Int. J. Production Economics 144 (2013) 223–242

Contents lists available at SciVerse ScienceDirect

Int. J. Production Economics

journal homepage: www.elsevier.com/locate/ijpe

An approach based on constraint satisfaction problems to disruptive event management in supply chains

Armando Guarnaschelli [a,*], Omar Chiotti [b], Hector E. Salomone [a]

[a] INGAR-CONICET-UTN, Avellaneda 3657, S3002GJC Santa Fe, Argentina
[b] CIDISI—UTN—FRSF, Argentina

ARTICLE INFO

Article history:
Received 15 February 2012
Accepted 12 February 2013
Available online 14 March 2013

Keywords:
Disruptive event
Execution control
Supply chain
Constraint satisfaction

ABSTRACT

This work introduces a generalized model for evaluating and restoring feasibility in the execution of supply chain processes. The model was designed to provide automation to the disruption management function of Supply Chain Event Management (SCEM) systems. The repair mechanism is based on a constraint satisfaction problem that can be automatically instantiated from self-contained descriptions of the ongoing schedules without previous knowledge of the supply chain structure. The proposed mechanism intends to make surgical modifications to the current schedule which do not affect the economical and operational considerations and the allowed changes are limited to the space of slacks already included by the original schedule. This level of repair can be safely delegated to automated systems and would facilitate the design of collaborative inter-organizational business processes to manage events along the supply chain. A case study validates the applicability of the proposed models.

1. Introduction

Enterprises engaged in collaborative planning processes create and maintain production and distribution schedules synchronized with the schedules of other members in the supply chain. During the execution, disruptive events affecting the schedules and their synchronization usually occur. To increase the ability of the supply chain to respond minimizing the impact of these disruptions, Scheduling Systems generate schedules prescribing some sort of resource reservation which might be able to absorb variability during execution such as buffers of material, resource capacity and time. These slacks are used to avoid the need of a new scheduling task, which can be costly and time consuming since all enterprises involved in the supply chain should agree on new synchronized schedules.

As schedules are affected by disruptive events they require an execution control process capable of disruption management. Current Supply Chain Management Systems lack of systematic approaches to disruption management. In practice the decision process taking place given the disruption is loosely structured, managers are seldom supported by systematic methodologies to cope with the problem caused by the disruption, and when they do, the solution is usually a full re-scheduling task.

Information systems that support execution control processes capable of disruption management have been called Supply Chain Event Management (SCEM) Systems (Masing, 2003; Zimmermann, 2006). SCEM systems should provide functionality for: capturing/predicting disruptive events that could affect the schedule execution; checking if schedules are still feasible after the occurrence of a disruptive event; and searching for solutions to locally repair schedules affected by a disruptive event (Guarnaschelli, 2012).

Focusing on the last two functional requirements, this paper presents a model driven development approach (Hailpern and Tarr, 2006), to automate the activities of *feasibility checking of schedules* in presence of disruptive events and *repairing of disrupted schedules* using a mechanism based on constraint satisfaction models.

The approach proposes a reference model as a platform independent meta-model, which allows the representation of ongoing schedules previously generated by the Scheduling Systems of the supply chain partners. When representing the availability of resources and the specification of supply process orders (SPOs), the reference model provides systematic elements to capture the slacks of the original schedules in a way that is suitable for analyzing feasibility in presence of disruptive events. This common representation for schedules allows the analysis of concurrent execution schedules from different business partners in a supply chain.

From an instance of this reference model a constraint satisfaction problem (CSP) for schedule feasibility checking and repairing

* Corresponding author. Tel.: +54 342 453 5568; fax: +54 342 455 3439.
E-mail addresses: ag28@live.com.ar, guarnaschelli@santafe-conicet.gov.ar (A. Guarnaschelli), chiotti@santafe-conicet.gov.ar (O. Chiotti), salomone@santafe-conicet.gov.ar (H.E. Salomone).

0925-5273/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.ijpe.2013.02.007

can be automatically derived through a model-to-model transformation. Based on this CSP, a mechanism for the automatic repair of disrupted schedules is implemented. This mechanism is able to check if the schedules are still feasible after the occurrence of a disruptive event, and then, to search for solutions using scheduled slacks, avoiding the triggering of a new process of rescheduling and coordination among the Scheduling Systems of the supply chain partners.

The changes to be introduced in a schedule to restore its feasibility are limited by the slacks on resources and orders. The aforementioned repair mechanism keeps unchanged every economical and operational target set when schedules were created. The objective is to absorb as many disruptions as possible within previously agreed operational slack without resorting to a new rescheduling task.

This paper is organized in the following way. Section 2 discusses related works. Section 3 describes a component-based architecture of a collaborative SCEM system which is the information system framework for the models proposed here. Section 4 presents a reference model for disruptive event management. Section 5 presents a constraint satisfaction model for feasibility checking and repairing of schedules. Section 6 presents a mechanism for the automated repair of disrupted supply processes. Section 7 describes the implementation details of the transformation engine. Section 8 presents an empirical validation of the approach through a case study, and Section 9 presents conclusions and future work.

2. Related works

The subject of this work is mainly related to three research areas: event management in complex software systems; rescheduling of manufacturing and distribution processes; and SCEM systems.

Disruption management in the execution of supply processes can be seen from the perspective of exception handling in complex software systems such as Workflow Management Systems, Process Management Systems and Self-healing Systems. In Workflow Management Systems, the support for exception handling goes from the definition of exception handlers and methods for the specification of exceptional behavior to classification and forecasting of exceptions (Hwang and Tang, 2004; Song and Han, 2003; Yuan et al., 2008). In Process Management Systems the definition of exception handlers invoked under given conditions provide support for given types of exceptions (Hamadi et al., 2008; Weske, 2007). Self-healing Systems are intended to provide support for diagnosing faulty situations and selecting and/or searching for recovery strategy (Friedrich et al., 2010; Griffith et al., 2009). While these approaches are useful frameworks for managing exceptions of general business processes, the nature of disruptions in the specific domain of supply chain processes need to be further exploited to build more powerful corrective actions to re-establish feasibility at the same time the disruption is being handled. Inspired in these approaches, the proposal of this work provides this additional feature by capturing, in a systematic way, the aspects of the supply process feasibility needed to automate a reparation mechanism.

Rescheduling is the current practice for facing disruptions during the execution of production and distribution processes. In the literature, different formulations for manufacturing rescheduling problems have been reviewed (Aytug et al., 2005) and different strategies to restore schedule feasibility are evaluated (Pfeiffer et al., 2007).

Rescheduling research works on distribution processes consider logistics networks and the underlying transportation resources with a set of assumptions on the behavior of the vehicles and loading and unloading on depots as in the work of (Li et al., 2009a, b).

Rescheduling approaches as the mentioned above are mainly devoted to provide centralized decision making and largely dependent on typified processes. They cannot be easily extended to collaborative multi-organizational supply chains. They are suitable for supporting the individual rescheduling task of each sub-node in the supply chain but provide rather little support to help restoring a collaborative execution schedule. An approach that supports supply chain operations coordination based on a distributed search algorithm inspired by constraint programming has been proposed by (Gaudreault et al., 2012).

Supply Chain disruption management has been discussed in specific industrial contexts. A proposal for rescheduling under disruptions is introduced in the work of (Arief Adhitya and Karimi (2006)). It addresses the event management problem by the previous definition of a cause-effect graph between supply chain entities; upon this graph a rescheduling strategy is introduced. This strategy requires knowing the formulation of the scheduling models of every supply chain partner. In contrast, our approach does not make any assumptions on the supply chain structure and does not require any knowledge about the methodologies used to build schedules. The repair mechanism is fully functional through an automated transformation from the schedules, output of the scheduling systems, and the support for new processes or partners is automatic.

The schedule repair methodology described in this paper should not be considered a rescheduling approach but a subordinate decision problem suitable to be delegated to an autonomous repair mechanism with the aim of systematizing the usage of the slacks already provided in the original schedule

Therefore, the degrees of freedom in our approach are constrained within the original schedule slacks and should never require a re-assessment of the performance measures. For this reason, the feasibility model is only involved with capturing the requirements of the scheduled supply processes and their relationship with the availability profile of the resources.

This reduction of scope in the decision problem is the key enabler of a generalized formulation that can be applied to any sort of supply process and is intended to perform as an autonomous supervisory component in the execution control layer. A failure in the repair will always fall back on a re-scheduling.

Research in SCEM systems has mainly been focused in addressing the monitoring, the capture and the communication of disruptive events. The ability to exert corrective control actions has been identified as an area barely explored (Bearzotti et al., 2008; Zimmermann, 2006). In this sense, a method based on multi-agent negotiation of previously defined recovery plans for searching solutions to disruptions was presented (Cauvin et al., 2009). This approach does not take into account the planned availability of resources versus the resource utilization by orders, therefore the decision support is limited to give recommendations to a decision maker that will analyze them and relies on the previous definition of generic recovery plans. Autonomous disruptive event management functions for automatically deriving repair actions fully executable are not provided.

A summary of DM (Disruption Management) related works is shown in Table 1.

3. Business process and architecture for a collaborative SCEM system

Both the repair mechanism and the constraint satisfaction problems derived from instances of the proposed reference model

Table 1
Related works.

Knowledge area		Contribution to DM	In comparison with the approach of this work
Event management in complex software systems	Workflow management systems Process Management systems Self-healing systems	Classification and forecasting of exceptions Exception handlers Exception Handlers to give support to predefined exception types Faulty situation diagnosis. Selection of recovery strategies	Although a DM support system can be considered a complex software system, the nature of the disruptions in these approaches does not include the ones in the specific domain of supply chain processes. Powerful corrective actions to re-establish feasibility at the same time the disruption is being handled cannot be derived. Inspired in these approaches, the proposal of the present work provides powerful corrective actions by capturing, in a systematic way, the aspects of the supply process feasibility needed to automate a reparation mechanism
Rescheduling of manufacturing and distribution processes		Methods and mechanisms for rescheduling specific types of supply processes in manufacturing and distribution systems	Rescheduling is the current practice in supply chain management to tackle the problem of exceptions (effects of a disruption) in schedule execution. However the approach presented in this work allows restoring the feasibility of a schedule making minimal changes within its slacks, avoiding triggering a rescheduling task. The repair mechanism proposed can autonomously analyze whether it is possible to restore the feasibility or not. This is an important characteristic as frequent rescheduling can be extremely harmful and at the same time, slacks are used proactively to maintain a better adherence to schedule targets. Moreover the reference model can be used to define any kind of supply processes allowing disruption management across general supply chains
SCEM Systems		Information systems supporting the monitoring process of schedule execution and the communication of disruptive events across the supply chain	Fully applicable as every DM system should proactively detect and communicate disruptions. In this work the goal is more ambitious as the models allow implementing autonomous procedures to repair the schedule after a disruption, a characteristic mentioned in SCEM literature as unexplored and required

are the core tools to enhance an autonomous collaborative disruptive event management solution called CMDESC (Collaborative Management of Disruptive Events in Supply Chains) (Guarnaschelli et al., 2012).

CMDESC defines a collaborative business process for managing disruptions, identifying the participants of this process (which are information systems on behalf of supply chain partners), specifies their interactions and provides a foundation for its implementation as a SCEM system.

The collaboration is achieved by the interaction of three participants: two centralized managers and a distributed controller.

The participant Controller is distributed; it represents a supply chain member and is responsible for the execution control of every one of its schedules. The controllers interact with a single (centralized) Feasibility Manager responsible for checking the coordinated schedule's feasibility and conducting the collaborative repair. A single (centralized) Monitor is responsible for analyzing resources feasibility over time in comparison with orders progress in order to predict and detect disruptive events.

Controllers analyze and restore the feasibility of their schedules by using the services offered by the Feasibility Manager, and they predict and detect disruptive events through the services offered by the Monitor. In Fig. 1a summarized version of this business process modeled using the BPMN (Business Process Modeling and Notation) language is shown.

The process starts with the arrival of a *ScheduleMsg* containing a schedule which is registered by the Controller, which immediately requests its monitoring. Following, the Controller awaits two possible messages: a monitoring completion message generated by the Monitor (*Notify Completion* task) which ends the schedule control process, or a disruptive event message sent by the Monitor (*Notify Disruptive Event* task).

In presence of a disruptive event, the controller defines its scope (*Define Event Scope* task explained in Section 6), requests its feasibility check (*Request Feasibility Checking* task) and waits for a feasibility report sent by the Feasibility Manager.

If the feasibility report states that the schedule is still feasible despite the disruption, the controller keep waiting for the arrival of a new disruptive event message or the end of the monitoring process. If the infeasibility of the schedule is reported the controller requests a repair to the Feasibility Manager (*Request Schedule Repair* task).

The Feasibility Manager will start the Repair Schedule subprocess which is executed using the repair mechanism proposed in this work, and depicted in Section 6.

At the end of the Repair sub-process the Feasibility Manager sends a Feasibility Report Message (*Send Feasibility Report* task). The controller decides what to do next accordingly to the result of the repair process. If the schedule could be repaired, the controller introduces the changes recommended by the Feasibility Manager and start over a new controlling loop. If not, the controller will generate an alarm (to the corresponding planning sub-system) and the control process ends. For any result of the repair process the controller will send a cancellation message to the Monitor as the previously controlled schedule is not valid any more.

The architectural diagram in Fig. 2 depicts the main components of a collaborative SCEM system: Control Subsystems (Controllers), a Monitoring Subsystem (Monitor), and a Feasibility Management Subsystem (Feasibility Manager).

The Control Subsystem provides the Monitoring Subsystem with access to updated data from the Execution System. It also interacts with the Feasibility Management Subsystem requesting the feasibility verification and repairing function and engaging in collaborative repair when requested. It is also responsible for sending the solutions received from the Feasibility Management Subsystem to the Execution System or notifying exceptions to the Scheduling System for performing a rescheduling.

The logic of the Feasibility Management Subsystem is the focus of this work. It comprises: the generation of CSP models to evaluate feasibility; the reference model to express the schedules of supply chain partners; and the collaborative repair mechanism.

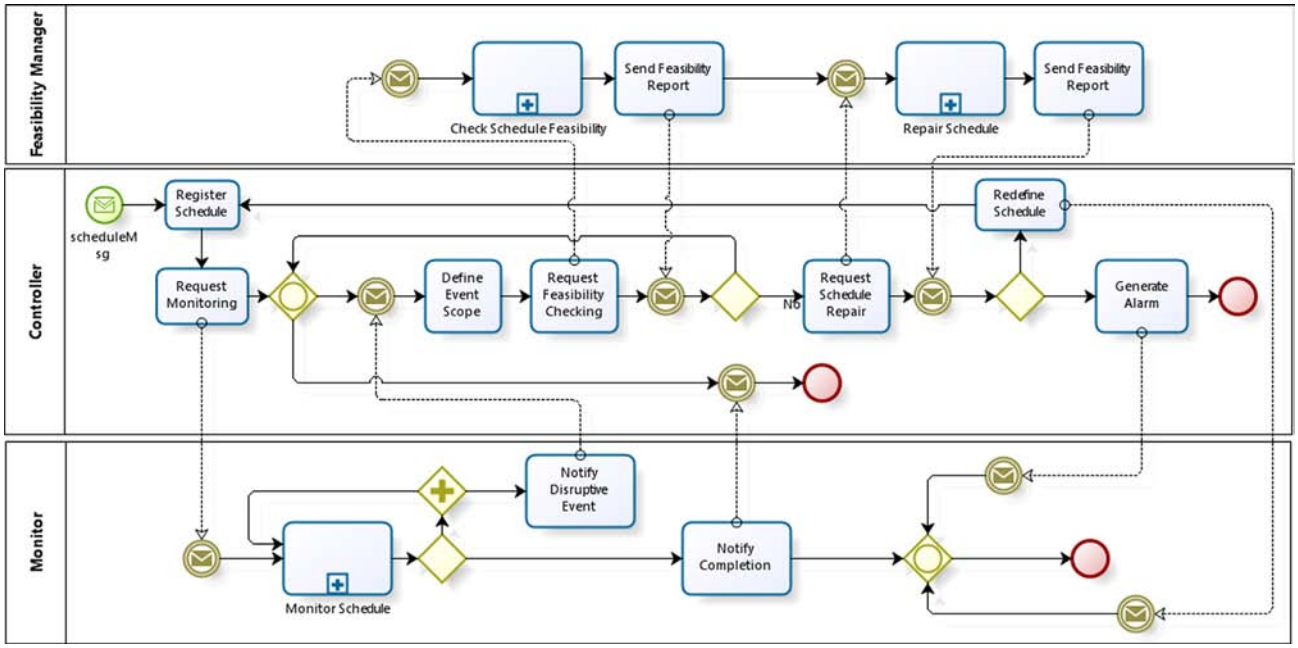

Fig. 1. Business process for managing disruptions.

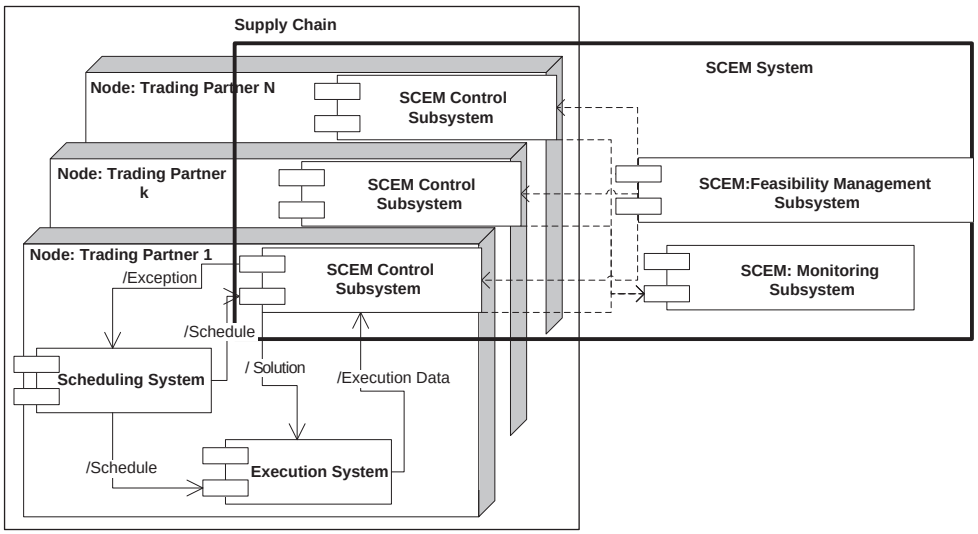

Fig. 2. SCEM system architecture.

4. Reference model for disruptive event management

The proposed reference model provides a common language to describe supply processes and the interaction among them and resources throughout the supply chain. As supply processes depend on resources for their execution, the model has the ability to describe the resources in a generic way, focusing on their feasibility aspects.

The reference model is specified using the unified modeling language UML 2.3 (OMG, 2010). It is represented by four main views: time modeling, supply processes modeling, resources modeling using feasibility dimensions, and SPO requirements modeling.

4.1. Modeling time

To specify a point in time a *TimeSpec* object is used, which can be absolute or relative to current time (Fig. 3). The *TemporalRelation* class allows the time specification exactly "At" a given *referenceValue* or more loosely as "Before" or "After" relationships. For example, $<$ *absoluteTime*=true, *value*=2013-03-15 13:25:00, *referenceValue*=2013-03-18 13:25:00, *temporalRelation*="*Before*"$>$ is a planar tuple representing a *TimeSpec* object and value can be any *dateTime* satisfying that *value*$\leq$*referenceValue*.

The *TimeWindow* class allows representing a time period through *startTime* and *endTime* attributes represented as *TimeSpec* objects. The precedence and synchronization between two entities could be represented through relations defined by *Precedence* or *Synchronization* objects between *startTime* and *endTime* attributes of *TimeWindow* objects.

DateTime objects representing specific time points are mapped to a continuous timeline when evaluating feasibility in Section 5 of this work. A continuous representation of time is more general and includes the discrete representation as a sub-case. Every discrete time point in the timeline of a scheduling horizon can be represented in a continuous timeline.

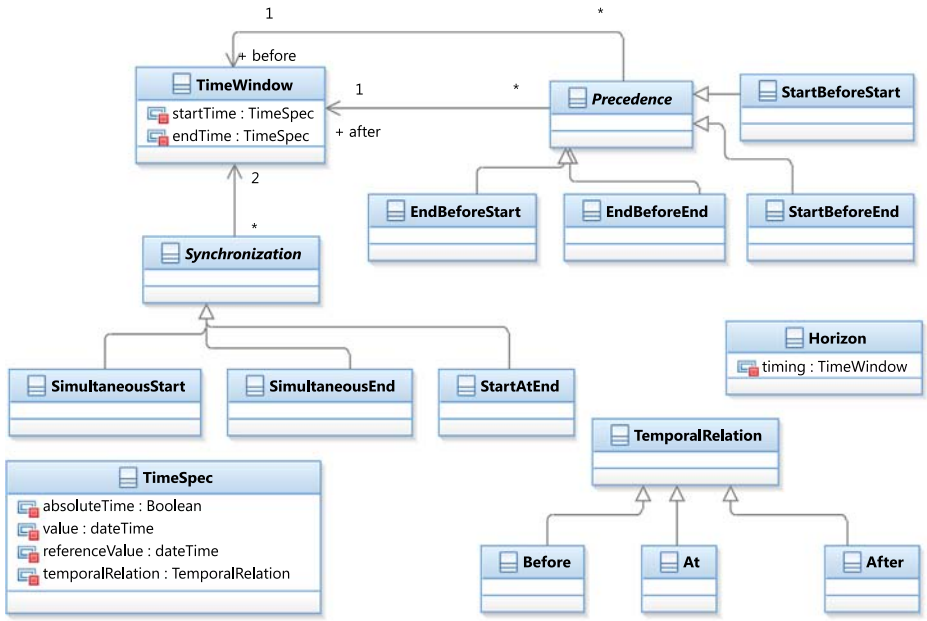

Fig. 3. Time Modeling.

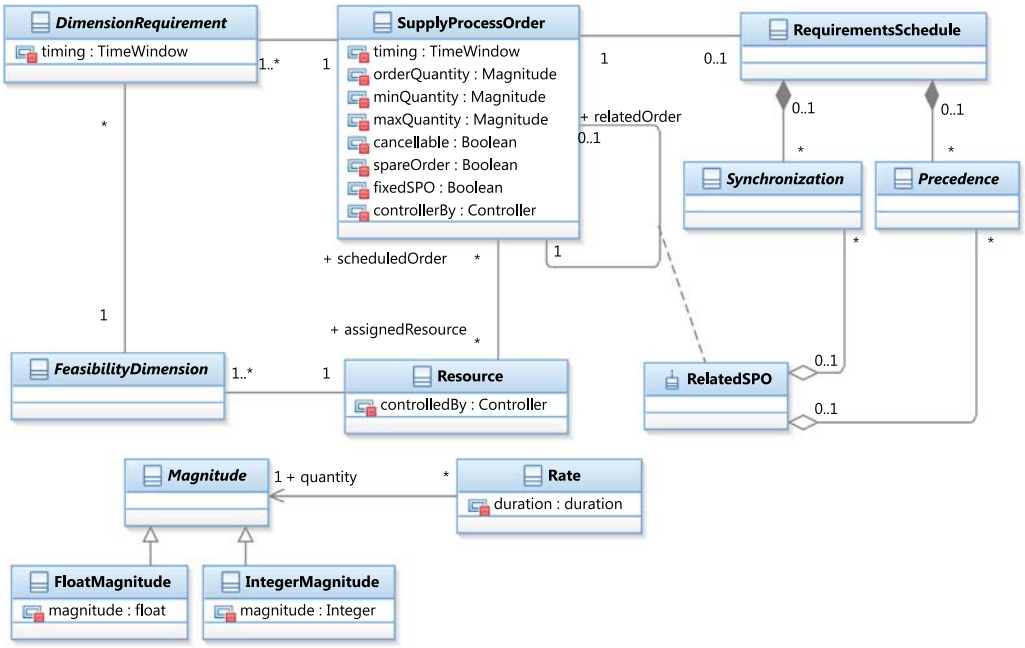

Fig. 4. Supply processes modeling.

Discrete representation of time is useful in several scheduling domains since it reduces the problem complexity and makes the model structure simpler and easier to solve but at expenses of losing generality since the main events are forced to fit in a pre-specified grid of time points. The models of this work can be easily be converted to a discrete time representation by providing pre-specified event points (explained in Section 5) already fixed to the periodic time grid.

4.2. Modeling supply processes

Any instance of a supply process is described by a supply process order coming from a Scheduling System. It is represented through the *SupplyProcessOrder* class (Fig. 4), and is temporally characterized by the *timing* attribute, which is a *TimeWindow*. The size of a SPO is defined by the *orderQuantity* attribute and its slack is described by the *minQuantity* and *maxQuantity* attributes. The *spareOrder* attribute allows the definition of an optional order that could be executed in case of a disruption; the *cancellable* attribute allows specifying that the SPO could be canceled; and the *fixedSPO* attribute allows specifying that the SPO cannot be modified.

Every resource is represented through the *Resource* class. The concept "feasibility dimension", represented through the *FeasibilityDimension* class, is used to describe any resource "ability" to satisfy the requirements of a SPO. For example, the feasibility of a transport unit to comply with the requirements of a transfer process between two geographical points can be described with two *FeasibilityDimension* objects, representing a load capacity and a geographic location.

Thus, the requirements of any *SupplyProcessOrder* on one or more resources can be defined using *DimensionRequirement* objects. A SPO assigned to a resource has a requirement for each feasibility dimension of the resource. In the way of the previous example, if the SPO has a transport unit resource assigned, it will have a requirement over the load capacity and geographical location of the resource (e.g. to be able to load cargo on that location).

A *SupplyProcessOrder* may have several requirements involving one or more *Resource* objects. These requirements may need a temporal orchestration, which is defined with a *RequirementsSchedule* comprising at least one *Precedence* or *Synchronization* object.

A supply process carried out between two enterprises can refer to two SPOs (one per enterprise). This is represented by an association called *RelatedSPO* class, which imposes the condition that both *SupplyProcessOrder* objects have the same value of their *orderQuantity* attribute. The temporal coordination will be defined by *Synchronization* or *Precedence* objects.

4.3. Modeling resources using feasibility dimensions

The feasibility of a *Resource* is described by a set of *FeasibilityDimension* objects specialized in two types, *StateBasedDimension* and *CapacityDimension* (Fig. 5). A *Horizon* defines the overall time window where the availability and requirements associated with the resource are scheduled.

A CapacityDimension represents feasibility aspects described by a numerical magnitude that quantifies a capacity of the resource that fluctuates over time according to the planned availability and to the requirements imposed by its allocated orders (e.g. the inventory of a semi-finished product, the capacity of a tank, number of milling machines available in a manufacturing shop are examples of capacity dimensions). Requirements over capacity dimensions will consume or produce capacity, in the example way: consuming inventory, occupying a milling machine or releasing it and filling and emptying a tank.

A *StateBasedDimension* represents feasibility aspects of a resource whose state may vary over time. Throughout its execution SPOs may require (through the appropriate dimension requirement) a resource to be in a given state. A resource *StateBasedDimension* has a finite discrete set of possible states and a set of feasible transitions between those states. The geographic location of a transport resource and the setup of a machine are examples of state based dimensions.

The availability of a resource over the horizon is modeled by an *AvailabilityProfile* class. Each feasibility dimension of a resource has a unique availability profile, which can be a *ScheduledCapacityProfile* for capacity dimensions or a *ScheduledStatesProfile* for state based dimensions.

4.3.1. Modeling the ScheduledCapacityProfile

The scheduled capacity profile (Fig. 6) is defined by three elements: an ordered set of *AvailableCapacityItem* objects, an ordered set of *CapacityUsageItem* objects and the *currentCapacity* attribute. The ordered set of *AvailableCapacityItem* objects provides minimum and maximum levels of capacity represented by the *maxCapacityBound* and *minCapacityBound* attributes, respectively. This set allows representing the minimum and maximum capacity value a resource can have in different periods within the horizon. These values define upper and lower bounds on both the current availability defined by the *currentCapacity* attribute and the projected availability. The projected availability is the remaining capacity at different times within the horizon when considering the variations due to the satisfaction of the capacity requirements (Fig. 11).

A capacity dimension may impose conditions on its usage. For instance, it may limit the rate, quantity or duration for the consumption or production requirement. The *ScheduledCapacityProfile* is therefore complemented by a set of *CapacityUsageItem* objects that impose bounds to formulate valid requirements.

The feasibility of a capacity dimension is verified when all requirements meet the conditions of the respective *CapacityUsageItem* objects, and the projected capacity satisfies the bounds defined by *AvailableCapacityItems* objects.

4.3.2. Modeling the ScheduledStatesProfile

A scheduled states profile is represented by the *ScheduledStatesProfile* class (Fig. 7). This profile is defined by an ordered set of *ScheduledState* objects, each representing the resource state scheduled for a time period in the corresponding state dimension. Thus, this ordered set defines the schedule of states of the dimension along the horizon.

The current state of the dimension is expressed in the profile by the *current* attribute, which is a *ScheduledState* object, and coincides with the first item of the ordered set of *ScheduledState* objects.

The profile of scheduled states can be specified by *ScheduledState* objects without any dependence among them or by using the relationship *nextState* to represent predefined sequences of states.

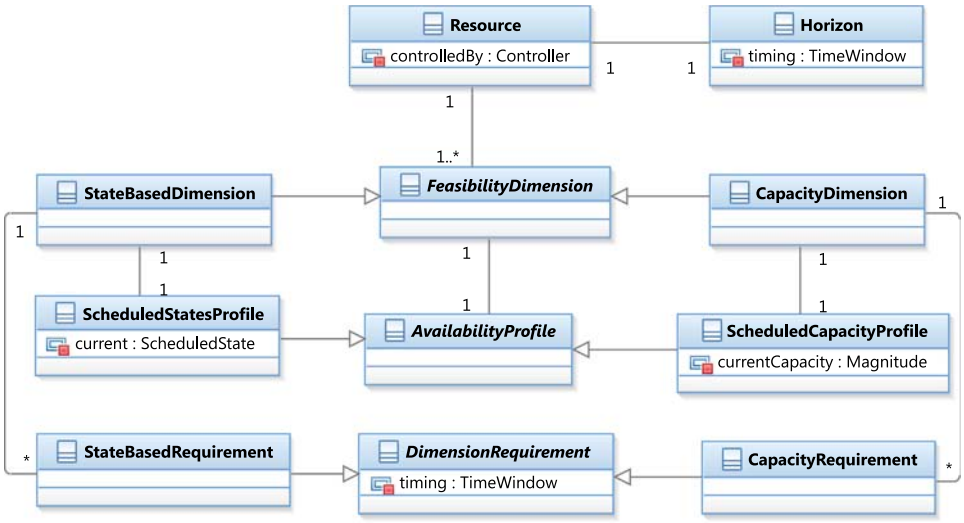

Fig. 5. Resources modeling.

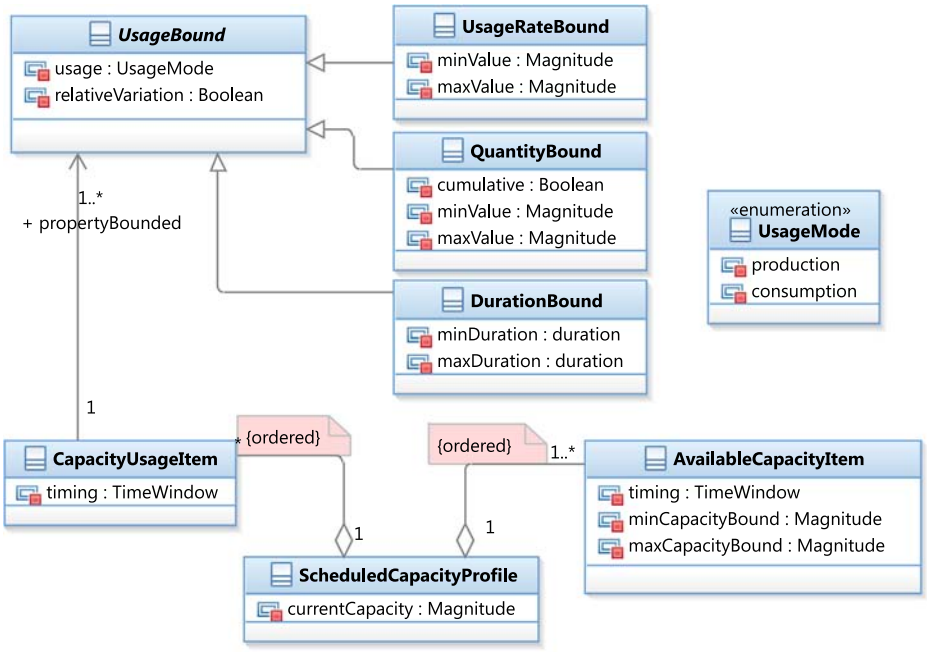

Fig. 6. *ScheduledCapacityProfile* modeling.

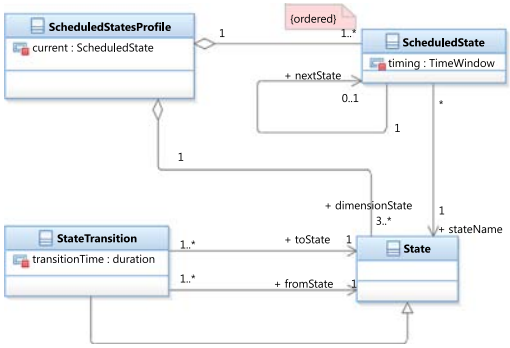

Fig. 7. ScheduledStatesProfile modeling.

A *ScheduledStatesProfile* also includes a description of feasible transitions between states, which are defined using objects of a particular subtype of the *State* class called *StateTransition*. The feasibility of a state dimension is verified when all state requirements are consistent with the scheduled states of the profile, that is to say when state *s1* is required the dimension is in state *s1*, as it was previously scheduled.

4.4. Modeling SPO requirements

Every *SupplyProcessOrder* has a set of requirements over the resources assigned for its fulfillment. As requirements are over feasibility dimensions, an SPO can have both capacity and state requirements, in any case they are generalized by the *DimensionRequirement* concept.

4.4.1. CapacityRequirement

Requirements on a *CapacityDimension* are represented by *CapacityRequirement* objects expressing the production or consumption of capacity according to the *usage attribute* (Fig. 8).

Capacity requirements can be of three types: *instantaneous*, which consumes or produces capacity instantaneously (used to model situations where the duration of the requirement is negligible compared to the duration of the SPO); *Continuous*, which have a consumption or production rate of capacity (Fig. 4); and *Renewable*, which uses a certain capacity in a time period and restores it when the request is completed. A Renewable requirement, if modified during a repair process, can have time slacks represented by *minDuration* and *maxDuration* attributes.

The *reqQuantity* attribute represents the produced or consumed amount of capacity. For every capacity requirement in a SPO its *reqQuantity* is derived from the *orderQuantity* attribute of the SPO. This relationship is defined according to the *scalingMode* and *scalingFactor* of the requirement allowing to model recipe proportions, stoichiometric factors and lot based orders.

4.4.2. State requirement

A state requirement is represented by *StateBasedRequirement* objects and comprises at least one *AtomicStateRequirement*, which defines a state required on a time window (Fig. 9). This representation allows a SPO to require the states of a resource in several ways: an individual state, a set of states with no relationship between them, or a sequence of states defined by *nextState* relations.

4.5. Analysis of the representational capabilities of the Reference Model

The Reference Model as stated previously serves as a modeling language to represent execution schedules. Execution Schedules are the result of solving their associated scheduling problems which can be quite diverse and complex. It is indeed relevant to ask what kind of schedules, result of scheduling problems it can represent. To answer that, we introduce a comparison of the reference model with the scheduling meta-model within ILOG OPL Scheduler (IBM-ILOG, 2009; Le Pape, 1994). The latter is a widely known scheduling problem builder and solver used in many research papers and industrial projects.

In ILOG OPL Scheduler, the building blocks to model a scheduling problem are resources and activities. Activities are simultaneously assigned to resources and sequenced (ordered in time) according to a set of: activity, resource and temporal constraints.

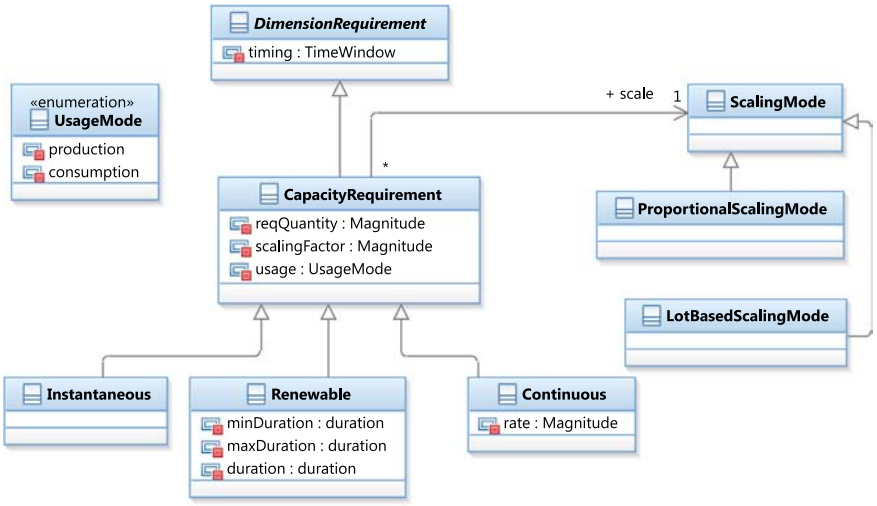

Fig. 8. Capacity requirement modeling.

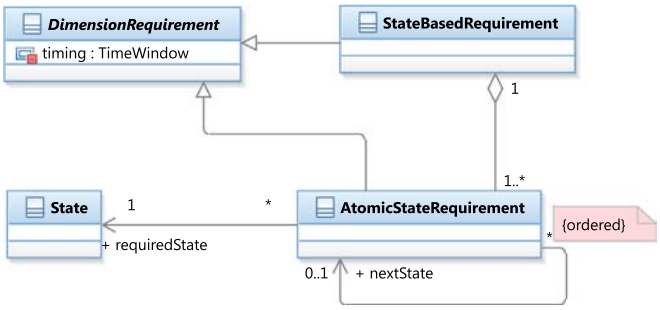

Fig. 9. State requirement modeling.

In any case the result of solving the problem is a set of tasks with start and end time on a given resource.

The Reference Model focuses in supply processes rather than individual tasks. In a supply process a task is represented by a *DimensionRequirement* over a resource feasibility dimension. A Supply Process may contain many requirements for its fulfillment and also a requirements schedule representing the timing and sequencing of these requirements.

In the output of any scheduling problem a task only states how much of a resource is produced, consumed or used and when, therefore we have checked whether it is possible to represent the universe of resources available in ILOG OPL Scheduler.

In Fig. 10 the ILOG OPL Resource meta-model is shown and in Table 2 a description of these modeling entities and how they can be represented using the Reference Model is given.

We highlight that the Reference Model does not represent the complexity of economical and performance objectives of scheduling problems but rather a set of supply process orders and how they use resources for its execution as it is not intended to generate schedules.

5. A Model for Feasibility Check and Restoration

This section presents a model for Feasibility Check and Repair formulated as a Constraint Satisfaction Problem (FCR–CSP) which is able to evaluate and restore feasibility through minimal impact changes in supply chain schedules.

FCR–CSP is a model capable of assessing the feasibility of one or more schedules expressed in terms of the reference model, i.e., every order in the schedule is represented as supply processes modeled with the reference model elements, including its dimension requirements over feasibility dimensions of resources. For every schedule expressed in such way an instance of FCR–CSP capable of assessing its feasibility can automatically be obtained. This is possible due to the way FCR–CSP has been designed: its data model only uses elements present on the Reference Model, and the equations are written over the feasibility dimensions of resources. As a consequence it can handle any kind of schedule as long as it is written in terms of the reference model.

To be able to use FCR–CSP, supply chain partners should implement transformation engines that take the output of their scheduling systems and express it in terms of the reference model. The structure of FCR–CSP consists of a data model representing reference model elements in a suitable form and a constraint model which holds the constraints that ensure a schedule is feasible according to resource availability

An instance of the data model is obtained by exploring the elements of one or more reference model instances (object models representing schedules). To introduce this data model and for its implementation, UML and OCL (Object Constraint Language) which is a part of UML are used. As a consequence sets, parameters and variables are introduced in a platform independent fashion.

5.1. Disruptive events

Disruptive events are classified in two kinds: resource events produced by changes in the availability of a controlled resource; and order events produced by changes in the requirements of a SPO. They can be described using the reference model as follows.

Resource events:

A. Changes in the *ScheduledCapacityProfile* of a resource capacity dimension, including
   - Change in future availability (change in the set of *AvailableCapacityItem*).
   - Change in current available capacity (change in the attribute *currentCapacity* of the dimension).

B. Changes in the *ScheduledStatesProfile* of a resource state based dimension including
   Change in future scheduled states (change in the set of *ScheduledStates*).

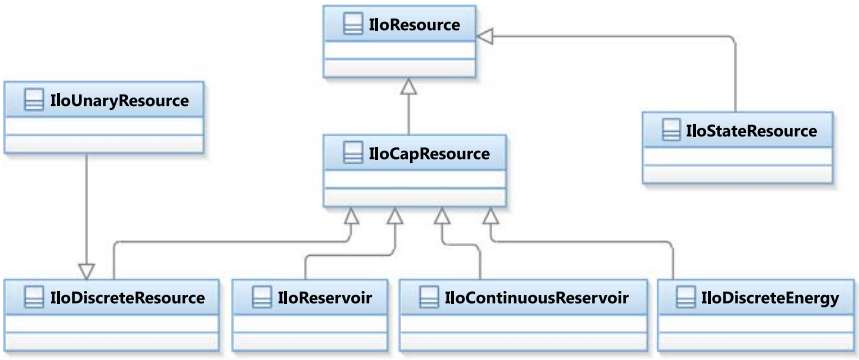

Fig. 10. ILOG OPL Scheduler resources meta-model.

Table 2
Modeling ILOG OPL resources using the Reference Model.

ILOG entity	Reference Model representation
IloDiscreteResource. A discrete resource represents a resource of finite capacity, which is defined to be a positive integer number. The capacity of a discrete resource can vary with time. At any time *t*, the capacity represents the number of copies or instances of the resource that are available. Each activity in a schedule may require some amount of the capacity of a discrete resource	A resource with a capacity dimension holding a *ScheduledCapacityProfile* whose currentCapacity value at the start of the scheduling horizon is its finite capacity (number of available copies of the resource). In general ILOG discrete resources are used in a renewable fashion. So requirements assigned to this type of resource (resource dimension) must be renewable capacity requirements. In cases where the resource is consumed, instantaneous requirements can be used. If there are bounds on the number of copies (units of capacity) of the resource that can be used by a single requirement, these bounds can be represented using capacity usage items holding quantity bounds
IloUnaryResource. Represents a resource whose capacity is one. An instance of IloUnaryResource is either occupied or free. No more than one activity can be executed at a given time	It is modeled just as the discrete resource but with a single unit of capacity and it is used only as a renewable resource. It only fulfills renewable capacity requirements of size one, therefore it is either occupied or free
IloDiscreteEnergy. An instance of the class IloDiscreteEnergy is divided into time buckets (for example, minutes, hours, months, years) that contain a certain amount of energy (for example, watt–hours or human-months) that can be made available over those intervals. The available energy is used by the activities defined on the resource, and as consequence capacity constraints on the discrete energy resource use energy rather than capacity	Suppose that the discrete energy resource has n time buckets. Each time bucket *i* has an available energy of $E(n)$. Then it can be represented using a resource with a capacity dimension whose availability profile also holds n *capacityUsageItems* whose time windows represents the time buckets in the scheduling horizon and hold cumulative quantity bounds (its cumulative Boolean attribute is true) that is: if i is a *capacityUsageItem*, the bounds express that during its time window no more than E(i) units of energy are available to be consumed
IloReservoir. An instance of *IloReservoir* represents a resource that can be dynamically replenished by producing activities. The capacity of an instance of *IloReservoir* can be simultaneously required and provided by activities. This class can be used to model situations in which certain types of activities consume capacity from the reservoir, while others produce capacity A reservoir must not overflow its *maximal capacity* or underflow its *minimal capacity*	A resource with a capacity dimension whose scheduled capacity profile has the following characteristics: Its *currentCapacity* attribute represents the current reservoir occupancy or available capacity. Consuming and producing instantaneous capacity requirements model the corresponding ILOG activities. Its maximal and minimal capacity are represented using an available capacity item with a time window from current time to the end of the scheduling horizon and corresponding min and max capacity bounds
IloContinuousReservoir. An instance of *IloContinuousReservoir* represents a resource which activities can either fill or empty in a continuous and linear process between the start and the end of the activity. If the duration of the activity is null, the filling (or emptying) process is instantaneous (so not continuous). The maximum and minimum levels of a continuous reservoir can vary over time	A resource with a capacity dimension whose scheduled capacity profile has the following characteristics: Its *currentCapacity* attribute represents the current reservoir occupancy or available capacity. The feasible rate for producing or consuming capacity in the continuous linear process is modeled using a capacity usage item with a rate bound, determining min and max rate for production and for consumption The maximum and minimum levels that may vary over time are represented using available capacity items, one for each pair of minimum and maximum levels. Therefore they define time windows for the period where the pairs of levels are valid
IloStateResource. An instance of the class *IloStateResource* represents a resource of infinite capacity whose state can vary over time. Throughout its execution, each activity may require a state resource to be in a given state (or in any of a given set of states). Consequently, two activities may not overlap if they require incompatible states during their execution. A state resource does not constrain the number of activities that can be executed in parallel, provided that the overlapping activities require compatible states of the resource. State transitions of a state resource can be modeled. A state transition can be instantaneous or not (requiring a transition time)	Such resource behavior is represented using *StateBasedDimension* objects. The *ScheduledStatesProfile* of the dimension defines the set of possible states for the resource, the set of feasible transitions and required transition times. In the profile the current state the resource is also shown as well as possible predefined set of states for the resource to be along the schedule, modeled using *ScheduledState* objects Activities, in the reference model captured by the concept of requirements may require to a resource to be in a given set of states. This is modeled using the concept of State Requirement that may be composed by single atomic (individual) state requirements or a set of state requirements for the resource

Change in the current state of a state based dimension (change in the *current* attribute of the *scheduledStatesProfile* of the dimension). Order events

C. Specification change in one SPO (changes in any SPO attribute as: *startTime*, *endTime* or *orderQuantity*).
D. The addition or the cancellation of a SPO.

5.2. A strategy to assess and restore feasibility

The feasibility of a schedule is evaluated by the ability of the resources to fulfill the requirements imposed by the set of SPOs. In the reference model, the requirements of a *SPO* and the availability of resources are defined as different entities. Both of

them can be independently specified along the horizon. The feasibility of the schedule depends on the matching of their specifications.

This is done by modeling how the requirements use the resource availability along the horizon. This strategy is implemented by introducing a set of variables modeling the use of every resource's dimension by SPO requirements, and a set of constraints modeling how those requirements affect their availability. Thus, detecting infeasibility is reduced to detect infeasibility of the constraints model, and repairing a damaged schedule is to allow these variables to use slacks, generating changes in one or more SPOs to restore feasibility.

FCR–CSP is composed by three models: a capacity dimension model, a state based dimension model and a main model associating them. The main model links the capacity and state based dimension models through a set of constraints relating capacity and state requirements from both types of dimension models.

5.3. FCR–CSP: capacity dimension model

The availability of a capacity dimension is modeled by a *ScheduledCapacityProfile* object, which has a *currentCapacity* attribute modeling the available capacity at current time that by convention is the beginning of the horizon. A capacity dimension is feasible when the projected value of capacity after considering the requirements lies within the bounds defined by the set of *AvailableCapacityItem* objects (that define min and max bounds along the horizon). Also, the requirement characteristics should satisfy the constraints imposed by the *CapacityUsageItem* objects.

To calculate the projection of capacity along the horizon, a continuous timeline is assumed. All significant time points along this timeline are modeled as an ordered set of *eventPoints* (this concept is explained by (Ierapetritou and Floudas, 1998) and approaches using it are reviewed in (Méndez et al., 2006)). Therefore, capacity requirements are assigned to start or end at one unique *eventPoint*.

The calculation of projected capacity at each *eventPoint* is done using a capacity balance starting from the projected value in the previous point and adding the consumptions or productions (made by possibly assigned requirements) of capacity in the current point.

EventPoints are assigned to *availableCapacityItems* which provide the capacity bound against the projection will be verified. These assignments define a time window for each *eventPoint* (*the time point it represents belongs to this window*) and consequently for each requirement assigned to it as well.

In Fig. 11, an example of a capacity projection over a continuous timeline with *eventPoints* is shown. The availability of the capacity dimension is modeled using two available capacity items *A*1 and *A*2, representing a change in the prescribed availability of the dimension within the scheduling horizon. In the figure a continuous capacity requirement *r*1 is assigned to start and end at *eventPoints* *e*2 and *e*3 so the time points both represent will be the start and end time of this requirement. Also in the figure *eventPoints* *e*1, *e*2 and *e*3 are assigned to available capacity item *A*1 and *e*4, *e*5 and *e*6 to *A*2. According to *A*1 and *A*2 the capacity projection at first group of *eventPoints* has tighter bounds to satisfy than the second group of *eventPoints*.

Every requirement is assigned to a unique capacity usage item, which also defines a time window for the requirement. The usage rules imposed by capacity usage items are checked for every requirement it has assigned. This translates into checking the values for the attributes quantity, duration and rate against the corresponding bounds defined in the capacity usage

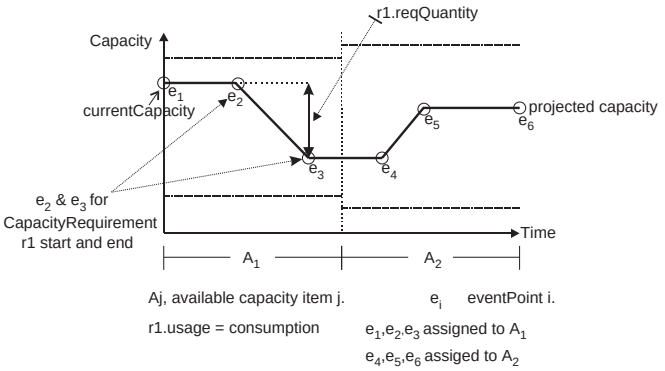

Fig. 11. Capacity projection for a capacity dimension.

item. A capacity dimension is feasible when the projected capacity at each *eventPoint* satisfies the bounds defined by its assigned available capacity item, and the requirements satisfy the usage rules imposed by the capacity usage item they are assigned to.

This capacity dimension model introduces independent binary variables for the following assignments (*r* is a *capacity Requirement*):

i). Every *r.timing.startTime* to one *eventPoint*
ii). Every *r.timing.endTime* to one *eventPoint*
iii). Every *r.timing.startTime* to one *CapacityUsageItem*
iv). Every *r.timing.endTime* to one *CapacityUsageItem*
v). Every *eventPoint* to one *AvailabilityCapacityItem*

All the above assignments imply a temporal compatibility between the related elements, e.g.: if *eventPoint e* is assigned to *availableCapacityItem a*, the time point corresponding to *e* is within the time window defined by *a*. Also if requirement *r* is assigned to start/end at *eventPoint e*, the start time/end time of *r* equals the time point of *e*.

5.3.1. Data model for the capacity dimension model in Tables 3–5

5.3.1.1. Constraints for the capacity dimension model

CD-1: *eventPoints* are timely ordered. As the set of *eventPoints* is ordered, the variables representing the time points they address in a feasible solution keep the same order. This constraint assures time consistency for the different requirements that may take place in every *eventPoint*.

$$t[r,c,e] \geq t[r,c,e-1],\ \forall(r \in R, c \in cDim[r],\ e = 2..e[r,c]:\ \#(cR[r,c]) > 0) \quad (1)$$

CD-2: Every capacity requirement d is assigned to an *eventPoint* from the corresponding capacity dimension

$$\sum(e = 1..e[r,c]) zCRS[d,e] = 1,\ \forall(r \in R, c \in cDim[r],\ d \in cR[r,c] : d.order \notin cSet \quad (2)$$

$$if\ (d \notin instCR) \sum(e = 1..e[r,c]) zCRE[d,e] = 1,\ \forall(r \in R, c \in cDim[r], d \in cR[r,c] : d.order \notin cSet) \quad (3)$$

If a capacity requirement belongs to a canceled SPO, it is not assigned to eventPoints

$$\sum(e = 1..e[r,c]) zCRS[d,e] = 1 - iC[d.order],\ \forall(r \in R, c \in cDim[r], d \in cR[r,c] : d.order \in cSet) \quad (4)$$

Table 3
Sets for the capacity dimension model.

R	set of *Resource* objects
SPO	set of *SupplyProcessOrder* objects
cDim[*r*]	*r*∈*R*, set of *CapacityDimension* objects modeling resource *r*
av[*r*,*c*]	*r*∈*R*, *c*∈cDim[*r*], ordered set of *AvailableCapacityItem* objects for the capacity dimension *c* of resource *r*
cU[*r*,*c*]	*r*∈*R*, *c*∈*cDim*[*r*], ordered set of *CapacityUsageItem* objects of the capacity dimension *c* of resource *r*
allReq	set of *DimensionRequirement* objects in the model
reqs[*o*]	{*d* | *d*∈*allReq*: *d.supplyProcessOrder*=*o*}, *o*∈*spo*. Set of *DimensionRequirement* objects of SPO o
reqs[*r*]	{*d* | *d*∈*allReq*: *d.resource*=*r*}, *r*∈*R*. set of *DimensionRequirement* objects assigned to resource *r*
allCR	{*d* | *d*∈∈*allReq*: *d.oclIsTypeOf*(*CapacityRequirement*)}. set of *CapacityRequirement* objects
cR[*r*,*c*]	= {*d* | *d*∈*allCR*: *d.resource*=*r* & *d.capacityDimension*=*c* }, r∈R, c∈∈cDim[r]. set of *CapacityRequirement* objects assigned to capacity dimension *c* of resource *r*
instCR	= {*d* | *d*∈*allCR*: *d.oclIsTypeOf*(*InstantaneousRequirement*)}. set of instantaneous capacity requirements
renCR	={*d* | *d*∈*allCR*: *d.oclIsTypeOf*(*RenewableRequirement*}. set of renewable requirements
contCR	= {*d* | *d*∈*allCR*: *d.oclIsTypeOf*(*ContinuousRequirement*)}. Set of continuous capacity requirements
timingSet	SPO ∪ allReq∪ av[*r*,*c*] ∪ cU[*r*,*c*], *r* en *R*, *c* en cDim[*r*]. set of entities with timing attribute
cSet	= {*o* | *o* in *SPO*: *mod*[*o*]=1 & *o.cancellable*=*1*}. set of cancelable SPO

Table 4
Parameters for the capacity dimension model.

mod[*o*]	*o* ∈*SPO*. This Boolean parameter states whether a SPO o is under a repair process or not
cCap[*r*,*c*]	*r*∈*R*, *c*∈*cDim*[*r*]. *currentCapacity* attribute of the *ScheduledCapacityProfile* of dimension *c* of resource *r*
e[*r*,*c*]	*r*∈*R*, *c*∈*cDim*[*r*]. number of eventPoints in capacity dimension *c*.
qMin[*d*]	*d*∈allCR. This value satisfies Eqs. (27) and (28) when d.supplyProcessOrder.orderQuantity=d.supplyProcessOrder.minQuantity minimum value for the requirement d quantity (qMin[d]) and is possible only when the SPO takes its minimum value
qMax[*d*]	*d*∈allCR. This value satisfies Eqs. (27) and (28) when d.supplyProcessOrder.orderQuantity=d.supplyProcessOrder.maxQuantity maximum value for the quantity of requirement d (qMax[d]) and is possible only when the SPO takes its maximum value
minB[*r*,*c*]	=*min*{*a.minCapacityBound*}, *a*∈*av*[*r*,*c*], *r*∈∈*R*, *c*∈*cDim*[*r*]. Minimum value allowable for the capacity projection of *c* along the horizon
maxB[*r*,*c*]	=*max*{*a.maxCapacityBound*}, *a* ∈*av*[*r*,*c*], *r*∈∈*R*, *c*∈*cDim*[*r*]. Maximum value allowable for the capacity projection of *c* along the horizon
mUse[*r*,*c*]	*r*∈*R*, *c*∈*cDim*[*r*]. boolean parameter with value 1 when the capacity dimension holds continuous requirements together with other types of requirements

Table 5
Variables for the capacity dimension model.

zAv[*a*,*e*]	*r*∈∈*R*, *c*∈*cDim*[*r*], *a*∈∈*av*[*r*,*c*], e∈ *eventPoints*[*r*,*c*]. Binary variable with value 1 when eventPoint e is assigned to *AvailableCapacityItem* a of *CapacityDimension c* from resource *r*
zUS[u,*d*] zUE[u,*d*]	*r*∈∈*R*, *c* ∈*cDim*[*r*], *u*∈∈*CapUsageItems*[*r*],*d*∈∈cR[*r*,*c*]. Binary variables with value 1 when requirement d is assigned to start (zUS) or end (zUE) within capacity usage item u time window
t[*r*,*c*,e]	*r*∈∈*R*, *c* ∈*cDim*[*r*], *e*=1…*e*[*r*,*c*]. Time point corresponding to eventPoint e. With domain [r.horizon.startTime.value, r.horizon.value]
Q[*o*]	*o*∈∈*SPO*. Quantity variable for SPO o. With domain [0, o.maxQuantity]
Q[*d*]	*d*∈ *allCR*. Quantity variable for requirement d. With domain [0, qMax[*d*]]. If *d*.scalingMode=LotBasedScalingMode Q[*d*] is a non-negative integer variable.
zCRS[*d*,e] *zCRE*[*d*,e]	*d*∈∈*allCR*, *e*=1…e[*d*.resource,*d*.feasibilityDimension]. Binary variables denoting the eventPoint where requirement d starts (zCRS) or ends (zCRE) respectively.
cRR[*d*]	*d*∈∈∈*contCR*.Variable representing the rate of consumption/production of requirement d. With domain [d.minUsageRate, d.maxUsageRate]
pC [*r*,*c*,e]	*r*∈∈*R*, *c*∈ *cDim*[*r*], *e*=1…*e*[*r*,*c*]. Projected capacity variable for dimencion c of resource r at eventPoint e
iC[*o*]	*o*∈ *SPO*. Binary variable with value 1 if SPO o is canceled

$$if(d\notin InstCR)\sum(e=1..e[r,c])zCRE[d,e]=1-iC[d.order],\quad \forall(r\in R,c\in cDim[r],d\in cR[r,c]: d.order\in cSet) \tag{5}$$

CD-3: Every *eventPoint* e has at most one capacity requirement assigned

$$\sum(d\in cR[r,c])\,(zCRS[d,e]+zCRE[d,e])\le 1,\forall(r\in R,c\in cDim[r],e=1\ldots e[r,c]) \tag{6}$$

CD-4: If capacity requirement d is assigned to end at *eventPoint* e, it has to be assigned to start at any $e' < e$

$$\sum(e'=1..e[r,c]: e'<e)\,zCRS[d,e']\ge zCRE[d,e],\quad \forall(r\in R,c\in cDim[r],d\in cR[r,c],e=1\ldots e[r,c]) \tag{7}$$

CD-5: If *capacity dimension* c has *Continuous capacity requirements* mixed with other types of requirements and eventPoint e has been assigned to be the start or the end of a requirement; then eventPoint e-1 must not be assigned to any requirement's start or end. This constraint allows checking whether the *capacity dimension* is feasible just before another requirement is attended.

$$\sum(d\in cR[r,c])(zCRS[d,e-1]+zCRE[d,e-1]) \le 1-\sum(d\in cR[r,c])(zCRS[e]+zCRE[e]),\quad \forall(r\in R,c\in cDim[r],e=2\ldots e[r,c]: mUse[r,c]=1) \tag{8}$$

CD-6: If capacity requirement d is assigned to *eventPoint* e, their times are equal.

$$\forall(r\in R,c\in cDim[r],d\in cR[r,c],e=1..e[r,c])\quad if(zCRS[d,e]=1)start[d]=t[r,c,e],\forall(r\in R,c\in cDim[r],\quad d\in cR[r,c],e=1\ldots e[r,c]) \tag{9}$$

$$if(zCRE[d,e]=1)\{end[d]=t[r,c,e]\},\quad \forall(r\in R,c\in cDim[r],d\in cR[r,c],e=1\ldots e[r,c]) \tag{10}$$

CD-7: Every *eventPoint* is assigned to a unique available capacity item.

$$\sum(a\in av[r,c])\, zAv[a,e]=1,\forall(r\in R,c\in cDim[r],e=1\ldots e[r,c]) \quad (11)$$

CD-8: If eventPoint e is assigned to an available capacityItem a, then their times are related

$$if(zAv[a,e]=1)start[a]\le t[r,c,e];\ end[a]\ge t[r,c,e],\ \forall(r\in R,c\in cDim[r],a\in av[r,c],e=1..e[r,c]) \quad (12)$$

CD-9: Capacity projection for dimension c of resource r.

$$\begin{aligned}pC[r,c,1]&=cCap[c.r]+\sum(d\in instCR\\ &: d\in cR[r,c]\&d.usageMode=(\text{Production}))Q[d]^*zCRS[d,1]\\ &+\sum(d\in instCR : d\in cR[r,c]\&d.usageMode\\ &=(Consumption))(-Q[d]^*zCRS[d,1])\\ &+\sum(d\in renCR : d\in cR[r,c]\&\ d.usageMode\\ &=(\text{Production}))(Q[d]^*zCRS[d,1]-Q[d]^*zCRE[d,1])\\ &+\sum(d\in renCR : d\in cR[r,c]\&d.usageMode\\ &=(Consumption))(-Q[d]^*zCRS[d,1]\\ &+Q[d]^*zCRE[d,1]),\forall(r\in R,c\in cDim[r])\end{aligned} \quad (13)$$

$$\begin{aligned}&\text{pC}[r,c,e]=\text{pC}[r,c,e-1]+\sum(\text{d}\in\text{instCR} : \text{d}\in\text{cR}[r,c]\&\ \text{d.usageMode}\\ &=(\text{Production}))(Q[d]^*\text{zCRS}[d,e])\\ &+\sum(\text{d}\in\text{instCR} : \text{d}\in\text{cR}[r,c]\&\ \text{d.usageMode}\\ &=(\text{Consumption}))(-Q[d]^*\text{zCRS}[d,e])\\ &+\sum(\text{d}\in\text{renCR} : \text{d}\in\text{cR}[r,c]\&\ \text{d.usageMode}\\ &=(\text{Production}))(Q[d]^*\text{zCRS}[d,e]-\ Q[d]^*\text{zCRE}[d,e])\\ &+\sum(\text{d}\in\text{renC} : \text{d}\in\text{cR}[r,c]\&\ \text{d.usageMode}\\ &=(\text{Consumption}))(-Q[d]^*\text{zCRS}[d,e]+Q[d]^*\text{zCRE}[d,e])\\ &+\sum(\text{d in contCR} : \text{din cR}[r,c]\&\ \text{d.usageMode}=(\text{Production}))\\ &[(t[r,c,e]-t[r,c,\text{e}-1])^*\text{cRR}[d]^*(1-\sum(\text{e}'\text{ in e}[r,c] :\\ &\text{e}'\le\text{e}-1)(\text{zCRS}[d,\text{e}']-\text{zCRE}[d,\text{e}-1]))]\\ &+\sum(\text{d in contCR :}\ \text{d in cR}[r,c]\&\ \text{d.usageMode}\\ &=(\text{Consumption}))\\ &[-(t[r,c,e]-t[r,c,\text{e}-1])^*\text{cRR}[d]^*(1-\sum(\text{e}'\text{ in e}[r,c] :\\ &\text{e}'\le\text{e}-1)(\text{zCRS}[d,\text{e}']-\text{zCRE}[d,\text{e}-1]))],\\ &\forall(\text{r}\in\text{R},\text{c}\in\text{cDim}[r],\text{e}=2..e[r,c])\end{aligned} \quad (14)$$

CD-10: The projected capacity variable (pC) must satisfy the available capacity items bounds along the horizon.

$$\forall(r\in R,c\in cDim[r],e=1\ldots e[r,c])\ pC[r,c,e]\le\sum(a\in av[r,c])(a.maxCapacityBound^*zAv[a,e]),\ \forall(r\in R,c\in cDim[r],e=1\ldots e[r,c]) \quad (15)$$

$$pC[r,c,e]\ge\sum(a\in av[c.r])(a.\min CapacityBound^*zAv[a,e]),\ \forall(r\in R,c\in cDim[r],e=1\ldots e[r,c]) \quad (16)$$

CD-11: Capacity requirements are assigned to capacity usage items of the corresponding capacity dimension.

$$\sum(u\in cU[r,c])zUS[u,d]=1,\forall(r\in R,c\in cDim[r],d\in cR[r,c] : d.order\notin cSet) \quad (17)$$

$$if(d\notin instCR)\sum(u\in cU[r,c])zUE[u,d]=1,\ \forall(r\in R,c\in cDim[r],d\in cR[r,c] : d.order\notin cSet) \quad (18)$$

If the SPO to which a requirement belongs is canceled such assignment does not take place

$$\sum(u\in cU[r,c])zUS[u,d]=1-iC[o],\ \forall(r\in R,c\in cDim[r],d\in cR[r,c] : d.order\in cSet)$$

$$if(d\notin instCR)\sum(u\in cU[r,c])zUE[u,d]=1-iC[o],\ \forall(r\in R,c\in cDim[r],d\in cR[r,c] : d.order\in cSet) \quad (19)$$

CD-12: If requirement d is assigned to start or end within capacity usage item u time window, its startTime/endTime belongs to that time window

$$if(ZUS[u,d]=1)\{start[u]\le start[d];\ end[u]\ge start[d]\},\ \forall(r\in R,c\in cDim[r],u\in cU[r,c],d\in allCR : d\in cR[r,c]) \quad (20)$$

$$if(zUE[u,d]=1)start[u]\le end[d];\ end[u]\ge end[d],\ \forall(r\in R,c\in cDim[r],u\in cU[r,c],d\in allCR : d\in cR[r,c]) \quad (21)$$

CD-13: The quantity of a SPO remains as planned unless it is under a repair process

$$if(\text{mod}[o]=0)Q[o]=o.orderQuantity,\forall(o\in SPO) \quad (22)$$

$$o.\min Quantity\le Q[o]\le o.\max Quantity,\ \forall(o\in SPO) \quad (23)$$

$$Q[o]\ge o.quantityLowerBound-o.\max Quantity^*(iC[o]),\ \forall(o\in SPO : o\in cSet) \quad (24)$$

$$Q[o]\le o.quantityUpperBound+o.\max Quantity^*(iC[o]),\ \forall(o\in SPO : o\in cSet) \quad (25)$$

$$Q[o]\le o.quantityUpperBound^*(iC[o]),\forall(o\in SPO : o\in cSet) \quad (26)$$

CD-14: The quantity variable of every capacity requirement is related to the quantity variable of the SPO holding the requirement.

$$if(d.scalingMode=\text{ProportionalScalingMode})Q[d]=d.scalingFactor^*Q[o],\ \forall(o\in SPO,d\in reqs[o] : d\in allCR) \quad (27)$$

$$if(d.scalingMode=LotBasedScalingMode)\ d.scalingFactor^*(Q[d]-1)+1\le Q[o]\le Q[d]^*d.scalingFactor,\ \forall(o\in SPO,d\in reqs[o] : d\in allCR) \quad (28)$$

CD-15: Instantaneous requirements start and end times are equal.

$$end[d]=start[d],\forall(d\in instCR \quad (29)$$

CD-16: A renewable requirement duration remains as planned, unless its associated SPO is under a repair process. In such cases is defined by the difference between end and start times.

$$if(\text{mod}[d.\text{supply}\text{ProcessOrder}]=0)d.duration=end[d]-start[d],\ \forall(d\in renCR : d.order\notin cSet) \quad (30)$$

$$d.\min Duration\le end[d]-start[d]\le d.\max Duration,\forall(d\in renCR : d.order\notin cSet) \quad (31)$$

$$if(iC[d.order]=0)d.\min Duration\le end[d]-start[d]\le d.\max Duration,\ \forall(d\in renCR : d.order\in cSet) \quad (32)$$

CD-17: If a capacity usage item specifies bounds for required quantities, then requirement quantities must fulfill the following constraints (shown only for the case of a production requirement, but valid for consumptions as well).

if(d.usageMode = Production){

$$Q[d] \le \sum(u \text{ in } cU[r,c])(u.productionQuantity.\max Value^{*} zUS[u,d]);$$
$$Q[d] \ge \sum(u \text{ in } cU[r,c])(u.productionQuantity.\min Value^{*} zUS[u,d]);$$
$$Q[d] \le \sum(u \text{ in } cU[r,c])(u.productionQuantity.\max Value^{*} zUE[u,d])$$
$$Q[d] \ge \sum(u \text{ in } cU[r,c])(u.productionQuantity.\min Value^{*} zUE[u,d])$$
$$\forall(r \in R, c \in cDim[r], d \in cR[r,c]) \quad (33)$$

CD-18: The duration of a renewable requirement is verified using the capacity usage item assigned for its start time.

$$(end[d] - start[d]) \le \sum(u \in cU[r,c])(u.productionDuration.\max Value^{*} zUS[u,d]),$$
$$\forall(r \in R, c \in cDim[r], d \in cR[r,c]) \quad (34)$$

$$(end[d] - start[d]) \ge \sum(u \in cU[r,c])(u.productionDuration.\min Value^{*} zUS[u,d]),$$
$$\forall(r \in R, c \in cDim[r], d \in cR[r,c]) \quad (35)$$

If capacity usage items constraint the rate of any continuous requirement d, such rate is verified with respect to every capacity usage item that spans the duration of the requirement. The constraint for the case where d.usage = production, and is valid also for consumptions.

$$cRR[d] \le k.productionRate.\max Value + (d.\max Rate - d.\min Rate)^{*}(2 - ZUS[i,d] - ZUE[j,d]);$$
$$cRR[d] \ge k.productionRate.\min Value - (d.\min Rate - d.\min Rate)^{*}(2 - ZUS[i,d] - ZUE[j,d]),$$
$$\forall(r \in R, c \in cDim[r], d \in contCR : d \in cR[r,c] \,\&\, d.usageMode = (\text{Production})),$$
$$\forall(i,j,k \text{ in } cU[r,c] : i \le k \le j) \quad (36)$$

5.4. FCR–CSP: state based dimension model

The availability of a *StateBasedDimension* is modeled by sets of *ScheduledState* objects defining which states the dimension must take along the horizon. The attribute *current* shows the current *ScheduledState* which could be a *StateTransition* or a "*proper state*" (a state which is not a *StateTransition*).

The requirements for a state based dimension require a specific state during a time window. Therefore the feasibility of the resource will depend on the compatibility of required states and the states the dimension will take along the horizon.

The strategy of the model for a state based dimension consists in the generation of a set of blocks used as markers of the start and end time of a state. These blocks are time ordered and have assigned a unique state from the set of possible states of the dimension. For example in Fig. 12 the *x*-axis shows a set of blocks, and the *y*-axis has the set of states they may take. The assigned states are painted.

This ordered set of blocks with their states determines a state projection for the dimension. To be feasible a state based dimension has to be able to attend both the scheduled states and the state requirements (described in Sections 4.3.2 and 4.4.2). For that purpose each scheduled state of the availability profile of the dimension is assigned to one block. Every block has a single state from the set of states of the profile. The assignment of the current and another scheduled state to blocks 1 and 2 is shown in Fig. 12.

Atomic state requirements are also assigned to a unique block, but there is no limit in the number of requirements assigned to one block. Therefore every state based requirement should have every one of its atomic requirements assigned to blocks. Finally those blocks should satisfy the *nextState* relationships imposed by requirements and by scheduled states as well, Fig. 12 shows a valid state requirement assignment.

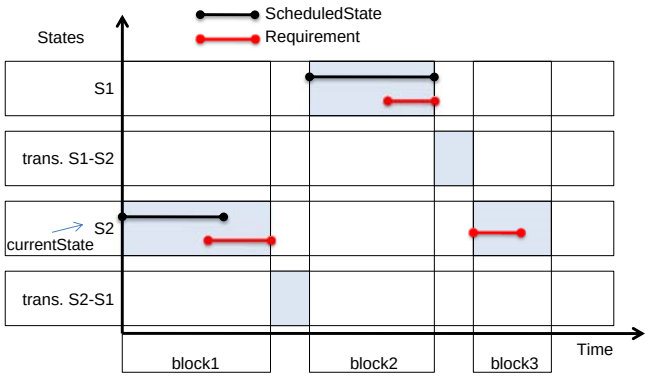

Fig. 12. Blocks for a state based dimension model.

The scheduled states profile might not specify states for the whole horizon. In such case it is possible for the dimension to take unplanned states without affecting scheduled states. This flexibility constitutes a useful slack to absorb disruptions.

In order to implement the aforementioned strategy the state based dimension model introduces binary variables for the following assignments:

i) *One state to every block.*
ii) *Every atomic state requirement to one block.*
iii) *Every scheduled state to a dedicated block.*

All the above assignments imply a temporal compatibility between the related elements.

5.4.1. Data model for the state based dimension model in Tables 6–8

5.4.1.1. Constraints for the state based dimension model

SD-1 Every scheduled state is assigned to a block of the state dimension, and conversely a block has no more than one scheduled state assigned.

$$\sum(i = 1..nB[r,s]) wBSc[i,d] = 1, \forall(r \in R, s \in sDim[r], j \in sched[r,s]) \quad (37)$$

$$\sum(j \in sched[r,s]) wBSc[i,d] \le 1, \forall(r \in R, s \in sDim[r,s], i = 1..nB[r,s]) \quad (38)$$

SD-2 The first block corresponds to the current attribute of the scheduled states profile of the dimension

$$wBSc[1,j] = 1, \forall(r \in R, s \in sDim[r], j \in sched[r,s] : j = s.scheduledStates\text{Profile}.current) \quad (39)$$

SD-3 Let *a*, *b* be scheduled state objects, if *b* is *nextState* of *a*, both are assigned to consecutive blocks

$$wBSc[i-1,a] \le wBSc[i,b], \forall(r \in R, s \in sDim[r], i = 2..nB[r,s], a,b \in sched[r,s] : b = a.nextScheduledState) \quad (40)$$

SD-4 Every block determines a unique state in the set of projected states (blocks holding a state), that is: the block has an unique state, and if it has been assigned to ScheduledState *j*, it has the state of *j*, *j*.state

$$\sum(k \in sSet[r,s] : if(i \ge 2) k \in properS) wBS[i,k] = 1, \forall(r \in R, s \in sDim[r], i = 1 \ldots nB[r,s]) \quad (41)$$

$$wBS[i,j.state] \ge wBSc[i,j], \forall(r \in R, s \in sDim[r], i = 1 \ldots nB[r,s], j \in sched[r,s]) \quad (42)$$

Table 6
Sets for the state based dimension model.

sDim[*r*]	*r*∈*R*. set of StateBasedDimension objects in resource *r*
sSet[*r*,*s*]	*r*∈*R*, *s*∈sDim[*r*]. Set of State objects of the dimension *s*
sched[*r*,*s*]	*r*∈*R*, *s*∈sDim[*r*]. Set of ScheduledState objects of the dimension *s*
sReq[*r*,*s*]	*r*∈*R*, *s*∈sDim[*r*]. Set of StateBasedRequirement objects of the dimension. Each element l in stateReq[*r*,*s*] is composed by a set of AtomicStateRequirement objects
properS[*r*,*s*]	*r*∈*R*, *s*∈sDim[*r*]. Set of states of the dimension which are not of the type TransitionState

Table 7
Parameters for the state based dimension model.

fTran[*s*1,*s*2],	*r*∈*R*, *s*∈*sDim*[*r*], s1∈*sSet*[*r*,*s*], s2∈ *sSet*[*r*,*s*]: s1,s2∈proprtS[*r*,*s*]
fTT[*s*1,*s*2]	fTran[*s*1,*s*2] is a boolean parameter with value 1 if there is a transition from properState s1 to properState s2, with value 0 otherwise fTT[s1,s2] is the transition time from s1 to s2
nB[*r*,*s*]	*r*∈*R*, *s*∈*sDim*[*r*]. Number of blocks for dimension *s*
hL[*r*]	Horizon length

Table 8
Variables for the state based dimension model.

start[*k*], end[*k*]	*k*∈{scheduleStates[*r*,*s*] ∪ {a, a∈ l, l ∈sReq[*r*,*s*]} *r* ∈*R*, *s*∈sDim[*r*]}
wBSc[i,j]	r∈R, s∈sDim[r], i=1…nB[r,s], j∈sched[r,s]. Boolean variable with value 1 if scheduledState j is assigned to block *i*
wBS[i,k]	r∈R, s∈sDim[r], *i*=1…nB[*r*,*s*], k ∈sSet[r,s]: if (*i*≥2) *s*∈properS. Boolean variable with value 1 if block i assumes state k, 0 otherwise
tS[s,i],tE[s,i]	r∈R, s∈sDim[r], i=1…nB[r,s]. With domain [r.horizon.startTime.referenceValue, r.horizon.endTime.referenceValue]. tS is the start and tE the end time of block i of dimension s
tranT[s,i,i′],	r∈*R*, s∈sDim[r], i, i′=1…nB[*r*,*s*]: i′=i+1. transition time from block i to block i′ of dimension s. A calculated variable that considers the final assumption of states for consecutive blocks i and i′
tran[s,i,i′,k,k′]	r∈R, s∈sDim[r], i, i′=1...nB[r,s], k, k′∈sSet[r,s]: i′=i+1 & k′∈ properS and if(*i* > 1) k∈properS. Boolean variable with value 1 if there is a transition from state k to k′ between consecutive blocks i and i′
wSR[d,i]	r in R, s in sDim[r], l in sReq[r,s], d en l, i en 1…nB[r,s]. *Boolean variable with value 1 if atomic state requirement d is assigned to block i of dimension s*

SD-5 Block *startTime* and *endTime*

$$tS[s,i] \le tE[s,i], \forall (r \in R, s \in sDim[r], i = 1..nB[r,s]),$$
$$\forall (r \in R, s \in sDim[r], i = 1 \ldots nB[r,s]) \quad (43)$$

SD-6 As the current scheduled state is assigned to the first block of the state based dimension, the following time relationship holds.

$$\forall (r \in R, s \in sDim[r], j \in sched[r,s] : j = s.scheduledStates\text{Pro}file.current)$$
$$tS[s,1] = start[j], \forall (r \in R, s \in sDim[r], j \in sched[r,s] :$$
$$j = s.scheduledStates\text{Pro}file.current) \quad (44)$$

$$tE[s,1] \ge end[j], \forall (r \in R, s \in sDim[r], j \in sched[r,s]$$
$$: j = s.scheduledStates\text{Pro}file.current) \quad (45)$$

SD-7 If scheduled state *j* is assigned to block i the scheduled state timing is within the time window of *i*.

$$tS[s,i] \le start[j] + hL[r]^*(1 - wBSc[i,j]), \forall (r \in R, s \in sDim[r],$$
$$i = 1 \ldots nB[r,s], j \in sched[r,s]) \quad (46)$$

$$tE[s,i] \ge end[j] - hL[r]^*(1 - wBSc[i,j]), \forall (r \in R, s \in sDim[r],$$
$$i = 1 \ldots nB[r,s], j \in sched[r,s]) \quad (47)$$

SD-8 If two consecutive blocks have different states assigned there must be a feasible transition between them.

$$wBS[i,t] \le 1 - (wBS[i-1,k] - fTran[k,t]), \forall (r \in R, s \in sDim[r],$$
$$i = 2 \ldots nB[r,s], k, t \in properS[r,s]) \quad (48)$$

SD-9 The following constraints determine whether there is a transition between state *k* to *k*′ when going from block i to *i*+1. When the current attribute is a state transition, it is assumed that the transition is taking place and there is no transition to the next block.

$$tran[s,i,i',k,k'] \ge wBS[i,k] + wBS[i',k'] - 1; tran[s,i,i',k,k']$$
$$\le wBS[i,k]; tran[s,i,i',k,k'] \le wBS[i',k'],$$
$$\forall (r \in R, s \in sDim[r], i, i' = 1 \ldots nB[r,s], k, k' \in sSet[r,s] : i' =$$
$$i + 1 k, k' \in properS) \quad (49)$$

$$tran[s,1,2,k,k'] = 0, \forall (r \in R, s \in sDim[r], k, k' \in in\ sSet[r,s]$$
$$: k' \in properS\ and\ k \notin properS) \quad (50)$$

SD-10 Transition times between blocks are calculated variables according to the state transitions that take place when going from block *i* to block *i*+1. These transition times affect the start and end times of every block.

$$tranT[s,i,i'] = \sum (k,k' \in properS) tran[s,i,i',k,k']^* fTT[k,k'],$$
$$\forall (r \in R, s \in sDimr, [i, i' = 1 \ldots nB[r,s] : i' = i+1) \quad (51)$$

$$tS[s,i'] \ge tE[s,i] + tranT[s,i,i'], \forall (r \in R, s \in sDim[r], i,$$
$$i' = 1 \ldots nB[r,s] : i' = i+1) \quad (52)$$

SD-11 Every atomic state requirement is assigned to a state base dimension block having a compatible state.

$$\sum (i = 1 \ldots nB[r,s]) wSR[d,i] = 1, \forall (r \in R, s \in sDim[r],$$

Table 9
Sets for the main model.

TimingSet	SPO ∪ allReq∪ av[*r*,*c*] ∪ cU[*r*,*c*] ∪scheduleStates[*r*,*s*] ∪ {a: a∈ l, l∈∈sReq[*r*,*s*]} *r* in *R*, *s*∈∈sDim[*r*], *c*∈∈cDim[*r*]. set of entities with a timing attribute
ReqScheduleSet	set of *RequirementsSchedule* objects (schedule to be analyzed with FCR–CSP)

Table 10
Variables for the main model.

start[*k*], *end*[*k*]	*k*∈∈*timingSet*. Variables for the start and end time of all entities with a *timing attribute*. With domain [r.horizon.timing.startTime.value, r.horizon.timing.endTime.value]

$$l \in sReq[r,s], d \in l : l.order \notin cSet) \quad (53)$$

$$\sum(i = 1 \ldots nB[r,s]) wSR[d,i] = 1 - iC[l.order], \quad \forall(r \in R, s \in sDim[r], l \in sReq[r,s], d \in l : l.order \in cSet) \quad (54)$$

$$wSR[d,i] \le wBS[i, d.requiredState], \quad \forall(r \in R, s \in sDim[r], l \in sReq[r,s], d \in l, i = 1 \ldots nB[r,s]) \quad (55)$$

SD-12 Atomic state requirement objets having a *nextState* are assigned to consecutive blocks.

$$wSR[d1, i-1] \ge wSR[d2,i], \forall(r \in R\ s \in sDim[r], l \in sReq[r,s], d1, d2 \in l, i = 2 \ldots nB[r,s] : d2 = d1.nextState) \quad (56)$$

SD-13 An atomic state requirement starts and ends within its assigned block start and end times

$$start[d] \ge tS[s,i] - hL[r]^*(1 - wSR[d,i]), \forall(r \in R, s \in sDim[r], l \in sReq[r,s], d \in l, i = 1 \ldots nB[r,s]) \quad (57)$$

$$end[d] \le tE[s,i] + hL[r]^*(1 - wSR[d,i]), \quad \forall(r \in R, s \in sDim[r], l \in sReq[r,s], d \in l, i = 1 \ldots nB[r,s]) \quad (58)$$

5.5. FCR–CSP main model

Every instance of the reference model has a variety of *Timing* objects defining time windows and time points in the horizon of resources. These timing objects are often interrelated as in an SPO's *requirementSchedule* that holds synchronizations or precedencies amongst the requirements of an SPO. The main model captures these time relationships and individual timing constraints; as it checks the coordination of both types of requirements (capacity and state requirements), it comprises the coordination of both types of feasibility dimensions as well.

5.5.1. Data model for the main model in Tables 9–10

Glob-1 The following set of constraints model the feasible values of every object in an instance of the reference model that holds a *timing* attribute with respect to its start and end times.

$$start[k] \le end[k] \quad (59)$$

$$if(k.startTime.temporalRelation = At) start[k] = k.startTime.referenceValue \quad (60)$$

$$if(k.startTime.temporalRelation = Before) start[k] \le k.startTime.referenceValue \quad (61)$$

$$if(k.startTime.temporalRelation = After) start[k] \ge k.startTime.referenceValue \quad (63)$$

$$if(k.endTime.temporalRelation = Before) end[k] \le k.endTime.referenceValue \quad (64)$$

$$if(k.endTime.temporalRelation = After) end[k] \ge k.endTime.referenceValue \quad (65)$$

(59)–(65), ∀(k∈∈timingSet)

Glob-2 Unless a SPO is allowed to changes its specification, its timing attribute keeps its originally planned values without using any slack.

$$if(\mathrm{mod}[o] = 0) start[o] = o.startTime.value; end[o] = o.endTime.value, \forall(o \in SPO) \quad (66)$$

Glob-3 The timing of a requirement is bounded by the timing of its associated SPO.

$$start[d] \ge start[o]; end[d] \le end[o], \forall(o \in SPO, d\ i \in reqs[o]) \quad (67)$$

If a SPO o is within a repair process constraints 0 are relaxed for every requirement k of the SPO o. So requirements times move freely within the SPO times as long as the requirement schedule (Glob-4 and Glob-5) of the order is feasible.

Glob-4 Constraints modeling precedences within objects of type *RequirementSchedule*

$$if(p = StartBeforeStart) start[p.before] \le start[p.after] \quad (68)$$

$$if(p = StartBeforeEnd) start[p.before] \le end[p.after] \quad (69)$$

$$if(p = EndBeforeEnd) end[p.before] \le end[p.after] \quad (70)$$

$$if(p = EndBeforeStart) end[p.before] \le start[p.after] \quad (71)$$

(68)–(71), ∀(rs in ReqScheduleSet, p∈∈rs.precedence)

Glob-5 Constraints modeling synchronizations within objects of type *RequirementSchedule*
∀(*rs in ReqScheduleSet, p in rs.synchronization*)

$$if(p = Simultaneous Start) start[p->at(1)] = start[p->at(2)] \quad (72)$$

$$if(p = SimultaneousEnd) end[p->at(1)] = end[p->at(2)] \quad (73)$$

$$if(p = StartAtEnd) start[p->at(1)] = end[p->at(2)] \quad (74)$$

(72)–(74), ∀(*rs in ReqScheduleSet, p in rs.synchronization*)

6. Mechanism for automated repair

The mechanism introduced in this section is designed to perform in the context of the collaborative business process for

managing disruptions described in Section 3. Controllers hold their resources information consisting in their availability and their scheduled SPOs. If a Controller requests the Feasibility Manager for feasibility repair, it will send a message with the request including this information.

At every stage of the mechanism the Feasibility Manager will hold an *activeNet* composed by resources and SPOs involved in the repair process. An *activeNet* is composed by four sets of entities that join the mechanism: *activeFixedSPO*, SPOs which their specifications remain fixed but later can be modified; *activeVarSPO*, SPOs that can be modified; *activeResources*, resources in the mechanism; and *fixedSPO*, SPO whose specification cannot be ever modified. A resource *r* in *activeResources* has its scheduled SPOs in any of the SPO sets.

If a SPO is in the set *activeVarSPO*, the domain of its related variables (regarding time and quantity) encompasses all the slacks available in the SPO.

When a Controller is alerted by the Monitor of a disruptive event (as depicted in Fig. 1) it will define an event scope (the minimal required information to evaluate feasibility) and send it to the Feasibility Manager. If the event is a change in the availability of a resource, the event scope is the affected resource and all its scheduled orders, and then the *activeNet* is defined by: adding the affected resource in *activeResources* and all its scheduled orders in *activeFixedSPO* (the other sets are empty). The mechanism always evaluates the feasibility of the whole *activeNet* by trying to solve its associated FCR–CSP (as an *activeNet* is a set of reference model supply processes).

If the FCR–CSP of the *activeNet* has a solution (every constraint is satisfied), which means the event does not compromise the execution of any SPO, there is not an exception and the outcome of the mechanism is the updated net (the new specification of the affected resource is the only update).

If a solution is not found, the mechanism advances to the next stage, expanding the *activeNet*. This is repeated until the stopping condition is fulfilled. The expansion consists in.

6.1. Stage1

1) Include in *activeVarSPO* every *spareOrder* in the set of *scheduledOrders* of any resource already present in *activeResources* (*if there is not any spareOrder go to Stage2*). This enforces the utilization of optional orders that can be executed in case of disruptions (Section 4.2), before modifying scheduled orders.
2) Add into *activeResources* the resources assigned to these *spareOrders*; and add into *activeFixedSPO* the *scheduledOrders* of these resources. This enables to check whether the resources involved in executing the *spareOrders* can also execute their previously scheduled orders.
3) If this new *activeNet* is feasible the net of SPOs is updated with the changes introduced in the schedule and the repair mechanism ends, otherwise go to *Stage2*.

6.2. Stage2

1) Include in *activeVarSPO* every SPO in *activeFixedSPO*; this step makes variable the set of fixed orders of the resources involved in the mechanism.
2) Add into *activeResources* the resources assigned to the SPOs recently added to the *activeVarSPO* set and add into *activeFixedSPO* the *scheduledOrders* of these resources which were not added to the *activeVarSPO* set. This enables to check whether the resources involved in executing the orders which were just made variable can also execute their previously scheduled orders.
3) If this new *activeNet* is feasible the net of SPOs is updated with the changes introduced in the schedule and the repair mechanism ends, otherwise return to *Stage1*.

The global stopping condition is fulfilled whenever a solution is found or the set of *activeFixedSPO* is empty, that is, the *activeNet* cannot be expanded.

7. FCR–CSP transformation engine implementation

Section 4.5 established that for every schedule defined with the reference model, a transformation into the data model of the FCR–CSP allows to have a FCR–CSP instance for the schedule. The model to model transformation required was implemented as follows:

1) A XML Schema (XSD model) (W3C, 2004) for the reference model was obtained. This task is accomplished using a standard UML to XSD transformation provided by IBM Rational Software Architect (IBM, 2012).
2) The data model structure of FCR–CSP (composed by the capacity, state and main data models) was defined as ILOG OPL (IBM, 2011) data model. As constraints models are solved in the following case study using ILOG OPL.
3) A JAVA project ILOG-InputOutputManager was developed in order to transform a XML instance of the reference model (a schedule) into an ILOG OPL data instance, and vice versa.

8. Case study

The tools used in this case study validation are: ILOG-OPL Studio to solve the FCR–CSP constraints models; Scheduling System to generate feasible supply chain schedules; and the automatic transformation of Section 7 to generate the FCR–CSP data model.

8.1. Case study description

The studied supply chain produces and distributes urea. Production and factory warehousing is done in Bahia Blanca, Argentina (Factory Warehouse-Bahia Blanca). From a proprietary loading dock in the local port urea is distributed by ships to three distribution centers located at San Lorenzo, Argentina; Montevideo, Uruguay; Rio Grande, Brasil.

Shipments are managed by a third party logistics provider. Three ships with a predefined route and schedule are used: *Ship-DCSanLorenzo*, *Ship-DCUruguay* y *Ship-DCBrasil*, they attend the demands of the distribution centers at: San Lorenzo, Argentina; Montevideo, Uruguay and Rio Grande, Brasil, respectively.

Urea availability at Factory Warehouse in Bahia Blanca is practically unlimited; therefore demand and supply are managed using a Planning System for each Distribution Center attending the constraints imposed by: coordination of shipments according to availability and possible fluvial/marine routes (Table 11), mean ship trip times, loading and unloading times, safety stock and inventory sizes

The Scheduling System builds a schedule and provides slacks with the following characteristics. Dispatches at Distribution Centers have an associated time window for their fulfillment, consisting in: if the dispatch is originally planned for day d, it can be moved only within the week which d is within. Some SPOs representing dispatches allow quantity changes but those changes cannot exceed ten percent of their original quantity value. These policies are captured in the reference model using capacity usage

Table 11
Mean trip times (hours).

	San Lorenzo	Montevideo	Rio Grande
Factory Warehouse-Bahia Blanca	96 h. (Ship-DCSanLorenzo)	144 h. Ship-DCUruguay	168 h. (Ship-DCBrasil)
Montevideo	–	–	60 h.(Ship-DCBrasil)

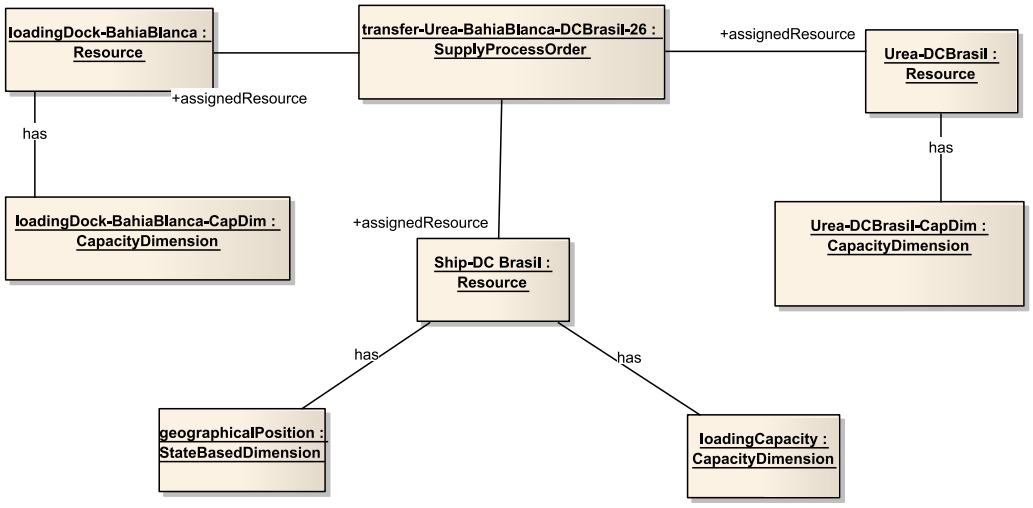

Fig. 13. An urea transfer SPO from Bahia Blanca to Brasil.

items. Urea transfers from Factory Warehouse-Bahia Blanca to Distribution Centers are scheduled with a specific timing, but in case of a repair process, their timing can change as long as the capacity of every resource involved (inventories, loading dock and ships) allow such change, and the involved states (geographical position of ships) are compatible with the change.

Ships are loaded at the Factory Warehouse-Bahia Blanca at a rate of 1000 t (metric ton)/h, and unloaded at San Lorenzo and Uruguay at rate of 250 t/h, and at a rate of 425 t/h at Brasil.

In exceptional situations transfers from Factory Warehouse-Bahia Blanca to the Distribucion Center in Brasil can be performed by another logistics provider that provides a transportation capacity of 17.000 t with a delivery time of 7 days. This supply process is modeled as a *spareOrder* according to the reference model. Daily demand at DCs is aggregated into daily dispatch SPOs for the 33 days of the horizon.

The main supply process, urea transfer from Factory Warehouse-Bahia Blanca to Distribution Centers is modeled using the reference model (Fig. 13). The transfer in the figure has three resources: loadingDock-BahiaBlanca representing the dock where ships are loaded with urea, Urea-DCBrasil representing the inventory of urea in the Distribution Center in Brasil, Ship-DCBrasil representing the ship used to transport the urea.

8.2. Data collection

To obtain information on stock movements of the urea supply chain, records of port movements were collected within a 33 days horizon, together with stock movement reports from the urea factory at Factory Warehouse-Bahia Blanca in the same period. These records were consolidated in a data base and this data was analyzed to infer both the typical demand pattern in every DC and the ship scheduling policy. Then, a Distribution Resource Planning based Scheduling System was used to generate detailed schedules for the urea supply chain. The output of this system was expressed as a set of supply processes modeled using the artifacts provided by the reference model.

8.3. Disruptive event management

The performance of the FCR–CSP based repair mechanisms was studied in several disruptive situations. Following two situations are described, which are chosen for their simplicity to follow the repair steps and evaluate whether FCR–CSP can provide common sense solutions that an Operations Manager could obtain and recommend.

8.3.1. Disruptive event on a resource

Shortly after initializing the execution of the schedule the shipping company communicated the unavailability of Ship-DCBrasil from day 16 (384 h) and on (horizon is 33 days). This was reflected as change in the availability profile of the resource. Feasibility of the affected resource was assessed by means of the FCR–CSP. The result was that the schedule becomes unfeasible and therefore the repair mechanism was executed. Table 12 summarizes the search for a solution as the *activeNet* was expanded.

The FCR–CSP on Expansion 2 found a solution. The most important changes were:

SPO *transfer from Bahia Blanca to Brasil at day* 26 was canceled.

SPO Spare order of Urea from Bahia Blanca to Brasil was activated and its involved resources (spare ship, loading dock and urea inventory in Distribution Center in Brasil) properly coordinated. This coordination satisfies the logistics provider policies specified in the capacity usage items of the spare ship resource. Its resulting specification is

- *SPO: quantity:* 17000*; startTime:* 383*; endTime:* 608
- *Requirements:*
- *CapacityRequirement*: *load urea in Bahia Blanca*, involving resource loadingDock-BahiaBlanca starts at 383 h and ends at 400 h.
- *CapacityRequirement*: *load urea in BahiaBlanca*, involving resource *spareShip* starts at 383 h and ends at 400 h.
- *StateBasedRequirement*: *arrive and stay in state Rio Grande*, to download urea at Distribution Center in Brasil, involving resource *spare ship* starts at 568 h ends at 608 h.

Table 12
Resource event managed using FCR–CSP.

Expansion	ActiveResources	ActiveFixedSPO	ActiveVariableSPO	Supply chain collaborators
Event scope	Ship-DCBrasil	The set of transfers of Ship-DCBrasil	∅	Controller-Ships
Expansion 1	Ship-DCBrasil; Urea-DCBrasil; loadingDock-BahiaBlanca	Set of 33 daily dispatch SPOs of Urea from Distribution Center in Brasil; the set of scheduled orders that use loadingDock-BahiaBlanca that is: every transfer in the supply chain schedule apart from those that use Ship-DCBrasil	Transfer from BahiaBlanca to Brasil at day 7 and transfer at day 26.	Controller-Ships; Controller-Brasil; Controller-Factory Warehouse
Expansion 2	Ship-DCBrasil; Urea-DCBrasil; loadingDock-BahiaBlanca; spareShip	Set of 33 daily dispatch SPOs of Urea from Distribution Center in Brasil; the set of scheduled orders that use loadingDock-BahiaBlanca that is: every transfer in the supply chain schedule apart from those that use Ship-DCBrasil	Transfer from Bahia Blanca to Brasil at day 7 and transfer at day 26; spare order of urea from Bahia Blanca to Brasil	Controller-Ships, Controller-Brasil; Controller-FW; Controller-3PL (A third party logistics provider controls the spare ship)

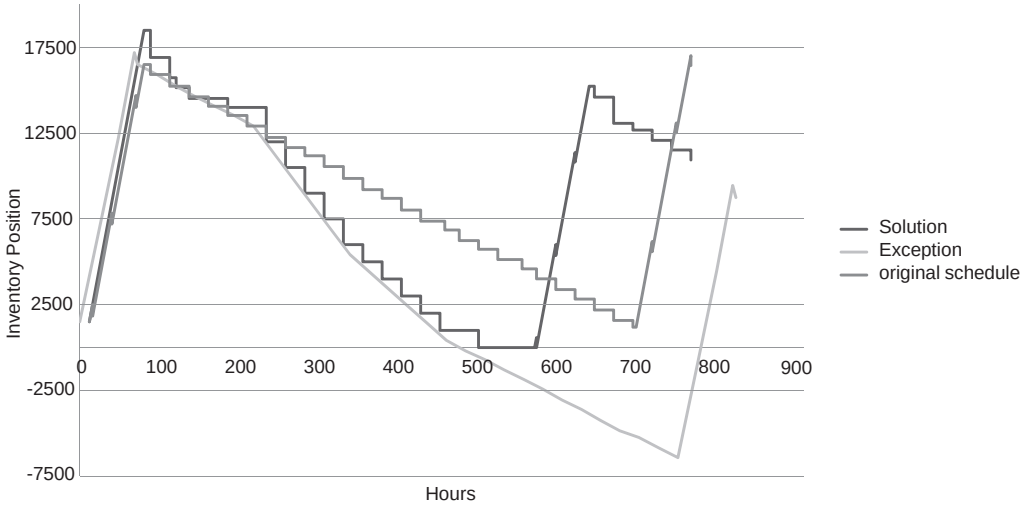

Fig. 14. Urea-DCUruguay Projected Available Inventory.

- *CapacityRequirement*: *download urea to DC in Brasil*, involving resource Urea-DCBrasil starts at 568 and ends at 608.

8.3.2. disruptive event on a SPO

At Uruguay the competitor's Distribution center temporarily runs out of stock, this obligated its clients to supply from the Distribution Center represented by resource Urea-DCUruguay (the inventory of urea at Distribution Center in Uruguay). As a result shipments scheduled from day 10 to 19 were substantially increased their quantity. A FCR–CSP evaluation of the schedule returned an exception. The scenario is depicted in Fig. 14 that shows the infeasibility of the inventory projection.

The FCR–CSP found a solution (Fig. 14) where 9 SPOs were modified. In the figure it is visible how the solution was closely related to the original schedule. The second urea transfer is put forward and the other modifications consisted in slightly reducing some orders quantities (within the original slacks) preserving most of the original schedule.

8.4. Results analysis

The events analyzed demonstrated that the mechanism based on the FCR–CSP can perform an autonomous search for a solution in a collaborative schedule after a disruption. The only requirement relies on the willingness to share information on resources and SPOs.

By just modeling the schedule using the reference model a full collaborative repair mechanism was readily available. Solutions for both disruptions were found and the changes introduced used only available slacks.

The type of solutions obtained by the mechanism is quite similar to the ones obtained after a classical three party negotiation between client, supplier and logistics provider. The mechanism succeeded in finding them autonomously and almost immediately without involving the parties into a probably extended manual negotiation. In most situations it outperforms a manual approach as it has full visibility of on-going supply processes across different partners and if required new processes are added in a distributed expansion mechanism.

The mechanism is innovative for both research and practice of schedule execution as it opens the possibility of delegating the schedule restoration, whenever possible, to an autonomous information system that enables a collaborative execution control among supply chain partners. This delegation is possible without engaging Controllers to new negotiations regarding the creation of a new schedule due to:

- Each individual Controller does not have any information about the schedule of the others. Therefore the problem of sharing information to other supply chain partners is reduced to share portions of their schedule with a neutral Feasibility Manager.
- The repair mechanism solves the coordination problem that arises from changes in the usage of resources as consequence of

the re-specification of SPOs done to restore schedule feasibility. Every change introduced is guaranteed to satisfy the availability and usage constraints of resources already provided in the schedules.
- The restoration of schedules is accomplished without any knowledge of the internal processes and the negotiations between supply chain partners required for building them.

The main benefit of using this mechanism is a reduction of the number of routine execution disruptions that can be repaired without translating into a full rescheduling task, avoiding the harmful effect of scheduling system nervousness: increases in setup times and costs (Kabak and Ornek, 2009).

A vast literature on supply chain risk planning, robust planning and buffer allocation, as in (Lenny and Saad, 2006; Koh et al., 2002; Landeghem and Vanmaele, 2002; Srivastava, 2000; Tang, 2006), deals with different strategies to tackle uncertainty resulting in schedules and plans with slacks, and all the effort done to provide them is better used by the adoption of this mechanism.

This constitutes the first proposal for disruptive event management in supply chains that takes advantage of the explicit (as a result of buffer allocation) and implicit (as a result of analyzing concurrently a set of active resources and its usage) slacks present in schedules in an autonomous fashion.

In contrast there are also two limitations for the approach. First, the repair capacity is limited as modifications in the schedule are constrained by the existence slacks. Second there is an implicit collaboration contract required for the methodology to work: supply chain partners acting as controllers must share small portions of their schedules, and these portions should be expressed in terms of the reference model.

8.5. Performance analysis

The performance of the repair mechanism depends on the solution times of the CSP problems associated to its stages. In the case study introduced in this work, schedules controlled by different business partners in a supply chain are coordinated.

The size of the case is representative of a real world scenario and extends to 107 modifiable orders (with its requirements) and 8 critical resources for a 33 days horizon.

The case study required solving 4 CSPs, the largest totalizing 10705 variables, 4995 of them were binary and the number of constraints grew up to 25839.

The solution time of a CSP averaged 74 s with a maximum of 115 s for each stage of the mechanism and a total computing time of 295 s to find feasible changes for a horizon of 33 days.

Unlike scheduling problems formulations, FCR CSP only searches for feasible solutions in the slack space of already defined schedules without optimizing any objective function, that is the key for seamlessly solve big problem instances.

9. Conclusions and future work

The CSP based approach to disruptive events management in supply chains presented in this work, gives a comprehensive treatment to the functions of evaluating disruptive events, detecting an exception and deciding corrective actions. The generality of the proposal is given by the fact that it does not rely on any supply chain configuration or internal structure. Generic supply processes are described as composed by SPOs and its required resources, focusing on the assessment of the feasibility of the execution. This feasibility is evaluated through general descriptions of availability and requirements that allows the modeling of any type of processes and resources within the same framework.

To provide systematic autonomy to current event management and execution control systems, a reference model was proposed. It has two main features: (i) it provides a self-contained description of any ongoing schedule of SPOs with the information required to assess its feasibility and (ii) it allows the automatic transformation into a CSP suitable to search for local solutions.

The mechanism for automated repair intends to make surgical modifications to the current schedule which do not affect the economical and operational considerations made at the moment the schedule was created. On purpose, the allowed changes are limited to the space of slacks already included by the original schedule.

This repairing task can be safely delegated to automated systems and would facilitate the design of collaborative inter-organizational business processes to manage events along the supply chain.

The limitations of the FCR–CSP approach are related to the existence of slacks in schedules. This is aligned with current trends of generating more robust schedules by including appropriate buffers (Radjou et al., 2002; Verderame et al., 2010).

The need for a monitoring system capable of detecting changes in both resources and orders is another critical requirement.

Finally the problematic of information sharing between business partners whenever an exception crosses inter-organizational borders arises. The present work implicitly assumed that information on resources and orders are shared with no restrictions, but this is not always the case. Further research on this subject is needed.

Financial support for this work by Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET) and Agencia Nacional de Promoción Científica y Tecnológica (ANPCyT) is gratefully acknowledged.

References

Arief Adhitya, R.S., Karimi, I.A., 2006. A model-based rescheduling framework for managing abnormal supply chain events. Computers and Chemical Engineering 31, 496–518.

Aytug, H., Lawley, M.A., McKay, K., Mohan, S., Uzsoy, R., 2005. Executing production schedules in the face of uncertainties: a review and some future directions. European Journal of Operational Research 161, 86–110.

Bearzotti, L., Salomone, E., Chiotti, O., 2008. An autonomous multi-agent approach to supply chain event management, Service Operations and Logistics, and Informatics, 2008. IEEE/SOLI 2008. IEEE International Conference on, pp. 524–529.

Cauvin, A.C.A., Ferrarini, A.F.A., Tranvouez, E.T.E., 2009. Disruption management in distributed enterprises: a multi-agent modelling and simulation of cooperative recovery behaviours. International Journal of Production Economics 122, 429–439.

Friedrich, G., Fugini, M., Mussi, E., Pernici, B., Tagni, G., 2010. Exception handling for repair in service-based processes. Software Engineering, IEEE Transactions on 36, 198–215.

Gaudreault, J., Pesant, G., Frayret, J., D'Amours, S., 2012. Supply chain coordination using an adaptive distributed search strategy. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 42, 1424–1438.

Griffith, R., Kaiser, G., Lopez, J.A., 2009. Multi-perspective evaluation of self-healing systems using simple probabilistic models, Proceedings of the Sixth International Conference on Autonomic computing. ACM, Barcelona, Spain, pp. 59–60.

Guarnaschelli, A., Fernandez, E., Chiotti, O., Salomone, H.E., 2012. A service-oriented approach to collaborative management of disruptive events in supply chains. International Journal of Innovative Computing, Information and Control 8, 5341–5368.

Hailpern, B., Tarr, P., 2006. Model-driven development: the good, the bad, and the ugly. IBM Systems Journal 45, 451–461.

Hamadi, R., Benatallah, B., Medjahed, B., 2008. Self-adapting recovery nets for policy-driven exception handling in business processes. Distributed and Parallel Databases 23, 1–44.

Hwang, S.-Y., Tang, J., 2004. Consulting past exceptions to facilitate workflow exception handling. Decision Support Systems 37, 49–69.

IBM-ILOG, 2009. Scheduler V6.7 Users Manual. IBM, United States.

IBM, 2011. IBM ILOG CPLEX Optimization Studio.

IBM, 2012. IBM Rational Software Architect.

Ierapetritou, M.G., Floudas, C.A., 1998. Effective continuous-time formulation for short-term scheduling. 1. Multipurpose batch processes. Industrial and Engineering Chemistry Research 37, 4341–4359.

Kabak, K.E., Ornek, A.M., 2009. An improved metric for measuring multi-item multi-level schedule instability under rolling schedules. Computers and Industrial Engineering 56, 691–707.
Lenny, Koh S.C., Saad, S.M., 2006. Managing uncertainty in ERP-controlled manufacturing environments in SMEs. International Journal of Production Economics, 109–127.
Koh, S.C.L., Saad, S.M., Jones, M.H., 2002. Uncertainty under MRP-planned manufacture: review and categorization. International Journal of Production Research 40, 2399–2421.
Landeghem, H.V., Vanmaele, H., 2002. Robust planning: a new paradigm for demand chain planning. Journa of Operations Management 20, 769–783.
Le Pape, C., 1994. Implementation of resource constraints in ILOG Schedule: a library for the development of constraint-based scheduling systems. Intelligent Systems Engineering 3, 55–66.
Li, J.-Q., Mirchandani, P.B., Borenstein, D., 2009a. A Lagrangian heuristic for the real-time vehicle rescheduling problem. Transportation Research Part E: Logistics and Transportation Review 45, 419–433.
Li, J.-Q., Mirchandani, P.B., Borenstein, D., 2009b. Real-time vehicle rerouting problems with time windows. European Journal of Operational Research 194, 711–727.
Masing, N., 2003. Supply Chain Event Management as Strategic Perspective—Market Study: SCEM Software Performance in the European Market. Hochschule Bremen, Bremen.
Méndez, C.A., Cerdá, J., Grossmann, I.E., Harjunkoski, I., Fahl, M., 2006. State-of-the-art review of optimization methods for short-term scheduling of batch processes. Computers and Chemical Engineering 30, 913–946.
OMG, 2010. Object Management Group, Unified Modeling Language (UML) 2.3.
Pfeiffer, A., Kádár, B., Monostori, L., 2007. Stability-oriented evaluation of rescheduling strategies, by using simulation. Computers in Industry 58, 630–643.
Radjou, N., Orlov, L.M., Nakashima, T., 2002. Adapting To Supply Network Change, in: Research, F. (Ed.), The TechStrategy Report.
Song, Y., Han, D., 2003. Exception specification and handling in workflow systems, Proceedings of the Fifth Asia-Pacific Web Conference on Web technologies and applications. Springer-Verlag, Xian, China, pp. 495–506.
Srivastava, V.D.R.G.J.a.R., 2000. A review of techniques for buffering against uncertainty with MRP systems. Production Planning and Control 11, 223–233.
Tang, C.S., 2006. Perspectives in supply chain risk management. International Journal of Production Economics 103, 451–488.
Verderame, P.M., Elia, J.A., Li, J., Floudas, C.A., 2010. Planning and scheduling under uncertainty: a review across multiple sectors. Industrial and Engineering Chemistry Research 49, 3993–4017.
W3C, 2004. XML Schema Part 0: Primer Second Edition, in: (W3C), W.W.W.C. (Ed.).
Weske, M., 2007. Business Process Management: Concepts, Languages, Architectures. Springer-Verlag, New York, Inc.
Yuan, H.-t., Ding, B., Sun, Z.-x., 2008. Workflow Exception Forecasting Method Based on SVM Theory, Computational Intelligence and Design, 2008. ISCID '08. International Symposium on, pp. 81–86.
Zimmermann, R., 2006. Agent-based Supply Network Event Management. Birkhauser-Verlag, Basel, Switzerland.