*Transdisciplinary Engineering: Crossing Boundaries*
*M. Borsato et al. (Eds.)*

*doi:10.3233/978-1-61499-703-0-279*

# Managing Dependencies in Heterogeneous Design Automation Systems

Tim HJERTBERG [1], Roland STOLT and Fredrik ELGH
*School of Engineering, Jönköping University, Sweden*

**Abstract.** Increasing competition in cost efficiency, lead-times, product quality, quotation accuracy, and abilities to provide customization drives companies toward development and adoption of new methods. To re-use knowledge gained from previous projects in order to avoid producing the same knowledge again and to circumvent previously encountered obstacles is an approach which is more or less used by most companies. Utilization of Design Automation (DA) systems in the engineering design process have proven to increase process efficiency and to enable new ways of working by systematic re-use of engineering knowledge. In order to ensure system longevity, industrial practitioners and researchers have pointed at implementation and long term management as important aspects to consider during development. The systems are often built on top of commercial software and legacy systems integrated by different types of scripts and system descriptions which becomes dependent of each other in different ways. Changes made during maintenance in one of these artifacts propagates through the dependency structure making traceability and transparency key factors for keeping the system valid over time. This paper presents a description of the problem in a real industrial setting together with a suggestion of an approach, based on set-up and management of dependencies between sections inside and across different types of system components, which is aimed to aid implementation and management of DA tools. A prototype system which informs the user, of functional sections related to a functional section to be updated, have been developed. The prototype is applied on a multidisciplinary heterogeneous system environment used for simulation based knowledge build up and concept evaluations of jet engine components.

**Keywords.** Design Automation, Dependency Management, Customization

## Introduction

OEMs and consumers are more frequently demanding high levels of customization of products. Subcontractors are significantly affected by this since they need to compete with other subcontractors with providing the most appealing quotation. To be able to meet the demands, the subcontractors' infrastructure have to be flexible enough to be able to provide a large range of external variety and at the same time keep costs low. This increased demand of customization can be reflected in an increasing amount of research within the field of customization [1, 2]. Allowing high levels of customization in manufacturing and development processes generally results in high internal variety, which in turn is related to increased complexity [3] and increased cost [4] of the processes. One way to counteract the increased cost, resulting from increased

[1] Corresponding Author, E-Mail: Tim.Hjertberg@ju.se

customization, is to actively make use of previously produced knowledge. By re-using knowledge, creation of the same knowledge over and over is avoided and thereby avoiding costs related to the synthesis of the knowledge. Re-use of knowledge also have potential in reducing time since in many cases, less time is needed to implement already existing knowledge than to reproduce it. In technology and product development processes, many attempts have been made to re-use knowledge by letting computer based support tools make use of it in different ways. Design Automation (DA) is an approach which acts on well formalized engineering processes by automatically performing tasks with the help of stored knowledge. There are many approaches to DA which can be used differently depending on i.e. the maturity of the considered process, where in the development process the DA system is supposed to act, or the level of customization the system should enable. What could be seen as the simplest type of customization enabling DA system is a configuration system. Configuration systems make use of a modular product structure which is configured in the most suitable way for a specific customer [5, 6]. Other types of systems, with the ability to provide higher degrees of customization, deals with i.e. parametric design [7], or Knowledge Based Engineering (KBE) [8]. Systems also exists with the purpose of generating and evaluating versions of design concepts, often with the help of simulation software [9, 10]. The Design Automation tools introduces new ways of working when implemented but also comes with an investment. This investment is re-gained if the system can be active long enough to enable savings, or produce value in other ways, proportional to the investment. Over time, processes can change, new requirements can be added to system functionality, new knowledge is added, knowledge is changed, and environmental changes can be made to the environment in which the system is built and operates. These types of changes provides challenges in keeping the system operational over time and companies adopting Design Automation systems have described this as problematic. Amongst others, two areas have been described as underlying reasons as of why these types of changes are challenging and hard to handle. Transparency of the system itself, and traceability of the knowledge which the system makes use of or is built upon [11]. The same paper presents a walkthrough of existing methodologies for system development focused on DA related applications. Concluded from this investigation, it can be seen that existing methodologies does not support implementation and maintenance of the systems being developed. The methodologies however states that these are aspects with importance and affects the success of the final implemented system. Transparency have been pointed out as a factor which affects longevity of DA systems [12, 13] and refers to the ability of accessing the system and its components as well as knowledge used by the system. Low transparency of the system results in time consuming processes for performing maintenance and updates. If parts cannot be accessed, they could over time be rendered invalid, thus reducing functionality, performance and/or accuracy of the system. Traceability will in this context be referred to the ability to follow an artifact, and the knowledge fragments of which it is built from, through its development and life.

In this paper an attempt to provide an approach, which aids implementation and management by proactively introducing traceability during DA system development, is presented. The traceability is gained by keeping track of dependencies between functional sections in the system and through this, providing several possibilities to facilitate management of the system.

## 1. Dependency management

Throughout engineering processes, large numbers of documents are created. The documents have varying scope and purpose. These documents describe the product from different viewpoints in different levels of abstraction. They can describe legal limitations of products and processes, specify customer requirements, contain knowledge of how to design and evaluate concepts. Dependencies often exists between the documents which can refer each other in many ways depending on the format. Dependencies can act on specific parts of documents, creating a complex dependency structure. Extensive work is required to track these sub-document level dependencies when changes have been made and the document collection have to be checked for consistency.

Documents are often subjected to change during the engineering processes and it is important that they are consistent with each other [14]. The management of documents in such heterogeneous environments have frequently been pointed out as important in order to maintain consistency in document clusters and thereby keeping systems and documents valid [15]. Monticolo et al. [16] addresses this problematic, focused on the engineering design process and expert models connected to CAD and CAE models. They describe the problem in a concurrent engineering perspective where information such as parameters, expert rules, and mathematical relations are shared by several users in different disciplines. They further state that tools existing today is not capable of managing encapsulated knowledge and cannot ensure that information is consistent through different heterogeneous expert models. A Knowledge Configuration Model (KCModel) is proposed with the aim to allow for acquisition, traceability, re-use, and consistency of explicit knowledge used in configuration. The solution for consistency is based on checking every knowledge instance used in a knowledge configuration with all other configurations. Their approach is constrained to explicit knowledge. Scheffczyk et al. [17] proposes the use of strict explicit formal consistency rules in order to obtain consistency in heterogeneous repositories. They present a tool which can be used to automatically achieve consistency or to pinpoint inconsistencies in document structures. By setting priorities to the rules, an impact assessment can be extracted from the inconsistency analysis. Hutter et al. [18] presents a system called MAYA. The system is described as a tool which maintains formal developments. To interact with MAYA, the user translates specifications to a formal specification language. The specifications contain theories in which, when the specification is translated to the formal language, proof obligations are defined to indicate relations to other theories. External theory provers, such as the one presented in [19], can be connected to the software in order to operate the proof obligations.

Most research found which deals with consistency of document clusters are presenting methods of how to automatically achieve consistency by enforcing a set of rules on the content of the documents. Egyed [20] presents a method for automatically detecting and tracking inconsistencies in software design models. Engineers have to define consistency rules which is used by the system in order to automatically detect violations of the rules. The violations are presented to the user which has to evaluate if the inconsistencies are relevant to deal with or not. Xiong et al. [21] introduces a language, called Beanbag, for the purpose of creating automated fixing procedures in software development environments. The language is based on languages for writing consistency relations but is also adapted for the adding of semantics which is used in order to provide a description for the fixing procedure. Spanoudakis et al. [22] have developed a model and a prototype system on the model, used to generate traceability relations. Thus, traceability rules have to be defined manually. These rules are represented in XML from which the prototype system is able to produce four types of traceability relations. A very similar model can be seen in [23].

A lot of research have been done to the considered topic. Methods and tools exists, which helps software developers or other practitioners to keep their document and system environments consistent and updated. Tools exists which can automatically keep track of relations between documents or make changes to code in order to re-obtain consistency. However, in order to build the environments required for the tools to work, a lot of manual work will have to be done prior to obtaining automatic consistency checks. Most of the tools are developed with focus on large scale software development, specific problems or system entities, and are supposed to be used by pure software developers. In the engineering design field, a lot of smaller software tool development projects are performed, without the intention to be part of a larger system in the future, relatable to the System-of-Systems concept. The individual software tools are often developed by the design engineers themselves who are not very prone to doing extensive documentation work, not by software developers.

No solutions have been found which have the ability to explore the content inside different types of documents, keeping track of relations between sections in one document type to sections in another document type, and doing this with a low amount of set-up effort.

## 2. Dependency management in DA systems

In this section an approach, Figure 1, of how to work with dependency management in DA system environments is presented.

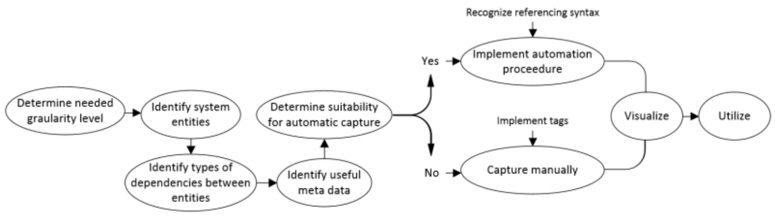
Recognize referencing syntax
Yes
Implement automation proceedure
Determine needed graularity level
Identify system entities
Determine suitability for automatic capture
Visualize
Utilize
Implement tags
Identify types of dependencies between entities
Identify useful meta data
No
Capture manually

**Figure 1.** Proposed approach for dependency management in DA environments.

*Granularity Levels* - Dependencies can be captured in different levels of granularity depending on needs in specific cases. A fine granularity level enables visualization of the system structure in different views. Depending on the purpose for using the system or which person it might be desirable to have this possibility. Setting up the system dependency structure in fine granularity enables different stakeholders to filter the view to suit their discipline or wanted level of abstraction. An example of granularity levels can be seen in Figure 2 where the children of a parent is a finer grained representation of the parent.

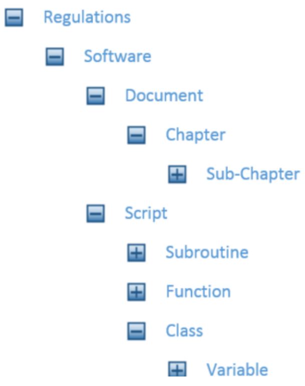

**Figure 2.** Example of granularity levels

*Meta Data* - By adding meta data to captured dependencies, or while capturing dependencies, the efficiency of the utilization of the stored dependency structure can potentially be increased. Information about the person who captured a specific dependency enables the possibility to contact this person for consultation when a change is planned for a considered dependency. Descriptions of the purpose of the dependency and how the affected system entities interact technically, enabled engineers to be quickly informed and saves them from going through code or documentation in order to figure this out. If there are any specific demands which are required in order to keep the dependency valid, this could be added here. These could be that a script needs to work against a specific version of a commercial software in order to work, or that a variable need to be kept within a certain range.

*Capturing Dependencies* - Depending on what type of dependencies exists in the system, they can be captured in different ways. Types of dependencies can be divided in many ways. In this paper dependencies will be divided in two groups, structural, and passive dependencies, as described in [24]. Dependencies can be captured manually or automatically depending on how they are formalized in the system or documentation. If the dependencies occur in a standardized format, these could be found by an algorithm and automatically captured. Dependencies which are not described in a predictable way or if it for some reason is not worth to build the structure needed for automatic capture, they can be captured manually. In this case it is proposed to introduce tags, containing the desired information, to the entities. These tags can be built in such way that an algorithm can find them and thereby enabling a semi-automated capture. Programming languages usually describes several types of dependencies which easily can be captured automatically. These could be relations between subroutines, functions, classes, and libraries. Dependencies which are typically hard to capture automatically are the passive dependencies. These are often described in natural, non-formal language and might have to be captured manually. Cross-platform dependencies can also be hard to capture automatically since communication between two platforms can occur in several different ways. One must ensure that all ways of communication is covered in the algorithm to ensure that all dependencies are captured and that they are captured in the correct way.

*Visualization* - The captured dependencies can be used to visualize the system structure in different ways in order to obtain overviews of the system. Informative views can be obtained by configuring the dependencies using the meta data and the granularity levels. Utilization of filtering and clustering techniques provides possibilities to create discipline specific views by removing irrelevant parts or by putting focus on relevant parts. By using the granularity levels, views which require more or less previous knowledge about the system can be obtained. This enables creation of visualizations adapted for stakeholders with focus on varying degrees of technicality or abstraction.

*Transparency/Accessibility* - The dependency structure can be used in order to obtain transparency of the system environment. By providing direct access to system components such as scripts or descriptive documents through the utilized visualization approach, the engineers would not have to search for the files, and could also be guided to the correct place inside the considered system entity without manual navigation or interaction with PLM or version control systems. Interfacing functionality could also provide previews and editing capability of system entities without having to open them in their native development environment.

*Impact assessment and change propagation* - During maintenance of system entities, it can be hard to assess the effect of a change, to other system entities. The scope of the affected area can vary a lot with different types of changes. Through the dependency structure the engineers can get estimations of the impact of a change depending on what types of relations it have to other system entities, or how many dependencies the entity considered for change have to other parts of the system. The finer granularity of which the system is described in, the higher the accuracy, of the impact assessment, will be.

When a change is made, it will propagate through the system via the dependency structure. Depending on the nature of the change, it might affect components of the system, outside of the changed component. Further change might have to be done to affected components in order to regain consistency. This behavior can thereby keep propagating through the system. By investigating meta data captured in the dependency structure, engineers could determine if change have to be performed to interfacing components.

## 3. Case Study

In this chapter, a description of the problem in a real industrial setting is presented together with a suggestion of an approach, based on modelling and management of dependencies between functional sections inside and across different types of system components, which is aimed to aid implementation and management of DA tools.

A case study have been performed in collaboration with a company in the aerospace industry. Workshops and interviews were held with several people from the company, working mainly with technology or product development but who also were heavily involved with development of DA systems. Focus of the activities was to further develop the understanding of needs presented in [11].

The company is a global actor in the area of development, production, service and maintenance of components for aircraft engines, rockets and gas turbines with high technology content. The company provides products that are completely custom engineered in an international market with high competition. The products are integrated in complex systems working in extreme environments for long time periods with both customer and legal demands for complete documentation and traceability. The company takes full responsibility for the functionality of their products during its operation including service, maintenance and updates. Fulfilling these harsh requirements is a challenge but at the same time an opportunity to sustain a competitive edge. Automation of design and production preparation by the use of knowledge based engineering (KBE) has been used at the company for more than a decade to enable quick adaptation to changes in customer specifications and evaluation of different design solutions. In order to aid the concept development phase, a multidisciplinary analysis system containing KBE applications is currently being developed by the company. The purpose of the system is to provide knowledge of how changes of the design parameters affects a concept. This knowledge is obtained by performing analyses in a number of different disciplines. Simulations of cycle lifetime, stiffness, buckling, producibility, thermal effects, and more are performed and the results are compiled and sent to the concept developers. The system consists of several different commercial software, controlled and stitched together with in-house developed software and scripts written in several different programming languages. When realizing the systems, the company engineers follow method descriptions called Design Practices together with other documents and knowledge sources. The design practices are directed towards describing the execution of a certain task on a certain component e.g. meshing a CAD model. Connecting these documents to program code are seen as challenging but important in order to obtain high traceability through the system. Over time the design practices as well as the program code are updated and subjected to changes which creates problems in keeping the connection valid. The integration of this kind of systems in its intended environment are seen as an important aspect although problematic. Aspects such as knowledge traceability through the system as well as system output representation are thought to have an impact on the success of the implementation. From the workshops and interviews a set of success criteria, thought to have potential to overcome the main obstacles for DA system success, were derived. For each success criteria a set of enablers, thought to have the ability to enable the fulfillment of the success criteria, were derived. Emphasis of the result from the interviews and workshops could be found around the aspects connected to system transparency and knowledge traceability which was thought to be enabled by connecting related parts of the system to each other.

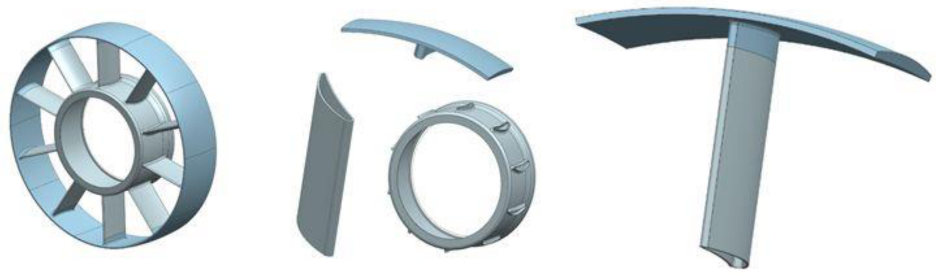
**Figure 3.** Welding assembly sequence of a structural jet engine component.

A system that is currently developed at the company was used as subject for introducing dependency management as a means to achieve increased system transparency and facilitated knowledge traceability. The system is used as a module in a larger system which performs a set of analyses in order to build knowledge about concepts. This specific module is used to perform producibility evaluations by analyzing a components geometrical features in relation to available manufacturing processes. The Producibility Assessment System (PAS) is built on two commercial software (Siemens NX, and MS Excel), three different programming languages (VB, VBA, and NX Knowledge Fusion), and has connections to normative descriptions written in non-formal natural language in MS Word documents. The system has been used to perform producibility assessments of a structural component, connecting a jet engine to the air craft wing, Figure 3. 128 versions of the component are automatically generated from a base-line model and run through the system which evaluates the different versions with consideration of, for the company available, welding techniques.

## *3.1. Applying dependency management*

In the test-case with the PAS, most dependencies were captured manually. Automatic capture was performed on one type of dependency. An algorithm was written in python for automatic capture of dependencies between knowledge fusion scripts. The knowledge fusion language is developed by Siemens and is used to perform actions in the CAD software Siemens NX. When the dependency structure was set up on the PAS, the finest granularities consisted of chapters in natural language documents, and subroutines/functions/classes in scripts. This resulted in 81 structural dependencies and 2 passive dependencies. 5 dependencies were caught automatically and 78 were caught manually. 63 of the 78 are directly connected to how the used programming languages calls or executes other entities of the system. Capture of these dependencies have potential in being automated in the same way as the capture of dependencies between the Knowledge Fusion scripts. This means that 82% of the dependencies in this system has potential in being captured automatically with simple algorithms. This is without including possible automatic capture of cross-platform dependencies or attempts to standardize natural language descriptions. In the test case, two different ways of visualization were tested. A natural way of presenting the dependency structure is in a regular tree structure. However, when the system grows and more dependencies are introduced, it can be hard to keep a clear overview at fine granularity levels. Filtering techniques can be used in order to improve the ease of use. For the second visualization approach an open source software for network exploration, Gephi [25], was used in order to build graphs. The graphs show system entities as nodes and dependencies as lines between nodes. Several different layout algorithms can be applied to the graphs in order to produce clear views of the structure. Filtering and clustering techniques can also be applied in Gephi to further improve the usability of the visualized dependencies. A python script was used in order to generate input files, representing the dependency structure of the PAS system, for Gephi. The generated input files were imported into Gephi and resulted in the plots shown in Figure 4, where the color scale from green to red indicates how many interactions a system entity has with other entities. Dependencies can also be weighted in order for certain dependencies to affect the visualization in a way which reflects its importance.

Meta data can be displayed in the graphs and they can be filtered and searched in order to provide suitable views for certain situations.

Transparency was in this case study introduced by providing access to the system entities registred in the dependency structure. Two different technical solutions for achieving this were tested. The user of the system was given the possibility to open the system entity, in its native environment, directly from the visualization tool. Text based entities, such as code or natural language documents, can be displayed in the visualization tool in order to enable quick previews. If a dependency acts on a specific part inside such documents, this specific part is located and displayed for the user in the visualization tool.

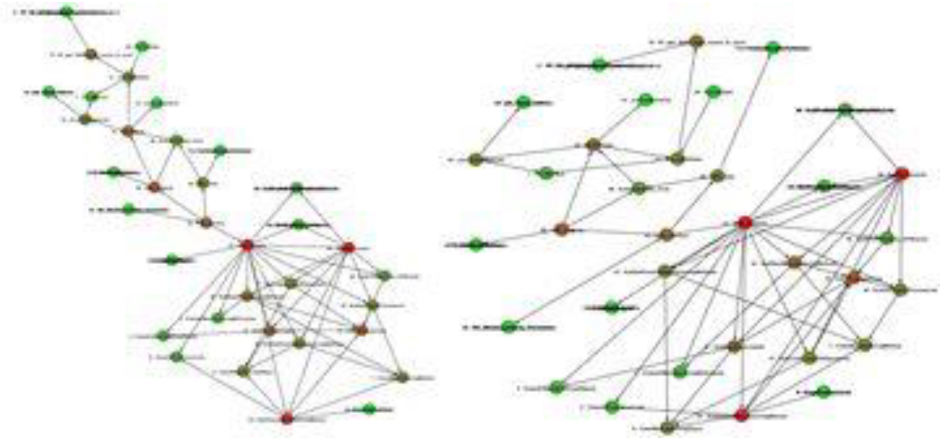

**Figure 4.** Gephi visualizations of the PAS system.

## 4. Conclusions

The objective of this work was to introduce an approach with potential of achieving traceability and transparency in heterogeneous system and document environments such as the environment of a typical DA system. The traceability and transparency were introduced with the intention to enable a more efficient maintanance process of the considered environements. An atempt to achieve this was made by introducing dependency management on a sub-document level, allowing cross-document type dependency capturing. Manual labour was cut by introducing automatic capture of certain dependency types. In the test case 82% of the total amount of dependencies had potential for automatic capture. Two important parts of the approach is the consideration of granularity levels and the capturing of meta data. These can be used to create clear and explanatory overviews of the system in which the flow of information and knowledge can easily be traced through the system structure without having to obtain this through document and code scrutinization. A conclusion based on reviewed literature, industrial input, and the case study presented in this article, is that there is a need for approaches which provides traceability and transparency to the concidered type of environments, and that dependency management and visualization seem to have potential in achieving this. However, further evaluation will have to be performed in an industrial setting in order to obtain further verification and validation. Future work will include a more extensive evaluation of the presented approach in the industrial environment of the case company. Visualization techniques and meta data representation will be further investigated.

## Acknowledgements

The authors would like to express gratitude towards the participating companies in the study as well as The Knowledge Foundation who partly funds the project.

## References

[1] F.S. Fogliatto, G.J.C. da Silveira, and D. Borenstein, The mass customization decade: An updated review of the literature, *Int. J. of Production Economics,* Vol. 138, 2012, No. 7, pp. 14-25.

[2] G. Da Silveira, D. Borenstein and F. S. Fogliatto, Mass customization: Literature review and research directions, *International Journal of Production Economics,* Vol. 72, 2001, pp. 1-13.

[3] D. Krause, G. Beckmann, S. Eilmus, N. Gebhardt, H. Jonas and R. Rettberg, Integrated Development of Modular Product Families: A Methods Toolkit, In W. T. Simpson et al. (eds.) *Advances in Product Family and Product Platform Design: Methods & Applications*, Springer, New York, pp. 245-269, 2014.

[4] L. Hvam, M. Malis, B. Hansen and J. Riis, Reengineering of the quotation process: application of knowledge based systems, *Business Process Management Journal,* Vol. 10, 2004, pp. 200-213.

[5] L. Hvam, N.H. Mortensen and J. Riis, *Product customization*, Springer-Verlag Berlin, 2008.

[6] A. Claesson, *A Configurable Component Framework Supporting Platform-based Product Development*, PhD thesis, Chalmers University of Technology, 2006.

[7] K. Amadori, M. Tarkian, J. Ölvander and P. Krus, Flexible and robust CAD models for design automation, *Advanced Engineering Informatics,* Vol. 26, 2012, No. 4, pp. 180-195.

[8] J. Johansson, Manufacturability analysis using integrated KBE, CAD and FEM, *Proceedings of the ASME Design Engineering Technical Conference*, 2008, pp. 191-200.

[9] J. Johansson, A flexible design automation system for toolsets for the rotary draw bending of aluminium tubes, in *2007 Proc. of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, DETC2007*, 2008, pp. 861-870.

[10] O. Isaksson, A generative modeling approach to engineering design, *DS 31: Proceedings of ICED 03, the 14th International Conference on Engineering Design,* Stockholm, 2003.

[11] T. Hjertberg, R. Stolt, J. Johansson, M. Poorkiany and F. Elgh, Implementation and Management of Design Systems for Highly Customized Products – State of Practice and Future Research, In: R. Curran et al. (eds.), *Transdisciplinary Lifecycle Analysis of Systems. Proceedings of the 22nd ISPE Inc. International Conference on Concurrent Engineering*, Delft, July 20-23, IOS Press, Amsterdam, 2015, pp. 165 - 174.

[12] M. Cederfeldt, Planning Design Automation : A Structured Method and Supporting Tools, PhD thesis, Chalmers University of Technology, Göteborg, 2007.

[13] W.J.C. Verhagen, P. Bermell-Garcia, R.E.C. van Dijk and R. Curran, A critical review of Knowledge-Based Engineering: An identification of research challenges, *Advanced Engineering Informatics,* Vol. 26, 2012, pp. 5-15.

[14] S. M. Becker, T. Haase and B. Westfechtel, Model-based a-posteriori integration of engineering tools for incremental development processes, *Software & Systems Modeling,* vol. 4, pp. 123-140, 2005.

[15] D. Hutter, Semantic Management of Heterogeneous Documents, In A. Aguirre et al. (eds.) *MICAI 2009: Advances in Artificial Intelligence*. vol. 5845, Springer Berlin Heidelberg, 2009, pp. 1-14.

[16] D. Monticolo, J. Badin, S. Gomes, E. Bonjour and D. Chamoret, A meta-model for knowledge configuration management to support collaborative engineering, *Comp. in Industry,* Vol. 66, 2015, pp. 11-20.

[17] J. Scheffczyk, U. M. Borghoff, P. Rödig and L. Schmitz, Consistent document engineering: formalizing type-safe consistency rules for heterogeneous repositories, *Proceedings of the 2003 ACM symposium on Document engineering*, Grenoble, France, 2003, pp. 140-149.

[18] D. Hutter and S. Autexier, Formal Software Development in MAYA, In: D. Hutter et al. (eds.) *Mechanizing Mathematical Reasoning*, Vol. 2605, Springer Berlin Heidelberg, 2005, pp. 407-432.

[19] A. Bundy, Automated theorem provers: a practical tool for the working mathematician?, *Annals of Mathematics and Artificial Intelligence,* Vol. 61, 2011, pp. 3-14.

[20] A. Egyed, Automatically Detecting and Tracking Inconsistencies in Software Design Models, *Software Engineering, IEEE Transactions on,* Vol. 37, 2011, pp. 188-204.

[21] Y. Xiong, Z. Hu, H. Zhao, H. Song, M. Takeichi and H. Mei, Supporting automatic model inconsistency fixing, in *ESEC-FSE'09 - Proc. of the Joint 12th European Software Engineering Conference and 17th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 2009, pp. 315-324.

[22] G. Spanoudakis, A. Zisman, E. Pérez-Miñana, and P. Krause, Rule-based generation of requirements traceability relations, *Journal of Systems and Sftware,* Vol. 72, 2004, No. 7, pp. 105-112.

[23] T. Olsson and J. Grundy, Supporting traceability and inconsistency management between software artifacts, *Proceedings of the 6th IASTED International Conference on Software Engineering and Applications, SEA 2002*, 2012, pp. 484-489.

[24] J. Malmqvist, A classification of matrix-based methods for product modeling, in *DS 30: Proceedings of DESIGN 2002, the 7th International Design Conference,* Dubrovnik, 2002.

[25] M. Bastian, S. Heymann, and M. Jacomy, Gephi: an open source software for exploring and manipulating networks, *ICWSM,* Vol. 8, 2009, pp. 361-362.