz) with each running part of the multiagent system.

4.2. Performance and stress test experiments

As previously indicated, a key aspect in the final stages of the development process for the automation and control software is the ability to evaluate the behavior and performance of the system under different workload scenarios. Likewise, it is also of great interest to know the maximum workload that may render the control system incapable of responding in real time to the demands of the system and cause it to become unstable or erratic.

Load testing is the simplest means of evaluating performance. Load testing generally helps understand the behavior of the system under a specific forecast load. In our case, the load is generated by concurrent execution of the set of agents that make up the system and that generate a specific number of transactions in the form of messages that contain events, requests for tasks and responses between agents. The capacity for monitoring the virtual simulation environment provides a simple form of testing that can itself point to any bottleneck in the system's processing capacity. A drastically increase in the system's workload can be used as a stress test. This test is normally used to determine the upper limits of responsiveness of the system under an extreme workload and to determine the robustness of the application when subject to an extreme load. It also helps developers determine whether the application will have sufficient response capacity if the current load exceeds the forecast maximum.

The evaluation model used has been based on the capacity of the virtual simulation environment to optionally increase the workload in the system. Scenarios with increasing workloads have been used for behavior analysis. The same precision feeding process has been used for each scenario, which is equivalent to approximately one day duration in real time. In each of the tests presented in this section, the following parameters have been measured: (1) workload expressed as number of messages processed in the entire multi-agent system per unit of time; (2) absolute number of sent messages and messages received by the whole system during the simulation time and (3) percentage occupation of the CPU on the PC running the simulation in the centralized control mode or of the set of CPUs in the distributed control mode.

To analyze the behavior of the virtual model under different scenarios, we have used the metrics of “messages pending execution”, referenced as $\Delta M_p$. This metric factor is the difference between the absolute number of messages that have been sent by the agents ($M_s$) and the absolute number of messages received as responses to the foregoing ($M_r$) during the simulation time ($\Delta M_p = M_s - M_r$). Using these metrics, we have been able to analyze variations in the value of $\Delta M_p$ under different workloads. This analysis allows us to determine the workload at which the system maintains robust behavior ($\Delta M_p$ close to 0) and when its behavior becomes undesirable ($\Delta M_p$ significantly greater than 0), and it is rendered incapable of responding in real time to the requirements of the system itself. When $\Delta M_p$ stays close to 0, we can determine the throughput of the system by using the metrics of number of messages per second that flow through the communication network. In addition, the percentage occupation of the CPU of the PC on which the simulation is run provides us with information on when the hardware supporting the process does not have sufficient capacity to execute the workload demanded by the system in real time. In such a case, the value of $\Delta M_p$ will also increase and become $> 0$.

4.3. Experiment results

Different simulation scenarios have been conducted on the virtual precision feeding system described to analyze the behavior of the system under two different implementation architectures – one centralized and the other distributed. The results we are presenting here only analyze the behavior of the system from the point of view of its overall performance, i.e., the set of all virtual components being run concurrently. Based on this assumption, a comparative analysis has been conducted of the different tests by measuring the message flow originating from the entry and exit of animals at all feeders measured according to the metrics described above. These metrics have helped us determine, in each case, the load levels at which the system stops responding in real time. Moreover, use of these metrics by means of a test with dynamic workload has allowed us to evaluate the robustness of the system, especially when temporarily subjected to extreme workloads. Simulation of an increasing workload would be equivalent in reality to an increase in the number of feeders and animals on the farm.

![Fig. 6. Dynamic behavior of the precision feeding virtual prototype subjected to an increasing workload and executed under a centralized process architecture (1 CPU).](image-url)
The rate of occupation of the CPU reaches 100% and the number of messages not processed by the system increases over time.

(3) For load values over 50.1 messages/s, the rate of unprocessed messages \(\Delta M_p\) increases proportionally with the increase in workload.

Fig. 7 presents the results of the behavior of the virtual precision feeding system when temporarily subjected to load stress. During the simulation process, the workload has increased from a value of approximately 25 messages/s up to a maximum value of 112 messages/s at second 35 and then decreases until arriving at close to approximately 25 messages/s up to a maximum value of 112 messages/s at second 49 (at which time the load falls below the limit value). As of that instant, the system processes both the new events produced to-gether with those accumulated in the queues of the agents before dropping to the value of 0 at second 75 in the simulation. At that moment, the system returns to functioning in real time. This behavior demonstrates the significant robustness of the architecture proposed for the implementation of new control system for precision feeding systems. Taking into account that the maximal expected workload in a farm of 1000 animals is approximately of 8.2 messages/s, the proposed system design largely satisfy the requirements of the real system.

Fig. 8 presents the results of the behavior of the same virtual precision feeding system when executed under a distributed process architecture, specifically on two PCs (see Section 4.2) connected by means of an Ethernet network of 100 Mb/s. A group of 5 of the virtual feeder agent components is run on each PC for a total of 10 between the two (as in the experiment shown in Fig. 6). As in that experiment, the virtual distributed feeding system has been subjected to an increasing workload until it significantly exceeds the maximum load, at which point the system as a whole stops working in real time. Under a distributed process (2 PCs in this case), the workload capacity of the system significantly increases with respect to the centralized system, with the maximum workload situated at 102.1 messages/s, a value that almost doubles that of the centralized process mode. Thus, the execution in a distributed environment is clear advantageous from the point of view of increased performance of the whole system. This feature of the architecture also gives it a more flexible design capacity in keeping with the needs and requirements of real systems.

5. Conclusions

Real implementation of the concept of farm precision feeding requires the design of new and complex equipment. The implementation of these systems presents significant challenges relating to their complexity, reliability, cost effectiveness of the design process and its final validation. Virtual prototyping of precision feeding systems allows rapid verification of the design at low cost and provides immediate feedback. This also allows for the exploration of design alternatives that could lead to a higher performing final system. This article proposes an agent-based simulation framework facing those challenges through virtual prototyping used as a design method and as a stage prior to their actual use on the farm. The basic concept applied is to use virtual components that model the physical and logical components of a real system. Composable simulation is achieved by integrating the set of virtual components into a multi-agent system that models the farm automatic precision feeding as a whole. Evaluation of the designed system is obtained by simple means using the capacity of the simulation environment to modify the working conditions of the system (e.g., the workload), measure its performance and detect possible bottlenecks in the final system. A set of experiments conducted show that this approach is capable of simulating the functioning of actual precision feeding equipment and evaluating its dynamic behavior, performance and maximum load when subjected to different workload scenarios. The results show that the distributed control architecture has a workload capacity that is significantly greater than that of the capacity of the centralized system. This flexibility at the implementation stage has helped verify the advantages of exploring design alternatives and to achieve a more robust and higher-performing final system and as shown in this study, distributed control architecture can be more advantageous for implementing precision feeding equipments in large farms. The proposed agent-based simulation framework is an efficient tool for assisting in the process of design, implementation, validation and testing of automatic precision feeding systems.
Acknowledgements

This research has been partially funded by the Spanish Ministerio de Ciencia y Tecnología under Project PTR-0955. The authors thank Normande Ouellet from Agriculture and Agri-Food Canada, Sherbrooke, Québec, for her collaboration during software evaluation.

References


