Using LEL and scenarios to derive mathematical programming models. Application in a fresh tomato packing problem

Alejandra Garrido[a,b], Leandro Antonelli[a], Jonathan Martin[a], M.M.E. Alemany[c], *Josefa Mula[c]

[a] *LIFIA, Fac. de Informática, Univ. Nac. de La Plata, La Plata, Argentina*, garrido@lifia.info.unlp.edu.ar, lanto@lifia.info.unlp.edu.ar, jonathan.martin@lifia.info.unlp.edu.ar,
[b] *CONICET, Argentina*
[c] *Research Centre on Production Management and Engineering* (*CIGIP*), *Universitat Politècnica de València, Valencia, Spain,* mareva@cigip.upv.es,**Corresponding author:* fmula@cigip.upv.es

Abstract

Mathematical programming models are invaluable tools at decision making, assisting managers to uncover otherwise unattainable means to optimize their processes. However, the value they provide is only as good as their capacity to capture the process domain. This information can only be obtained from stakeholders, i.e., clients or users, who can hardly communicate the requirements clearly and completely. Besides, existing conceptual models of mathematical programming models are not standardized, nor is the process of deriving the mathematical programming model from the concept model, which remains ad hoc. In this paper, we propose an agile methodology to construct mathematical programming models based on two techniques from requirements engineering that have been proven effective at requirements elicitation: the language extended lexicon (LEL) and scenarios. Using the pair of LEL + scenarios allows to create a conceptual model that is clear and complete enough to derive a mathematical programming model that effectively captures the business domain. We also define an ontology to describe the pair LEL + scenarios, which has been implemented with a semantic mediawiki and allows the collaborative construction of the conceptual model and the semi-automatic derivation of mathematical programming model elements. The process is applied and validated in a known fresh tomato packing optimization problem. This proposal can be of high relevance for the development and implementation of mathematical programming models for optimizing agriculture and supply chain management related processes in order to fill the current gap between mathematical programming models in the theory and the practice.

Keywords: Language extended lexicon (LEL); scenarios; software engineering; mathematical programming; fresh tomato packing.

1. Introduction

There is an increasing interest in mathematical programming models for optimal decision support applications (Dominguez-Ballesteros et al., 2002). Indeed, the development of optimization and decision support tools is needed to obtain all the benefits of transactional information technology (IT), improving the economic performance and customer satisfaction of supply chains (Grossmann, 2005). Along these lines, mathematical programming models have been demonstrated to be powerful optimization tools to support decision makers in many supply chain processes such as: production planning (Alemany et al., 2013), order promising (Alemany et al., 2018; Grillo et al., 2017), shortage planning (Esteso et al., 2018), supply chain production and transport planning (Mula et al., 2010), among others. The agriculture sector also faces many complex problems for optimization (Saranya & Amudha, 2017) as it has been reported in some recent works (Cid-Garcia & Ibarra-Rojas, 2019; Grillo et al., 2017; Liu et al., 2019). Some revisions about mathematical programming models applied to different problems in agriculture can be found for supply chain design (Esteso, Alemany, & Ortiz, 2018), fresh fuit supply chain management (Soto-Silva et al., 2016), agribusiness supply chain risk management (Behzadi et al., 2018) and crop planning (Jain et al., 2018), among others.

Once formulated, mathematical programming models are often implemented as part of a decision support system (DSS), which is thus called model-driven DSS. We refer readers to the work of Udias et al. (2018) regarding an example of recent agricultural model-based DSS. These types of DSSs allow the user to make what-if analysis and define different scenarios without the need to understand the complexities of mathematical programming models (Mundi et al., 2013). Mir et al. (2015) provide an extensive revision of DSS application in agriculture noting their main weaknesses, most of them related to the poor involvement of stakeholders in the DSS construction process, which our methodology aims to overcome: failure to support stakeholder participation before and after development stages, failure to support the relationship between stakeholders and experts/developers, low adaptation, complexity with user inputs and under-definition of end users.

Moreover, the construction of mathematical programming models is a complex and very time-consuming process, which requires an expert to acquire a deep understanding of the modelling domain, context of use, decision making activity, and to learn the complete set of constraints from the problem under study. All this problem knowledge should be acquired before the model is constructed, because any change that occurs afterwards might imply a whole redesign and implementation of the model. For this reason, mathematical programming model construction should be preceded by a conceptual modelling activity whose natural use in the field of applied mathematics has been pointed out by Lesh (1981) and Lesh et al. (1983).

During conceptual modelling, different tools are used, like Bizagi, BPWin, iGrafx, Process Modeler, System Architect and Visio, to help understand the domain, the flow of data, products, decisions and the interaction among parties, and to elicit requirements as completely as necessary (Armengol et al., 2015; Giannoccaro & Pontrandolfo, 2001; Hernández et al., 2008; Mula et al., 2006; Pérez Perales et al., 2012). However, there has been no consensus or standardization in this regard.

The process of creating a conceptual model may be closely compared with the process of requirements elicitation for any software system. In particular, we claim that a very important aspect of conceptual models is that they should allow their iterative, incremental and collaborative construction. Indeed, agile methods in requirements engineering have demonstrated the importance of managing the inherent complexity of a system specification in an incremental and iterative manner (Schön et al., 2017). A relevant technique to elicit requirements and get a clear and complete understanding of a domain is the use of scenarios (Leite et al., 2000). The description of scenarios ranges from visual (storyboards) to narrative (structured text) (Young, 2004). They are constructed iteratively on the basis of a universe of discourse (UofD), i.e., a domain's vocabulary or lexicon. Leite and Franco (1993) named it the language extended lexicon (LEL). It is a meta-language used to gather or elicit requirements, which aims at describing the meaning of words and phrases specific to a given application domain. It has three convenient characteristics in the context of analytical modelling: easy to learn (Cysneiros & Leite, 2001), easy to use (Gil et al., 2000) and good expressiveness (Kaplan et al., 2000). Moreover, there exist specific rules to derive LEL elements into scenario elements, and scenarios retrofit LEL's vocabulary in a very incremental and iterative construction process (Leite et al., 2000). Here, we claim that the above quality characteristics and the construction process of the pair LEL + scenarios make them an adequate conceptual model from where mathematical programming models could be systematically derived.

In this article, we propose a novel methodology to guide the derivation of mathematical programming model elements from a conceptual model created with LEL + scenarios. Our derivation proposal consists of several rules that map conceptual model elements (either LEL vocabulary items or scenario elements previously derived from LEL) into mathematical programming model elements. The added benefit of this methodology is that it provides traceability from vocabulary and requirements specification to each mathematical programming model element. This traceability becomes very important when a change is necessary in the conceptual model, to know the particular place in the mathematical programming model specification where the change will have an impact.

In order to give further support to the derivation process, we relate conceptual model creation to a knowledge building process of collective creation between stakeholders and analysts. This process emphasizes the production and continuous improvement of knowledge parts (Moskaliuk et al., 2009), and it is usually supported with a web-based knowledge building community like a mediawiki (Baraniuk et al., 2004). Thus, we have constructed a semantic mediawiki based on an ontology for the collaborative creation of the conceptual model based on LEL and scenarios. Moreover, the semantic mediawiki is used to semi-automatically derive mathematical programming models.

Thus, the paper proposes a novel methodology that connects the areas of requirements engineering and agile methods with conceptual modelling in order to create mathematical programming models that capture the agriculture research domain more effectively and completely. Summarizing, the main contributions of this paper are: (i) a proposal for utilizing the pair LEL + scenarios to create conceptual models that gather the vocabulary of decision makers and specify the domain knowledge necessary to build a mathematical programming model; (ii) a rule-based methodology for the systematic derivation of mathematical programming models from the conceptual model generated by LEL + scenarios; (iii) a knowledge model designed over an ontology based on the description of LEL + scenarios; and (iv) an extensible tool consisting of a semantic mediawiki that allows to partially automate the mathematical modelling derivation. To the best of our knowledge, the methodology proposed in this article is the first approach to standardize the development of a conceptual model and its consequent translation to a mathematical programming model, and no other requirement elicitation method in the field of software engineering has been adapted for the derivation of mathematical programming models. Moreover, the derivation process provides traceability from requirements to mathematical programming model elements to deal with changes more effectively.

The rest of the paper is organized as follows. Section 2 provides a literature review on mathematical programming models, conceptual models, requirements elicitation, LEL and scenarios and knowledge building. Section 3 describes our methodology for the systematic derivation of mathematical programming model elements from the conceptual model generated by LEL+ scenarios. Section 4 applies the method to derive a linear mathematical programming model for fresh tomato packing. Section 5 defines an ontology for LEL + scenarios on which we based the construction of a semantic mediawiki for the semi-automatic derivation of the mathematical programming model. Section 6 provides conclusions and future research directions.

2. Literature review

2.1 Mathematical programming models

Mathematical programming involves finding the values of some variables that, subject to certain constraints, maximize or minimize an objective function. We assume that deterministic mathematical programming models have a generic structure: a definition part and a modelling part (Pérez et al., 2010; Shapiro, 1993). Table 1 provides a description of mathematical programming model parts.

2.2 Conceptual mathematical modelling

Siau (2004) defines conceptual modeling as the process of formally documenting a problem domain to achieve understanding and communication between the different participants. Developing conceptual models means specifying the essential objects, or components, of the system to be studied, and the relationships or types of exchanges between the objects that affect the functioning of the system (Lezoche et al., 2012). From the abstraction of conceptual models emerges the concept of reference models, which are generic conceptual models that formalize recommended practices for a given domain (Pesic & van der Aalst, 2005).

Table 1. Mathematical programming model parts.

MATHEMATICAL PROGRAMMING MODEL PART	ELEMENT	DESCRIPTION	EXAMPLES
Definition part	Indexes	Objects or concepts of the model. The number of elements of a class of objects provides the number of instances of this class	Machines (*m*) Products (*i*)
	Sets	Group of instances of one or several indexes that meet certain characteristics or constraints	Group of products that can be processed by each machine $P(m)$
	Parameters	Known characteristics of one or several elements (indexes) over which is not possible to act	Capacity of each machine (Cap_m) Production cost for each product on each machine (PC_{im})
	Decision variables	Unknown characteristics of one or several elements (indexes) over which it is possible to act (decision-maker can determine their value)	Quantity to be produced of each product on each machine every time period (X_{imt})
Modelling part	Objective function	Goal/s to be optimized (minimize or maximize)	Maximize profits Minimize costs Minimize time
	Constraints	Problem limitations that should be respected for every combination of the decision variables	Availability of resources (e.g. machines' capacity) Company policies (e.g. service level) Logic or implicit constraints (e.g. flow balance, positive quantities)
Decision-maker	Decision-maker	Person who makes the decision	Planner, manager
Temporal characteristics	Time-horizon	The length of time (with a beginning and end date) over which a problem is optimized	A year, six months, etc.
	Time-period	Space of time into which a time-horizon is divided	Seconds, minutes, hours, days, weeks, months or years
	Replanning time period	Space of time in which the plan is calculated again	Seconds, minutes, hours, days, weeks, months or years

In the field of conceptual mathematical modelling, Schneeweiss (2003a) identifies different classes of distributed decision making (DDM) problems in supply chain management. The same author derives the coupling equations for the most usual cases in coordinating the supply chain (Schneeweiss, 2003b). However, the coupling equations are still of a very general, almost verbal character. In this context, Alemany et al. (2007) propose a reference mathematical programming model for collaborative planning that addresses two of the challenges of DDM: the spatial and temporal interdependencies. Alemany et al. (2011) developed an application to support the integrated modelling and execution of the supply chain collaborative planning process made up of several decisional centers which make decisions based on mathematical programming models. However, the formulation of the own specific decisional characteristics of each decision center (micro-decision view) mainly relies on the ability of the mathematical programming modeler. Moreover, Pérez-Perales et al. (2012) propose a framework to support modelling the decisional view of collaborative planning through mathematical programming models.

While the above studies are very useful, this article provides further tool-supported guidence and a precise specification in terms of derivation rules to derive mathematical programming models from a conceptual model, as opposed to general descriptions or relying in the modeler's ability.

2.3 Process of mathematical programming model formulation

Since conceptual models are not standardized, neither is the process of deriving the mathematical programming model from the concept model, which remains ad hoc. In this sense, Raghunathan (1996) proposed a methodology to design a DSS with its underlying mathematical programming and data models. The methodology includes six steps: (i) problem domain analysis, (ii) database design, (iii) modelbase design, (iv) database/modelbase integration design, (v) problem/decision maker characteristics and (vi) specific DSS design. However, the setting of Raghunathan (1996) is a classroom, so the problem statement is completely specified from the start. Alternatively, our proposal is inspired in the current, agile way of system specification, which recognizes that the construction of any model should be iterative and incremental. Additionally, the use of entity-relationship modelling by Raghunathan (1996) has two implications: a) the problem must be simple, otherwise the diagram is not even readable; and b) stakeholders may not be able to understand it. Contrarily, we use a natural language and a semi-structured scenario specification for the conceptual model, which is more appealing for a system specification that incorporates stakeholders in the process.

Furthermore, Dominguez-Ballesteros et al. (2002) define different stages in the process of deterministic and stochastic linear programming model formulation and implementation: conceptualisation (data collection and study of the problem), data modelling (categorisation and abstraction of the data), algebraic form (modeller's form), translation (matrix generator/modelling language), machine-readable form (algorithm's form), solution and solution analysis. However, the stages can be understood as guidelines for the mathematical programming modeler more than a derivation procedure. That is, unlike Raghunathan (1996) and to the best of our knowledge, no methodology or structured modeling language is proposed for the conceptualization stage.

2.4 Requirements elicitation

Requirements elicitation is the process that analysts follow to ensure a correct understanding of stakeholders' needs and the domain specification before a system is designed and implemented (Leite et al., 2000). In this regard, Geisser and Hildenbrand (2006) state that software requirements are very complex and a multitude of stakeholders participate in their description (Geisser & Hildenbrand, 2006). They propose a method called CoREA that covers collaborative requirements elicitation in a distributed environment as well as quantitative decision support for distributed requirements prioritization and selection. Our proposed approach also relies on collaborative knowledge acquisition and description, with the added advantage of using this knowledge base as a conceptual model from where a mathematical programming model can be derived through the application of a set of rules.

Closer to our work, Laporti et al. (2009) propose an approach to develop system requirements in an iterative and collaborative way. Experts in the domain collaborate to build narrative descriptions of stories. Then, these stories are used as input to describe scenarios, which are in turn used to define use cases. In this regard, our proposal is similar since it considers the collective construction of LEL + scenarios and a mapping between LEL, scenarios and mathematical programming models. The difference is that the output of the transformation in the work by Laporti et al. (2009) is still a textual, semi-structured representation (use cases), which does not distance much of the previous products, whereas in our case the output is a structured mathematical programming model, so the mapping requires a more complex strategy that includes a precise representation of the relations among model elements. Our approach uses two existing techniques for requirements elicitation: the LEL (Leite & Franco, 1993) and scenarios (Leite et al., 2000). The LEL is a very convenient tool for both stakeholders with no technical skills and analysts, since it conforms to the mechanism used by the human brain to organize knowledge (Oliveira et al., 2007), which makes it easy to learn while having good expressiveness. The process to build the LEL is comprised of six steps (Breitman & Leite, 2003; Kaplan et al., 2000), which allow constructing a list of terms classified in four categories (see Table 2).

Turning into scenarios, they can be used in different stages of software development, from clarifying business processes and describing requirements to providing the basis of acceptance tests (Alexander & Maiden, 2004). Leite et al. (2000) propose a template with six elements to describe scenarios (see Table 2), which are derived from the LEL following a methodology consisting of five steps: (i) to identify main and secondary actors, i.e., LEL symbols that belong to the subject type; (ii) to identify scenarios within the behavioral responses of symbols chosen as actors; (iii) to define the scenario goal based on the notion of the verb symbol in which the scenario is based; (iv) to identify the scenario resources, searching in the notion of the verb that created the scenario, for LEL symbols of the object category; and (v) to derive episodes from each behavioral response of the verb that identified the scenario.

Table 2. LEL categories and scenarios elements.

Category	Characteristics	Notion	Behavioral responses
Subject	Active elements which perform actions	Characteristics or condition that subject satisfies	Actions that subject performs

Object	Passive elements on which subjects perform actions	Characteristics or attributes that object has	Actions that are performed on object
Verb	Actions that subjects perform on objects	Goal that verb pursues	Steps needed to complete the action
State	Situations in which subjects and objects can be located	Situation represented	Actions that must be performed to change into another state

Attribute	**Description**
Title	Name that describes the scenario
Goal	Conditions and restrictions to be reached after the execution of the scenario
Context	Conditions and restrictions that are satisfied and constitute the starting point of the scenario execution.
Actors	Agents that perform actions during the scenario starting from the context to reach the goal
Resources	Products and elements used by the actors to perform actions
Episodes	Steps executed by the actors using the resources starting at the context to reach the goal

2.5 Ontologies and knowledge building

Ontologies define the common vocabulary in which shared knowledge from a domain of discourse is represented (Gruber, 1993; 1995). They can be constructed in two ways, domain dependent and generic. CYC (Lenat, 1995), WordNet (G.A. Miller, 1995) and Sensus (Swartout et al., 1996) are examples of generic ontologies. A benefit of using a domain ontology is to attain the shared and agreed definition of a semantic model of domain data and the links between different types of semantic knowledge, which makes it suitable in formulating data searching strategies for information retrieval (Munir & Sheraz Anjum, 2018).

Furthermore, a semantic mediawiki defined over an ontology provides a web-based support for a knowledge building community (Baraniuk et al., 2004). We have used a semantic mediawiki in this work to allow for the collaborative definition of LEL + scenarios of the problem domain and for the semi-automatic derivation of the mathematical programming model using the mediawiki's query engine.

3. Methodology for mathematical programming model derivation

3.1. Conceptual model construction and methodology overview

The mathematical programming model derivation process starts from an existing conceptual model consisting of a complete or close to complete specification of LEL + scenarios of the system. There are three variations that we propose to the original definition of LEL and scenarios for the specific goal of generating mathematical programming models. The first is that we do not use the terms in the "state" category of the LEL, because no mathematical programming model element is derived from them. The second is that we distinguish attributes inside the notion of symbols, especially those that become scenario's actors and resources. That is, an actor is a LEL subject, and as such it will have a notion with its conceptual definition. We call attributes to the terms that characterize the actors and appear in their notion, usually after the verb "has". Similarly, a resource is a LEL object with a notion that names its attributes. In turn, attributes are also defined as LEL objects, and this is the reason that attributes are underlined in the notion of the actor or resource that they characterize, describing a relation between LEL terms. The third variation that we propose is related to specifying the temporal location inside scenarios' context element with more detail, identifying three fields: time horizon, time period and replanning time period.

In order to provide a better understanding of the proposed methodology, Figure 1 depicts its two main stages (Conceptual model construction and Mathematical programming model construction), the phases in the construction process for each stage, and two different levels of iteration. There is one iteration cycle that occurs often in the construction of the conceptual model, where scenarios may retrofit the LEL, and a second level or global iteration cycle, between the conceptual model and the mathematical programming model, which should not be as usual. The methodology provides traceability by way of rules that specify the source of each mathematical programming model element. Therefore, this methodology is robust enough to actually afford changes in the mathematical programming model construction.

The input for the whole process, as Figure 1 shows, is the domain knowledge obtained from stakeholders and documents. The first phase consists in the LEL specification, which is created by a system analyst together with the stakeholders and, if possible, the expert in mathematical modelling, thus creating a multi-disciplinary team. They should identify the sources of knowledge, define the LEL, verify and validate it. The second phase consists in specifying scenarios using the knowledge captured in the LEL. If during the description of the scenario, it is noticed that more knowledge from the domain is needed, the process goes back to phase 1.

When the knowledge captured by the LEL + scenarios appears adequate and complete, the third phase (Mathematical Programming Model Derivation) begins. This phase uses the knowledge captured in LEL and scenarios to derive the mathematical model. The mathematical programming derivation proposal consists of several rules that map conceptual model elements (either LEL vocabulary items or scenario elements previously derived from LEL) into mathematical programming model elements. The derivation process may be carried out manually by the mathematical programming expert, possibly together with the system analyst. In addition, we provide tool support for a semi-automatic derivation through a semantic mediawiki. Even with a tool support, manual revision from the mathematical programming expert will be necessary, since it is not possible to automatically create the equations that model constraints from a textual description, although we can isolate the sentence that contains a constraint from the conceptual model. In Fig. 1, the numbering of steps in the mathematical programming model derivation phase denotes a sequence in which rules should be applied. Moreover, at the end of each of these steps, the manual intervention of the mathematical programming expert is advised to prevent an overly complex mathematical programming model (as it could happen with a large number of indexes) or to spot missing items in the conceptual model. Furthermore, it could also be detected that more knowledge from the LEL and Scenarios is needed, and it that case, the process goes back to the conceptual model construction.

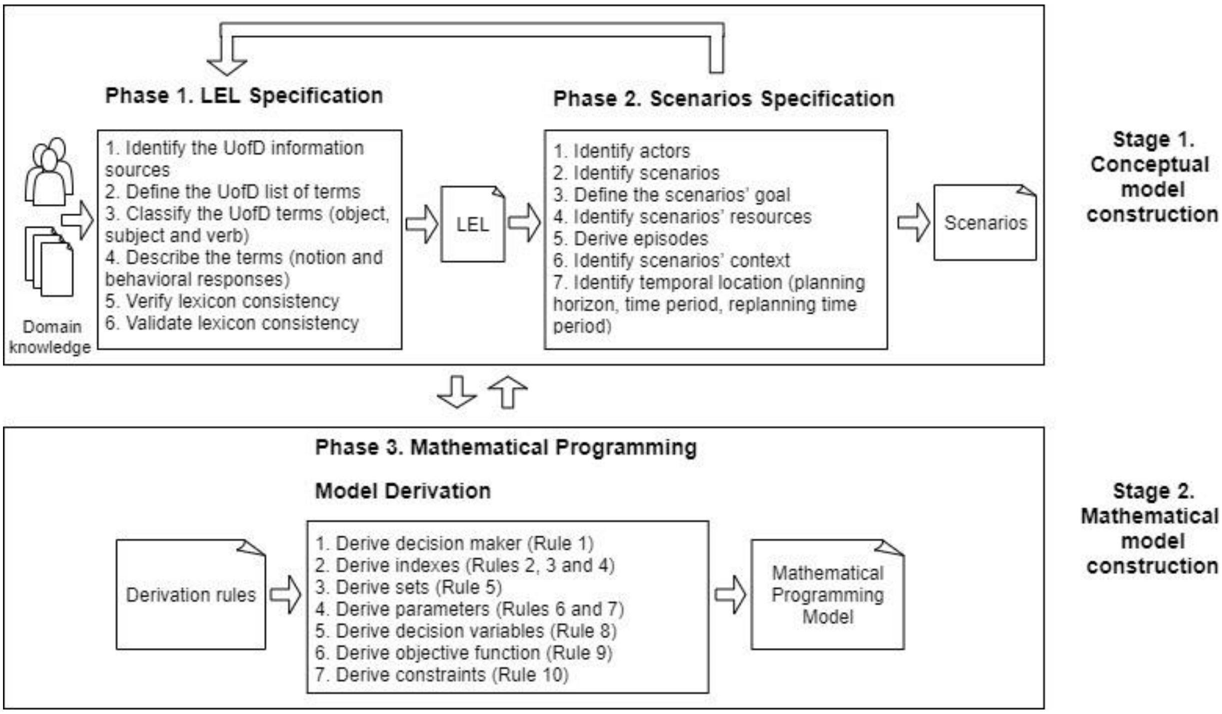


Fig. 1. Methodology for mathematical programming model derivation.

3.2. Mathematical programming model derivation

This section presents the rules that allow deriving a mathematical programming model from a conceptual model composed of LEL + scenarios. Below we present a detailed description of each derivation rule, listed by its rule number. Note that rule numbers do not dictate an order of application except for the order dictated by the methodology and outlined in Figure 1. Following this description, Table 3 provides a summary of the rules.

Rule 1. The main actor of the base scenario becomes the decision maker in the mathematical programming model

Main actors are those who execute actions in the **domain**, in this case, those making decisions. We consider the main actors to be a single person or several persons playing the same role, i.e., making a centralized decision. There will be a base scenario that derives from the behavioral response of the single main actor, who will be the decision maker in the mathematical programming model.

Rule 2. The time period of the planning horizon defined in the temporal location, inside the context of the base scenario, becomes an index of the mathematical programming model. Moreover, other data objects related to time in the temporal location could also become indexes.

The base scenario should specify the temporal location in its context attribute. Particularly, the time period specifies the regular intervals in the time horizon at which different decisions are to be made. If such time period exists, Rule 2 is applied to derive an index from it. Other LEL objects specifying time considerations (shipping day or maturity day, among others) could also appear in the temporal location and probably become indexes.

Rule 3. Scenarios' actors that have multiple instances become *indexes* of the mathematical programming model

Actors in scenarios are derived from LEL subjects. A subject in LEL could denote a specific person or a role. If it is a role, which is filled by several persons, there should be an index in the mathematical programming model to represent them. If the cardinality of a particular actor is likely to grow from 1 into several people, the mathematical programming expert could decide to include the index to make the model more flexible to accommodate this change in the near future.

Rule 4. Scenarios' resources that have multiple instances become indexes of the mathematical programming model

Resources represent relevant physical elements or information used by scenarios' actors to achieve their goal. Resources derive from LEL's objects. Objects can be singletons (a single instance) or denote a class of elements. When an object denotes a class, it becomes an index of the mathematical programming model. Similar to Rule 3, if the object could grow into a class in the future, the mathematical programming expert could decide to include the index.

Rule 5. Actors and/or resources which are related by a notion in the LEL become *sets* in the mathematical programming model when their relation denotes a restriction

Two or more actors and/or resources are related when they appear in the same notion. In the case where the relation among them is restricted for some cases, this restriction should be defined as a set. Conversely, if the relation is many-to-many, it would not be necessary to define the sets.

Rule 6. The number of instances of the indexes could become parameters in the mathematical programming model

An actor that becomes an index derives from a LEL subject with multiple instances. The number of instances of an index is known, and therefore, it could become a parameter of the mathematical programming model, although this is not always the case. The same occurs with resources and temporal data objects. An example is the number of instances of the time period, which matches the decision time horizon and could be defined as a parameter.

Rule 7. Attributes of scenarios' actors and attributes of scenarios' resources and attributes of their relationship with known values become parameters of the mathematical programming model. Each parameter is indexed by those indexes related to it by the same notion and for which its value remains known

In the case of attributes that did not become indexes by Rule 4 and denote a known value, by this rule they become parameters of the mathematical programming model. Moreover, a parameter derived by this rule should be indexed by the indexes that are related to it. We define two LEL terms as related if they appear in the same notion, i.e., either one of them appears in the notion of the other, or both terms appear in the notion of a third term. These indexes could include other actors but also other resources. However, note that not all indexes that appear in the notion will be assigned to the parameter, only those that refer to a known value. Thus, the mathematical programming expert should determine which subset of indexes refers to a known value, and the parameter should be indexed only by that subset of indexes. The notion gives the whole subset, but the expert decides what indexes should be used.

Rule 8. Attributes of scenarios' actors and attributes of scenarios' resources and attributes of their relationship that have unknown values become decision variables of the mathematical programming model. Each decision variable is indexed by those indexes related to it by the same notion and for which its value is unknown

This rule is similar to Rule 7 but for unknown values. That is, attributes that appear in the notion of actors or resources, which values should be assigned in the decision process, become decision variables. Moreover, decision variables should take the indexes that are related to it by the same notion and refer to unknown values. It could be necessary to define artificial decision variables (without economic/physical interpretation) in order to mathematically represent a reality or to force some logical constraint.

Rule 9. The goal of the base scenario contains the objective function

The base scenario should specify in its goal attribute, the purpose of the main actor (which becomes the decision maker by Rule 1) in executing the scenario. This goal is specified as a complex sentence with a relative clause that starts with "so as to" followed by the verb "minimize" or "maximize". From this verb to the end, this relative clause becomes the objective function. Moreover, the expert may look for further details of each objective function in the notion of the LEL symbols involved in the goal.

Rule 10. The set of context sentences of all scenarios become the set of constraints of the mathematical programming model

Scenarios should specify in its context attribute, the conditions to comply. These conditions are described in natural language, as sentences that relate to the actors and resources of each scenario.

Experts in mathematical programming modeling should use the restrictions described in the scenarios' context and relate them to parameters and indexes previously defined by other rules to derive the set of mathematical programming model constraints. These constraints usually contain logical constraints that represent business rules. These rules should appear during the requirements elicitation that a system analyst carries out to construct the LEL, and therefore they would also be derived as part of the scenarios' context. Additionally, there are added other artificial constraints (for instance, a positive boundary to variables) in order to avoid erroneous results. These constraints will not generally appear in the conceptual model and they should be added by a subsequent analysis of the mathematical programming expert.

Table 4 summarizes the relationship among LEL, scenarios and mathematical programming model elements. Rules are grouped by the mathematical programming model element that is derived from them. The second and third columns use indentation to represent nested concepts (for example, in Rule #2, the scenario element that generates an "Index" is the "Time period", inside the "Temporal location", which is in the "Context" of the "Base scenario").

Table 3. Equivalences among LEL, scenario and mathematical programming model elements.

Rule Number	**LEL model**	**Scenario model**	**Mathematical model**
1	Subject	Base scenario Main actor	Decision Maker
2	Object	Base scenario Context Temporal location Time period	Index
3	Subject (multiple instances or single instance--opt)	Any scenario Actor	Index
4	Object (multiple instances or single instance--opt)	Any scenario Resource	Index
5	Subject / Object Notion (restriction)	Any scenario Actor / Resource	Set
6	Subject / Object Notion (no. of instances)	Any scenario Actor / Resource (index)	Parameter
7	Subject / Object Attribute (known value)	Any scenario Actor / Resource	Parameter

		Attribute	
8	Subject / Object Attribute (unknown value)	Any scenario Actor / Resource Attribute	Decision Variable
9	Subject Behavioral response (verb)	Base scenario Goal	Objective function
10	Object	Any context	Constraint

4. Application

This section applies the above methodology to the problem of fresh tomato packing addressed by Miller et al. (1997). The purpose of using an existing problem is to contrast the result of our derivation rules with a real, published mathematical programming model while avoiding the long description that a new mathematical program would require.

4.1 Conceptual model of the tomato packing problem: LEL + scenarios

This section presents the conceptual model for the tomato packing problem created in terms of LEL + scenarios. The team assembled for this task was multidisciplinary, that is, composed of system analysts and a mathematical programming expert. The information sources of the UofD to construct the LEL were provided by the article from Miller et al. (1997), interviews with local tomato producers that played the role of customers and other documentation sources online. After three iterations, the team arrived at the LEL that appears in Table 4.

Table 4. LEL of the tomato packing problem.

Term	Role	Notion
Packinghouse management (PM)	Subject	Conducts the business of the packinghouse. The PM decides when to harvest tomatoes matured in the present or previous cycle, communicates its decision to growers and packs the harvested tomatoes to fulfil the market demand
Grower	Subject	Person responsible for a tomato field (has been assigned a certain number of acres of tomatoes), including harvesting the tomatoes, for which it has some harvest capacity. There are several growers. Each grower produces a certain yield of bins of tomatoes per acre to take them to the packinghouse
Market	Subject	Customers of the packinghouse, who buy tomatoes to sell them. The market has a market demand in number of boxes of tomatoes they would like to buy each day
Acres of tomatoes	Object	Land assigned to a grower with tomatoes that get matured on a certain maturity day, and are harvested on a certain harvesting day. Acres may have a fraction of "vine ripe tomatoes" that are sold "as is"
Tomato	Object	Produce planted by growers on the acres of their fields. Tomatoes that have not been harvested in 2 cycles generate a cost of damaged tomato
Harvesting day	Object	Day in which tomatoes already matured are harvested. Every day in the horizon may be a harvesting day
Maturity day	Object	Day in a period in which tomatoes in an acre get ready to be harvested. The acres of a specific grower may have different maturity days within the decision horizon

Decision horizon	Object	Largest time in which the readiness of tomato fields for harvest can be accurately predicted. It is 3 days
Harvest capacity	Object	The capacity of a grower to harvest during a certain day
Bin	Object	Container where the grower places the harvested tomatoes
Cost of damaged tomato	Object	Penalty cost due to dissatisfaction of a grower because of delayed harvest of fields matured in the previous and present cycles in $ per bin
Fraction of "vine ripe tomatoes" (v.r.t.)	Object	Tomatoes sold without gassing
Yield of bins of tomatoes per acre	Object	The number of bins of harvested tomatoes per acre of a certain grower
Packinghouse	Object	Place where tomatoes are packed in boxes and stored. It has a packing capacity, and a gassing capacity per day. The packinghouse requires some fraction of hour needed to pack a bin at a certain packing cost. The packinghouse generates a fraction of tomatoes ready after gassing. The packinghouse works on regular hours (9 to 5) and overtime hours (after 5pm) to generate an inventory level at the end of the day
Packing cost	Object	The total packing cost consist of: (1) cost of damaged tomatoes; (2) inventory holding cost, (3) shortfall cost, (4) overtime and (5) regular hour packing costs
Packing capacity	Object	No more tomatoes may be packed once the packinghouse reaches the packing capacity
Gassing capacity	Object	Capacity of the packinghouse to gass tomatoes in a harvesting day. It is measured in number of boxes of tomatoes
Fraction of hour needed to pack a bin	Object	Time required for packing 1 bin of harvested tomatoes and put them in boxes
Fraction of tomatoes ready after gassing	Object	Tomatoes ready after a gassing session
Regular packing hours	Object	Hours when the packinghouse is operating on a day, that generates a regular packing cost. It goes from 9 am to 5 pm
Overtime packing hours	Object	Extra hours required to complete the packing at the packinghouse on a day. They generate a higher cost than the regular packing hours. Overtime hours are after 5pm
Regular packing cost	Object	Cost of packinghouse operation in $ per hour during regular packing hours
Overtime packing cost	Object	Cost of packinghouse operation in $ per hour during overtime packing hours
Inventory level	Object	Number of boxes of tomatoes of the packinghouse at the end of a certain day. It has a certain inventory holding cost in box/day
Inventory holding cost	Object	Cost of packinghouse storage in boxes/day
Market demand	Object	Number of boxes of tomatoes that the packinghouse customers require in each harvesting day. If the market demand is not covered it generates a shortfall
Shortfall	Object	Number of missing boxes of tomatoes needed to reach the market demand on a certain harvesting day
Shortfall cost	Object	Cost of being short on satisfying the market demand in $ per box short
Box	Object	Container where tomatoes are placed during packing
Harvest	Verb	To cut the tomatoes that are matured
Pack	Verb	Put in boxes the harvested tomatoes. Packing is done on regular hours at a certain cost or overtime hours to complete packing all harvested tomatoes
Gass	Verb	Technique used to ripen tomatoes that are not completely matured by exposing them to ethylene gas

The first version of the scenarios was created after the first iteration of the LEL and was used to retrofit the second iteration of the LEL.

The following four tables describe the scenarios of the tomato packing problem. First, Table 5 presents the sole Level 0 scenario, called *base scenario*. Then, Tables 6 through 8 show the three scenarios of Level 1, which derive from episodes of the base scenario.

Table 5. Scenario Level 0 of the tomato packing problem.

Scenario 0:	**Plan the harvest and the packing of fresh tomatoes**	
Goal	Make a plan at the beginning of the present cycle to harvest and pack tomatoes matured on the present and last cycles, so as to minimize the total packing cost	
Actors	**Main actor**	Packinghouse management (PM)
	Secondary actors	Growers, market
Resources	**Physical resources**	Packinghouse; acres of matured tomatoes for each grower in the present and past cycles; tomatoes; bins; boxes.
	Information resources	Market demand in number of boxes for each day in the present cycle; for each grower, the number of acres of matured tomatoes in the present and past cycles and the number of bins of tomatoes generated after harvesting.
Context	- Not all tomatoes that get matured in a cycle are harvested in the same cycle. - Tomatoes harvested in the next cycle after they get matured may be sold but have less quality. - Tomatoes not harvested in the next cycle after they get matured must be discarded. - A grower may only harvest up to their capacity. - The packinghouse may only pack and gass a limited number of tomato boxes a day. - Temporal location: • Decision horizon: 3 days • Decision period: harvesting day; every day in a decision horizon. • Other temporal variables: maturity day; any day in the decision horizon.	
Episodes	- PM makes a plan with the harvesting day of matured tomatoes, for each day in the present cycle, in number of acres for each grower. - Each grower harvests the amount decided and communicated by the PM. - Each grower takes the harvested tomatoes in bins to the packinghouse. - PM packs the harvested tomatoes in the packinghouse, labelling some boxes as "vine-ripe tomatoes". - PM gasses all boxes of tomatoes which are not labelled as "vine ripe". - Tomato boxes are shipped to cover the market demand, and the surplus remains as inventory of the packinghouse.	

Table 6. Scenario 1.1 at Level 1 of the tomato packing problem.

Scenario 1.1:	**Plan the the harvesting day of matured tomatoes**	
Goal	Decide how many acres of matured tomatoes to harvest for each grower in each day of the present cycle	
Actors	**Main actor**	Packinghouse management (PM)
Resources	**Physical resources**	Acres of tomatoes.
	Information resources	For each grower: harvest capacity, acres of matured tomatoes for each day and yields of bins of tomatoes per acre. Also: cost of tomatoes damaged due to delayed harvest, packing and gassing capacity of the packing house.
Context	- A grower may only harvest in a day up to their harvest capacity. - The number of bins of harvested tomatoes to pack from all growers should be less than the available gassing capacity of the packinghouse for that day. - The number of bins of harvested tomatoes to be packed per day should be less than the combined regular and overtime packing capacity of the packinghouse.	
Episodes	- Considering the information resources available, the PM calculates, for each day in the present cycle and for each grower, the number of acres to harvest of tomatoes matured of the previous and the present cycles.	

Table 7. Scenario 1.2 at Level 1 of the tomato packing problem.

Scenario 1.2:	**Harvest**	
Goal	Harvest the tomatoes and take them to the packinghouse.	
Actors	**Main actor**	Grower
Resources	**Physical resources**	Acres of tomatoes; tomatoes; bins; packinghouse.
	Information resources	Harvest capacity; acres of matured tomatoes for each day; number of acres to harvest each day
Context	- Tomatoes to be harvested are matured.	
Episodes	- The grower cuts the tomatoes from the acres already matured that the PM decided to cut on the present day. - The grower places the tomatoes in bins. - The grower takes the bins to the packinghouse.	

Table 8. Scenario 1.3 at Level 1 of the tomato packing problem.

Scenario 1.3:	**Pack the harvested tomatoes**	
Goal	Put the tomatoes in boxes to be transported.	
Actors	**Main actors**	Packinghouse personnel
Resources	**Physical resources**	Bins of tomatoes; boxes; packinghouse.
	Information resources	Gassing capacity; regular packing hours; regular packing hour cost; overtime packing hours; overtime packing hour cost
Context	- Packing occurs during regular packing hour plus overtime packing hours, which combined are at most 12 hours. - Regular packing hours are at most 8 hours a day, and overtime packing hours are at most 4 hours a day. - No more tomatoes may be packed once the packinghouse reaches the packing capacity	

4.2 Derived mathematical programming model

We show the derived mathematical programming model elements in separate tables according to the rules applied. First, Table 9 shows the decision maker and indexes generated by applying Rules 1 – 4.

Table 9. Derivation of decision maker and indexes.

Rule Number	**LEL model**	**Scenario model**	**Mathematical programming model**
1	Subject: Packinghouse Management (PM)	Scenario 0 Main actor PM	Decision Maker *PM*
2	Object: Harvesting day	Scenario 0 Context Temporal location Decision period	Indexes t
2	Object: Maturity day	Scenario 0 Context Temporal location Other temporal vars	j
3	Subject: Grower	Scenario 0 and 1.2 Actor: Grower	i

In the process of deriving the indexes, some scenarios' resources with multiple instances were considered as candidates on which to apply Rule 4. After that, the expert reviewed the relations among actors and resources but did not derive any sets because there are none restricted relations. Applying Rule 6 to the number of instances of the index for time period t yielded the time horizon as the parameter T. The number of instances of the index for maturity day j is the same parameter T. The number of instances of the index for grower i yielded parameter K. Other parameters came from analyzing the attributes of scenarios' actors and resources. For example, the attribute acres of actor grower yielded parameter H. Rule 7 indicates that H should be indexed by the indexes “related to it by the same notion”. In this case, the expert inspected the notion of acres looking the underlined terms (i.e., related elements) that were defined as indexes: i for “grower”, j for “maturity day” and t for “harvesting time”. However, the known values of acres are their grower and their maturity day, but their harvesting time is unknown. Therefore, the indexes assigned to the parameter are i and j, and the parameter is H_{ij}. Other parameters were derived similarly. The whole list of parameters appears in Table 10.

Table 10. Derivation of parameters.

Rule Number	**LEL model**	**Scenario model**	**Mathematical programming model parameter**
6	Object: Harvesting day Index: t	Scenario 0 Context Temporal location Decision period	T
6	Subject: Grower Index: i	Scenario 0 and 1.2 Actor: Grower	K
7	Subject: Grower Attribute: Acres Related indexes: Grower (i), Maturity day (j)	Scenario 0 and 1.2 Actor: Grower Attribute: Acres	H_{ij}
7	Subject: Grower Attr: Harvest capacity Related indexes: i	Scenario 0 and 1.2 Actor: Grower Attr: Harvest capacity	U_i
7	Subject: Grower Attr: Yields of bins Related indexes: i	Scenario 0 and 1.2 Actor: Grower Attr: Yields of bins	b_i
7	Subject: Market Attr: Market demand Related indexes: Harvesting day (t)	Scenario 0 Actor: Market Attr: Market demand	D_t
7	Object: Acres Attr: Fraction of v.r.t. Related indexes: -	Scenario 0 Resource: Acres Attr: Fraction of v.r.t.	δ
7	Object: Packinghouse Attr: Packing capacity Related indexes: -	Scenario 0 Resource: Packinghouse Attr: Packing capacity	P
7	Object: Packinghouse Attr: Gassing capacity Related indexes: t	Scenario 0 Resource: Packinghouse Attr: Gassing capacity	G_t
7	Object: Packinghouse Attr: Packing cost Notion: Cost of damaged tomato Related indexes: -	Scenario 1.3 Resource: Packinghouse Attr: Packing cost Notion: Cost of damaged tomato	C
7	Object: Packinghouse Attr: Packing cost Notion: Inventory holding cost Related indexes: -	Scenario 1.3 Resource: Packinghouse Attr: Packing cost Notion: Inventory holding cost	Ch

7	Object: Packinghouse Attr: Packing cost Notion: Shortfall cost Related indexes: -	Scenario 1.3 Resource: Packinghouse Attr: Packing cost Notion: Shortfall cost	*Cs*
7	Object: Packinghouse Attr: Packing cost Notion: Regular packing cost Related indexes: -	Scenario 1.3 Resource: Packinghouse Attr: Packing cost Notion: Regular packing cost	*Cr*
7	Object: Packinghouse Attr: Packing cost Notion: Overtime packing cost Related indexes: -	Scenario 1.3 Resource: Packinghouse Attr: Packing cost Notion: Overtime packing cost	*Co*
7	Object: Packinghouse Attr: Fraction of hour needed to pack a bin Related indexes: -	Scenario 0 Resource: Packinghouse Attr: Fraction of hour needed to pack a bin	f
7	Object: Packinghouse Attr: Fraction of tom. ready after gassing Related indexes: -	Scenario 0 Resource: Packinghouse Attr: Fraction of tom. ready after gassing	α

The next step was to derive the decision variables from the analysis of the attributes of scenarios' actors and resources, but this time, with unknown values. For example, the attribute acres of actor grower, with a certain maturity day and with an uncertain harvesting day. This attribute yielded decision variable X. Similar to the case for parameter H, to find the indexes for variable X the expert looked at the underlined indexes in notion of acres, which are: i for grower, j for maturity day and t for harvesting time, and the 3 of them are assigned to X to yield X_{ijt}. Other decision variables were derived similarly by applying Rule 8 (I_t, R_t, O_t, S_t), and appear in Table 11. Further analysis on the scenarios and the context, caused the expert to split the decision variable X_{ijt} in 2 variables: X_{ijt} to refer to the acres matured on the present cycle and L_{ijt} to refer to the acres matured on the last cycle. Additionally, 2 more decision variables were needed to represent the acres not harvested, both in the present cycle (Y_{ijt}) and the past cycle (A_{ijt}).

To define the objective function[1], Rule 9 was applied. Then, the expert had to manually write the final equation:

Minimize

$$\sum_{i=1}^{K}\sum_{j=1}^{T}\sum_{t=1}^{T} C * (A_{ijt} + Y_{ijt}) + \sum_{t=1}^{T} Ch * I_t + \sum_{t=1}^{T} Cs * S_t + \sum_{t=1}^{T} Cr * R_t + \sum_{t=1}^{T} Co * O_t \tag{1}$$

Table 11. Resulting decision variables.

Rule	LEL model	Scenario model	Mathematical programming model
8	Subject: Grower Attribute: Acres Rel. indexes: *i*, *j*, *t*	Scenario 0 and 1.2 Actor: Grower Attr.: Acres	X_{ijt}
8	Object: Packinghouse Attr.: Inventory level Related indexes: *t*	Scenario 0 Resource: Packinghouse Attr.: Inventory level	I_t
8	Object: Packinghouse Attr.: Regular packing hours Related indexes: *t*	Scenario 0 Resource: Packinghouse Attr.: Regular packing hs	R_t
8	Object: Packinghouse Attr.: Overtime packing hours Related indexes: *t*	Scenario 0 Resource: Packinghouse Attr.: Overtime packing hs	O_t
8	Object: Market demand Attribute: Shortfall Related indexes: *t*	Scenario 0 Resource: Market demand Attribute: Shortfall	S_t
-	Derived manually by the expert	Scenario 0 and 1.2 Actor: Grower. Attr.: Acres (matured on *last* cycle)	L_{ijt}
-	Derived manually by the expert	Scenario 0 and 1.2 Actor: Grower. Attr.: Acres (matured *present* cycle *not* harvested)	Y_{ijt}
-	Derived manually by the expert	Scenario 0 and 1.2 Actor: Grower. Attr.: Acres (matured on *last* cycle *not* harvested)	A_{ijt}

Deriving the constraints was mostly handcrafted taking all the information available in the context sentences of scenarios to create the corresponding equations, plus the addition new constraints to balance quantities and make the mathematical programming model work. Constraints appear in Table 12.

Table 12. Resulting constraints.

Rule	Scenario model	Mathematical programming model
10	Scenario 0 & 1.1 Context sentence: A grower may only harvest in a day up to his harvest capacity.	$\sum_{i=1}^{K} X_{ijt} + L_{ijt} \le U_i \ \forall_{it}$
10	Scenario 1.1 Context sntc: The number of bins of harvested tomatoes to pack from all growers should be less than the gassing capacity of the packinghouse for that day.	$(1-\delta)\sum_{i=1}^{K}\sum_{t-j=0}^{T-1} X_{ijt} * b_i + \sum_{i=1}^{K}\sum_{t+T-j=1}^{T} L_{ijt} * b_i \le G_t \ \forall_t$
10	Scenario 1.1 Context sntc: The number of bins of harvested tomatoes to be packed per day should be less than the combined regular and overtime packing capacity of the packinghouse.	$O_t + R_t - f\sum_{i=1}^{K}\sum_{t-j=0}^{T-1} X_{ijt} * b_i - f\sum_{i=1}^{K}\sum_{T-j=1}^{T} L_{ijt} * b_i = 0 \ \forall_t$

10	Scenario 1.3 Context sntc: Packing occurs during regular packing hour plus overtime packing hours, which are at most 12 hours.	$O_t + R_t \leq 12 \ \forall_t$
10	Scenario 1.3 Context sntc: Regular packing hs. are at most 8 hours/day	$R_t \leq 8 \ \forall_t$
-	Added by expert: balance the acres matured in the previous cycle but not yet harvested	$A_{ijt} - A_{ijt-1} + L_{ijt} = 0 \ \forall_{ijt}$
-	Added by expert: balance the acres matured in the present cycle but not yet harvested	$Y_{ijt} - Y_{ijt-1} + X_{ijt} = H_{ij} \ \forall_{ijt}$
-	Added by expert: balance the end-of-period inventory level (equal to the preceding end-of-period level + the quantity of “vine ripe” tomatoes packed + the number of boxes of tomatoes ready after gassing - the forecasted demand)	$I_t - S_t - I_{t-1} + S_{t-1} - \delta \sum_{i=1}^{K} \sum_{t-j=0}^{T-1} X_{ijt} * b_i + \delta \sum_{i=1}^{K} \sum_{t-T-j=1}^{T} L_{ijt} * b_i = D_t + G_t \ \forall_t$
-	Added by expert: assure a continual flow of tomatoes to the market. The mature green tomatoes to be packed should be at least equal to a fraction α of tomatoes ready on the day of gassing.	$(1-\delta) \sum_{i=1}^{K} \sum_{t-j=0}^{T-1} X_{ijt} * b_i + \sum_{i=1}^{K} \sum_{t+T-j=1}^{T} L_{ijt} * b_i \geq \alpha G_t \ \forall_t$

5. Semantic mediawiki construction for the LEL and scenarios definition

A domain ontology is proposed to attain the shared and agreed definition of a semantic model for the LEL and scenarios. Moreover, ontology-based information retrieval allows us to formulate queries based on our derivation rules, which will help in the semi-automatic derivation of mathematical programming model elements. Finally, tool support is provided through a semantic mediawiki constructed over the ontology, which allows for the knowledge building process of a conceptual model and its derivation into math model elements.

5.1 Ontology Model

The proposed ontology is depicted in Figure 2. Thus, *Subjects* symbols are related to Scenario’s *Actors* while *Objects* symbols are related to Scenario’s *Resources.* Moreover, *Verbs* symbols are related to *Scenario* and *Episode* from Scenario’s representation as they represent the activities or actions that are realized. LEL symbols are represented by the *Symbol* element, that has two properties: a *notion* and a list of *behavioral responses*. Thus, the *notion* property just contains *has* relations with *Subjects* and *Objects* that represent attributes of the described Symbol. Since we need to differentiate known attributes (parameters) from unknown ones (decision variables), the *has* relation is in fact modeled as two separate relations: *has known* and *has unknown*. In turn, the *behavioral response* of a LEL's Symbol is represented using a relation to the *Verbs* that represent the actions were the Symbol participates, and therefore, its responsibilities.

From the scenario's perspective, the element *Scenario* is represented by a property to define a *title* as plane text. The rest of the Scenario's properties are represented as relations with other model elements. Thus, there is a relation called *bounded* to a *Context*. There are two relations from Scenario to the *Actor* element called *involves main actor* and *involves secondary actor*, to represent the main and secondary actors of the Scenario respectively. Scenario also has a property called *executed over*, to related it with the *Resources* over which it is executed. The Scenario's episodes are described by the property *performs* related with the *Episode* elements. Finally, Scenarios are connected by the property *has* to the *Goal* element.

In turn, a *Context* has a text property *description* and a *has* property related to a *TemporalLoc* that represents the temporal location of the scenarios with its properties for *time-horizon, time_period* and *replanning_time_period.* The *Actor* element has *name* and *instance_number* as properties, where the latter is used for the mathematical programming model derivation, to identify actors with more than one instance as indexes. *Resource* has a structure identical to *Actor* although their semantic meaning is completely different. The *Episode* element has a *sentence* that describes it and a relation *described in* that connects the Episodes to the parent Scenario. Moreover, a *Goal* has a property *description* and a relation *optimize* to an *Optimization Goal*. This Optimization Goal is not part of the original definition of scenarios but added for mathematical programming model derivation purposes. It is described by two properties: *Operation* and *TargetVariable* to define the max/min operation for one particular variable as objective of the Scenario's *Goal*. Finally, there is a *Constraint* element that could be related with *Episode, Actor, Resource or TemporalLoc* elements by the property *is restricted by*. A Constraint is defined by the property *expression*, which is applied on the corresponding elements.

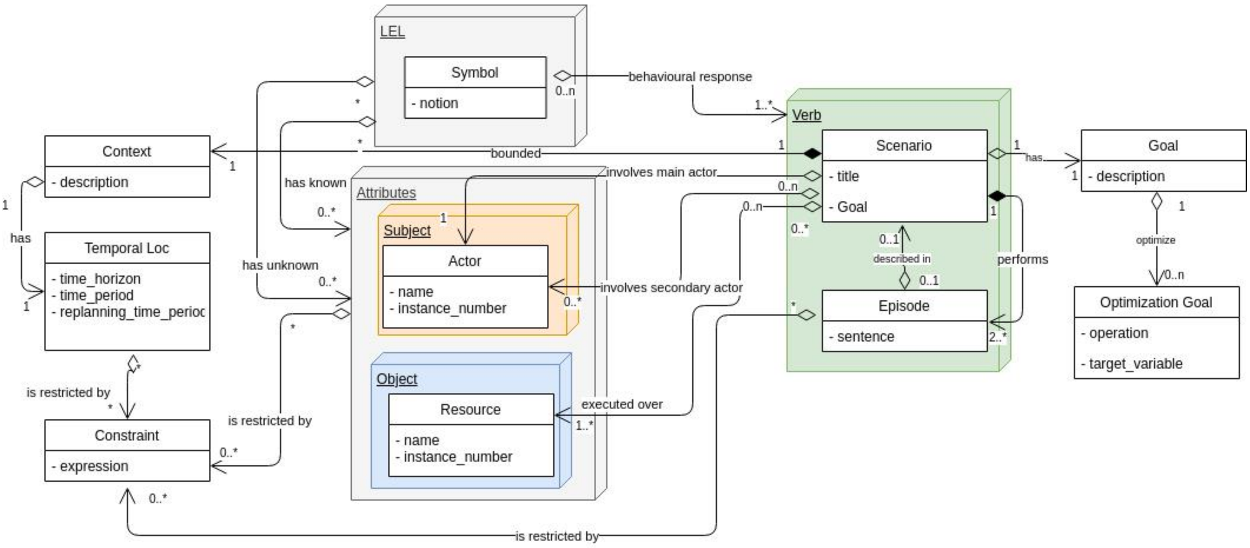


Fig. 2. The LEL and scenarios ontology.

5.2 Semantic queries

The queries rely on the definition of a specific Scenario instance defined as Base Scenario. The result of these queries will be used by the mathematical programming expert to obtain a preliminary version of the final mathematical programming model, as explained before. Table 13 summarizes the queries, in a pseudo-code that makes them more readable.

Table 13. Queries based on each rule to derive potential mathematical programming model elements

Rule Number	Mathematical model output	Semantic Query
1	Decision Maker	Base Scenario involves main actor: ?Actor
2	Index	Base Scenario bounded: ?Context ?Context has: ?TemporalLoc ?time_period
3	Index	Scenario involves main actor: ?Actor ?Actor instance_number >1: ?Actor Scenario involves secondary actor: ?Actor ?Actor instance_number >1: ?Actor
4	Index	Scenario executed over: ?Resource ?Resource instance_number > 1: ?Resource
5	Set	Symbol has known: ?Attribute1 Symbol has known: ?Attribute2 (?Attribute1 is restricted by: ?Constraint) == (?Attribute2 is restricted by: ?Constraint) ?Attribute1, ?Attribute2
6	Parameter	Scenario executed over: ?Resource ?Resource instance_number > 1: ?Resource instance_number Scenario involves secondary actor: ?Actor ?Actor instance_number > 1: ?Actor instance_number

7	Parameter	(Actor has known: ?Attribute)+ (Resource has known: ?Attribute)
8	Decision Variable	(Actor has unknown: ?Attribute)+ (Resource has unknown: ?Attribute)
9	Objective function	Base Scenario has: ?Goal ?Goal optimize: ?Optimization Goal
10	Constraint	Constraint

Rule 1 determines the decision maker by requesting the main *Actor* from the *Base Scenario.* Rule 2 determines the temporal index from the time period, accessing the *Context* of the *Base Scenario* through the *bounded* property, and from the *Context,* using the properties *has* to reach the properties of the *Temporal Location*. Rules 3 and 4 are also intended for deriving indexes. In these queries, all the instances of *Actor* and *Resource* in all the scenarios are collected by the properties *involves main actor*, *involves secondary actor* and *executed over*, and they are selected if the property *instance_number* is greater than 1. Rule 5 determines relations among *Actors* and *Resources* that are candidates to become sets, specifically, if they appear in the same notion and are related with the same *Constraint* by the property *is restricted by*. Rule 6 derives parameters from the indexes obtained by rules 3 and 4, specifically, the number of instances described by the property *instance_number*. Rule 7 derives parameters from known *Attributes* of *Actors* and *Resources*. Particularly, this rule uses the property *has known* that belongs to *Symbol* elements representing *Subjects* (for *Actors*) and *Objects* (for *Resources*). Similarly, Rule 8 determines decision variables but using the property *has unknown* to denote the unknown *Attributes.* To define the optimization goal of the mathematical programming model, it is necessary to access to the *Goal* of the *Base Scenario* by the property *has* and subsequently, to its *Optimization Goals* by the property *optimize.* Finally, any other restriction for the model could be derived from the *Constraint* elements.

5.3 Mediawiki implementation

We have built a semantic mediawiki in order to provide support for the collaborative construction of a knowledge base. The wiki provides the capability of creating and editing articles by way of a user-friendly interface guided by forms that will be used by stakeholders, analysts and mathematical modelling experts. These forms are based on the ontology proposed for the LEL and scenarios. Figure 3 shows a form to describe a Scenario. The figure shows that some attributes as goal and context are plain text, while others are described with a kind of button or *token*. These tokens describe relations to other elements of the model already created.

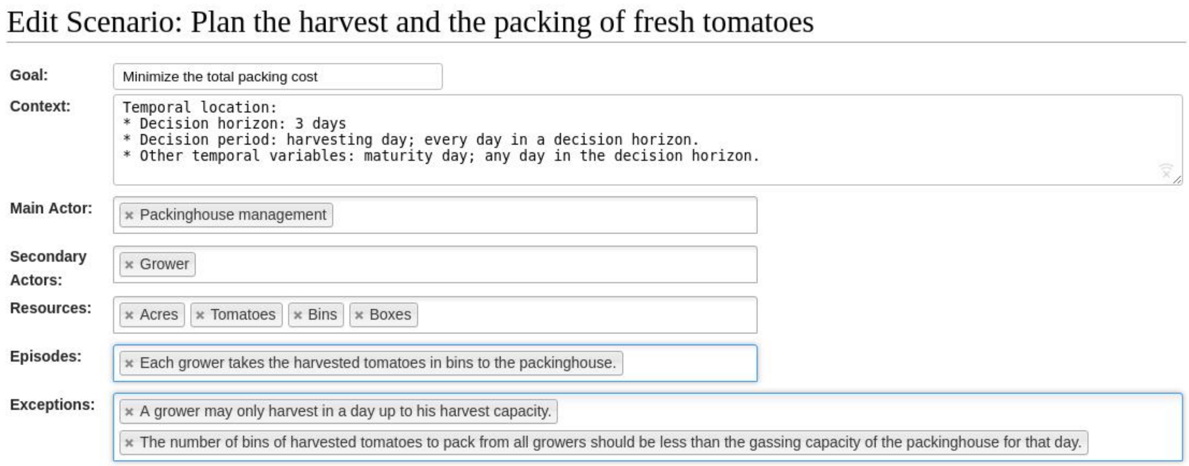
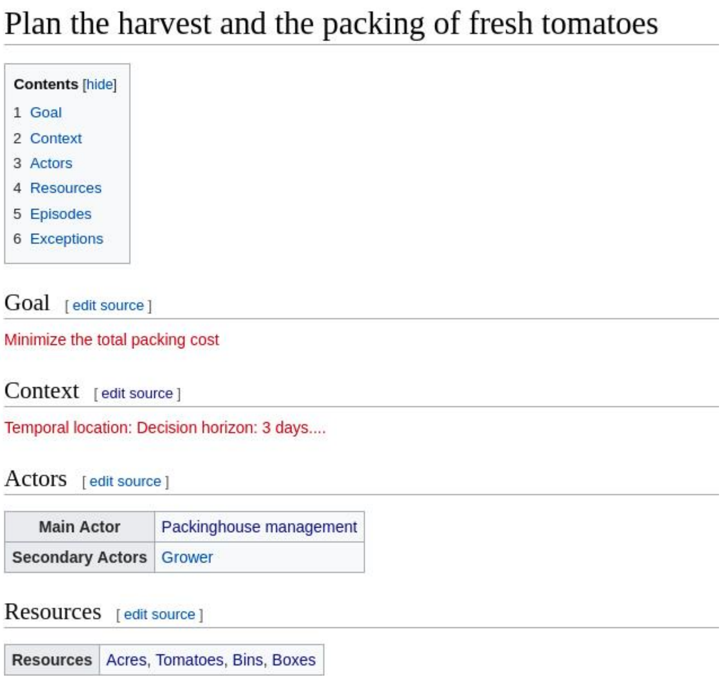


Fig. 3. Mediawiki form based on LEL and scenarios' ontology.

In turn, Figure 4 shows a form to navigate a Scenario. The wiki-links could be blue or red describing whether the related article is already defined or not, respectively.


Plan the harvest and the packing of fresh tomatoes

Contents [hide]

Goal [edit source]

Minimize the total packing cost

Context [edit source]

Temporal location: Decision horizon: 3 days....

Actors [edit source]

Main Actor	Packinghouse management
Secondary Actors	Grower

Resources [edit source]

Resources	Acres, Tomatoes, Bins, Boxes

Fig. 4. Scenario's article.

Finally, the capability of application of ontologies as a semantic knowledge model allows to implement the semantic queries described previously to semi-automatically derive mathematical programming model elements from LEL + scenarios elements. Thus, once the knowledge base is constructed, the mathematical modelling expert will be able to generate new articles with a preliminary version of the mathematical programming model. Figures 5 displays an example, with the article generated from the LEL and scenarios of the case study. Note that the support is not completely automatic because the inference only allows for an approximation to the mathematical programming model, and the expert is still needed to verify the correctness of the mediawiki derivation and provide the algebraic form.

Decision Maker [edit | edit source]

Packinghouse management

Objective function [edit | edit source]

Description: Minimize the total packing cost.

Type: **Minimization**

Objective variable: Packing cost

Indexes [edit | edit source]

By rule 2 [edit | edit source]

- Harvesting day
- Maturity day

By rule 3 [edit | edit source]

- Grower

By rule 4 [edit | edit source]

- Acres
- Tomatoes
- Bins
- Boxes

Sets [edit | edit source]

Constraints [edit | edit source]

From contrains expresions: [edit | edit source]

- A grower may only harvest in a day up to his harvest capacity.
- The number of bins of harvested tomatoes to pack from all growers should be less than the gassing capacity of the packinghouse for that day.
- The number of bins of harvested tomatoes to be packed per day should be less than the combined regular and overtime packing capacity of the packinghouse.
- Packing occurs during regular packing hour plus overtime packing hours, which are at most 12 hours.
- Regular packing hs. are at most 8 hours/day

Expert's considerations [edit | edit source]

Rule 4

Remove indexes:

- Acres
- Tomatoes
- Bins
- Boxes

Rule 7

Packing cost:

- Cost of damaged tomato
- Inventoring holding cost
- Shortfall cost
- Regular packing cost
- Overtime packing cost

Rule 8

Acres:

- Matured on *last* cycle

Fig. 5. Article of mediawiki with automatic mathematical programming model derivation.

6. Conclusions

This paper has presented a novel methodology that connects the areas of requirements engineering with conceptual modelling in order to build mathematical programming models that capture the business domain more effectively and completely. Specifically, the methodology has proposed for the first time the use of the LEL and scenarios for creating a conceptual model of a domain from where a mathematical programming model can be derived. The construcion of the conceptual model invites the participation of all stakeholders, which is deficiency of other proposals for DSS construction in agriculture. In comparison with other approaches for conceptual mathematical modelling, this article provides further tool-supported guidence about how to obtain the problem definition and how to derive a mathematical programming model from a precise specification in terms of derivation rules, as opposed to relying on mere textual descriptions or in the modeler's ability. Moreover, we have proposed an ontology that provides the basis for a semantic mediawiki that serves both, sharing knowledge of the conceptual domain model among the different stakeholders, as well as semi-automating the derivation of the mathematical programming model. The usefulness of this proposal can be understood from several perspectives: research, academic and managerial. From the research and academic points of view, we may highlight the main contributions as follows: (i) it provides a novel step-by-step methodology based on the LEL and scenarios that allows both: to obtain the required information to derive the definition part of a mathematical programming model, and to define the optimization problems that constitute the modelling part of the model; (ii) our approach provides a structure to the problem that allows to identify the elements of the problem clearly; (iii) using the LEL and scenarios to create a conceptual model iteratively and incrementally in collaboration with stakeholders allows applying an agile development approach to mathematical modelling; (iv) the use of LEL and scenarios provides traceability from the requirements to the mathematical programming model implementation to cope with possible changes of requirements and a better understanding of their impact on the model; and (v) the process of creating a conceptual model with LEL + scenarios also generates a complete specification of requirements for a potential model-based DSS. Regarding the managerial perspective, we believe that the ease of use and good expressiveness of the proposed methodology will facilitate the implementation of mathematical programming models in agriculture, as well as provide new tools for teaching mathematical programming and foster research in the combined areas of agile methods in requirements engineering, mathematical programming and decision support system development. Further research includes validating the proposed methodology in real world case studies from agriculture. Finally, we intend to extend the approach to the derivation of mathematical programming models under uncertainty, such as stochastic programming and fuzzy mathematical programming.

Acknowledgments. This work was supported by the European Commission, project H2020-RUC-APS, grant number H2020-MSCA-RISE-2015-691249; and the Argentinian National Agency for Scientific and Technical Promotion (ANPCyT), grant number PICT-2015-3000.

References

Alemany, M., Ortiz, A. & Fuertes-Miquel, V.S. (2018). A decision support tool for the order promising process with product homogeneity requirements in hybrid Make-To-Stock and Make-To-Order environments. Application to a ceramic tile company. *Computers & Industrial Engineering 122*: 219–234.

Alemany, M.M.E., Alarcón, F., Lario, F.C. & Boj, J.J. (2011). An application to support the temporal and spatial distributed decision-making process in supply chain collaborative planning. *Computers in Industry 62*: 519–540.

Alemany, M.M.E., Lario, F.C., Ortiz, A. & Gómez, F. (2013). Available-To-Promise modeling for multi-plant manufacturing characterized by lack of homogeneity in the product: An illustration of a ceramic case. *Applied Mathematical Modelling 37*(5): 3380–3398.

Alemany, M.M.E., Pérez Perales, D., Alarcón, F. & Boza, A. (2007). Planificación Colaborativa para Redes de Suministro-Distribución (RdS/D) mediante programación matemática en entornos distribuidos. In *Int. Conference on Industrial Engineering & Industrial Management*.

Alexander, I. & Maiden, N. (2004). Scenarios, stories, and use cases: the modern basis for system development. *Computing Control Engineering Journal 15*(5): 24–29.

Armengol, A., Mula, J., Díaz-Madroñero, M. & Pelkonen, J. (2015). Conceptual model for associated costs of the internationalisation of operations. *Lecture Notes in Management and Industrial Engineering* 181–188.

Baraniuk, R., Burrus, C., Johnson, D. & Jones, D. (2004). Sharing knowledge and building communities in signal processing. *IEEE Signal Processing Magazine 21*(5): 10–16.

Behzadi, G., O'Sullivan, M.J., Olsen, T.L. & Zhang, A. (2018, September 1). Agribusiness supply chain risk management: A review of quantitative decision models. *Omega (United Kingdom)*. Elsevier Ltd.

Breitman, K.K. & Leite, J.C.S.P. (2003). Ontology as a requirements engineering product. In *Proceedings of the IEEE Int. Conference on Requirements Engineering (RE)*.

Cid-Garcia, N.M. & Ibarra-Rojas, O.J. (2019). An integrated approach for the rectangular delineation of management zones and the crop planning problems. *Computers and Electronics in Agriculture 164*.

Cysneiros, L.M. & Leite, J.C.S.P. (2001). Using the language extended lexicon to support non-functional requirements elicitation. In *Proceedings of the Workshops de Engenharia de Requisitos, Wer'01*. Buenos Aires, Argentina.

Dominguez-Ballesteros, B., Mitra, G., Lucas, C. & Koutsoukis, N.S. (2002). Modelling and solving environments for mathematical programming (MP): A status review and new directions. *Journal of the Operational Research Society 53*(10 SPEC.): 1072–1092. Retrieved from https://doi.org/10.1057/palgrave.jors.2601361

Esteso, A., Alemany, M., Ortiz, A. & Peidro, D. (2018). A multi-objective model for inventory and planned production reassignment to committed orders with homogeneity requirements. *Computers & Industrial Engineering 124*: 180 – 194.

Esteso, A., Alemany, M.M.E. & Ortiz, A. (2018). Conceptual framework for designing agri-food supply chains under uncertainty by mathematical programming models. *International Journal of Production Research 56*(13): 4418–4446.

Geisser, M. & Hildenbrand, T. (2006). A method for collaborative requirements elicitation and decision-supported requirements analysis. In R. G. Ochoa S.F. (ed.), *Advanced software engineering: expanding the frontiers of software technology*. Boston, MA: Springer.

Giannoccaro, I. & Pontrandolfo, P. (2001). Models for Supply Chain Management : A Taxonomy. In *Production and Operations Management 2001.Conference POMS mastery in the new millennium.* Orlando, FL, USA.

Gil, G.D., Figueroa, D.A. & Oliveros, A. (2000). Producción del LEL en un dominio técnico. Informe de un caso. In *Proceedings of the Workshops de Engenharia de Requisitos, Wer'00*. Rio de Janeiro, Brazil.

Grillo, H., Alemany, M.M.E., Ortiz, A. & Fuertes-Miquel, V.S. (2017). Mathematical modelling of the order-promising process for fruit supply chains considering the perishability and subtypes of products. *Applied Mathematical Modelling 49*: 255–278.

Grossmann, I. (2005). Enterprise-wide optimization: A new frontier in process systems engineering. *American Institute of Chemical Engineers 51*(7): 1846–1857.

Gruber, T. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition 5*(2): 199–220.

Gruber T. (1995). Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies 43*(43): 907–928.

Hernández, J.E., Mula, J., Ferriols, F.J. & Poler, R. (2008). A conceptual model for the production and transport planning process: An application to the automobile sector. *Computers in Industry 59*(8): 842–852. Retrieved from https://www.sciencedirect.com/science/article/pii/S0166361508000778

Jain, R., Malangmeih, L., Raju, S.S., Srivastava, S.K., Immaneulraj, K. & Kaur, A.P. (2018). Optimization techniques for crop planning: A review. *Indian Journal of Agricultural Sciences 88*(12): 1826–1835. Retrieved from https://www.researchgate.net/publication/329736318

Kaplan, G., Hadad, G., Doorn, J. & Leite, J.C.S.P. (2000). Inspección del léxico extendido del lenguaje. In *Proceedings of theWorkshops de Engenharia de Requisitos, Wer'00*. Rio de Janeiro, Brazil.

Laporti, V., Borges, M.R.S. & Braganholo, V. (2009). Athena: A collaborative approach to requirements elicitation. *Computers in Industry 60*(6): 367–380.

Leite, J.C.S.P. & Franco, A.P.M. (1993). A strategy for conceptual model acquisition. In *Proceedings of IEEE International Symposium on Requirements Engineering, 1993*. San Diego, California: IEEE Computer Society Press.

Leite, J.C.S.P., Hadad, G., Doorn, J. & Kaplan, G. (2000). A Scenario Construction Process. *Requirements Engineering Journal 5*(1): 38–61. Retrieved from http://link.springer.com/article/10.1007/PL00010342

Lenat, D.B. (1995). CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM 38*(11): 33–38.

Lesh, R, Landau, M. & Hamilton, E. (1983). Conceptual models in applied mathematical problem solving research. In R. Lesh and M. Landau (ed.), *Acquisition of Mathematics Concepts and*

Processes. NY: Academic Press.

Lesh, Richard. (1981). Applied mathematical problem solving. *Educational Studies in Mathematics 12*(2): 235–264. Retrieved from https://doi.org/10.1007/BF00305624

Lezoche, M., Yahia, E., Aubry, A., Panetto, H. & Zdravković, M. (2012). Conceptualising and structuring semantics in cooperative enterprise information systems models. *Computers in Industry 63*(8): 775–787. Retrieved from https://www.sciencedirect.com/science/article/pii/S016636151200125X

Liu, L., Wang, H. & Xing, S. (2019). Optimization of distribution planning for agricultural products in logistics based on degree of maturity. *Computers and Electronics in Agriculture 160*: 1–7.

Miller, G.A. (1995). WordNet: a lexical database for English. *Communications of the ACM 38*(11): 39–41.

Miller, W.A., Leung, L.C., Azhar, T.M. & Sargent, S. (1997). Fuzzy production planning model for fresh tomato packing. *International Journal of Production Economics 53*(3): 227–238. Retrieved from http://linkinghub.elsevier.com/retrieve/pii/S0925527397001102

Mir, S.A., Qasim, M., Arfat, Y., Mubarak, T., Bhat, Z.A., Bhat, J.A., … Sofi, T.A. (2015). Decision support systems in a global agricultural perspective-a comprehensive review. *Journal of Agriculture Sciences 7*(1): 403–415.

Moskaliuk, J., Kimmerle, J. & Cress, U. (2009). Wiki-supported learning and knowledge building: effects of incongruity between knowledge and information. *Journal of Computer Assisted Learning 25*(6): 549–561.

Mula, J., Poler, R., García-Sabater, J.P. & Lario, F.C. (2006). Models for production planning under uncertainty: A review. *International Journal of Production Economics 103*(1): 271–285. Retrieved from https://www.sciencedirect.com/science/article/pii/S0925527306000041

Mula, Josefa, Peidro, D., Díaz-Madroñero, M. & Vicens, E. (2010). Mathematical programming models for supply chain production and transport planning. *European Journal of Operational Research 204*(3): 377–390. Retrieved from http://dx.doi.org/10.1016/j.ejor.2009.09.008

Mundi, I., Alemany, M., Boza, A. & Poler, R. (2013). A model-driven decision support system for the master planning of ceramic supply chains with non-uniformity of finished goods. *Studies in Informatics and Control 22*(2): 153–16. Retrieved from https://doi.org/10.24846/v22i2y201305

Munir, K. & Sheraz Anjum, M. (2018). The use of ontologies for effective knowledge modelling and information retrieval. *Applied Computing and Informatics 14*(2): 116–126. Retrieved from https://www.sciencedirect.com/science/article/pii/S2210832717300649#b0035

Oliveira, A. de P.A., Leite, J.C.S. do P., Cysneiros, L.M. & Cappelli, C. (2007). Eliciting Multi-Agent Systems Intentionality: from Language Extended Lexicon to i* Models. In *XXVI International Conference of the Chilean Society of Computer Science (SCCC'07)*. IEEE. Retrieved from http://ieeexplore.ieee.org/document/4396976/

Pérez, D., Lario, F.C. & Alemany, M.M.E. (2010). Descripción detallada de las Variables de Decisión en Modelos basados en Programación Matemática en un contexto de Planificación Colaborativa de una Red de Suministro/Distribución (RdS/D). In *4th Int. Conf. on Industrial Engineering and Industrial Management; XIV Congreso de Ingeniería de Organización*. Donostia - San Sebastián.

Pérez Perales, D., Lario, F.C., Alemany, M.M.E. & Hernández, J. (2012). Framework for Modelling the Decision. *International Journal of Decision Support System Technology 4*(2): 59–77. Retrieved from http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/jdsst.2012040104

Pesic, M. & van der Aalst, W.M.P. (2005). Towards a reference model for work ditstribution in workflow management systems. In & M. N. E. Kindler (ed.), *Proceedings of the First*

International Workshop on Business Process Reference Models (BPRM,05). Nancy: University Henri Poincaré.

Raghunathan, S. (1996). A structured modeling based methodology to design decision support systems. *Decision Support Systems 17*(4): 299–312.

Saranya, S. & Amudha, T. (2017). Crop planning optimization research - A detailed investigation. In *2016 IEEE International Conference on Advances in Computer Applications, ICACA 2016*. Institute of Electrical and Electronics Engineers Inc.

Schneeweiss, C. (2003a). Distributed decision making in supply chain management. *International Journal of Production Economics 84*(1): 71–83. Retrieved from https://www.sciencedirect.com/science/article/pii/S092552730200381X

Schneeweiss, C. (2003b). Distributed decision making—a unified approach. *European Journal of Operational Research 150*(2): 237–252. Retrieved from https://www.sciencedirect.com/science/article/abs/pii/S0377221702005015

Schön, E.-M., Thomaschewski, J. & Escalona, M.J. (2017). Agile Requirements Engineering: A systematic literature review. *Computer Standards & Interfaces 49*: 79–91. Retrieved from https://www.sciencedirect.com/science/article/abs/pii/S0920548916300708

Shapiro, J.F. (1993). Chapter 8 Mathematical programming models and methods for production planning and scheduling. *Handbooks in Operations Research and Management Science 4*: 371–443. Retrieved from https://www.sciencedirect.com/science/article/abs/pii/S0927050705801884

Siau, K. (2004). Evaluating the Usability of A Group Support System Using Co-Discovery. *Journal of Computer Information Systems 44*(2): 17–28. Retrieved from https://www.tandfonline.com/doi/abs/10.1080/08874417.2004.11647563

Soto-Silva, W.E., Nadal-Roig, E., González-Araya, M.C. & Pla-Aragones, L.M. (2016, June 1). Operational research models applied to the fresh fruit supply chain. *European Journal of Operational Research*. Elsevier B.V.

Swartout, B., Patil, R., Knight, K. & Russ, T. (1996). Toward distributed use of large-scale ontologies. In *Proc. of the Tenth Workshop on Knowledge Acquisition for Knowledge-Based Systems*. Banff, Canada.

Udias, A., Pastori, M., Dondeynaz, C., Carmona Moreno, C., Ali, A., Cattaneo, L. & Cano, J. (2018). A decision support tool to enhance agricultural growth in the Mékrou river basin (West Africa). *Computers and Electronics in Agriculture 154*: 467–481.

Vicens, E., Alemany, M.., Andrés, C. & Guarch, J.. (2001). A design and application methodology for hierarchical production planning decision support systems in an enterprise integration context. *International Journal of Production Economics 74*(1–3): 5–20. Retrieved from https://www.sciencedirect.com/science/article/pii/S0925527301001037

Young, R. (2004). *The Requirements Engineering Handbook*. Artech HouseISBN: 978-1-58053-266-2.