Tracking Product Specification Dependencies in Collaborative Design for Conflict Management

Mohamed Zied Ouertani, Lilia Gzara

Tracking Product Specification Dependencies in Collaborative Design for Conflict Management

M. Z. Ouertani [1], L. Gzara
CRAN (Research Centre for Automatic Control), CNRS UMR 7039, Nancy University, Faculty of Sciences and Techniques, BP239, Vandoeuvre-les-Nancy, 54506, France

Abstract
The main difficulty associated with a collaborative design process is understanding the product data exchanged during design. Efficient and effective coordination of design activities relies on a thorough understanding of dependencies between shared product specifications throughout the entire development cycle. This paper explores the linkages between design process features and product specification dependencies, and suggests ways of identifying and managing specification dependencies to improve collaborative process performance. Using a UML[2] specification, we propose a process traceability tool to track the design process in an ongoing manner. Based on the information captured, dependencies between specifications involved in the tracked process are identified and inserted in a dependency network, maintained throughout the design process. A set of mechanisms is then proposed to qualify the identified dependencies. Extracting and qualifying specification dependencies could be useful in many design situations, for example, during an engineering change management process to assess impacts and study change feasibility, or during a conflict management process to assist designers in resolving conflicts and maintaining the coherence of the design process (knowing that change management is a tool to conduct conflict management). Special attention is paid to the conflict management process. By means of a case study, we show how the solution we propose can assist designers during the conflict management process.

Keywords*: Collaborative process, Design process traceability, Specification dependencies, Conflict resolution, Change management*

1. Introduction

Although in most design processes, coordination entails clear communication between designers, the real reason for this coordination is not for communication but for resolving dependencies between product specifications [1]. Design is constraint oriented, and comprises many interdependent parts. A change in one part may have consequences for another part, and designers cannot always oversee these interdependencies and consequences.

Product specifications are not always trivial and explicit. Current PDT (Product Data Technology) tools are not able to extract these dependencies from informal and textual descriptions and hence the dependencies cannot be revealed by the currently available PDT tools.

Several researchers have investigated specification dependencies, for example: Eppinger et al. [2], Kusiak and Wang [3], Dong and Agogino [4], Wang and Jin [1], Browning [5] and Yassine and Braha [6]. Most have proposed a representation of specification dependencies (with a DSM[3] matrix, a network or a set of patterns) but none has shown how to obtain these

[1] Corresponding author. *E-mail address*: Mohamed-Zied.Ouertani@cran.uhp-nancy.fr
[2] Unified Modelling Language
[3] Design Structure Matrix

representations or proposed mechanisms to identify dependencies. Moreover, these studies have addressed specifications involved in a design process that *has already been carried out*, whereas the usefulness of specification dependencies knowledge is primarily *during* the design process, to help designers in performing their activities and resolving interdependency problems. In addition to this reported work, all studies reported to date have only investigated the case of two dependent design activities (an upstream activity feeds specifications to a downstream activity) belonging to the same decomposition level of the design process. For a complete identification of specification dependencies, it is necessary to consider the dependencies between activities carried out at various decomposition levels of a design process.

In terms of dependency qualification, some researchers have provided interesting proposals on this issue. We particularly note the framework developed by Krishnan et al. [7] to measure dependency based on two dimensions: upstream specification *evolution* and downstream specification *sensitivity*. We also cite the work of Kusiak and Wang [3] dealing with *qualitative dependency* (the direction of the change of a product specification that is affected by another) and *quantitative dependency* (the rate of change of a product specification that is affected by another). However, these dependency qualifications and quantifications do not assist in identifying whether the dependency is strong and should be kept or whether it can be removed. We should highlight that all of the previous studies have assumed the importance of qualifying product specification dependencies but none of them has proposed mechanisms to support these dependencies.

We have therefore developed a solution called DEPNET[4], which explicitly captures product specification dependencies, inserts them in a dependency network that is maintained throughout the design process, and assists designers in resolving dependencies during the design process, particularly when design conflicts occur or when engineering changes are requested.

The DEPNET solution differs from the previous studies [1-7] in three major aspects, as follows.

- It proposes a method to identify specification dependencies and defines concepts to qualify the discovered dependencies.
- It takes into account the predefined specifications as well as the emerging specifications resulting from non-planned design activities.
- It considers the design process in a more realistic way and seeks to identify product specification dependencies among sets of dependent activities belonging to various decomposition levels.

In addition to the introduction, this paper consists of four more sections. Section 2 illustrates the problem definition, using the example of a turbocharger design problem. Section 3 focuses on the DEPNET solution: it describes the dependency network constructs and shows how to build this network. Section 4 illustrates, by means of a case study, the use of the DEPNET solution during conflict management, while Section 5 provides conclusions and future prospects.

[4] DEPNET – product specification **DEP**endencies **NET**work identification and qualification.

2. Example – A turbocharger design process

The mechanical concept of a turbocharger revolves around three main parts: a *turbine wheel*, driven by the exhaust gas from a pump, most often an internal combustion engine, which spins the second part; an *impeller*, whose function is to force more air into the pump's intake supply; and a *centre hub rotating assembly* (CHRA), which contains bearings, an oil circuit, a cooling system and a shaft that directly connects the turbine and impeller.

The design process of a turbocharger is composed of four main concurrent sub-processes: the turbine wheel design sub-process, the impeller design sub-process, the CHRA design sub-process, and the centre housing assembly customisation (i.e., matching the impeller housing and turbine housing to the vehicle engine). The designers have then to exchange preliminary product specifications to enable the process to move forward.

At the beginning of the turbocharger design process, the concerned parties each have a set of data at their disposal as well as a set of requirements to follow, such as:

- the vehicle/application specifications: vehicle or equipment manufacturer, gearbox type and maximum weight of the vehicle,
- the engine specifications: engine type, configuration, displacement and 3D CAD drawings,
- the engine performances: power, torque, engine speed and air flow, and
- the turbocharger specifications: regulation type, actuator type, engine turbocharger position, turbocharger weight and specific speed limits.

According to these specifications, the *impeller designer* starts the planned activity (to define the impeller part). The designer has to define the impeller attributes, which include: wheel cast-material, wheel cast-process, expected compressor inlet temperature and compressor outlet temperature. Once these attributes are fully defined, the impeller designer defines the exducer and inducer diameters of the compressor wheel. The impeller 3D CAD drawing is then created. The final task in the impeller design sub-process is defining the impeller housing by calculating the parameters 'trim'[5] and A/R[6]. Once these attributes are calculated, the designer is able to complete the 3D CAD drawing of the impeller part.

Based on the customer specifications, the turbocharger specifications and the impeller-defined attributes, the *turbine designer* commences the planned activity (to define the turbine wheel) at the same time as the impeller designer commences the planned activity. First the interdependent parameters, namely the wheel, nozzle ring and insert ring materials, the maximum limit of the turbine inlet temperature, and the inlet/outlet turbine pressure are defined in such a way as to achieve the target turbocharger performance. Once these parameters are defined, the turbine designer can begin to define the wheel dimensions, and create the 3D CAD drawing of the turbine wheel. Defining the wheel dimensions involves calculating the exducer and inducer diameters. The designer concludes this part of the design by defining the turbine housing (by calculating the turbine attributes 'trim' and A/R).

Concurrent with the impeller and turbine definition activities, the *CHRA designers* specify their parts based on the impeller and turbine definitions and the turbocharger specifications.

---

[5] The 'trim' attribute, which is an area ratio used to describe both turbine and compressor wheels, is calculated using the inducer and exducer diameters.

[6] A/R describes a geometric characteristic of all compressor and turbine housings. It is defined as the inlet cross-sectional area divided by the radius from the turbo centreline to the centroid of that area.

The CHRA designers have to define the bearing system, the cooling system, the oil circuit and the shaft. Afterwards, and in a collaborative way, the CHRA, impeller and turbine designers finalise the centre housing assembly by determining how this part should be connected to the vehicle engine.

The basic principle behind turbocharging is fairly simple, but a turbocharger is a very complex piece of machinery. Not only must the components within the turbocharger itself be precisely coordinated, but the turbocharger and the engine it services must also be accurately matched. If they are not, then the engine is inefficient and it could even be damaged. This is why it is important that the concerned parties collaborate closely by coordinating their activities as well as their communication of the specifications. The different parts are highly interdependent, as modifying one of them impacts significantly on the others. For instance, the shaft sub-part defined within the CHRA sub-process directly connects the turbine and compressor wheels.

Fig. 1 illustrates a partial view of the turbocharger design process, highlighting the input/output dependencies between the product specifications among the design activities.

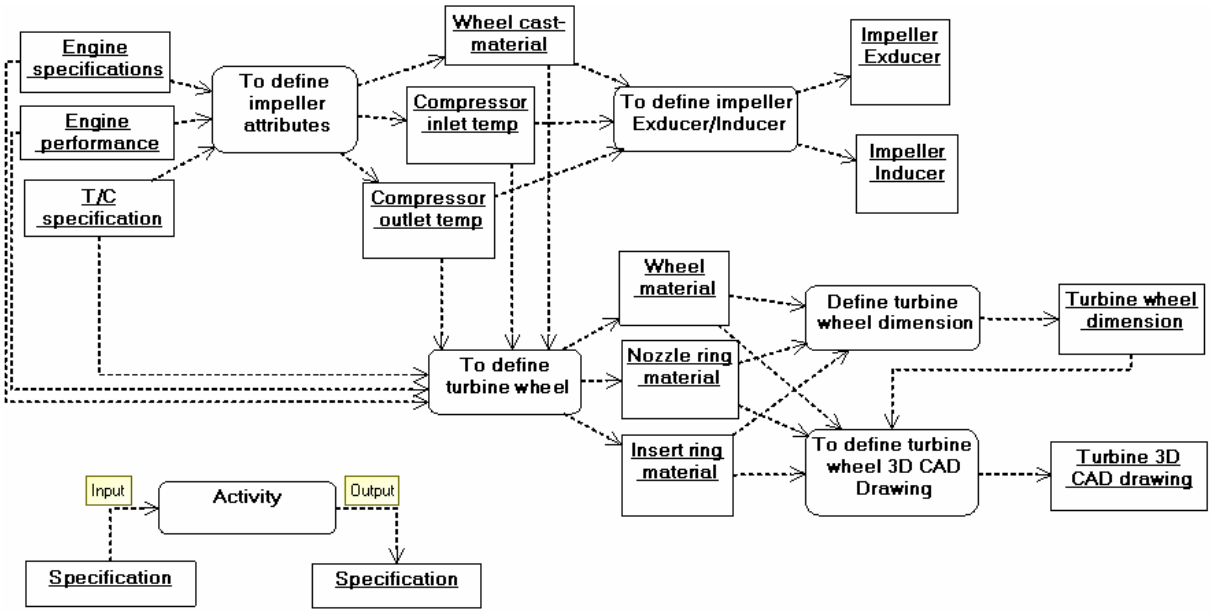

Fig. 1. Partial view of the turbocharger design process.

3. DEPNET

The main aim of the DEPNET solution is to capture dependencies and to explain them within a dependency network. This network is an oriented graph[7] composed of *nodes* corresponding to the product specifications handled during the design process and *arcs* corresponding to the dependency relationships between these specifications. In a context of collaborative design, dependency between two sets of data could be on *forward* or *feedback* direction. Forward-dependent data are those that require input from other activities, but not from themselves. Feedback-dependent data are those that need input from other activities, including from themselves. The feedback links are to be considered since they are a source of rework and thus are resource consuming and time consuming. Thus, two sets of data are said to be

[7] Arcs are directed to indicate whether a piece of data D1 depends on a piece of data D2, or vice-versa.

dependent in the case of forward dependency or interdependent in the case of feedback dependency. Some parts of the dependency network may therefore be cyclic.

3.1 Network nodes

Design work is dependent on a succession of tasks for defining a new product through the use and generation of various product specifications. The specifications can be, among many others, structural, functional, behavioural and geometrical. They correspond to the various product descriptions elaborated by designers during the development process, in terms of geometrical parameters, functions, bills of material, CAD drawings, simulation data and mechanical calculation results.

Depending on the flexibility given to the designer for elaborating product specifications, and on the values of specification properties, product specifications can evolve through different states until the product is finally defined. These different states can indicate whether the product specification is already fixed or under negotiation [8]. Grebici *et al.* [9] identify four product specification states: draft, exhibit, enable and deliver.

3.2 Network arcs

Among the criteria that characterise links between elements, the dependency link criterion remains the most studied in the literature and is the most complicated to treat in collaborative design modelling. According to Finger *et al.* [10], it is not sufficient to satisfy a set of constraints; it is important to represent and track dependencies within a set of constraints and to evaluate the qualitative and quantitative impacts of one product specification on another. Many definitions of dependency links have been proposed in literature. According to Kusiak and Wang [3], *a dependency between two specifications is the effect of the change in one specification's value on another*. According to Wang and Jin [11], *two specifications are said to have a dependency relationship if any of the pairs can not be complete without the other*. These definitions reveal the existence of two kinds of dependencies between product specifications: *dependency at creation* and *dependency at modification*. In addition to these dependency relationships, we distinguish two other kinds of dependencies between product specifications: *redundancy* and *consistency*. Each of these dependency kinds is presented in the following sections.

3.2.1 Redundancy relationship

In a concurrent design environment, it is important to provide different viewpoints of a product for the different task domains. This requires multiple models of a product part corresponding to the different domains [12]. Multiple-view modelling could be a good basis for an integrated modelling of different views. The approach for integrated modelling can be very helpful to an engineer who is solely responsible for the development of a complete product. In practice, however, several engineers are usually involved in the development of a product. As each has his/her own point of view on a particular product specification, redundancy of product specification could occur. *Two product specifications are said to be redundant when both of them describe the same entity and are expressed differently*. This could occur when the product specifications belong to different product model views. Furthermore, each product part has to be related to one or more parts that, together, implement the functionality of the conceptual component to be designed. Hence, several product specifications define a common interface (connection between two product parts) and describe the same entity; using a different expression for each of them. For example, in Fig. 2,

depending on the actor involved in the sub-assembly design, the specification of the ‘shaft diameter’ could have three different expressions signifying the same objective: ‘shaft diameter’ (for the shaft designer), ‘internal diameter of the turbine wheel’ (for the turbine wheel designer) and ‘internal diameter of the impeller wheel’ (for the impeller wheel designer).

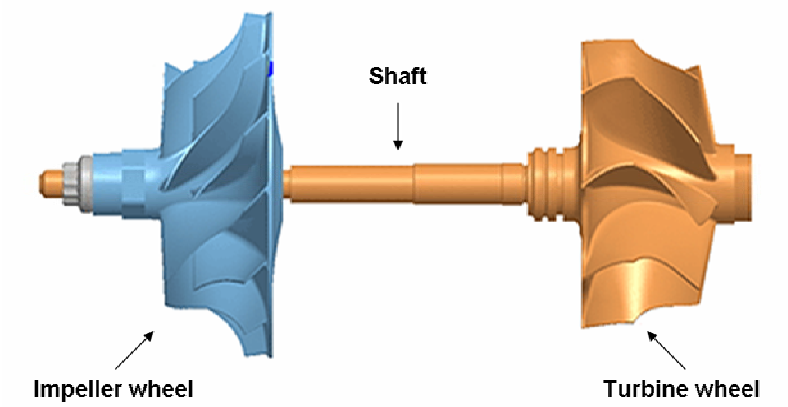

Fig. 2. Turbine wheel, impeller wheel and shaft sub-assembly.

Consequently, mechanisms to check these redundancies between product specifications are required, such as those mentioned in references [3] and [13], where specification redundancy is detected and removed. We note that a redundant specification should be removed without changing the design objective.

3.2.2 Consistency relationship

According to Nuseibeh et al. [14], we deal with consistency when *two product specifications do obey some relationship that is prescribed to hold between them*. This relationship between descriptions can be expressed as a constraint, against which specifications can be checked [13] [15]. Constraints can include equations, qualitative constraints and computer-based procedures. These constraints have an influence on rules. For example, the turbine wheel performance and the A/R specifications obey a qualitative constraint that is prescribed to hold between them. The turbine performance is greatly affected if the A/R of the housing is changed, since it is used to adjust the flow capacity of the turbine. Using a smaller A/R will increase the exhaust gas velocity into the turbine wheel. This provides increased turbine power at lower engine speeds, resulting in a quicker boost rise. However, a small A/R also causes the flow to enter the wheel more tangentially, which reduces the ultimate flow capacity of the turbine wheel. This will tend to increase exhaust backpressure and hence reduce the engine’s ability to *breathe* effectively at high RPM[8], adversely affecting peak engine power.

Several studies deal with the consistency maintenance issue, such as the multiple-view feature modelling for the integral product development approach as proposed by Bronsvoort and Noort [13]. This approach presents maintenance policies to keep different model views consistent, such as consistency definition, the way the consistency can be checked and the way the consistency can be recovered if needed. Currently, such consistency rules are captured in design project documents, others are embedded in tools and some are not captured

[8] Revolutions Per Minute

anywhere [14]. A common constraint repository is then necessary to capture all these consistency relationships.

3.2.3 Dependency at creation

Two specifications are said to be 'dependent at creation' if the creation of one of them depends on the creation of the other. Among the attributes proposed by Culley et al. [16] to qualify the dependency link, the *relevance*, *usage* and *completeness* attributes are the most relevant to qualify dependency at creation. We are particularly interested in the *completeness* attribute that is used to draw the actual product specification variation interval.

The designer will express how large the variation interval of the consumed product specification should be. The higher the completeness attribute value, the smaller the input product specification variation interval is. The completeness of a specification is often defined by its user with regard to his/her needs to produce other product specifications[9]. A measurement scale is proposed to evaluate this attribute. It comprises verbal descriptions of four different levels. A numerical indicator is assigned to each level as shown in Table 1.

Table 1.
Constructed attribute for product specification completeness

Attribute level	Description of the attribute level
0	Weak: the input product specification should be given below a certain maximum value
1	Not vital: the input product specification should be given within a certain value range
2	Vital: the input product specification should be given with the smallest value range
3	Extremely vital: the input specification should be precisely given

3.2.4 Dependency at modification

Two specifications are said to be 'dependent at modification' if the change in one of them implies modifying the second one. Among the attributes proposed in science management and engineering design works defining this dependency link, the following are the most relevant to qualify the dependency at modification: *level number* [17], *importance ratings* [18], *probability of repetition* [19], *evolution and sensitivity* [7]. We are particularly concerned with the estimation of the last two measures of dependency, since gathering input is difficult to obtain for simulating a development process that involves iteration. We should note that the *evolution* attribute, proposed to qualify activity dependency, refers to the variation of an activity. To express the variation of a product specification value we choose to use the term *variability* instead of the term evolution.

The *variability* can then be defined as *the likelihood that the output specification provided by one task would change after being initially released*. A product specification exhibits a high variability if its designer is incapable of estimating a value, or range of values, for the specification output. On the other hand, a product specification is said to exhibit low variability if its designer provides a good estimate of the output value.

It should be noted that some research works [20] deal with specification *stability*, which is the reverse of variability. It corresponds to the likelihood that a specification no longer changes during the remainder of the process.

[9] A product specification can be considered as complete in order to be used for a given product specification and not for another one.

The *sensitivity* is *the degree to which work is changed as the result of absorbing a transferred product specification.* If a product specification has a major sensitivity to its predecessor product specification, then a small change in predecessor product specification has a huge impact on the final results. On the other hand, if a product specification is not sensitive to preceding design changes, then the predecessor product specification has to change drastically before the dependent product specification is affected by this change.

Tables 2 and 3 describe, respectively, the four subjective levels of a product specification *variability* and *sensitivity*. This scale was developed using the techniques for constructing subjective attributes, as described by Keeney [21]

Table 2.
Levels of product specification variability – adapted from [22]

Attribute level	Description of the attribute level
0	Not variable: the output product specification does not vary
1	Low variability: the output product specification varies a little
2	Moderate variability: the output product specification is unstable
3	High variability: the output product specification is very unstable

Table 3.
Levels of product specification sensitivity – adapted from [22]

Attribute level	Description of the attribute level
0	Not sensitive: output product specification sensitivity is null to most input changes
1	Minor sensitivity: output product specification sensitivity is low to most input changes
2	Moderate sensitivity: output product specification sensitivity is medium to most input changes
3	Major sensitivity: output product specification sensitivity is high to most input changes

Once we have devised an instrument to measure completeness, sensitivity and variability, the next logical question is: how do we assign completeness, sensitivity and variability values to product specifications? A questionnaire based on structured expert interviews is compiled to support designers during the assessment. For an extensive discussion on subjective assessments of uncertain quantities utilising a structured expert interview process, please refer to references [23] and [24].

3.2.5 Synthesis

Product specifications can be interrelated by four kinds of dependency relationships, yielding an oriented network structure. These kinds of relationships are not disjointed; two existing specifications can be dependent at creation and at modification while being related with a consistency relationship. To help designers to evaluate dependencies between two existing specifications, the dependency measures at creation and at modification that are described in §3.2.3 and §3.2.4 are aggregated to form one criterion to express the *dependency degree* between the two product specifications (Eq. 1). As they are complementary attributes, a multiplicative utility function is used to aggregate the variability and the sensitivity attributes (V*S). Furthermore, the more the completeness is *high* (from *weak* to *extremely vital*) the more the required rework is long. Thus, for a given variability and sensitivity values, the more the completeness is elevated, the more the iterations are long and the more the dependency degree is high. In the case where the variability value is 0 and the completeness value is different from 0, the dependency degree value must be different from 0, since a “not null completeness” implies a dependency at creation. The dependency degree formula between two product specifications is then:

$$\text{Dependency Degree} = \text{Completeness} * (1 + (\text{Variability} * \text{Sensitivity})). \quad \text{(Eq. 1)}$$

Accordingly, the resulting range value of the dependency degree is an integer between 0 and 30, where {0, 1, 2, 3 and 4} denotes a null or weak dependency and a low risk of rework, and {15, 20, 21 and 30} denotes a very high dependency and a high risk of rework. The values {5, 6, 7, and 8} and {9, 10, 12 and 14} describe, respectively, a moderate and a high dependency, and risk of rework.

3.3 Approach taken to build the dependency network

The approach taken to build up the dependency network corresponding to one design situation involves, first, *keeping trace of the design process progression* by recording it in a database system and, second, applying a set of *SQL queries* on the obtained information to extract the network (see Fig. 3).

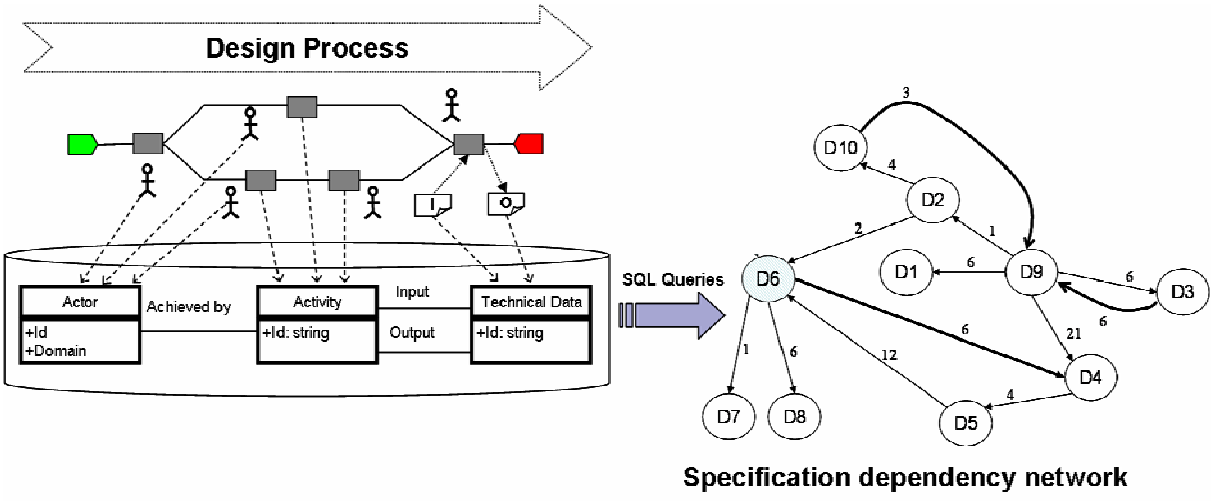

Fig. 3. Specification dependency network identification approach.

In order to keep track of the design process progression, we consider the questions proposed in the Zachman Framework [25]: **What** are the traceable items (product specifications); **Where** are the traceable items (design actions handling the product specifications); **Who** are the participants playing different roles in the creation, modification and exchange of product specifications; **Why** and **How** are product specifications created, modified and/or evolved in the way they are (the design rationale behind the design actions); and **When** are the product specifications created, modified and/or evolved (chronology of design actions).

Hence, we should highlight the following.

- In terms of design actions (**Where**), two management levels of the design process exist: *prescribed* and *emerging*. At the *prescribed* level, the process comprises phases, which in turn comprise planned activities. The *emerging* level corresponds to the non-planned activities occurring during the design progress.
- The key issues for the justification herein adopted (**Why**) are articulated as questions, with each *issue* followed by one *solution*, among several *alternatives*, that respond to the issue. *Arguments* support the adopted solution and could be requirement based, rule based, case based, and so on.

In order to store the various records tracing the design progression in a database system, the various elements presented previously are formalised in a UML class diagram. For more details about this diagram, please refer to reference [26].

This UML model is then used as a specification for a traceability tool, which can be seen as an *a posteriori* workflow engine to declare the ongoing design. It allows each actor involved in the design process to declare various pieces of information when performing his/her design action, for example: the phase (planned activity or non-planned activity being executed, name, identification), the objective of this design action, the actor domain and competencies, the input and the output product specifications used and generated (with the associated sensitivity, variability and completeness measures, as well as the state attributes), the various constraints between specifications that have to be followed while achieving the required design action (a constraint is expressed as a relation between two or more product specifications and represents a consistency relationship between these specifications) and, finally, the justification of the choices the actor made during the design action.

We should note that designers assess the completeness, variability and sensitivity attributes according to their expertise. However, if they have trouble in assessing these attributes, they can be assisted by questionnaires based on structured expert interviews. Examples of the questions are given in Table 4.

Table 4.
Assessment of variability, sensitivity and completeness attributes

Attribute	Questions
Variability	-How fixed are the design requirements? -What is the estimated date of change? -What are the possible causes of changes? For example, the specification has a connection with other specifications to be modified; assessing the specification is a complex design problem.
Sensitivity	-What is the design risk when a change occurs? -What is the rework risk when a change occurs? -What is the iteration duration estimation?
Completeness	-Are there other equivalent product specifications? -What is the level of expertise of the designer? -How uncertain is the specification?

Fig. 4 illustrates the developed traceability tool used to track the design progress. This screenshot shows the turbine wheel designer declaring the design process's related information.

Once the design workflow is declared, the captured elements are stored in a database, the tables of which correspond to the various classes of the UML traceability model. A set of SQL queries is then applied in the database to extract the specification dependency network. Fig. 5 gives a partial view of the network related to the turbocharger design process progression.

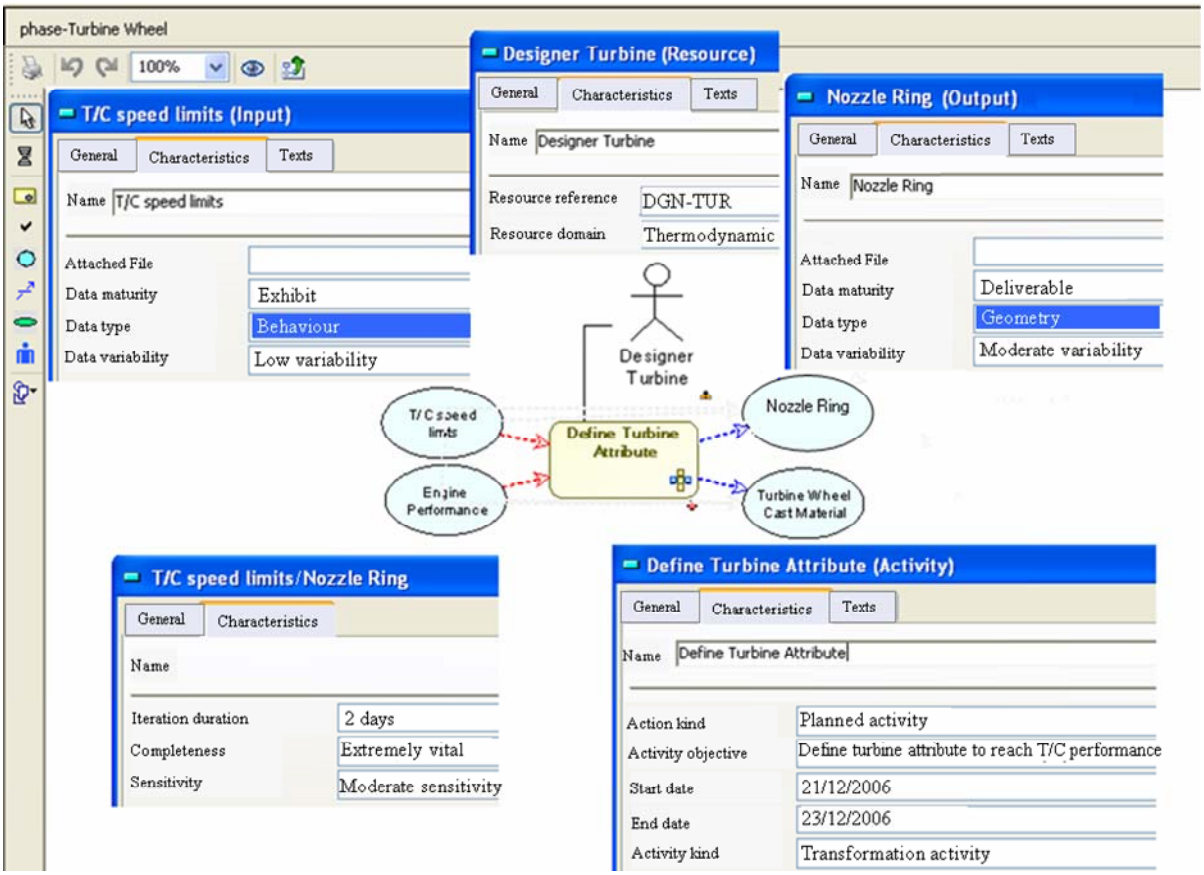

Fig. 4. Traceability tool to track the design process progression.

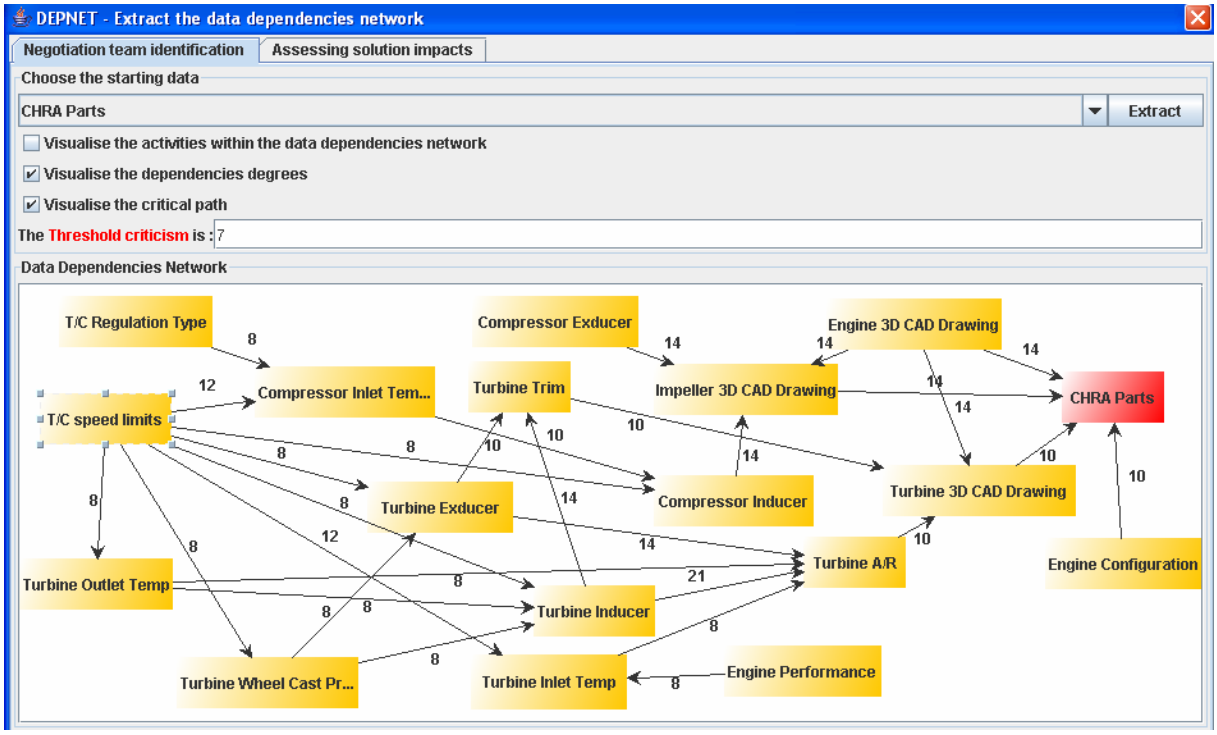

Fig. 5. Partial specification dependency network for the turbocharger design process progression.

4. Application of DEPNET in conflict management

The interaction of experts in a collaborative design context may give rise to conflict. According to Klein [27], conflict can occur when at least two incompatible design commitments are made, or when a design party has a negative critique of another design party's action. Hence, the conflict management process is a critical element of collaborative design. It can be seen as a succession of mainly five phases: conflict detection, identification of the conflict resolution team, negotiation management, solution generation and solution impact propagation. The focus of this section is on the conflict resolution team identification (prerequisite for conflict resolution) and the solution impact propagation (solution evaluation) phases. It illustrates how the dependency network supports designers to conduct these two phases.

4.1 Conflict resolution team identification

Conflict resolution cannot be achieved by one single actor; it requires the interplay of different areas of expertise. To avoid iteration in the conflict resolution process, it is highly advisable to do it in a collaborative way that seeks the input of many actors to reach a consensus quickly. These actors refer to those designers producing the product specifications leading to the source of the conflict. In order to identify these negotiators, the specification dependency network is extracted and the product specifications, on which the source of conflict depends, are identified through network backtracking (the source of conflict being the starting point). SQL queries are then applied in order to identify the negotiators forming the conflict resolution team.

In the case of the turbocharger example, a conflict is detected when the CHRA designers are defining their respective parts. The dependency network corresponding to this source of conflict is illustrated in the screenshot in Fig. 5. This network refers to the precedence relationship existing between the handled product specifications, to define the CHRA parts. The dependency degree between the handled specifications is also shown. The parties responsible for the realisation of the identified product specifications are then notified about the conflict occurring on the product specification 'CHRA parts', in order that they must participate in the negotiation process to find a solution, thus forming the conflict resolution team.

4.2 Impact assessment

The technical solution selected through the negotiation phase corresponds to the change of one or more product specifications involved in the design process, leading to the elaboration of the source of conflict. Hence, evaluating the impact of the chosen solution is carried out through the propagation of the product specification changes. This propagation is done through a downstream traversal of the dependency network, starting from the product specification 'solution'. In the considered case study, the solution given to resolve the detected conflict involves modifying the specification 'turbocharger speed limits'. A dependency network is then identified, the starting data being the 'turbocharger speed limits'. The impact of modifying this product specification is illustrated in Fig. 6, with a forward coverage of the identified network. A list of the product specifications to be modified is established as well as a list of the activities for applying these modifications.

The dependency network shown in Fig. 6 is enormous. Identifying the solution-dependent nodes and then the design activities to be re-executed may be an ad hoc and costly task. Not

all the activities responsible for the identified dependent product specification have to be re-executed; such an operation is costly and time consuming. Thus, the concept of a *critical dependency network* is introduced in order to reduce the number of solution-dependent nodes to be considered for design process re-execution. It involves eliminating product specifications with a *low dependency degree*, and everything stemming from them, among the impacted product specifications. Fig. 7 illustrates a critical specification dependency network where the selected minimum threshold is equal to 8.

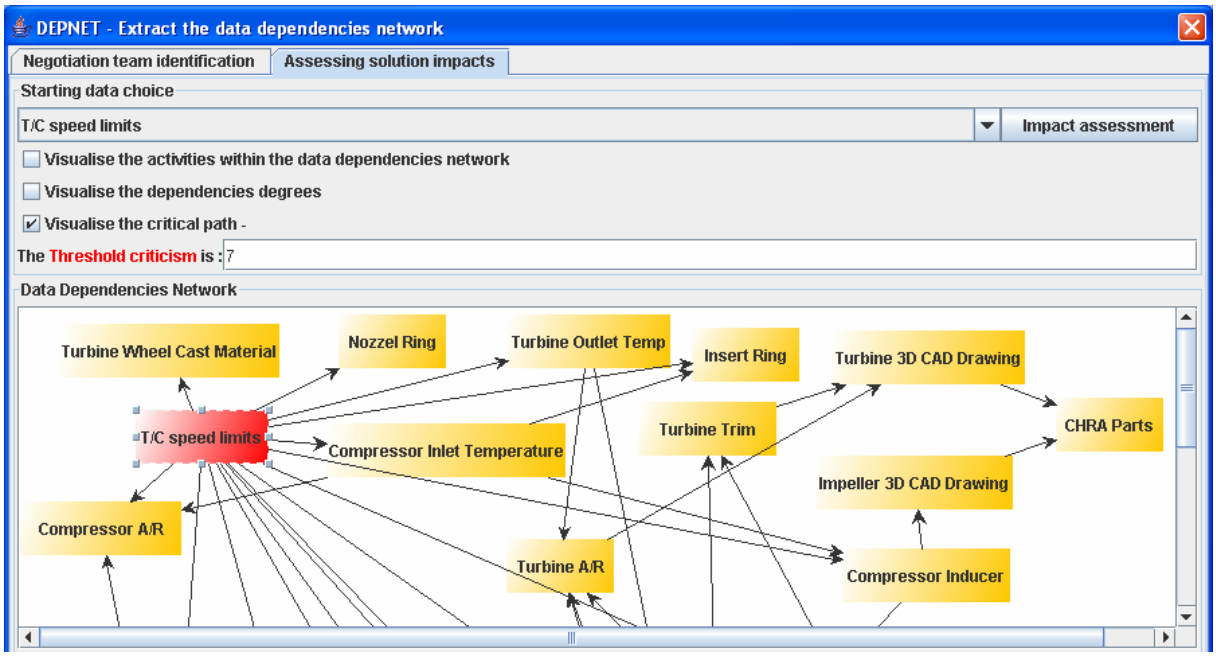

Fig. 6. Partial view of the product specification dependency network.

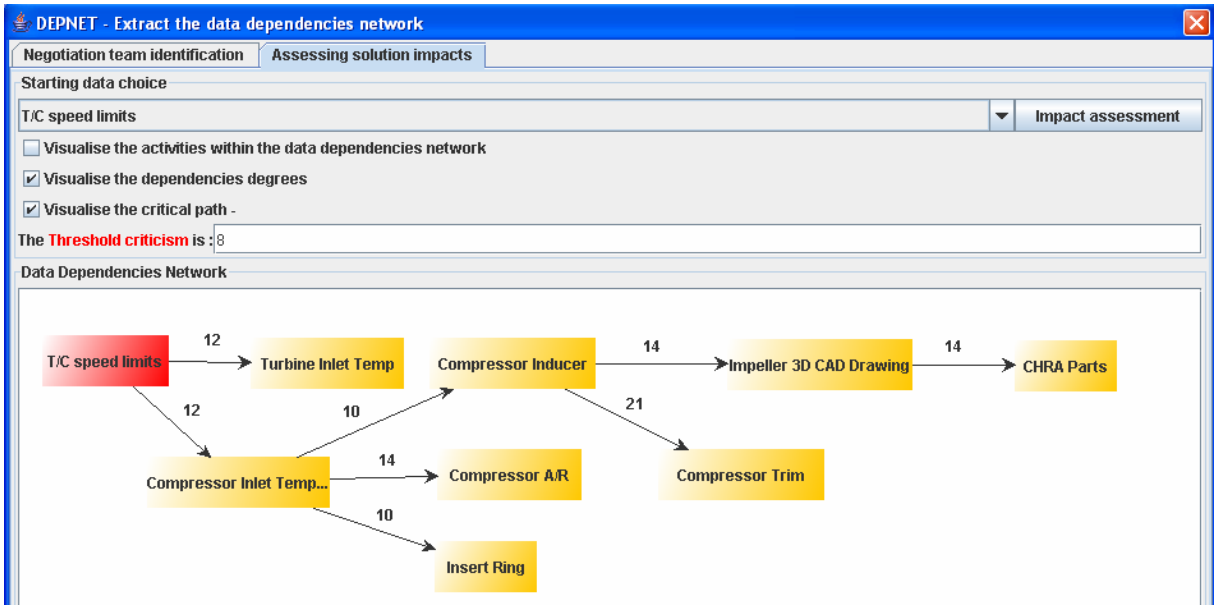

Fig. 7. Critical dependency network.

5. Conclusion and future prospects

Product specification dependencies are not always trivial and explicit, and current PDT tools are not able to extract these dependencies from informal and textual descriptions. As a

consequence, they cannot be discovered by the current PDT tools. Many studies have focused on data dependencies, but none of them has proposed mechanisms to identify these dependencies and the methods to manage them. The DEPNET solution presented in this paper captures product specification dependencies and structures them in a network. It proposes mechanisms to qualify these dependencies and defines ways of using these dependencies during the conflict management process, first, to identify the negotiation team, and second, to assess the solution impacts. The success of the DEPNET solution depends, like any other computerised application, on the involvement of people to populate and update the network. Under time pressure, actors may not be willing to record all information consistently. In this case, they are asked to capture a minimum of information (the activity name, input data and generated data). In some large cases, since the use of the DEPNET solution reduces iteration, design actors should adhere to the proposed solution.

Further points remain to be considered on the issue of specification dependencies management.

First, since the dependency network is dynamic and evolving during the process progress, an automated tool for charting and maintaining the coherence of this network would be helpful, especially in the case of complex products. Secondly, the DEPNET solution currently addresses specifications already generated in the design process. Extending this solution to cover all product dependencies, i.e., specifications that are not yet generated, would reduce conflict occurrence and assist designers in their work by taking into account the various interactions between product specifications. Thirdly, the dependency network proposed in this paper can provide a means for representing and for reasoning about the dependencies between product specifications. It may be useful to reduce the number of specification dependencies to the necessary minimum. For this purpose, mechanisms to check specification consistency would be proposed. These mechanisms can also be used as a decision tool to control product complexity and test various product configurations by eliminating some dependency arcs and analysing the impact of the corresponding alternatives. For this purpose, a set of criteria to analyse configurations, a constraints database and a verification mechanism are proposed. In both cases, graph theory can be used to achieve this goal.

On the other hand, in addition to tracing process design and generating a dependency network, the DEPNET solution can evolve into a rationale design capitalisation tool. Text mining techniques could be applied to the ‘justification’ table to extract new knowledge from stored cases. Moreover, specification dependencies can be transferred to other designs. A turbocharger for a petrol engine might differ significantly from, but may still feature the same specifications dependencies as, a diesel engine turbocharger. Dependency network reuse is especially interesting in the case of routine design. In the case of innovative design, transfer should be carried out gradually. When a piece of data, generated during a new design project, is identified as equivalent to another piece of data previously generated, only immediate dependencies should be transferred, i.e., arcs connected to that piece of data. The same reasoning applies to the transferred nodes.

Finally, the DEPNET solution has to be integrated, first with the CO2MED tool [28], already developed within the CRAN laboratory, to manage conflict resolution and, second with a Product Lifecycle Management (PLM) environment.

References

[1] Wang K.L. and Jin Y. Modelling dependencies in engineering design. The ASME Design Theory and Methodology Conference, DETC99/DTM-8778, September 12-15. Las Vegas, Nevada (1999).

[2] Eppinger S.D., Whitney D.E., Smith R.P. and Gebala D.A. A Model-Based Method for Organizing Tasks in Product Development. Research in Engineering Design, Vol. 6, (1994) 1-13.

[3] Kusiak A. and Wang J. Dependency analysis in constraint negotiation. IEEE Transactions on Systems, Man, and Cybernetics, Vol. 25, (1995) 1301- 1313.

[4] Dong A. and Agogino A.M. Managing design information in enterprise-wide CAD using ‘smart drawings’. Computer-Aided Design, Vol. 30(6), (1998) 425-435.

[5] Browning T.R. Applying the design structure matrix to system decomposition and integration problems: a review and new directions. IEEE Transactions on engineering management, Vol. 48(3), (2001) 292-306.

[6] Yassine A. and Braha D. Four Complex Problems in Concurrent Engineering and the Design Structure Matrix Method. Concurrent Engineering: Research and Applications, Vol. 11(3), (2003) 165-176.

[7] Krishnan V., Eppinger S.D. and Whitney D.E. A Model-Based Framework for Overlapping Product Development Activities. Management Science, Vol. 43(4), (1997) 437-451.

[8] Giannini F., Monti M., Biondi D., Bonfatti F. and Monari P.D. A modelling tool for the management of product data in a co-design environment. Computer-Aided Design, Vol. 34(14), (2002) 1063-1073.

[9] Grebici K., Blanco E. and Rieu D., Toward non-mature information management in collaborative design processes, in International Conference on Engineering Design, ICED'05. 2005, The Design Society: Melbourne, Australia.

[10] Finger S., Fox M.S., Navinchandra D., Prinz F.B. and Rinderle J.R. Design Fusion: A Product Life-Cycle View for Engineering Designs. Second IFIP WG 5.2 Workshop on Intelligent Computer Aided Design. 19-22 September. Cambridge, U.K. (1988).

[11] Wang K.L. and Jin Y. Managing dependencies for collaborative design. The ASME Design Engineering Technical Conferences and Computers and Information in Engineering Conference. September 10–13. Baltimore, MA. (2000).

[12] Bidarra R., Kranendonk N., Noort A. and Bronsvoort W.F., A collaborative framework for integrated part and assembly modeling in Proceedings of the seventh ACM symposium on Solid modeling and applications. 2002, ACM Press. 389-400: Saarbrücken, Germany.

[13] Bronsvoort W.F. and Noort A. Multiple-view feature modelling for integral product development. Computer-Aided Design, Vol. 36(10), (2004) 929-946.

[14] Nuseibeh B., Easterbrook S. and Russo A. Making inconsistency respectable in software development. Journal of Systems and Software, Vol. 58(2), (2001) 171-180.

[15] Raghothama S. and Shapiro V. Consistent updates in dual representation systems. Computer-Aided Design, Vol. 32(8-9), (2000) 463–477.

[16] Culley S.J., Davies S., Hicks B.J. and McMahon C.A. An assessment of quality measures for engineering information sources. 15th International Conference on Engineering Design. August 15-18. Melbourne, Australia: The Design Society (2005).

[17] Steward D.V. System Analysis and Management: Structure, Strategy and Design, New York: Petrocelli Books, NY, USA. (1981).

[18] Pimmler T.U. and Eppinger S.D., Integration Analysis of Product Decompositions, in The ASME Design Theory and Methodology Conference (DTM'94). 1994. p. 343-351.
[19] Browning T.R. and Eppinger S.D. Modelling Impacts of Process Architecture on Cost and Schedule Risk in Product Development. IEEE Transactions on Engineering Management, Vol. 49(4), (2002) 428-442.
[20] Terwiesch C., Loch C.H. and DeMeyer A. Exchanging Preliminary Information in Concurrent Engineering: Alternative Coordination Strategies. Organization Science, Vol. 13(4), (2002) 402 - 419.
[21] Keeney R.L. Value-Focused Thinking: A Path to Creative Decision making, Published by Harvard University Press, Cambridge MA. (1992).
[22] Yassine A., Falkenburg D.R. and Chelst K. Engineering Design Management: An Information Structure Approach. International Journal of Production Research, Vol. 37(13), (1999) 2957- 2975.
[23] Merkhofer M. Quantifying judgmental uncertainty: methodology, experiences, and insights. IEEE Transaction on System, Man and Cybernetics, Vol. 17, (1987) 741-752.
[24] Shephard G. and Kirkwood C. Managing the judgmental probability elicitation process: a case study of analyst/manager interaction. IEEE Transaction on Engineering Management, Vol. 41, (1994) 414- 425.
[25] Zachman J.A. A Framework for Information Systems Architecture. IBM Systems Journal, Vol. 26(3), (1987) 276-292.
[26] Ouertani M.Z., Grebici K., Gzara L., Blanco E. and Rieu D. DEPNET: A Methodology for Identifying and Qualifying Dependencies Between Engineering Data. The future of product development, Proceedings of 17th CIRP International Design Seminar. 26-29 March. Berlin, Germany: Springer-Verlag (2007).
[27] Klein M. iDCSS: Integrating Workflow, Conflict and Rationale-Based Concurrent Engineering Coordination Technologies. Concurrent Engineering: Research and Applications, Vol. 3(1), (1995) 21-27.
[28] Lombard M. and Rose B., Conflict management in engineering design: industrial evidence from CO²MED software, in International Conference on Engineering Design, ICED'05. 2005, The Design Society: Melbourne, Australia.