# Configurable Resource Models for Treatment Planning in the Medical Domain

Denny Schneeweiss
Berlin Institute of Technology
dschneeweiss@mailbox.tu-berlin.de

Petra Hofstedt
Brandenburg University of Technology
Cottbus-Senftenberg
hofstedt@b-tu.de

## ABSTRACT

This paper presents configurable resource definitions based on feature models which can be used by domain experts without the immediate assistance of IT-experts. They are part of an approach to ease the handling of automated constraint-based scheduling systems in medical facilities by non-IT-experts with treatment process models defined in the BPMN workflow-language. These resource objects can be interactively configured by the end-users to describe the available resources (rooms, devices) and staff in their environment which can then be used in activities specified in the treatment models. The configured resource objects and the BPMN models can then be automatically transformed into resource scheduling CSPs.

## Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Logic and constraint programming*

## Keywords

BPMN, CP, Planning, Scheduling

## 1. CONSTRAINT-BASED TREATMENT PLANNING WITH BPMN

Public health care services have experienced a constantly growing cost pressure for decades. Therefore, measures to increase efficiency in health care are needed without reducing the patients' safety, comfort or treatment quality. Automated planning systems can be used for optimized planning and scheduling of medical and administrative activities such as operation planning, dialysis-management or staff rostering. One challenge is the usually high resistance of medical staff to new technology-based solutions. Therefore, new technology has to adapt to their requirements, be easy to use and should be introduced with the involvement of the users to enhance user-acceptance. Many automated planning systems use finite domain constraint-solvers at their core, which require a problem statement in the form of a *Constraint Satisfaction Problem* (CSP) [1]. These CSPs are usually modelled by IT-experts. Medical staff or hospital administrators are rarely able to adapt these CSPs by themselves when changes occur and processes have to be updated. In recent years, *Business Process Management* was introduced into health care promising an increase in efficiency, quality as well as safety. Here, graphical notations exist to describe coordinated activities like medical treatments as workflows. The *Business Process Model and Notation*-language (BPMN) [2] offers a simple and easily understandable notation that can be comprehended even by non-IT-experts. We want to leverage this advantage to ease constraint-based planning in medical facilities.

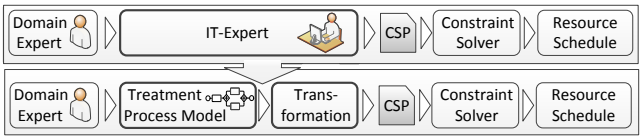


**Figure 1: Automatic transformation of BPMN treatment models into CSPs**

The overall goal of our approach is to enable domain experts like clinic administrators or physicians to adapt or extend their medical processes on their own when the need arises, mainly without the help of IT-experts. The central element is an automatic transformation from treatment process models in the BPMN-language into CSPs for time- and resource scheduling as depicted in Figure 1, which would traditionally have to be created and maintained by IT-experts. Through this contribution hospitals which use automated planning systems could react to changes faster and introduce process improvements or new processes in a shorter time since no IT-project would have to be set up to propagate the medical or administrative process changes to the planning system. which would save time and could also further increase the popularity of constraint-based planning systems by lowering the inhibition threshold for non-IT-experts.

### 1.1 Transformation of BPMN into CSPs

Our transformation takes a BPMN process description and generates a constraint problem that contains all possible schedules. Our current target platform is the JAVA-based CHOCO solver although this can be changed to other finite

BICT 2014, December 01-03, Boston, United States

DOI 10.4108/icst.bict.2014.257907

domain constraint solvers as the transformation generates constraints that are available in most solver systems.

**Activities** in a process instance are mapped directly to *tasks* in the generated CSP. A *task* is comprised of three integer variables $A_{start}$, $A_{length}$ and $A_{end} \in \mathbb{N}_0 \cup \{-1\}$ that model start- and ending time as well as the length of the activity. A task also includes the constraint $(A_{start} + A_{length} = A_{end})$ that binds the three component variables together. Activities may not be executed during the process runtime, for example when an exclusive gateway must choose between two different branches. To reflect this fact in the CSP, we use reification: An additional boolean variable $A_{active} \in \{true, false\}$ is created for each task, that represents it's scheduling status. The value of this variable is reified with the constraint $(A_{active} <> true) \Leftrightarrow (A_{start} = -1 \wedge A_{length} = 0)$, which must hold if the task is not active. This forces the task to the starting time $-1$ and it's length to zero when it's activity is not executed.

The foundation of the transformation are simple local precedence constraints, that take into account the predecessor and successor elements of activities and gateways. Sequence flows between two activities $A$ and $B$ are represented by simple precedence constraints $(B_{start} \geq A_{end})$. Gateways are represented by corresponding constraints. Figure 2 depicts two simple, exemplary process models with exclusive (1) and parallel gateways (2), for which the following constraints are generated:

$$(1) \quad ((B_{start} \geq A_{end} \wedge D_{start} \geq B_{end} \wedge C_{active} = 0) \oplus (C_{start} \geq A_{end} \wedge D_{start} \geq C_{end} \wedge B_{active} = 0))$$

$$(2) \quad (F_{start} \geq E_{end}) \wedge (G_{start} \geq E_{end}) \wedge (H_{start} \geq max(F_{end}, G_{end}))$$

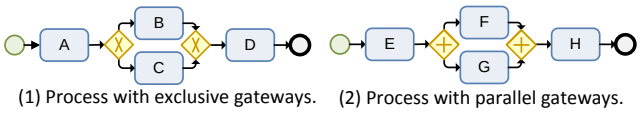


**Figure 2: Exclusive and parallel gateways**

By using the local view on predecessor and successor elements non-well-formed models can be supported, which do not always contain a closing join-gateway for every split-gateway. An additional generation of global constraints that takes into account whole sub-paths for enhanced filtering and pruning of non-solutions is also commenced.

## 2. CONFIGURABLE DOMAIN OBJECTS FOR RESOURCE ABSTRACTION

In order to create schedules a constraint solver needs to know not only about the order and relations of the process activities, but also which resources (patient and medical staff, treatment rooms, etc.) exist and how the process activities depend on them. Although the specification of the BPMN-language [2] is quite extensive, it only specifies resources abstractly as something that can be referenced by one or more activities. The specification merely distinguishes between human and non-human resources [2, 8.3.12]. This design decision was done by choice as BPMN is extensible, but the core language itself is *"constrainted only to concepts of modeling that are applicable to Business Processes"*, meaning that other types of modelling like data and information models, business rules or *"organizational models and resources"* are explicitly out of scope of the BPMN-specification [2, 7.1].

**Resources in CSPs** Resources are finite and may be used over a restricted time by each activity. However, they can be used by several activities at the same time, where the activities must not be synchronized. Thus, the modeling of the usage of a resource is done not only with respect to one single activity, but with respect to the whole process or even interconnected processes. The *Cumulative Resource Constraint* is an important global constraint for scheduling and resource allocation [3]. It models a non-consumable resource with a maximum capacity. Portions of this capacity can be allocated by tasks for given time intervals. The constraint makes sure that the capacity is never exceeded. Although the concept of the cumulative resource constraint as such is relatively simple, it requires a lot of detailed information. If more complex properties such as regular breaks or cleaning times are required for a cumulative resource the complexity significantly increases. Modelling these resources for a constraint solver even on an abstract level requires at least general knowledge about constraint-based principals and is in no way suitable for medical staff and would therefore contradict our overall goal. Nevertheless, they have to specify their human and non-human resources required by the activities in their treatment process models. Otherwise, automated planning systems cannot create a valid schedule.

**CDO-Abstraction Layer** As one can never entirely replace the assistance of IT-experts, especially when special scenarios have to be mapped to CSPs, our idea is to at least maintain the flexibility for the majority of cases. Therefore, we propose an abstraction layer that hides the complex resource descriptions for the constraint-solver behind a facade of configurable medical domain objects. This layer covers the details of resource constraints definitions behind a facade of *Configurable Domain Objects* (CDO) and expose only domain relevant attributes to the end users. This approach aims to enable domain experts to plan with resource objects that they understand (exam rooms, stationary or mobile medical devices, etc.). Figure 3 visualizes the steps and actors of our approach as a process model. Data objects represent information artefacts that are created and used by the different activities. In the first step, an IT-expert creates configurable domain objects and makes them available to the user (domain experts). The CDOs are modelled as special extended feature models. Their design is discussed in Section 2.1. The user can then instantiate and configure the CDOs according to the resources available in their environment. This configuration is done interactively with the help of the *FdConfig* tool. Section 2.1.1 covers this part. Then they can create treatment process models with BPMN or adapt existing ones. In this step, the activities in the process models are associated with the CDO-instances that represent the resources required to execute the respective activities. The next step is performed during the daily operation of the medical facility. For each patient a treatment process instance is created. Then, the transformation uses the process instances and the associated CDO-instances to generate a CSP which is passed to the constraint solver. From the CSP the solver can calculate a valid schedule. Optimization

towards a certain goal (e.g., minimal patient waiting times) can also be performed in this step.

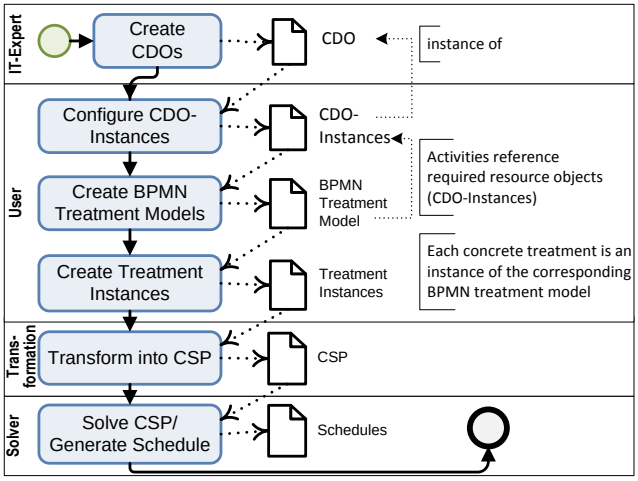


**Figure 3: Process model of our approach**

## 2.1 CDOs based on extended Feature Models

The first step in our approach, the creation of the CDOs, has to be performed by IT-experts as knowledge about constraint programming is required. The CDOs are based on *feature models* with numerical attributes. They are the interface through which the user can comfortably configure their CDO-instances. We extended them with CSP-elements that have an effect during the transformation phase.

**Extended Feature Models** A feature model [4] is a hierarchical structure of features, which are named parts, capabilities or properties of a product which the user can configure to be included (*selected*) or excluded (*removed*) in the product. Features can have optional or mandatory sub-features, which may be or respectively must be included in the product if their parent feature is included. Furthermore, *feature groups* describe that exactly one (XOR-group), at least one (OR-group), all or an arbitrary set of sub-feature have to be included, if the group's parent feature is included in the product. Furthermore, a feature model can also contain so-called cross-tree-relations, that link features across the feature hierarchy. The most common relations are *requirement* and *mutual exclusion*. In the first case, a feature $A$ can only be part of a product if a second feature $B$ is also included. Two mutually exclusive features can not both be part of a product. Extended feature models may also contain *Feature Attributes* with numerical values. They may also be affected by additional numerical constraints. A feature-model describes all possible products variants of a product line by a combination of product features and attribute values, which can be interrelated by additional constraints. In order to configure an actual product variant, the user must decide all features and valuate all attributes.

**CSP-Elements** For our approach we have extended the textual feature model language of the product configurator *FdConfig* [5]. Features can now additionally contain CSP-elements in the form of variables, constraints, resources definitions, tasks and resource requirements. Figure 4 specifies a simple cardio examination room. CSP-elements are preceded with the `CSP`-keyword. These elements have no effect when the user configures the CDO-instances and they are invisible to him. Their effects are carried out later, during the transformation phase (cf. Section 2.1.2). The normal feature model parts (features, groups, attribute, product constraints) represent the set of possible product variants, which the user can configure to define their CDO-instances. In this example, the user can configure the number of patients that the cardio room can hold (line 2). As an optional sub-feature (lines 5-13) he can decide to let the room have cleaning times, when the room is blocked. The cleaning interval can be specified by integer attributes `cleaningStart` and `cleaningEnd` in lines 6 and 7. A feature model constraint in line 8 makes sure that these values are in the right order.

```
1  feature CardioRoom {
2   int nbrOfPatients in [0..10];
3   CSP: public task[] patientsTasks;
4   CSP: cumulative CardioRoomRes {capacity = nbrOfPatients;};
5   feature CleaningTime : optional {
6     int cleaningStart in [0..23];
7     int cleaningEnd in [0..23];
8     constraint {cleaningStart < cleaningEnd};
9     CSP: task CleaningTask {
10         start = cleaningStart; end = cleaningEnd;}
11    CSP: CleaningTask requires nbrOfPatients of
12         CardioRoomRes from cleaningStart till cleaningEnd;
13  }
14 }
```

**Figure 4: FdConfig-notation of a cardio room-CDO**

### 2.1.1 Interactive product configuration

A product variant is only valid if all constraints hold. Therefore, the user must consider the implicit rules imposed by feature-subfeature relations as well as explicit constraints defined in the feature model. To make the configuration more user-friendly, a product configurator can assist the configuration process. In our approach we use *FdConfig* [5]. It provides interactive, backtracking-free product configuration. The backtracking-freeness property [6] reduces frustration in the configuration process. It means, that users can perform configuration steps in any order while it is assured that only valid options are offered so that the user can always get to a feasible configuration without reverting earlier decisions. To ensure this, *FdConfig* may respond to user decisions with *Consequence Decisions*. If, for example, two mutually exclusive features $A$ and $B$ existed in a feature model and the user would select feature $A$, then the configurator would automatically remove $B$ as a consequence, to ensure the model is always valid. If the user reverts his decision, the configurator will re-check the constraints in the feature model and do the same.

### 2.1.2 Transformation Phase

In the transformation phase, the configuration decisions (representing the CDO-instances) are used to generate variables, tasks and constraints that represent the resources from the CDOs. These are then added to the CSP along with the elements generated from the BPMN-workflows. The transformation only generates CSP-elements which are inside *selected* features. When feature attributes are referenced from inside CSP-elements (cf. line 4 in Figure 4), these are replaced by the values configured by the user.

The example contains an array of *public* tasks in line 3. They are placeholders which can later be referenced by the activities in the treatment process models. Line 4 holds the actual cumulative resource definition. It's maximum capacity comes from the integer attribute `nbrOfPatients` which the user can set during product configuration. If the user selects the `CleaningTime` feature (line 5), an additional task is created in the CSP (lines 9-10) with its start- and ending-times fixed to the values defined in the integer attributes `cleaningStart` and `cleaningEnd`. In order to block the cardio room during cleaning times, the cumulative resource representing the cardio room is allocated to this `CleaningTask` with it's full capacity by the statements in lines 11-12. This statement is a *Resource Requirement*. It allocates a resource or parts of it's capacity to tasks. This element can also be used to assign the activities to their required CDO-instances.

This exemplary configurable domain object hides the complexity of the underlying cumulative resource which may be subject to optional cleaning times. Further types of resources supported by our approach are, among others, exclusive resources (one resource for one activity at a time) and alternative resources (chose an exclusive resource from a set of alternatives), which also can be combined with boolean or numeric constraints to define more complex CDOs.

# 3. RELATED WORK

There are other approaches considering the combination of business process modeling and resource descriptions: Stroppi et. al. [7] extend BPMN with notations for resource modelling and visualization aiming at user-friendly task and resource modelling for non-it-experts, which is similar to our goal. Instead of UML-like notations our approach uses feature models for their clarity and explicitness as resource descriptions which can be interactively configured by end-users. [8] presents resource assignment constraints for BPMN defined in OCL, which are propagated to the process execution. Both papers [7, 8] evaluate their approaches with respect to the coverage of certain workflow resource patterns (WRP). This is not in the focus of our work; however, the transformation of the resource requirements onto a general constraint language (as in our approach) allows the realization of such WRP. Moreover, the WRP definitions in [8] are mainly based on boolean constraints and the OCL definitions are difficult to read and understand by non-IT-experts. Our approach builds on finite-domain and real interval constraints, and it is in principle generic and would allow to integrate any constraint solving tool. The constraint definition is supported in a user-friendly form. The constraint propagation takes place not only at execution time but before, interactively at modeling time to directly exclude inadmissible models at an early stage.

Wolf [9] presents constraint-based modeling of clinical pathways and task scheduling, Bartak [10, 11] focus workflow verification in a more general context. The approaches of mapping resource requirements to constraint problems are similar to ours. The resource constraints considered are typically exclusive, alternative and cumulative resources which are directly transfered to the scheduling problem. In contrast, we aim at non-IT-expert modeling such that we allow many forms of resource constraints (including FD and interval constraints) and in particular their configuration by the end user, such that certain invalid configurations can already be excluded interactively at modeling time.

# 4. REFERENCES


[1] Marriott, K., Stuckey, P.J.: Programming with Constraints: An Introduction. The MIT Press (March 1998)

[2] Object Management Group: Bpmn 2.0 specification - business process model and notation (bpmn) version 2.0. Technical report, Object Management Group (January 2011) `http://www.omg.org/spec/BPMN/2.0/examples/PDF`.

[3] Trojet, M., H'Mida, F., Lopez, P.: Project scheduling under resource constraints: Application of the cumulative global constraint. In: Computers Industrial Engineering, 2009. CIE 2009. International Conference on. (2009) 62–67

[4] Czarnecki, K.: Generative Programming: Methods, Techniques, and Applications Tutorial Abstract. Addison-Wesley (2002)

[5] Schneeweiss, D., Hofstedt, P.: Fdconfig: A constraint-based interactive product configurator. In Tompits, H., Abreu, S., Oetsch, J., Pührer, J., Seipel, D., Umeda, M., Wolf, A., eds.: INAP/WLP. Volume 7773 of Lecture Notes in Computer Science., Springer (2013) 239–255

[6] Hadzic, T., Subbarayan, S., Jensen, R.M., Andersen, H.R., Møller, J., Hulgaard, H.: Fast backtrack-free product configuration using a precompiled solution space representation. In: IN: PETO CONFERENCE, DTU-TRYK. (2004) 131–138

[7] Stroppi, L.J.R., Chiotti, O., Villarreal, P.D.: A BPMN 2.0 Extension to Define the Resource Perspective of Business Process Models. In: XIV Congreso Iberoamericano en Software Engineering – CIbSE. (2011)

[8] Awad, A., Grosskopf, A., Meyer, A., Weske, M.: Enabling Resource Assignment Constraints in BPMN. Technical report, Business Process Technology, Hasso Plattner Institut (2009) BPT Technical Report 04-2009.

[9] Wolf, A.: Constraint-Based Modeling and Scheduling of Clinical Pathways. In Larrosa, J., O'Sullivan, B., eds.: Recent Advances in Constraints - 14th Annual ERCIM International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP 2009. Volume 6384 of Lecture Notes in Computer Science., Springer (2011) 122–138

[10] Barták, R., Little, J., Manzano, O., Sheahan, C.: From enterprise models to scheduling models: bridging the gap. Journal of Intelligent Manufacturing **21**(1) (2010) 121–132

[11] Barták, R., Rovenský, V.: On verification of nested workflows with extra constraints: From theory to practice. Expert Systems with Applications **41**(3) (2014) 904–918