Interoperability in the Networked Design infrastructure

Jeroen COENDERS[1]

[1] BEMNext Lab, Department of Civil Engineering and Geosciences, Delft University of Technology, Delft, and Arup, Amsterdam, The Netherlands, *jeroen.coenders@arup.com*

Summary

Interoperability, the ability of different software applications to communicate with each other, is one of the biggest challenges for efficient and effective use of advanced software technology in structural design and engineering. In practice, the problem of interoperability exists very much for the end-user since agreement on common standards does not exist despite the many efforts over the years. This paper discusses an approach to interoperability based on the assumption that one final common standard will probably never exist and that a more practical federated approach needs to be taken. In this paper the author demonstrates the concepts which a new conceptual infrastructure, called NetworkedDesign, provides to address the challenges of interoperability. The infrastructure contains a rich set of concepts for the user to express information through lightweight open objects and logical constructs which can be used as a basis for an ad hoc standard between applications.

Keywords: *Interoperability; BIM; Building Information Modelling; Computational Design; NetworkedDesign; Structural Design; Engineering.*

1. Introduction

Interoperability, the ability of different software applications to communicate with each other, is one of the biggest challenges for efficient and effective use of advanced software technology in structural design and engineering, as well as building and infrastructure design in general. In building and infrastructural design many different stakeholders have to collaborate to conceive the design and the concepts behind it, but also to document the justification behind the design concepts and for the construction process important end-state of the design, the object to be build including all its different components, materials, quantities, etc.

In current design practice designers often use software for this documentation process and because there are many different stakeholders involved, interoperability between the different software applications becomes crucial for a successful design process. In current practice actually it often is one of the most laborious and time-consuming processes. For the future where software will become more powerful replacing various manual tasks of the engineer and more processes have to become more automated, the problem of interoperability the designers will face is a serious barrier for further development. Designers with programming or scripting abilities often have somewhat less problems than designer without these specialised skills. Although a solution might be that designers should be educated in programming and scripting, this is likely not the solution. Software development has become a specialist field on itself with evermore complicated concepts, structures, mechanisms, paradigms, etc. so that it will become very hard for a designer to maintain its knowledge and experience in both fields of design and software development high. Next to this many designers are not willing to invest time in the field of software development. Therefore, another approach is desirable.
Many different software-based strategies have been developed. However, effective strategies where multiple software applications are combined used, where data can be bi-directionally synchronised or better where information only exists once, efficiency suffers from interoperability issues.

One might question why to use so many different software applications in the first place. Is it not smarter to just adopt one application in the industry?
Unfortunately, this is not possible. Software applications are usually adopted in practice for their specialised capabilities, such as numerical structural analysis (such as finite element analysis software), three-dimensional geometrical modelling capabilities, information modelling capabilities (such as building information modelling – BIM [1]), etc. To integrate all these application in a single application would be an extremely complex if not impossible task if one would want to maintain the qualities of the individual specialist applications.
It is also very unlikely that a single application which has to spread its development effort over many different areas will be of similar high quality and have detailed specialist capabilities as a specialist application which has been custom developed for a single purpose and can focus all development effort on a single area and goal.

Another approach would be to develop a single software suite for the industry consisting of a number of collaborating software applications that tightly integrate through common definitions. This provides the opportunity that a single software developer controls the interoperating structure. Unfortunately, it seems that this approach is not working in practice. First of all, it can be observed that large software vendors attempt this approach, but struggle on the common definition. In practice it is very hard to tightly integrate definitions between very different applications, even for the people who control them. Second, the approach still suffers from the effect that when development effort needs to be spread, the quality of specialist software suffers, at least in the eyes of the specialists that need to use this software. Third, the approach has the danger it will create silos where within the silo everything works well, but interoperability with other silos is very poor. In practice it will be very difficult for the building industry to converge to a single silo as has occurred in other industries, because of the diverse nature in size and scope of the different stakeholders.

Another approach would be to force the industry to use a single application, but that approach will likely fail for the same reasons mentioned above. First of all, there is no single application or suite that is able to cope with all different stakeholders and their analysis and documentation processes (although some software vendors like to claim otherwise). Second, large barriers exist to develop such an application.

The last approach that will be mentioned is to develop standards which all application can use to interoperate between applications. This has been tried a number of times and until now is only successful for subsets of data. Actually, interoperability is not purely a scientific challenge: If software suppliers could agree on a common standard for data exchange there would be no issue. However, in practice the problem of interoperability exists very much for the end-user since agreement on this common standard does not exist despite the many efforts over the years. It can also be questioned if not more issues exist which lead to the fact that this standard is difficult to create, such as the fact that efficient and specialised software will contain a single purpose data structure and will suffer from inefficiency in case of a common data structure, or for instance the different interpretations the various stakeholders and disciplines will have for the same data object.

This paper discusses an alternative approach to interoperability based on the assumption that one common standard will probably never exist and that a more practical federated approach needs to be taken. This federated approach can be based on a conceptual infrastructure which provides concepts and functionality for modelling design and engineering data, information and logic.

2. NetworkedDesign

NetworkedDesign [2] is a conceptual infrastructure which aims to support the next generation of software tools, applications, frameworks and systems. This novel infrastructure has been proposed based on *computational design theory* [2], which is a theory the author is collaboratively researching and developing at Delft University of Technology (BEMNext Lab) and Arup to link key characteristics, values and concepts in structural design and engineering to technological concepts. The assumption for this theory is that adoption of software will be more likely when it is based on what the user needs and values rather than purely the next technological development as well as to understand what the barriers of adoption are for technology so that these can be addressed. In this paper the theory will not be further discussed, but the focus will be on demonstrating the concepts the infrastructure provides to address the challenges of interoperability. The infrastructure contains a rich set of concepts for the user to express information through lightweight open objects and logical constructs which can be used as a basis for an ad hoc standard between applications. The paper will also present a novel extension of the infrastructure which makes use of so-called Interface Rules to provide user-friendly interoperability for designers and engineers without programming knowledge which as states above is very important for adopting and success of the approach.
NetworkedDesign is related to parametric and associative design systems [3] but it needs to be noted that is not such a system. First of all, NetworkedDesign is not a system or software application, but an infrastructure on which systems, software applications, plug-ins, add-ins, etc. can be build and of which they can implement a number of concepts. Second, NetworkedDesign has no preference to the solver which is used to solve the logic as it has been expressed in the model. NetworkedDesign contains a parametric solver as one of its solvers, but can use other, more complex solvers to solve more complex logic arrangements.

2.1 Plugability

Plugability, the ability to extend NetworkedDesign by custom development plug-ins by the end-user is one of the most important base mechanisms in NetworkDesign's interoperability strategy. Because users are able to plug-in their own developed or modelled programming logic, highly flexible and customisable arrangements can be build which can adapt to almost any situation.

2.2 Custom data-structures

NetworkedDesign allows the user to define and build her own data-structure based on a number of entities in the infrastructure, which will be further discussed below: open (and extensible) objects, custom methods and custom assemblies. Again, because the user can build her own data structure which (partly) matches the data structure to interoperate with, the user is able to interoperate with almost any data structure a software application provides.

2.2.1 Open objects

Part of this data structure are the open objects the infrastructure provides. This means that objects are undefined, except for name and type, and act as a container of properties. The objects grow as more definition is added by the user by adding methods to the object. The properties are in their turn added and operated on by these methods. Any object that follows these rules can be constructed by making use of these mechanisms.

2.2.2 Custom methods

As mentioned methods expand the definition of objects and bring associativity (both single- and bi-directional) and behaviour to the infrastructure. Multiple method types exist in the infrastructure: Definition methods, Transformation methods, Calculation methods, Update methods and Interface

methods. Each of these types have a different combination of interaction with the user (and other systems) and behaviour they can provide. An important concept for interoperability is the concept of transformation that has been introduced in the NetworkedDesign infrastructure. This allows for bi-directional transformation of object definitions and provides a unique ability to interoperate with one system with one data structure while at the same time through bi-directional transformation the system is also able to interoperate with other data structures or the user with another data structure. Also the concept of methods adds to the flexibility to adapt objects to any other data structure for interoperability.

2.2.3 Custom assemblies

Assemblies consisting of objects, methods and their different relationships to build more complex arrangements of objects are used to provide pieces of logic that naturally belong to each other improving the understandability of the user. Additionally, they can be used for performance optimisation in the infrastructure. Because these assemblies can be build by the user, the user is able to use these entities to build more complex data structures potentially matching the data structures in the system to interoperate with.

2.3 Custom solvers

As stated NetworkedDesign has no preference for a single solver to solve the logic modelled to provide support for complex behaviour involving global mediation between objects, properties, etc. This allows for solving complex arrangements of logic and complex interoperability arrangements.

2.4 Rules

The concept of rules and rule-processing has been earlier introduced by the author for integration with parametric design [7,2]. Rules add the ability to select logic from the graph, to sort and to filter logic. This makes it very easy to define interoperating logic without dependency on the exact definition of the object of interoperation, which allows the user to build independent interoperability set-ups without making them dependent on the project or building.

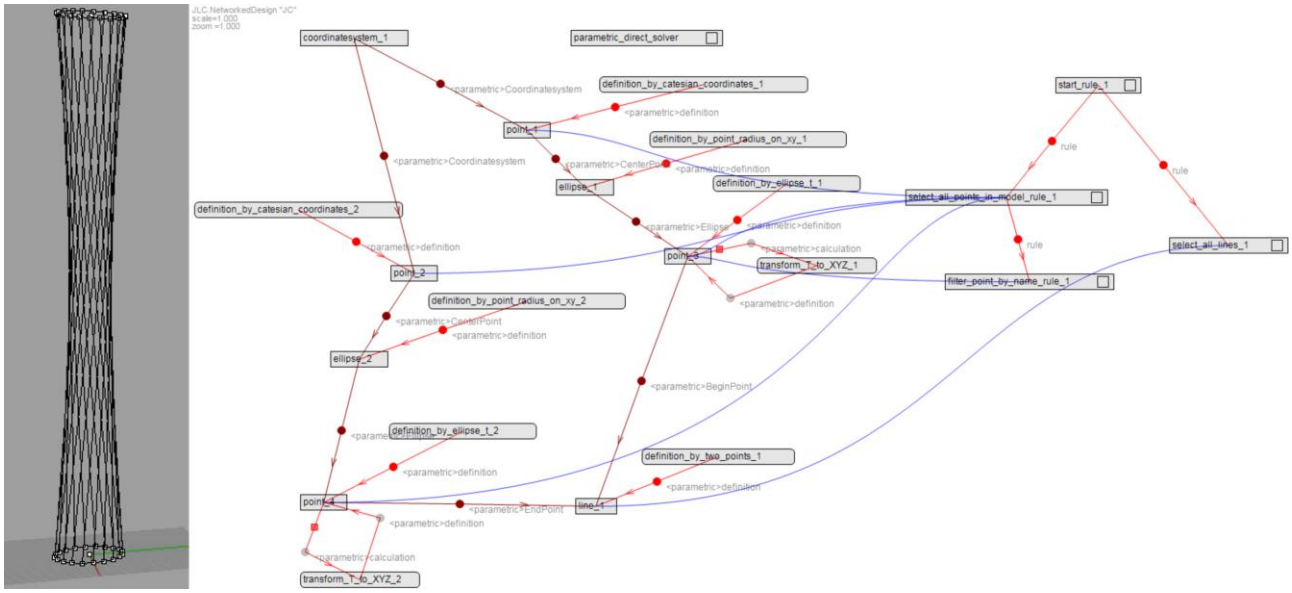

Figure 1: Example project in NetworkedDesign that demonstrates rule-processing.

Figure 1 shows an example of rule-processing in action for a tower project (shown on the left hand side). The right hand side shows the logic behind the model. Two separate portions of logic can be identified on closer inspection connected by the blue curved rule relationships. The left hand side of

the logic model is the actual tower model while the logic on the right hand side selects objects from this logic based on the principle of rule-processing. The blue rule relationships are relationships which depict the results of the rules as processed by the system.

2.5 Templates

Another powerful concept in the infrastructure are Templates. These entities allow users to create a template of objects, methods, assemblies, rules, etc. to package the logic in a portal format for easy and quick deployment during later use. Bi-directionality and multi-directionality [2] provide the ability to build highly flexible and reusable templates to be used in the interoperability setups.

2.6 Communication with applications

The current state of software makes communication between applications a lot easier than 5 to 10 years back when many applications were silos that stored their information in propriety, custom data formats often inaccessible for other applications. In the current state many software applications provide so-called APIs, which are Application Programming Interfaces [4], or use open data formats which are based on standardised structures, such as XML. This allows for software developers to extend these systems by making use of high-level languages and frameworks, such as .NET [5] and Java [6]. Many of these languages use a concept, called Reflection, which allows the programming language to query the data structures of another application in order to use these for interoperability. The interoperability concepts below make use of this concept.

2.7 Interface concepts

Recently, the NetworkedDesign infrastructure has been extended with novel concepts to further support interoperability which will be briefly introduced below.

2.7.1 Interoperability reflection interface

The interoperability reflection interface allows the user to load APIs of other systems and select desired properties and methods for use in the mapping and operating rules which will be introduced below. Under the hood of the infrastructure this interface can also provide low-level communication infrastructure in its implementation to be used by the different interface rules for communication with other system types. An example of the graphical variant can be seen in Figure 3.

2.7.2 Interface rules

Interface rules are a subtype of the Rule entity created specifically for interoperability purposes. These rules are aware of interoperability constructs provides by the reflection interface.

2.7.3 Interface mapping rules

Interface mapping rules are used for creating mappings between objects, often in the form of equivalency, e.g. “The property X on the Node object in system A is the same as property X1 on the Point object in NetworkedDesign” and allow the interface to communicate information in a single- or bi-directional manner.

2.7.4 Interface operator rules

Interface operator rules are used to perform certain operations in the interoperability setup. Examples include conditions (“If the type of object A matches ‘Point’, continue”), loops (“Foreach object in the collection do”), filters (“Create a subset of all objects in the collection of the type ‘Point’”) or sort (“Sort the objects in the collection in alphabetical order”).

2.8 Overview

Below a short overarching example will be given of what can be achieved by the strategy proposed above.

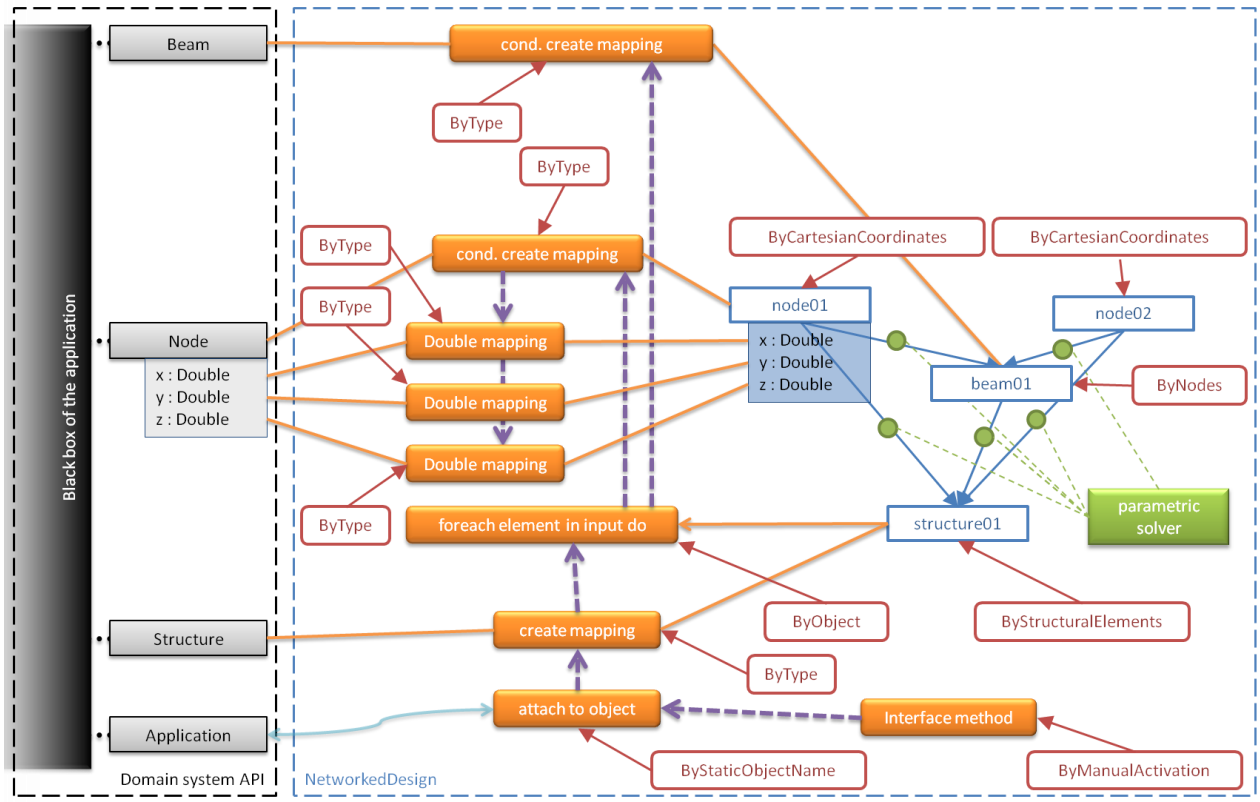

Figure 2: Example of a modelled interoperability set-up by the end-user.

Figure 2 shows an example of an API of a fictitious structural application (left hand side) where the black box logic of the application itself is exposed through the API. This API contains a static Application class which allows for other systems to hook onto the running application or starts the application in case it has not been started yet. Many systems in reality use a similar pattern. The API also exposes a Structure class which functions as a container for Node classes and Beam classes. On the Node class three Cartesian coordinate properties have been exposed in this example to explain the concept of property mapping.

On the right hand side of the figure the NetworkedDesign infrastructure can be seen. On this side using the parametric capabilities of the infrastructure the user has built her data-structure by instantiating two Node objects, a Beam object and a Structure object. The Node objects have been defined by Cartesian coordinates, the Beam object by the two Node objects and the Structure object contains the three different StructuralElement objects (Note that each of Node and Beam classes has been derived from a StructuralElement class). The parametric solver passes the data from one object to the other.

To create the interoperability set-up the user would start by adding the main interface rule, in this case called “interface method” and in this case defined by manual activation, which means that the system will not automatically update this rule, but that the user will fire it by hand. The next step is for the user to select a number of APIs that she wants to use and map too. Figure 3 shows an example of a selection interface where on the left hand side .NET assemblies can be added which contain the API to Tekla and on the right hand side the system reports all classes in the APIs. The

system can use reflection or a digital API description, such as a XML Schema, to identify the API data structure and expose this to the user.

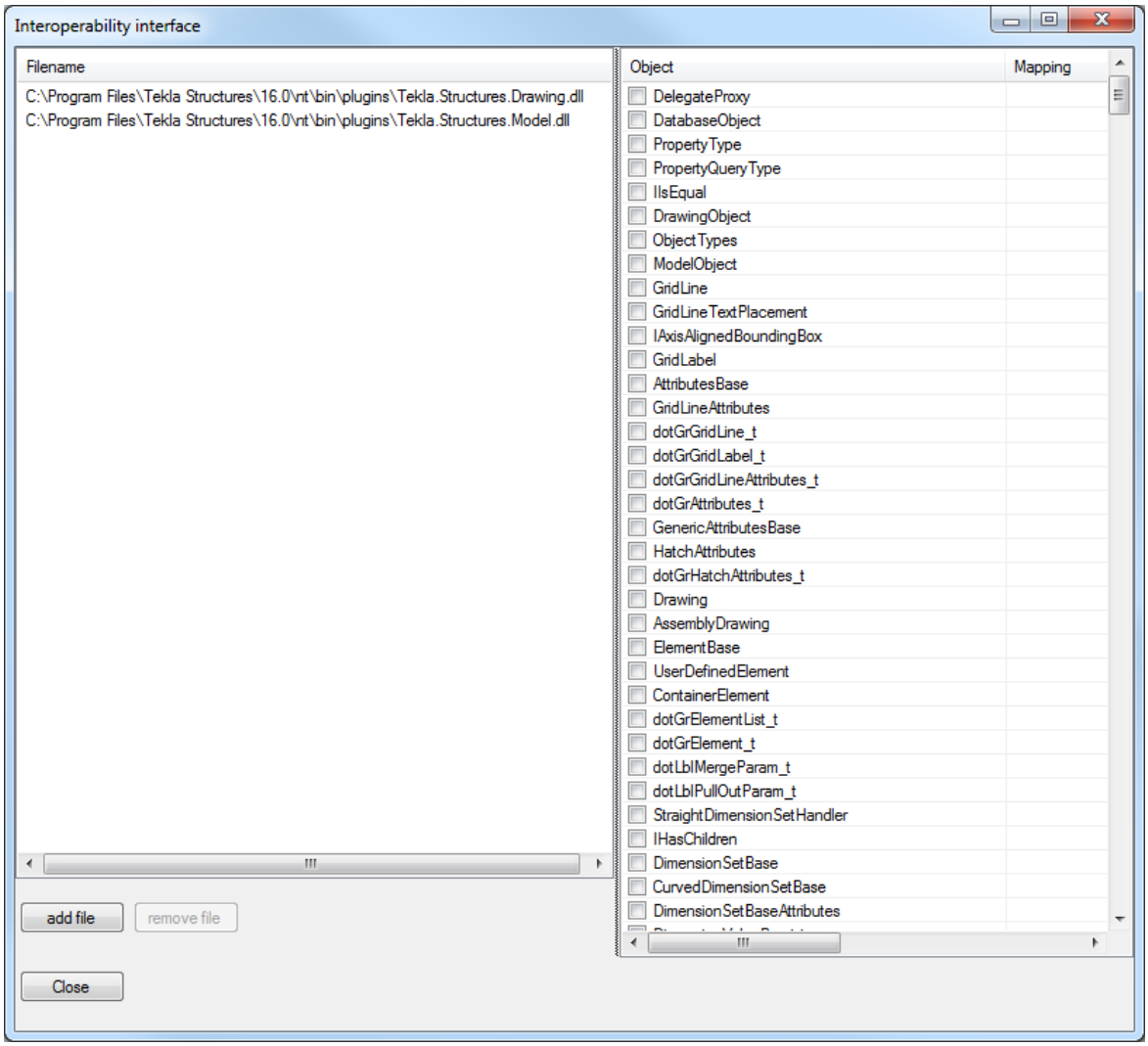

Figure 3: Example of the selection of API's, in this case the Tekla .NET API's.

Next step is for the user to start defining the process of interoperability. In this case first a hook needs to be created so that the system has an entry point into the application. This API uses a static Application class as hook. The user can catch this hook by instantiating an interface rule that is able to call this static class by making use of its name (in this case "Application"). This rule is called "attached to object" in this example. Please note that in Figure 2 the process flow is represented by the purple dotted arrows and that with the light-blue curved arrow the rule relationship is depicted which is created by the system. Now the hook exists, the user can further define the interoperability setup by selecting a mapping rule that creates a mapping between an instantiated object in the NetworkedDesign infrastructure, in this case "structure01" and a class type in the API which is define in the selection interface of Figure 3 by selecting one of the class (types). This mapping will instantiate the type in the API by using the Application instance.
Now that the Structure has been mapped, each of the structural elements will need to be instantiated. In this paper only the Node mapping will be discussed for brevity. To read each of the elements in the container object of "structure01" a "for each element in input do" interface operator rule will be used which reads a specified property of the input type defined by the property name and for each object in this property (assuming that the user has selected the property that exposes the container)

calls its following rules. By making use of a “conditional create mapping” interface mapping rule the nodes are selected based on the type name they will need to map. Similar to the mapping of the structure, the mapping instantiates a Node class in the API based on the attached Node object in the NetworkedDesign interface. Next mapping of the properties will need to be defined. In this case, a simpler interface mapping rule can be used to map the value of each of the properties on the NetworkedDesign object to its API equivalent. The user is now able to fire the rule as interface all objects to the other application.

3. Discussion

Please note that this application case has not covered all concepts as discussed above and much more complex set-ups could be build which provide even more flexibility, reusability and ease of use. For example, employing the rule-processing capabilities of the infrastructure further would make the mappings independent from the exact (number of objects), while packaging the rules in a Template can make deployment on a next project very easy.
Please note that this case only described a single-directional model, but that bi-directional communication can work in a similar manner.
The presented approach has a number of advantages: it provides a powerful and flexible mechanism for users without a lot of knowledge about programming to build their own interface setups to move data from one application to another.
However, the approach also has a number of limitations: 1) The approach will still be limited by API's provided by applications for interoperability. If these are not available, the approach will fail. 2) The approach is still limited by platforms, such as operating systems for client-sided software. 3) Still some knowledge is required on the workings of the API as especially the process cannot be derived by reflection. 4) The current implementation of NetworkedDesign is still very limited.

4. Conclusions

This paper has presented the problems that exist around the interoperability challenge that our industry faces currently and in the future where software use will be become more dominant in the design, engineering, construction processes as well as in the whole life-cycle of the building.
Furthermore, this paper has introduced the conceptual NetworkedDesign infrastructure and some of its concepts around the theme of interoperability.
Finally, this paper has presented a number of novel concepts recently added to this infrastructure that allows user to build their own interoperability standards and set-ups in an easy manner.
For the future this will mean that interoperability might become less of a problem which will save time and cost for the industry and will lead to higher quality products. The barriers for further adopting more and more complex software strategy in the future have been further lowered.

References

[1] Eastman C.M., 1999, Building Product Models: Computer Environments Supporting Design and Construction. CRC Press, 1999.

[2] Coenders, J.L., 2011, *NetworkedDesign, next generation infrastructure for computational design*. PhD dissertation, Delft University of Technology, VSSD, Delft, The Netherlands.

[3] Aish, R., 2005 *Introduction to GenerativeComponents, A parametric and associative design system for architecture, building engineering and digital fabrication,* White paper on http://www.bentley .com.

[4] Perry, S., 2006, *Core C# and .NET*. Pearson Education Inc.

[5] Microsoft, 2012, *.NET technology*, http://www.microsoft.com/net

[6] Oracle, 2012, *Java technology,* http://java.com/en/

[7] Coenders, J., 2008, “Rule-processing as a cross-cutting concern in parametric associative design systems”. In J.O. Salinas, editor, *IASS-SLTE 2008 simposium Acapulco Mexico, New materials and technologies, new design and innovations a sustainable approach to architectural and structural design*. IASS-SLTE, Acapulco, Mexico, pp87–88.