INTERNATIONAL DESIGN CONFERENCE - DESIGN 2012
Dubrovnik - Croatia, May 21 - 24, 2012.

DESIGN 2012

A DESIGN ASSISTANT ARCHITECTURE BASED ON DESIGN TABLEAUX

L. Hendriks and A. O. Kazakci

Keywords: formal framework for design, design tableau, design assistant

1. Introduction

One of the major topics in design research is the use of computational means to support knowledge based concept synthesis and creativity. While an abundance of systems has been presented in the past (based on analogy engines, case-based reasoning, genetic algorithms etc), starting with the second half of the 90s', we can observe a progressive shift towards the exploration of agent-based systems.
For example, [Grecu and Brown 1996] proposed a system for the parametric design of springs. They used agents called single function agents (SFAs) that have each a unique function such as the selection of parameters, estimating their values, evaluating alternatives, criticizing or recommending. The limited functionality of these agents reinforces their interaction. Later work has introduced learning into the SFAs framework.
[Campbell et al. 1999] present a bottom-up approach named A-design. Various types of agents exist in the system. Configuration agents are responsible for introducing the parts they represent when and where they can into the design that is being formulated. Instantiation agents fill out these configurations with components from a catalogue. Fragmentation agents can delete inappropriate configurations or add to the catalogue configurations created in the process if they are evaluated to be satisfactory. Managing agents are in charge of the agent population; they can create or destroy agents based on their level of contribution.
[Gero and Reffat 2001] used multiple representations for a single agent. The agent can perceive in various ways a design, which allows changing the trajectory of design in a situated process. [Reffat 2002] built on this system to produce a multi-agent version capable of creating concepts.
[Saunders 2001] presents curious agents that can detect novelty and originality based on Wund curves. He uses this framework for the design of art expositions to simulate the ways to maintain the public's interest during the visit.
This shift towards a Multi-Agent System paradigm in Design Support Systems research follows the general tendency of computer science striving for more and more massively distributed, autonomous systems. During the construction of design tools, to be able to cope with the fast-evolving computer science and technology, while also taking into account the particularities of design activities, one possible strategy is to build tools that make use of design theories and models. Such a strategy has the merit of having dedicated and focused solutions and also it would offer the possibility to discuss and benchmark various tools on a design theoretical level.
In the present work, we provide the basis for a tool that is both compatible with the most recent trends in agent technology and which is inspired from a particular design theory, namely, the C-K design theory [Hatchuel and Weil 2003]. We present a method, Design Tableau, which can be used as an automated reasoning engine for a local agent providing design assistance in the construction and verification of a design description. The system is based on Description Logics and it has the

advantage of being compatible with flexible and abstract ontologies such as the OWL DL, an ontology developed for the Semantic Web. This compatibility gives the agent the possibility to operate within a Multi-Agent System paradigm. The agent's reasoning process can thus interact with other agents, managing each their own local knowledge base, in order to query for missing knowledge or support other agents' reasoning.
The core method, Design Tableau, is a generalization of Semantic Tableau (ST) method. ST is a general method for logical theorem proving that can be used with a wide range of frequently used logics. Our method has the advantage of generalizing STs to be able to interact with other agents during reasoning operations (i.e. consistency checking, theorem-proving etc). Design tableaux are inspired by C-K design theory that describes design as a dual expansion process. In terms of computational design support, this implies the possibility to modify the design description while the knowledge base is being updated to provide the relevant knowledge for the current design concept. The method that mimics this process has already been presented in previous work [Hendriks and Kazakci 2010, 2011] based on the formalism of first-order logic. Here, the major change is the adaptation of the method to the description logic framework. Description Logics is a class of formalisms that provides significant advantages over first-order logics from an implementation point of view. While allowing logical decidability, they allow to work on ontologies, such as OWL, that allow the flexibility of conceptual graphs and semantic networks. A major consequence of this new version is the possibility to use the logical engine within an agent framework, thus, extending the design support by the possibility of interaction with distant databases within a multi agent system paradigm.
In the next section, we lay out some background. First, we give a brief overview of C-K theory that inspired the Design Tableaux. Then, a short discussion of description logics and OWL language is provided. In section 3, we begin with the definitions for our formal framework; design stages and design moves. Then, design tableaux are introduced and discussed with the help of examples. In section 4, a possible software architecture using design tableaux is described. A last section concludes with a general discussion.

2. Background and related work

2.1 C-K design theory and the process of dual expansion

C-K design theory [Hatchuel and Weil 2003] is a formal theory describing reasoning in design. According to the theory, design reasoning is based on two types of entities; concepts and knowledge. Concept are logical propositions (any type of logic is acceptable; [Hatchuel and Weil 2003] that cannot not be given a truth value with the currently available knowledge. They are said to be "without logical status". Consider, for example, C_0: '*tires made of dust*'. Without further specification or knowledge the status of this concept is not clear. On the other hand, Knowledge is a set of propositions that can be given an interpretation (a logical value), e.g., "K_1: tires are made of rubber". Concepts are especially interesting when they introduce a new notion, i.e. when the existence of an object falling under the concept is 'undecidable' with the available knowledge. Said in other terms, the available knowledge $K_1,\ldots,K_n$ would not be sufficient to *logically infer* the proposition 'there exists some C'.
C-K theory describes the reasoning process as the mutual interaction of design concepts and knowledge. When concepts are refined by successive partitioning (adding new properties), the concept space will activate available knowledge or trigger the expansion of knowledge (i.e. learning). For instance, one might ask "what kind of dust can provide rubber-like properties?" to expand the K space. Conversely, if the knowledge expansion is successful, it might provide further properties that can be used to still further refine the concepts. For example, if the learning attempt provides information about '*rubber powder and polypropylene*', the concept may be refined with C_1: $C_0 \wedge$ *'made of rubber powder*'. This process is named dual expansion process in C-K theory.
If we accept the knowledge space to be a formal description of some knowledge, then, in computational terms, we can assimilate it to a knowledge base. Likewise, concepts can be seen as formal descriptions of designs.

2.2 Description logics and formal reasoning with knowledge bases

The system we present uses description logic framework in order to be compatible with OWL and Semantic Web type knowledge representation. OWL is a language based on the RDF and XML formats extending the existing web standards. It is aimed at providing a universality of the syntax and flexibility. It is a language allowing building ontologies. This is a useful, or even required, property for a design support system in knowledge-based concept synthesis task. Most of the systems proposed in the literature use design specific ontologies such as the FBS schema.

Aside the significant expressive power of OWL, stemming from its relationship with conceptual graph and semantic network formalisms, it is also designed specifically for facilitating reasoning tasks for software agents having access to open-ended systems and databases. It allows describing a knowledge domain in terms of concepts (classes), roles and individuals.

A particular variant of OWL that makes it even more interesting is OWL DL that affords important inferential mechanisms based on Description Logics. DL corresponds to a decidable fragment of first-order logic. It provides thus sound and complete decision procedures for logical theorem-provers. An extensive amount of work on description logics has provided cutting-edge, highly optimized system implementations such as Pellet or Racer. DL benefits thus from a well defined semantics and well understood formal properties in terms of complexity and decidability.

A major use for reasoners is to make use of the explicitly coded information contained in the database to infer implicit information. As we shall see, our system targets rather to detect knowledge that is *not* contained in the knowledge base while attempting a reasoning task. The system is designed thus to discover the missing knowledge. In the currently available solvers, in such a situation, the system simply stops. In design, if the concept contains novelty, new knowledge is required to continue the reasoning. Thus, a system allowing pinpointing to the missing knowledge is required.

3. Design tableaux

3.1 Design stages and design moves

The language of description logic is built from primitive concepts (e.g. *properties*) and primitive relationships (called *roles*). These primitive terms make up the *vocabulary* of our language. We can distinguish *sentences* like Cat $\subseteq$ Animal, expressing cats are animals, and *concept descriptions*, concepts for short, like $\exists$owns.Cat, describing the concept of being the owner of some cat. We will write $x : C$ to express x is falling under the concept C (e.g. *Garfield : Cat*).

A *design stage* is given by a pair $s = \langle K; C \rangle$ such that K is a set of sentences capturing our knowledge at stage s and C is a concept containing the description of the artefact we are trying to design. The goal of the conceptual design process is to extend K and refine C such that based on the knowledge we can in fact prove the possibility of the existence of some obhect meeting the design description in C.

Definition 1

$s = \langle K, C \rangle$ is called

- Consistent $\Leftrightarrow K \nvdash \neg\exists x(x : C)$ (and inconsistent o.w.)
- Closed $\Leftrightarrow K \vdash \exists x(x : C)$
- Feasible $\Leftrightarrow$ s is consistent and closed
- Open $\Leftrightarrow$ s is consistent and not closed

The design process consists of moving from one design stage to another, trying to bridge the gap between our knowledge K and our design concept C, so that C can be proven by K. There are two main scenarios corresponding to K-expansion and C-expansion.

Scenario 1 [Select]: $\langle K; C \rangle \Rightarrow \langle K; C' \rangle$

Using the concepts from the vocabulary of K and C, build a new concept C'. For example if the concept C is a device to *protect an egg from breaking after a fall* and the knowledge K contains the concept of a parachute, the new C' could be a device to *protect an egg with a parachute from breaking after a fall*.

Scenario 2 [Query]: $\langle K; C\rangle \Rightarrow \langle K'; C\rangle$
Using the concepts in C and the knowledge in K, search for new knowledge K'. For example the concept of *protect an egg from breaking after a fall* includes a concept *protect form breaking*, which may guide us to new knowledge on protecting against breaking like using a parachute or a container filled with marshmallows.
Both scenarios assume an interaction with some environment that is interactively selecting concepts for refinement of the design goal and providing answers to the queries (the environment contains a user as well as other systems). The two scenarios can be combined to one design step $\langle K; C\rangle \Rightarrow \langle K'; C'\rangle$. An interesting constraint that can be imposed on such a design step, if the original *C* fulfills the design requirements, is that a solution for the resulting design stage is also a solution for the original concept. In description logic, where one can reason not only about sentences, but also about the relationships between concepts, one can state this formally as *C* is *derivable* from *K'* and *C'* (see below for more details).
Definition 2
A design step $\langle K; C\rangle \Rightarrow \langle K'; C'\rangle$ is called *sound* if and only if $K \subseteq K'$, $\langle K'; C'\rangle$ is consistent and

$K', C' \vdash C$.

In the design process not all design steps will necessarily be sound. Unsound design steps in a way indicate a significant change in the design requirements. In practice, this would require the designer to check in each unsound design step whether the client or the commissioner of the assignment agrees with the changes in the design brief.
Design steps are aiming at bridging the existing gaps that exists in the proof of the existence of an artifact answering the design concept description, but both scenarios 1 and scenario 2 are *extra-logical* steps. Scenario 2, extending the knowledge, can be seen as a kind of *abduction* where instead of seeking conclusions from given premises, one looks for extra assumptions that might prove a given conclusion. Scenario 1 is an even more creative step, directed by preferences of the designer.

3.2 Design tableaux

The rules of a Design Tableau combine both scenarios with the semantic tableau technique originally invented by E.W. Beth in the 1950's. In a semantic tableau one systematically seeks to refute a conclusion from a given set of premises. If this attempt fails, the tableau is said to *close*, the conclusion must follow from the premises and in fact a formal proof can be extracted from the tableau.
The most important difference between a Design Tableau and a semantic tableau is that in a Design Tableau one is allowed to add new premises: new knowledge. Additional knowledge is stacked on top of the tableau. Let us use a simple example to illustrate the basic principles.
In our example the first concept is that of a *Wagon*. Our next step is to add knowledge about wagons: trains have wagons ($Train \subseteq \exists has.Wagon$) and trains run on tracks ($Train \subseteq \forall runs\_on.Track$). Let us assume that new concept running on a track ($\forall runs\_on.Track$) is interesting for some reason. We want to add this to our initial wagon concept but now as a wagon not running on a track. Our next step is to check whether this design step from our initial stage to the new design stage is sound. So we try to

prove: $Train \subseteq \exists has.Wagon, Train \subseteq \forall runs\_on.Track, Wagon \wedge \neg \forall runs\_on.Track \vdash Wagon$.

Although in this case the proof is trivial, the development of the Design Tableau might be instructive.
In step 1 we introduced a (new) name, *d*, for the wagon we want to design. In step 2 we introduced new knowledge on the top of the tableau. Step 3 introduces the new design concept in the right branch (closing the right side of the tableau with a horizontal bar, to avoid confusion with the earlier concept) and a left branch to check soundness of the move towards the new concept. The conjunction of *wagon* and *running on a track* is split in step 4. After this step we see *d : Wagon* appear on both the left and the right side of the branch, which makes the branch close. This shows it is safe to proceed with the right branch and the new concept.
A basic idea of the tableau is that for each branch the formulas on the left hand side are assumed to be true and those on the right hand side false. A formula appearing both on the left and the right indicates

a conflict. In logical terms, this shows that it is impossible to construct a model making all formulas on the left true and those on the right false.

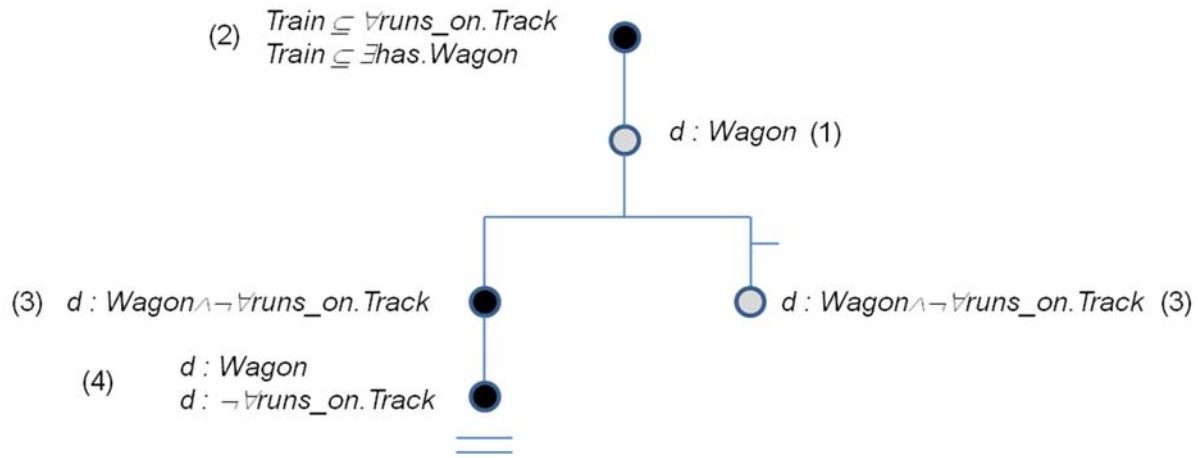

Figure 1. A Design Tableau example

Our first example introduced both extra-logical steps corresponding with scenario 1 and 2. The other rules of the Design Tableau are in fact purely logical and are meant to construct, if possible, models against the hypothesis that at least one of the formulas on the right has to be true if all formulas on the left are true. Such a counter-model is reached if for a certain branch in the tableau it does not make any sense to apply one of the logical rules again. As the logical rules break down each proposition in the tableau into smaller and smaller parts, a none closing (*open*) branch has atomic statements on its left and right, those of the form $d_i : C$ or $R(d_i, d_j)$, where C a primitive concept and R a primitive relationship and d_i and d_j names of objects (e.g. the new name d we introduced for the object for which the first concept applied). The logic rules of the Design Tableau guarantee that not only the model described by all the atoms on the left of an open branch does not fulfill any of the atomic statements on its right, but the same will be true for all statements on the left becoming true and those on the right becoming false. Figure 2 shows some (simplified) resulting models constructed from open branches of a Design Tableau.

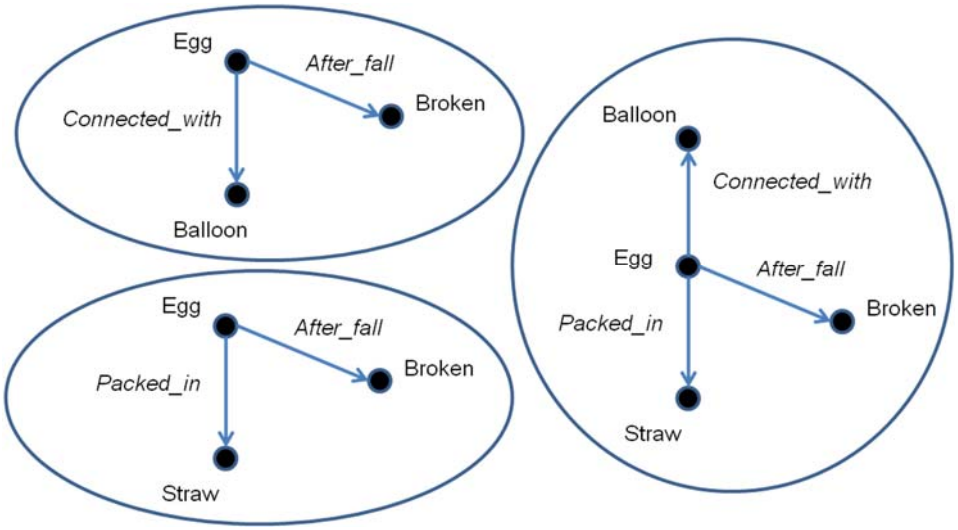

Figure 2. Counter-models resulting from an open Design Tableau

With the logical rules of the Design Tableau one is systematically searching for a counterexample against a design stage <*K; C*>. Starting with a situation *(K; • d:C)* formulas are decomposed and their constituents placed on the left or the right side of a branch, result ting in situations of the form *(K; L •*

Δ). Both the formulas in K and in L are on the left and the formulas Δ are on the right side of a branch. If we do not want to distinguish between formulas in K and L we will write a situation as $(\Gamma \bullet \Delta)$.
In figure 3 one can find the basic logical rules for the tableau. Here A and B are formulas and C and D are concepts, R is a role (a relationship between two individual objects). If we want to highlight that A is a formula in Γ and B a formula in Δ, we write $(\Gamma, A \bullet B, \Delta)$ or $(A, \Gamma \bullet \Delta, B)$.
Note that a conjunction of two formulas or two concepts on the right hand side of a branch causes a split in the tableau with two possibilities to construct a counterexample. In the rules for $\exists R.C$ in Γ (so appearing on the left side of the branch) and $C \sqsubseteq D$ in Δ a new object name y has to be introduced.
Once the basic rules have been introduced one can define other connectives and concept operations in the usual way. Let E and F be either both formulas or both concepts, than $E \vee F \equiv \neg(\neg E \wedge \neg F)$, $E \rightarrow F \equiv \neg(E \wedge \neg F)$, $E \leftrightarrow F \equiv (E \rightarrow F) \wedge (F \rightarrow E)$. Likewise one can define $\forall R.C \equiv \neg\exists R.\neg C$ and $C \equiv D$ as $(C \sqsubseteq D) \wedge (D \sqsubseteq C)$. With these definitions one easily also introduces the corresponding tableau rules.
In a semantic tableau, one is ready when the tableau is either closed (we have found a proof) or at least one branch stays open (we have found a counter model). In a Design Tableau we will not expect an immediate closure of the tableau without applying either scenario 1 or 2 (since concepts cannot be proven, nor disproven with the currently available knowledge). The idea of the Design Tableau is to construct a set of situations against the current design stage by applying the logical rules and let this set guide us in the application of either scenario 1 or scenario 2.

$(\Gamma, A \bullet A, \Delta)$	$\Rightarrow$	Closure
$(\Gamma, A \wedge B \bullet \Delta)$	$\Rightarrow$	$(A, B, \Gamma \bullet \Delta)$
$(\Gamma, \neg A \bullet \Delta)$	$\Rightarrow$	$(\Gamma \bullet \Delta, A)$
$(\Gamma, C \sqsubseteq D \bullet \Delta)$	$\Rightarrow$	$(\Gamma \bullet \Delta, x : C)$ **and** $(x : D, \Gamma \bullet \Delta)$
$(\Gamma, x : C \wedge D \bullet \Delta)$	$\Rightarrow$	$(x : C, x : D, \Gamma \bullet \Delta)$
$(\Gamma, x : \neg C \bullet \Delta)$	$\Rightarrow$	$(\Gamma \bullet \Delta, x: C)$
$(\Gamma, x : \exists R.C \bullet \Delta)$	$\Rightarrow$	$(R(x,y), y : C, \Gamma \bullet \Delta)$
$(\Gamma \bullet A \wedge B, \Delta)$	$\Rightarrow$	$(\Gamma \bullet \Delta, A)$ **and** $(\Gamma \bullet \Delta, B)$
$(\Gamma \bullet \neg A, \Delta)$	$\Rightarrow$	$(\Gamma \bullet \Delta, A)$
$(\Gamma \bullet C \sqsubseteq D, \Delta)$	$\Rightarrow$	$(y : C, \Gamma \bullet, \Delta, y : D)$
$(\Gamma \bullet x : C \wedge D, \Delta)$	$\Rightarrow$	$(\Gamma \bullet \Delta, x : C)$ **and** $(\Gamma \bullet \Delta, x : D)$
$(\Gamma \bullet x : \neg C, \Delta)$	$\Rightarrow$	$(x : C, \Gamma \bullet, \Delta)$
$(\Gamma, R(x,y) \bullet x : \exists R.C, \Delta)$	$\Rightarrow$	$(R(x,y), \Gamma \bullet \Delta. y : C)$

Figure 3. The logical rules for the Design Tableau

Taking our example of figure 1 one step further, using the logical rules, we get a tableau as in figure 4.

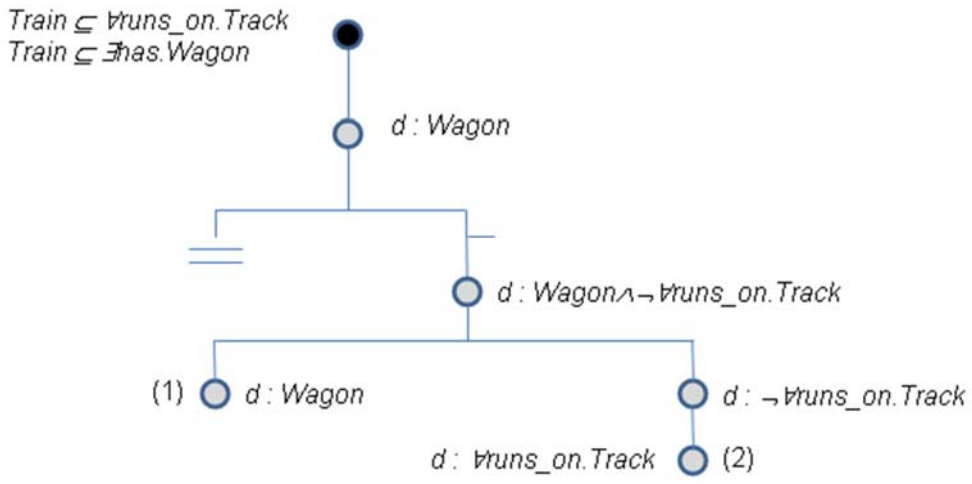

Figure 4. The Design Tableau example continued

The open branch with end node 1 now only has one untreated formula, *d : Wagon*, which is atomic and for which none of the logical rules applies. The second branch has as it only untreated formula *d : ∀runs_on.Track*. As no formula of the form *runs_on_Track(d, e)* is available the ∀-rule is not applicable. So we have two possible counter-models: one saying *d* is not a wagon and the other telling us *d* is running on tracks. To make both models impossible *d* should necessarily be a wagon not running on a track. Applying scenario 2 might lead us to the knowledge that a 'wagon' can be '*a heavy four-wheeled vehicle pulled by draught animals*'. To keep the example simple, we will use for now only a part of this knowledge and let us assume that if something is a vehicle with wheels we are willing to call that a wagon: $Vehicle \wedge \exists has.\ Wheel \subseteq Wagon$.
After adding our new knowledge (formula 1 in figure 5) and applying the logical tableau rules we end up with three open branches (with end nodes 2, 3 and 4). The resulting counter-models suggest we should search for a vehicle with wheels without tracks. Chances are that more information can be found on vehicles without tracks than for wagons without tracks (e.g. querying other agents knowledge bases), as 'wagon' would be more likely associated with trains.

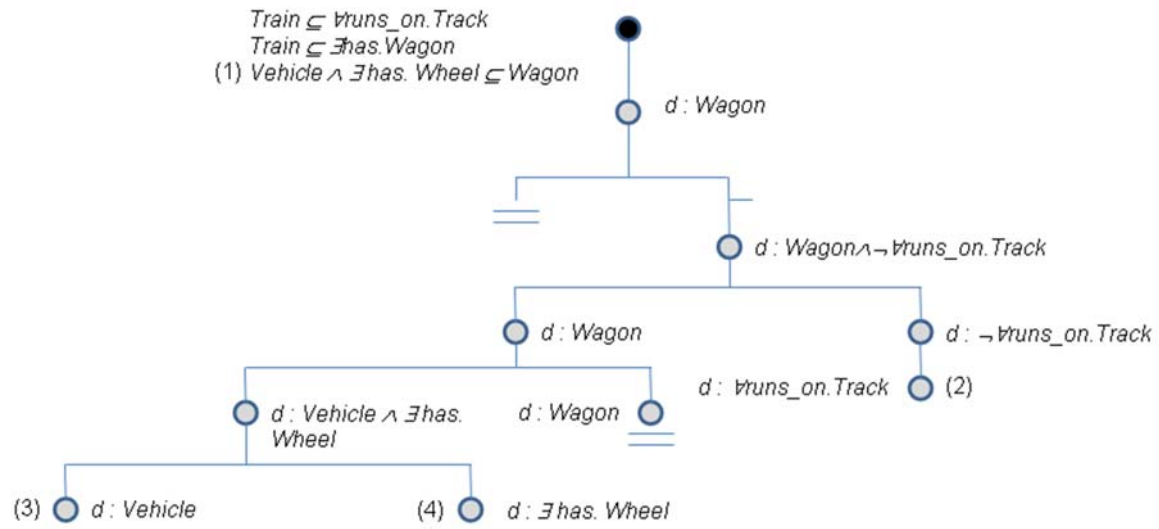

Figure 5. The Design Tableau example continued further

The example illustrates how comparing the situations produced from the open branches can support heuristics to construct a query for additional knowledge. Similarly, as in figure 1, it can help to identify interesting ways to change the design concept (e.g. by adding or deleting properties). The corresponding tableau rules for both scenarios are provided in figure 6.

(1) $(K; L \bullet \Delta) \Rightarrow (K; L, d : C \bullet \Delta)$ **and** $(K; L \bullet d : C)$
(2) $(K; L \bullet \Delta) \Rightarrow (K, A; L \bullet \Delta)$

Figure 6. The Design Tableau rules for the extra-logical scenarios

Applying scenario 1 includes, according to the first extra-logical rule of the Design Tableau the proof obligation of soundness for the design step introducing a new design concept *C*. If the tableau below the left branch contains an open branch (after all rules have been applied), the tableau below the right hand side is not save. If the designer decides to proceed with the new concept *C* a new Design Tableau of the form *(K • d : C)* would be required. Applying scenario 2 is simply adding new knowledge (a formula *A*) to the body of knowledge *K*.
Applying either scenario 1 or 2 introduces a real change in the design stage and it is by keeping track of such applications that with Design Tableaux one can follow the design process step by step.

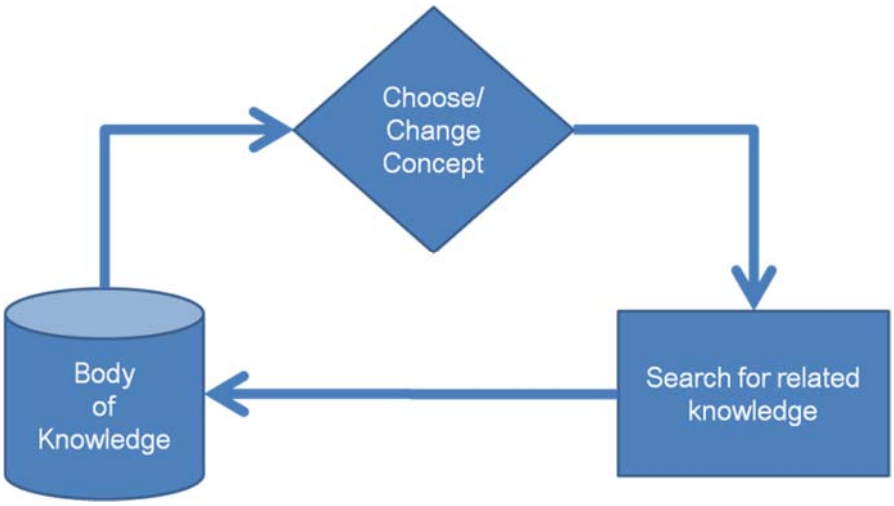

Figure 7. The basic principle for the Design Assistant

4. Architecture for a design assistant

In this section, we discuss an architecture for a design assistant that can be built around design tableaux. The architecture of a design assistant is combining a logical engine with a shell of interfaces to assist in applying scenario 1 and 2. The assistant will build a Body of Knowledge, which we can regard as a database. Scenario 1, changing the concept, and scenario 2, querying for new knowledge are the dynamic elements in the system.

Following the basic principle the architecture of our Design Assistant consists of three interfaces forming a shell around the main engine, a program building the Design Tableaux.

A typical session would start with an existing database containing some knowledge. In the Concept Manager interface one can study the concepts occurring in this body of knowledge and introduce a new concept, using combinations of concepts occurring in the body of knowledge. The next step then would be to start the Design Manager to build a Design Tableau.

The Design Manager will now usually find open branches in the Design Tableau (which can be studied with the Design Manager). These models can be used to either support formulating a research question (a query) to find new knowledge, or to suggest interesting concepts that emerged in the process.

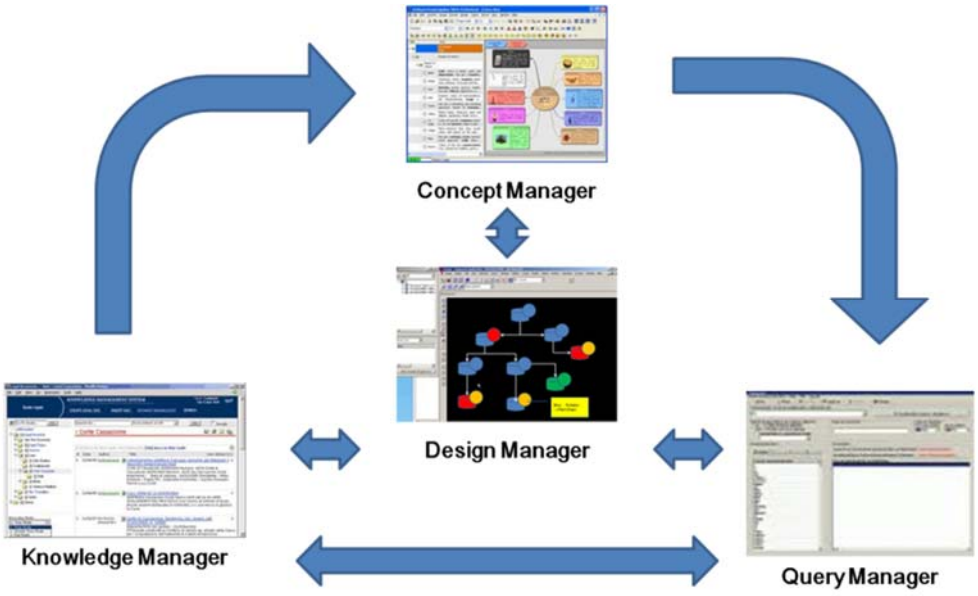

Figure 8. The architecture of the Design Assistant

The dynamic interplay between knowledge and concepts, the hall mark of C-K design theory, is implemented in this architecture by the appearance of new concepts in the answers to the queries and the use of these new concepts in the Concept Manager to refine the design concept, leading to new queries and so on, until a closed Design Tableau is found for a design concept satisfying the design team.

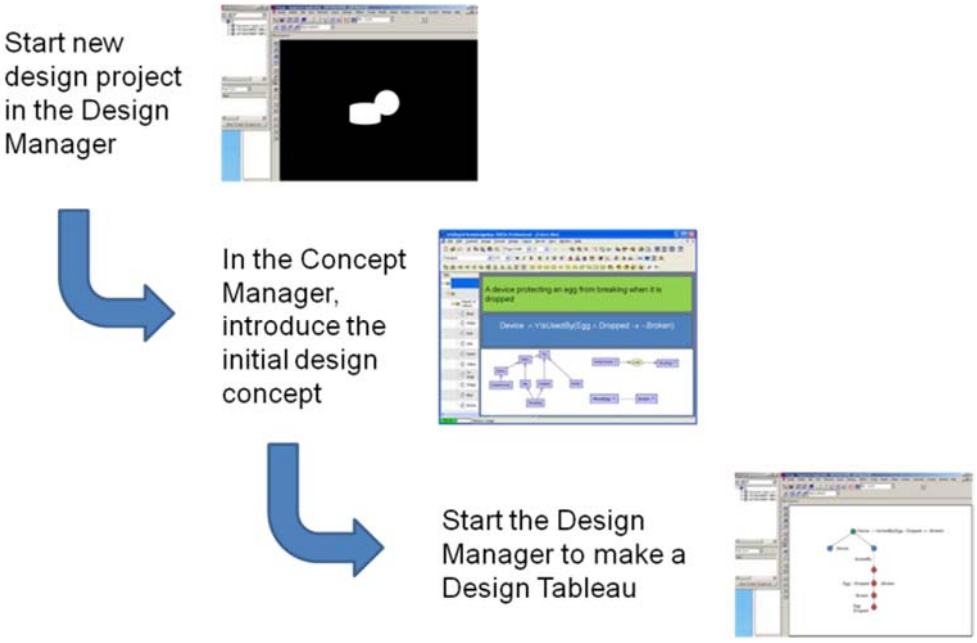

Figure 9. The first steps with the Design Assistant

4.1 The design manager

The main task of the Design Manager is to build the Design Tableaux and keeping track of the design stages by inducing and recording the application of the scenario 1 and 2 rules. During the design process the design team can decide to freeze certain design stages and proceed with others. For example in refining the design concept there will normally arise several alternatives. The design team may decide to develop several of them a few steps further to decide which seems to be the most promising one.
The Design Manager can show an overview of the design stages so far, such that the team may choose to open one of the frozen deign stages or start a new stage merging some of the frozen stages.

4.2 The concept manager

The Concept Manager is the interface to edit the design concept. It administers the vocabulary used in the Design Tableau, showing the relationships found between the concepts in use (the *ontology*).
In the Concept Manager the concepts in use can be graded for their preference and interest to the design team. Via the interest grades and the relationships found between concepts, the Concept Manager can suggest interesting concepts not yet included in the current design concept.
Both the actual ontology of the design stage and the design concept can be visualized as concept graphs.

4.3 The query manager

The Query Manager enables the editing of the queries for application of scenario 2. Based on the counter models provided by the Design Manager (from the open branches in the Design Tableau) the heuristics of the Query Manager can suggest queries. For example the counter models could suggest that most or even all branches would close if after a fall the pressure on the eggshell would be low, suggesting a query about *lowering the pressure on an eggshell*.

The query could in fact induce some scientific research into the issue, but for a start the query could also be used to find information in existing databases (including the body of knowledge managed by the Knowledge Manager) or on the internet.
The result of the query could be a text containing potentially useful information. In the Query Manager such texts (e.g. from the web browser) can be inspected and the most relevant paragraphs can be selected to be imported by the Knowledge Manager.
The first results of the query may not satisfy the design team and they may decide to edit the query. For example they could decide to change *lowering the pressure on an eggshell* into *lowering the pressure on a surface* hoping to find new results. In this case it is possible that *surface* does introduce a new concept in the Knowledge Manager later on.
In line with the graphical interface of the Concept Manager, the Query Manager will use a graphical interface along with the textual interface, using concept graphs.

4.4 The knowledge manager

The Knowledge Manager does not only store the collected knowledge, but also its translation in the formal language of the Design Tableau (e.g. Description Logic). As a full automatic translation of text into formulas is hardly feasible, the Knowledge Manager will allow for editing the formulas. Also in the Knowledge Manager a graphical interface using concept graphs is available.

5. Conclusion and discussion

In this paper we introduced the architecture of a design assistant based on Design Tableaux and its possible features. Consistent with the aim of the paper we have omitted several technical issues that have been studied elsewhere [Hendriks and Kazakci 2010, 2011] or under investigation. For example in our choice to introduce description logic in this paper we have been guided by complexity considerations. Design Tableaux were first introduced using first-order logic, for which the tableau method (using only the logical rules) is known to be undecidable. Restricting first order logic to a decidable fragment, e.g. description logic, is one way to make the Design Tableau method more feasible in practice. In introducing a restricted logical language one sacrifices a lot of expressivity. But the expressivity of description logic is be extended by OWL DL used for knowledge representation for authoring ontology's used for the Semantic Web.
In practice the computational complexity of developing a full Design Tableau would still be huge and heuristics are used to avoid a combinatorial explosion. The ontology that is kept of the Design Tableau by the Concept Manager can for example be used to select only those formulas in the body of knowledge that are in a sense related to the situation in the branch under construction, herewith avoiding unnecessary splitting of the tableau.
One could reasonably question the feasibility of the translation of knowledge formulated in natural language into logic formulas. We do however not suggest that such translations would be fully automated and hope that experience with prototypes of our design assistant will prove how a graphical interface can best support the human designers in this task.
Designing a Design Tableau based design assistant is also a theoretical exercise in analyzing the most basic operations in the design process and their relationships. Metaphorically speaking the design assistant is our 'Turing Machine', the theoretical device Alan Turing designed in the 1930's to formalize the concept of 'computation' and inspired the development of more realistic computers. Our analysis is not yet finished and more experimenting needs to be done. Nevertheless some lessons could be drawn already from the current version.
A closer look at scenarios 1 and 2 reveals that in both cases selection will play an important role. As suggested in the description of the Concept Manager, preferences and interests will play an important role to prevent a combinatorial explosion of all the possible choices to be explored. Such a basic role of the interests and preferences has either been taken for granted or is often omitted in formal approaches to design theory.
A more technical refinement of our formal framework results from the observation, thanks to the use of description logic, that a design stage $<K; C>$ will never close if C is a concept and K contains only general knowledge (e.g. description logic formulas of the form $C \subseteq D$). To close the Design Tableau

for *d : C* as the design concept description, at some point extra knowledge of the form *d : D* is needed. Formulas of the form *d : D* or *R(d,e)* form what is called in description logic an *ABox* whereas the general knowledge type of formulas, like $C \sqsubseteq D$ are what is called a *TBox*. Such a distinction between '*global'* knowledge and '*local*' knowledge is very useful, also in our more general formal framework for design, which is not necessarily restricted to description logic. The outcome of a closed design stage has in fact the form <*K,L; C*> where C is deducible from global knowledge *K* and *local* knowledge *L*. For example if *C* is the concept of a *wagon running without a track*, the local knowledge could contain concepts like a *Frame*, connected with *Wheels* connected to an *Engine*. The global knowledge *K* would teach us that *if* one would take a frame, connected with wheels that are connected with an engine, *then* the result would be a wagon running without a track. In description logic this would amount to: $K \vdash \sqcap L \sqsubseteq C$. The distinction between global and local knowledge is in line with the constructive approach in [Hendriks 2010] as the local knowledge provides us with the ingredients of the recipe for the artefact that corresponds to the design concept.

Our own overall conclusion from the theoretical exercise of designing a Design Tableau based design assistant is that such an endeavour can help to clarify the theoretical issues in formalizing design reasoning and supports our claim that also in design theory there is nothing more practical than a good theory.

References

Brown, D. C., S. E. Lander, et al. (1996). "The Application of Multi-agent Systems to Concurrent Engineering." Concurrent Engineering: Research and Applications 4(1): 2-5.

Campbell, M., J. Cagan, et al. (1999). "A-Design: An Agent-Based Approach to Conceptual Design in a Dynamic Environment." Research in Engineering Design 11(3): 172-192.

Gero, J. S. and F. Brazier, Eds. (2002). Agents in Design 2002. Sydney, Key center of design computation and cognition, University of Sydney.

Hatchuel. A.,Weil, B., "C-K design theory, an advanced formulation", Research in Engineering Design, Vol. 19, 2009, pp. 181 – 192.

Hendriks, A. Kazakci, A., "Design as Imagining Future Knowledge", Logic and Interactive Rationality. Seminar's yearbook 2010, D. Grossi (Ed), ILLC, University of Amsterdam, 2011, pp. 30-40.

Hendriks, A., Kazakci, A., "A formal account of the dual expansion of concepts and knowledge in C-K theory", Proceedings of the 11th International Design Conference - DESIGN 2010, D. Marjanovic (Ed.), FMENA, Zagreb, 2010, pp. 49- 58.

Kazakci, A. (2009). A formalisation of CK design theory based on Intuitionist Logic. International Conference on Research into Design. ICORD09. A. Chakrabarti. Banglore, India, Research Publising Services: 499-507.

Kazakci, A., Hendriks, A., "A method for design reasoning using logic: from semantic tableaux to design tableaux", Proceedings of the 18th International Conference on Engineering Design (ICED11), Vol. 2, S. Culley (Ed.), Design Society, 2011, pp. 275- 286.

Reffat, R. (2002). Intelligent Agents for concept invention in design. Agents in design. J. Gero and F. Brazier: 55-68.

Saunders, R.: (2001) Curious Design Agents and Artificial Creativity, Ph.D. Thesis, Faculty of Architecture, The University of Sydney.

Dr. Akin Kazacki
Mines ParisTech
60 boulevard Saint-Michel, 75272 PARIS Cedex 06
75272 PARIS Cedex 06 Paris, France
Telephone: 0(033) 1 40 51 92 08
Telefax: 0(033) 1 40 51 90 65
Email: akin.kazakci@ensmp.fr