Schedule Modeling and Expert Knowledge Elicitation Using Graphs

Paula A. Toselli[1], Jorge Palombarini[1,2], Eduardo Romero[1], Ernesto Martinez[3]
[1] *Universidad Tecnológica Nacional – Facultad Regional Villa María*
[2]*CIT Villa María (CONICET-UNVM)*
{ptoselli,jpalombarini,eromero}@frvm.utn.edu.ar
[3] *INGAR – CONICET (UTN)*
{ecmarti}@santafe-conicet.gob.ar

Abstract

Knowledge elicitation about repairs and adjustments in a production schedule is necessary to handle on-line modifications to a schedule and use this information to create heuristics and acquire knowledge about the expert good practices to repair schedules and to react to undesirable disruptive events.

In the set of the traditional tools to represent production schedules, the Gantt Diagram is the most popular option. However, it does not offer a simple vision of the elements that compose the schedule, or the changes made on it. Besides, Gantt Diagrams do not allow the implementation of knowledge elicitation that represents the operating mode of an expert user.

In this paper, a representation based on graphs is presented to model industrial schedules. A computational tool is made to gather attributes and characteristics that achieve pre-established production goals about schedule information systematically. The developed prototype allows creating Scheduling models of Batch-type industrial production.

The prototype has been designed to allow tracking of the sequence of changes made to an original schedule to achieve a given goal. The prototype favors user knowledge elicitation which highlights that graph-based representations have many advantages regarding schedule visualization, task changes and re-scheduling as well as details about schedule state sequences, resources and tasks involved

1. Introduction

Production systems, during their life cycle, rest on carrying out monitoring and control tasks to deal with unexpected events and uncertainty that are faced by adjusting production schedules. In this context, rescheduling knowledge is applied by experts in the production system domain are working on, to change and adjust the schedule applying corrective measures using their own experience [1].

Knowledge used by an expert to fix disruptive events is scarce, so it must be elicited, modeled and stored to be used later in similar states of a production schedule. In order to extract human expert knowledge, it is necessary to generate tools, develop heuristics and gather repair-based information systematically.

At present, scheduling of industrial production process is carried out through different methodologies and manual or informatics tools. As representative examples [2], the critical path diagram, PERT and the most popular option, Gantt Diagram can be mentioned [3].

In order to elicitate expert knowledge, these types of diagrams are not rich enough to provide data and information for rescheduling. Besides, tracing changes made to a given schedule is not amenable. The following points can be added [4]:

- Some schedules might be excessively complex to be represented using the mentioned diagrams.
- Gantt Diagram representation depicts only the time period where a set of tasks must be completed, but critical information like the number of resources needed for each task as well precedence and synchronization constraints are not explicitly modeled.
- During schedule execution, environmental conditions change and modifications have to be represented. If a Gantt Diagram is used, edition must be frequent and simple. In practice, these changes are difficult to implement as the planning problem complexity increases.
- Some data are not taken into account, for example costs and resource sharing constraints.

This works proposes representing schedules based on graphs to model synchronization/precedence constraints, considering adjustments and modifications (re-scheduling) along with schedule attributes that offer deeper and richer details which are lost in previous approaches. Besides, the proposed approach readily allows expert user knowledge elicitation.

Furthermore, using graphs to represent the dynamic of schedule states allows easy visualization of changes and discovers heuristics and repair rules using distance algorithms, weight functions, and others. Besides, some additional advantages are: representing compound resources easily, adding attributes and other complementary data.

Graphs enable the entire schedule or some part of it to be displayed focusing, for example, on repaired points,

meaning those planned tasks forced to be modified in view of a disruptive event. Finally, schedule changes are quick and simple to do which makes room to pinpoint modifications needed to reach a re-schedule state and the traceability of any change made along the way.

In order to convert schedule data into graph elements so to generate the corresponding representation it is necessary to map models validating the transformation from a schedule element to a graph one. Also, two abstraction levels are used to represent different types of data in the schedule state.

To validate the proposal, a prototype was developed to transform production schedule data into a graph model. Furthermore, the prototype presents some additional functionality like schedule states seen in a repair sequence and expert user knowledge elicitation.

To carry out the mentioned transformation, the QVT methodology is implemented. To automatize the tasks involved in elicitation and modeling tasks, the prototype performance is presented and detailed.

Finally, a multiproduct factory case study based on a schedule process is explained. The developed prototype is used to validate this work proposal, along with the advantages related to using graphs for scheduling modeling. This paper will further research work including distance calculation algorithms, inverse reinforcement learning for manufacturing cognitive systems giving support to decision making in industrial production schedules.

This paper is organized in the following sections: in the next section, relevant concepts of graph theory, UML and QVT are defined along with the more relevant applications. In the third section schedule modeling using graph is described. Then, in the fourth section, a case study is presented and used to validate this paper proposal. Finally, conclusions are enumerated and future works is presented.

2. State of the Art

A critical evaluation was made from published scientific literature specifically related to knowledge elicitation. In [5], different methods for knowledge elicitation are explained and discussed in detail. An example of a development product process is presented. Even though, knowledge models are mentioned, no particular representation model is explicitly used

The meaning of knowledge elicitation and its different levels are explained in [6]. Techniques are explained in detail for different contexts. Also, some tools are presented such as: PCPack, Protege and CmapTools. However, their benefits are only used to automatize the elicitation process and do not model knowledge.

Methods for information to be elicited using online collaborative networks are reviewed in [7]. Several techniques and algorithms are discussed, such as multiple random path traces, path distances and semantic proximity of seed context. Even though this paper has models, algorithms, and similar comparisons are presented, the scope of its contribution is still very different from the concepts presented in our work.

In [8], a study of knowledge representation and requirement elicitation is presented. Pre-conceptual schemas and conceptual graphs are applied to knowledge representation. Although graphs are used, the proposed approach is still different from the way graph models are used in our work. Finally, in [9], knowledge elicitation techniques are identified and organized in categories based on similarity. Strengths and weaknesses for each technique are mentioned. However, none models or representation tools are used.

3. Fundamentals

To develop this paper, several tools that constituted the theoretical frame of the presented proposal were employed, including Knowledge Elicitation, Graphs, Unified Modeling Language (UML) and Query, Views & Transformations (QVT).

3.1. Knowledge Elicitation. The process of collect from a human expert as source of knowledge, information that is relevant is called knowledge elicitation.

Through direct interactions with the expert, knowledge from the expert domain can be elicitated. Using a set of techniques and methods knowledge elicitation can be made [5].

Elicitation, representation and transmission of knowledge are very important activities that can be automatized. As a result, systems can emulate the experts experience and used as support in decision making.

Currently, many private and public sectors systems depend on an expert that control and manage all operations involved and related tasks. Often, this expert ends up being the only one that can operate the system. This poses a real problem if the human expert is absent.

Domain experts used to be the key in the knowledge elicitation process; multiple types of knowledge are often involved and human expertise has different roles in the system. The types of knowledge are [6]:

- Domain Knowledge: describes the concepts and elements in the domain and relations between them.
- Inference Knowledge: This is knowledge about how the components of expertise are to be organized and used in the overall system.

- Task Knowledge: also, called procedural knowledge, this is knowledge concerned with goals, sub-goals, tasks and sub-tasks.
- Strategic Knowledge: This is information that monitors and controls the overall problem solving process.

Knowledge elicitation allows the expert know-how to be modeled and emulated by a prototype.

3.2. Graph Theory– Concepts and definitions. A graph can be defined as a set of nodes or vertices (v) and a set of edges (A). The existence of the edges between different nodes allows the generation of binary relationships [10].

Given an infinite or finite set L of node and edge labels it can be defined a graph such as a tuple g = (V, E, μ, v) where:

- V is the node sets,
- E is the edge sets,
- u : V -> L node labeling function and
- v : E -> L edge labeling function.

A graph may have different classifications [4]:

- Directed: when the pair of edges are sorted.
- Undirected: when the pair of edges are not sorted.
- Rated: when the edges contain some value (weight, length long, others)
- Labeled: when each edge and node has its own label
- Unlabeled: it is a special case in which the same label is used on all the nodes and edges.

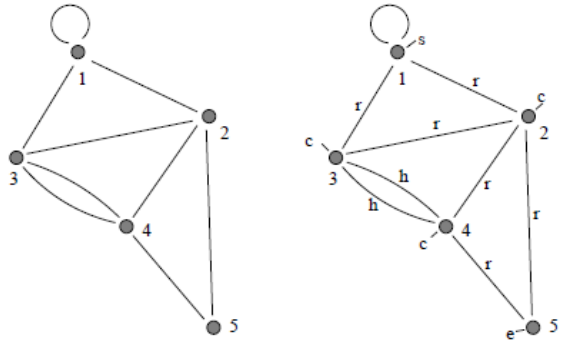

Figure 1. Graph Representations

In this paper directed and labeled graphs are used to represent the state of a schedule. In figure 1, a generic example of this kind of diagram is shown. A standard set of operations can be applied to graph nodes or edges. The mentioned sets of operations are defined as [5]:

- Node substitution (u -> v)
- Node eliminations (u -> ε)
- Node insertion (ε -> v)

These operations allow changes to be made in graph structures. Following the modifications, an execution trace of changes that is useful in debugging a rescheduling process among graphs can be made [2].

3.3. UML - Unified Modeling Language. The Unified Modeling Language (UML) is a graphic language that allows visualizing, specifying, constructing and documenting each part of the software development process. UML offers a way of shaping conceptual terms such as business processes, system functions, and particular objects of a determined language, database schemes and re-utilizable software components.

The mentioned graphic language defines a notation and a metamodel. The first one is the graphic material by which the models, the model language syntax, are observed. The metamodel is a diagram that defines this notation [12].

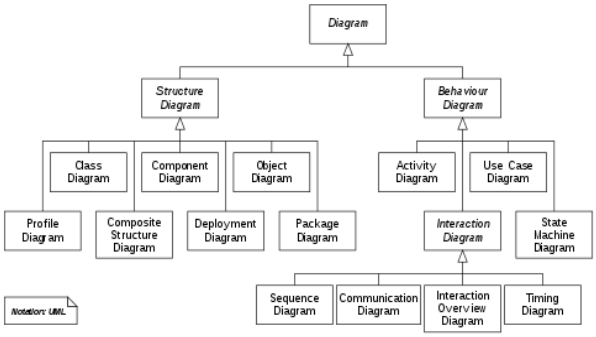

Figure 2. UML diagram structure

There are several types of diagrams in UML to model software as can be seen in figure 2. In this paper, the chosen ones are Class Diagrams.

This kind of diagrams represents the system components by the class and relationships between them. They are static diagrams that represent classes, with its attributes and methods, and the static relationships without showing the methods that are invoked between them.

A metamodel can be defined as a UML description on UML, that is to say, is a model from a model [13]. Occasionally, a mapping model is made when there is two metamodels and is necessary to convert one into another. The goal is to transform each object from a class (of the original model) to a similar object of the class into a destination model.

To carry out the mapping between the classes of a UML diagram, the Query, Views, Transformations

(QVT) methodology is used and is explained on the following section.

3.4. Metamodels Transformation - Query, Views, Transformations (QVT). Formally, a metamodel is a model that specifies the language concepts, the relations between them and the structural rules that restrict the possible elements or the valid models in it.

On the other hand, a metaclass is a class whose instances are also classes, i.e. as the objects are classes instances, the classes are instances of a metaclass [14].

It is important to take into account the information of the metamodels to carry out the transformation between models. These describe the representations of each model element and define the restrictions that must be enforced in a model transformation.

When talking about a transformation, we are making reference to the process of converting a model into another in the same system. Transformation is a process based on a set of rules, which defines the mechanisms of transforming an original model into a new one [15].

Transformations allows to:

- Identify a metaclass of the original model element.
- Transform the mentioned metaclass into the final model element.
- Keep the consistency of the established rules in the original model.

The process of model transformation is based on comparing *vis-à-vis* if all the defined relations are verified. For those relations who are not verified, the mentioned process tries to carry out by inserting, eliminating or modifying the final model [16].

3.4.1. QVT (Query, View, Transformations). The transformations are classified as relation and mapping. The first ones specify multi directional transformations. They do not create or modify models, but they can check the consistency among two or more related models. While mapping, transforms elements from a domain into the elements of another domain [17].

QVT allows carrying out transformations among models whose languages are defined in MOF (Meta Object Facility) using the following elements:

• Query: is an expression tested on a given model that has as a result one or more instances of types.

• View: is a model that is completely derived from another model. They are more general than Query. They tend to be read-only entities.

• Transformation: generates a final model from an original one. The relations specify the transformations while mapping implements them. QVT structure to transform models and elements relationships are shown in figure 3.

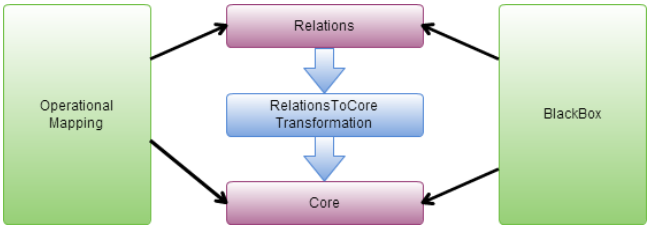

Figure 3. Relations among QVT metamodels

2.3.2. Relations. A relation in a language carries out a transformation among models called *candidates*, where a set of relations is used to make the model transformation.

A candidate model is a specification for the different types of elements that the original model can have.

```
Transformation ScheduleToGraph (sch: Schedule, gr: Graph)
    {
        top relation stateSchToNode()
        top relation batchToNode()
        top relation ResourcesToNode()
        top relation TasksToNode()
    }
```

Code 1. Transformation Example

Candidate models from the example are *sch* and *gr*. On the other hand, a *transformation* can be invoked to verify the consistency of two models as well as to transform a model into another. A transformation example can be seen on code 1.

The reserved identifier *transformation* is used to carry out a transformation in a one-way direction, i.e. considering that one of the models is chosen as the final one. The selected model must be empty or contain the necessary elements to make the transformation, meaning the destiny model must be totally empty or it should have some previous objects that must be mapped into the transformation model.

There are elements such as relations and domain within a transformation. The relations define restrictions that must be satisfied by the elements of the candidate models. The domains are variables that can be matched to a model of a given metamodel [18].

The domains that correspond to the models “sch” and “gr” can be seen in the example code 2. Each domain specifies a pattern, i.e. name and schema. In the example above the variable “name” is the same in both properties “name”, in consequence, they have the same value. A transformation has two kinds of relation: top-level where

all the relationships must be complied with and no-top level that are satisfied only when they are invoked directly or transitively.

```
top relation ResourcesToNode()
   {
      Name: String
      Domain sch   r: Resources {
          Nombre = Name
      }
      Domain gr   n: Node{
          Nombre = Name
          Label = "r"+Name
      }
   }
```

Code 2. Top Relation declaration example.

4. Schedule Modeling using Graphs.

Production planification and control activities can be considered as means to reduce the uncertainty and deal with disturbances that invalidate assumptions made.

The steps to create a task schedule, generally, are:

- Project analyzing
- Tasks and resources identification
- A work plan creation
- Assessment of results

Schedule models generate knowledge and allow the development of strategies to reach expected production goals [19]. Within the modern tools to represent schedules, Gantt Diagrams are one of the most popular options due to its simplicity and widespread use [20].

It is important to note that expert knowledge, which is formed and acquired by constant learning and professional experiences on creating schedules and modifications to reach production goals, is not represented, extracted or stored in Gantt diagrams.

Neither domain, task or strategic knowledge is represented with this diagrams, in consequence several data and information is missed. However, graph can elicitate this knowledge as is explained next.

To schedule modeling using graphs, the mentioned definition and concept in section 2 were applied. To model schedules as graphs, the UML methodology is first used, applying the object-oriented Paradigm [24]. To represent schedule data as graph elements, it is necessary to convert each production schedule element to the corresponding one in the graph. As both the schedule and graph domain are modeled using UML, it is possible to make a direct transformation between these metamodels using QVT. The transformation process is explained in more detail in the sections below.

4.1. Models Transformation. To begin with, a metamodel design is needed to transform models. Model-Oriented Objects (MOO) was chosen in the design process and it is implemented using UML.

MOO was selected as the modeling tool since structural representation of objects as well as dynamic and functional behavior of requirement is allowed. Besides, it can be used several abstraction levels and natural language is an important characteristic of this methodology [25].

In this paper, a class diagram was modeled with two domains that represent the schedule and the corresponding graph respectively [18]. Once design is finished, the original and the transformed domain are determined and selected. This means that, if model A is chosen as the original one, transformations are made in such manner that it is equivalent to model B (transformed model).

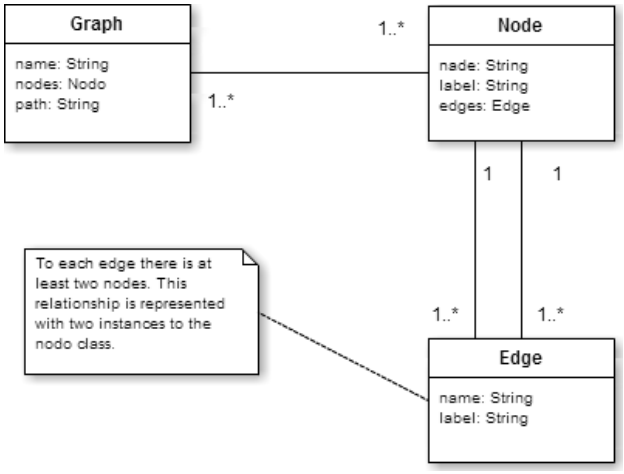

Figure 4. UML Graph model.

The graph model domain is shown in Figure 4. Each *Graph* and its elements, node and edges, are represented in three classes. Relations in a graph are the objects combination from these classes.

Two abstraction levels were established for the metamodels, as can be seen in table 1:

- In the first one, schedule elements are represented using nodes, edges, labels and relationships among them.
- In the second abstraction level, for each schedule element modeled as a graph object, characteristics and attributes are detailed. This allows storing

important information about schedule and its states.

Table 1. Schedule element transformed description.

Schedule Element	Graph Element Abstraction Level One	Attributes Abstraction Level Two
State	Node	Batch quantity, processing time, description.
Batch	Node	Processing time, due date, initial and end date, batch size, family, state, container, resources available.
Resource	Node	Resources type, description (simple, compound), processing time, top/low capacity, quantity of tasks.
Task	Node	Task type, initial/end time, total duration, precedent task, purpose.
Batches relationship	Edge	Previous and resulting batch are represented if they exist
Tasks Precedence relationship	Edge	Tasks precedence relationship is represented for each batch and/or resource.

Interaction among classes and objects of schedule and graph domains can be seen in figure 5. The goal is to transform, an action formally called mapping, each element from schedule domain to an equivalent one of graph domain.

Mapping is made using the concepts and definitions presented in section 2. A summary table is made from the mapped information and details about which element of the schedule domain corresponds to the one on the graph domain and which characteristics or attributes are needed to store the elicited knowledge.

For implementing QVT, four top-relation types were developed. Top-relations must be always executed. Each relation specify which variable in each domain are mapped directly and which ones has to be adapted to be truly equivalent in the transformed domain, like it was explained in previous sections.

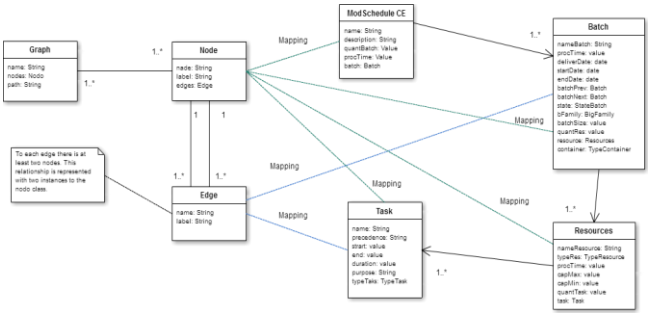

Figure 5. Schedule and Graph models transformation

QVT was selected since mapping can be automatized and used to validate and justify the modeling needed to convert the schedule information, in any state, to a graph model that provide more understanding and details, all in the same diagram [15].

5. Case Study.

To exemplify schedule modeling using graphs and the functioning of the prototype, a more complex case study is considered. This case study in about a multiproduct batch plant which has more than 80 equipments and includes the following production steps [21]:

(1) Formulation and mixing,
(2) Laboratory testing followed by product adjustment and release,
(3) Container filling,
(4) Carton packaging and
(5) Palletization

The plant consists of 59 formulation units, 1 laboratory and 9 fill-out trains. At the same time, the laboratory has a capacity for 15 batches. In table 2 different products and the type of container used are detailed [22].

Table 2. Product types description.

Big Family	Family	Container
Synthetic	Varnish	Metallic
	Enamels	Plastic
	Primers	
	Industrial coating	Plastic
		Metallic
	Pools	Metallic
	Demarcation	Packet
	Anilines	Metallic
Latex	Latex	

Regarding the production process, a formulation unit is used to carry out the formulation and product mixing. Each formulation unit consists of disperser and a tank. After each batch is processed, the disperser is disconnected from the tank to make it available for its use by a different batch in a possibly different unit. Four dispersers are used for the latex products and thirteen for the acrylics.

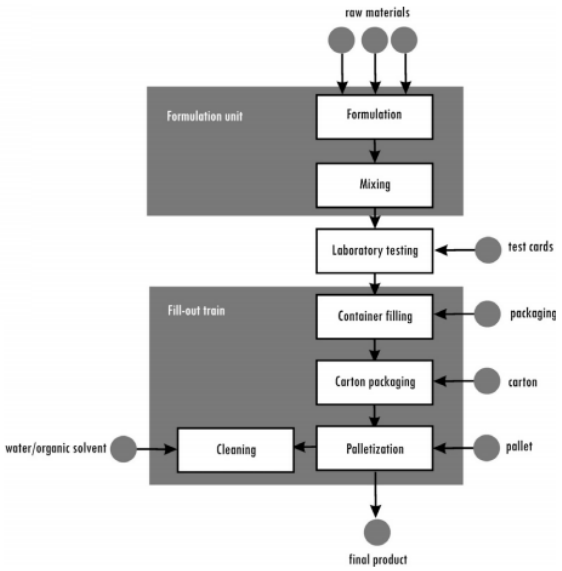

Figure 6. Production formula of the painting plant.

The filling belts are used to carry a batch from the correspondent formulation unit (tank), not only for the acrylic products but also for the latex ones, to their correspondent packages. The filling belts are also used for the filling, packaging, palletizing and cleaning of the containers. If a batch does not comply with the quality requirements, it goes back to the formulation and mixing unit.

Primarily, production batches are transported from tanks using alternative fill-out trains according to the interconnections available among the existing equipment at the shop-floor, and the restrictions that are constrain making up virtual formulation units (tank + disperser).

5.1. Schedule representation using graphs - Prototype developed. To evaluate the proposed modeling approach, a prototype was developed to automatize graph modeling and schedule information elicitation from an industrial process. The prototype was implemented in Netbeans IDE 8.0 (Java) and the database using PostgresSQL. To demonstrate the functionality of the prototype, schedules in the case study mentioned were used [23].

State schedule data and human expert knowledge are the inputs to the prototype. A state is a specific situation of the schedule, similar to a photograph describing how the schedule elements are in a determined place and time.

Multiple and different changes can be necessary to do on the schedule state, representing updating data for a task in the schedule, add production items, generate a solution to a disruptive event, etc. This activity of goal-directed changes to the schedule state is called re-scheduling.

To determine which changes must be made, expert knowledge is needed. The know-how to make the best repair is provided by the human expert as another input to be use in future re-schedules.

State, batch, resource and task attributes are data input of the prototype among with the user expertise. The schedule state is composed of this information and is provided from the industrial environment in which it is immersed. User expertise is elicitated from the know-how in schedule repairs made by the human expert.

To reach more generality, for each step or element basic attributes are represented, however the following data is required: specify if it is the first state or a modification of some other and similar one, select the previous state to create traceability and chose state first tasks to reference the starting point of graph analysis.

One or more state schedules can be added modified or eliminated from the prototype. Generally, schedule state modifications are not random. The human expert, mostly, has the experience to the best repair made. This knowledge is an input of the prototype to be stored and used to the changes made in schedule repairs.

Besides, schedule states can be stored and queried, having also access to the change traces on the basis of this information. Once the schedule state is registered, classes and objects of original model are created by the prototype, and the model is mapped into a graph schedule (graph domain) by the implementation of QVT methods like it was explained in previous sections.

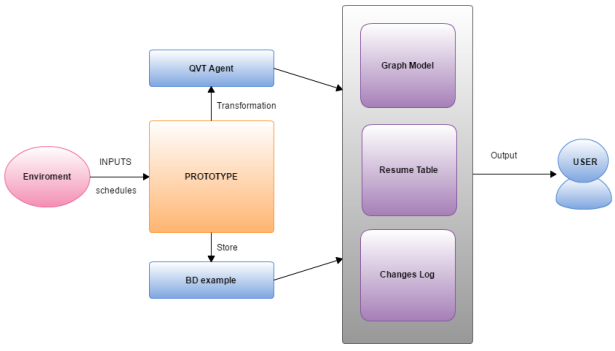

Figure 7. Prototype schema

Schedules modeled using graphs are generated by the prototype as can be seen on figure 6. A summary table detailing the transformed elements and their attributes is shown, also, by the prototype.

A modification to the actual schedule state is often necessary done as a result of a disruptive event, resource conflict or failure, and arrival of a new high-priority order. A new schedule state is generated and human expert decides by own expertise which changes are made on it. These modifications are stored in the prototype, representing each state obtained in a sequence of changes makes possible to compare information and trace the sequence of schedule states in the path to a goal, analyzing how the expert human reaches solutions to several problems that appear at the shop-floor level. Also, for each state modification, a detail of them are stored to create a change log. Steps to modify schedules and generate the mentioned logs are shown on code 3.

1. *Start*
2. *Schedule state query*
3. *Select state to modify*
4. *Select modify option*
5. *If attributes states change:*
   5.1.Change state attribute
   5.2. Save modification (original value, new value, place, date)
6. *If attribute batch change:*
   6.1. Change batch attribute
   6.2. Save modification (original value, new value, place, date)
7. *If attribute resources change:*
   7.1. Change resources attribute
   7.2. Save modification (original value, new value, place, date)
8. *If attribute tasks change:*
   8.1. Change tasks attribute
   8.2. Save modification (original value, new value, place, date)
9. *Save changes*
10. *Save changes log*
11. *End*

Code 3. Schedule state modification pseudo-code

Information stored in the change logs and the resulting traces of schedule states obtained are used to elicitate expert knowledge on schedule problem resolutions that is lost in other simpler diagrams such as Gantt diagrams. Heuristics and learning rules are created based on that stored knowledge and can be incorporated in a cognitive system using these systematic procedures to calculate state schedule similarities and graph distances [26].

According to the model conversion functionality, the prototype uses a detailed table of the converted elements, that is to say, using the stored information from a schedule state that performs the element transformation detailing which were the model correspondences between the schedule and graph domains are shown (see figure 8 for details).

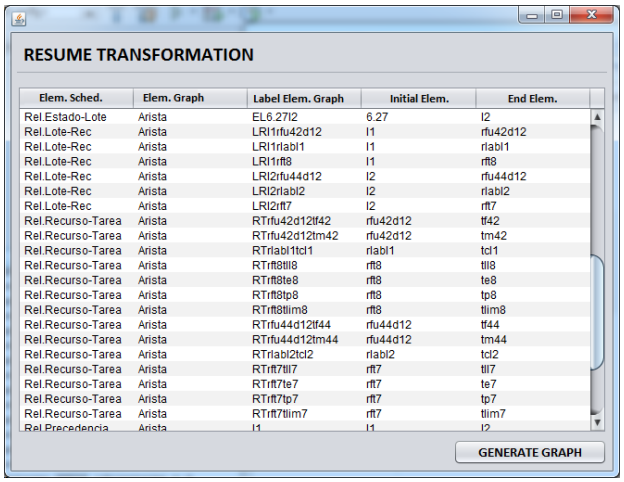

Figure 8. Model transformation summary table

The graph model for the schedule state is made from the overview table in figure 8, which will contain the abstraction level presented in foregoing sections.

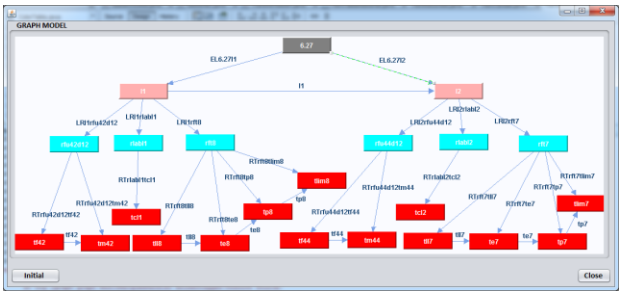

Figure 9. Schedule graph model example

The graph model, as can be seen in figure 9, presents a more informative representation than alternative tools to observe schedule states, batches, resources and tasks along with their name and precedence and execution relationships. Resource groups, model relations and task executions are discriminated by using different colors. Also, the first abstraction level information is observed in the graph diagram as was explained in previous sections. The schedule graph specifies element data obtained by steering over each node and resorting to the information stored in the second abstraction level.

6. Concluding Remarks

This paper proposes a new modeling methodology using graphs that results in a prototype implementation

that automates the model generation and the elicitation of knowledge for rescheduling. This proposal emerges of the need to satisfy the earlier limitations mentioned about current tools and representation methodologies to model production schedules. Be aware that current tools do not allow capturing expert human knowledge to address re-scheduling problems.

In the first place, concepts and definitions required to understand graphs, model languages and transformations were studied.

Secondly, the mentioned concepts were implemented to model, transform metamodels and develop a prototype that allows to automatize the schedule representation and to elicitate expert domain knowledge gained by trial-and-error experience in dealing with abnormal events and unplanned disturbances.

Finally, a case study of a multi-product batch factory is presented to demonstrate the use and advantages of the modeling and implementation of the methodology presented in this paper.

7. Future Work

Knowledge elicitation and production schedules modeling, along with the developed prototype, serve as a base of future works with Inverse Reinforcement Learning (IRL) [27].

Future developments will use learning algorithms where distances among schedules are need as input data. The proposal of this paper, using graph modeling is a fundamental advantage to quantify distances since exist several options of algorithms to calculate them. Finally, the proposed approach provides the foundations to future distance edition calculations and human expert knowledge modeling that will be used for automated learning in artificial cognitive systems.

8. References

[1] B. MacCarthy y J. Wilson, *Human Performance in Planning and Scheduling*, London: Taylor & Francis, 2001.

[2] Giudici y Bris Luch, *Introducción a la teoría de grafos*, Venezuela: Ediciones de la Universidad de Bolivar, 1997.

[3] F. González Fernández, *Teoría y práctica del mantenimiento industrial avanzado*, Madrid: Editorial FC, 2003.

[4] M. Gracia Ramos, M. Lopez-Jurado González, M. Yagüez Insa y J. Meringó Lindahl, *Guía práctica de economía de la empresa I: empresa y entorno (Teoría y ejercicios)*, Barcelona: Universidad de Barcelona, 2008.

[5] D. Ford y J. Sterman, «Expert knowledge elicitation to improve formal and mental models,» *System Dynamic Review*, vol. 14, nº 4, pp. 309-340, 1998.

[6] N. Shadbolt y P. Smart, *Knowledge Elicitation: Methods, Tools and Techniques*, Florida: J. R. Wilson & S. Sharples, 2015.

[7] V. Franzoni, A. Milani y S. Pallottelli, «Multi-path Traces in Semantic Graphs For Latent Knowledge Elicitation,» de *International Conference on Natural Computation (ICNC)*, Italia, 2015.

[8] C. Zapata Jaramillo, A. Gelbukh y F. Arango Isaza, *Pre-conceptual schema: a conceptual-graph-like kownledge representation for requierements elicitation*, Berlin: Springer-Verlag, 2006.

[9] N. Cooke, «Varieties of knowledge elicitation techniques» de *International Journal of Human-Computer Studies*¸USA, 1994.

[10] G. Wagner de García y A. Barredo, *Introducción a la teoría de grafos*, Armenia: Ediciones Elizcom, 2010.

[11] N. Martí Oliet, Y. Ortega Mallén y J. Verdejo López, *Estructura de datos y métodos algoritmicos: ejercicios resueltos*, Madrid: Pearson Educación, 2004.

[12] M. Fowller y K. Scott, *UML gota a gota*, Méjico: Addison Wesley, 1999.

[13] J. Rumbaugh, I. Jacobson y G. Booch, *El lenguaje de modelado unificado - Manual de referencia*, Madrid: Addison Wesley - Pearson, 2006.

[14] C. Ariste, J. Ponisio, N. Leopoldo y R. Giandini, *Diseñando transformaciones de modelos CIM/PIM: desde un enfoque de negocios hacia un enforque de sistemas*, 2015.

[15] M. Hebach, *MDA with QVT*, Presentation Boreland Together, 2006.

[16] L. Favre, «Model driven architectures for reverse engineering technologies: strategic directions and system evolution,» de *Engineering Sciece Referencia - IGI Global*, 2010.

[17] P. Barendrecht, *Modeling transformations using QVT operational mappings*, Eindhoven: University of Technology, 2010.

[18] M. Djedjigo, D. Maurad y P. Mokan, *Aspect-Oriented security hardening of UML design models*, Switzerland: Springer, 2015.

[19] M. González Riesco, *Gestión de la producción: cómo planificar y controlar la producción industrial*, Vigo: Ideas Propias, 2016.

[20] J. Verdoy y E. al., *Manual de control estadistico de calidad: teoria y aplicaciones*, España: Universitat Jaume, 2006.

[21] M. Rolón, M. Canavesio y E. Martinez, «Agent based modelling and simulation of intelligent distributed scheduling systems,» *Computer aided chemical engineering,* nº 26, pp. 985-990, 2009.

[22] M. Rolón, M. Canavesio y E. Martinez, «Agent based generative simulation of an intelligent distributed scheduling world with Netlogo,» de *Proceedings of CICA Applied Computational Intelligence Conference*, Buenos Aires, 2009.

[23] J. Palombarini, *Un enfoque cognitivo a la reparación automática de planes y schedules integrando aprendizaje por refuerzos con abstracciones lógicas y relacionales,* Santa Fe, 2014.

[24] R. Musier y L. Evans, «An aproximate method for the production scheduling of industrial batch processes with parallel units,» *Computers and chemical engineering,* nº 13, p. 229, 1989.

[25] D. Coleman, Object-oriented development - The fusion method, Michigan: Prentice Hall, 1994.

[26] Van Otterlo, *The Logic Of Adaptive Behavior: knowledge representation and algorithms for adaptative sequential decision making under uncertainty in first-order and relational domains,* Amsterdam: IOS Press, 2009.

[27] S. Levine, Z. Popović y V. Koltun, «Nonlinear inverse reinforcement learning with gaussian processes,» de *Advances in neural information processing systems 24 (NIPS)*, 2011.