ELSEVIER

Automation in Construction 16 (2007) 674–684

AUTOMATION IN CONSTRUCTION

www.elsevier.com/locate/autcon

# Tool support for computer-aided requirement traceability in architectural design: The case of *DesignTrack*

Ipek Ozkaya [a,*], Ömer Akin [b]

[a] *Software Engineering Institute, Carnegie Mellon University, United States*
[b] *School of Architecture, Carnegie Mellon University, United States*

Accepted 17 November 2006

**Abstract**

Requirement traceability is the commonly used term to refer to the management of the relationships that exist between design requirements and solutions throughout the design life-cycle. Enabling traceability within computer-aided design assists improving change management, consistency checking, and design compliance verification. *DesignTrack* is a prototype tool providing an integrated design environment for requirement specification and form exploration in the same design session. *DesignTrack* provides a navigation environment for complex design information spaces via enabling requirement traceability. This paper describes the design and functionalities of *DesignTrack*. It evaluates the applicability, power, and drawbacks of requirement traceability enabled computer-aided design by structuring the Leadership in Energy and Environmental Design (LEED) standard requirements with *DesignTrack* as a case study.


*Keywords:* Requirement traceability; Computer-aided design; Design knowledge; Design process; Information processing

## 1. Introduction

Requirement management related activities are often viewed as front–end activities in the architectural design life-cycle rather than as activities associated with design exploration [1–3]. Studies show that designers use multiple media such as sketches, drawings, charts, matrices, and text, to capture and manage requirements as design progresses [4,5]. Subsequently, designers usually need to use several different types of applications to bring these multiple media together to manage design requirements computationally. These applications most often are general purpose office applications, such as databases or spread sheets, which need to be forced to fit design information management. This approach is not only slow, inefficient, and error-prone, but it also does not support the natural shift between drawing and textual media that designers use for requirement management.

Computational design support tools for integrating requirement management with design exploration do not exist. Moreover, architects do not have access to computational strategies to assist them in using composite media, e.g. textual and graphical, for requirement management during architectural design exploration. Requirement management is an inseparable part of design and has to be considered in correlation with form exploration, rather than as a front–end task or as an activity which is addressed marginally. Identifying and formulating the relative algorithm sets for requirement manipulation such as how to search, filter and structure data effectively are crucial. Computational requirement management and engineering strategies need to evolve, along with algorithms to manipulate requirements for architectural design as well.

We set out with a goal to investigate how to fill requirement management related task gaps in computer-aided architectural design. We pursued computation of requirement–design relationships and traceability as strategies. An effective requirement management framework that offers requirement–design relationship management and traceability has the following immediate benefits:

1. Requirement data can be interfaced with other computational design tools, such as simulation tools, to measure performance of designs based on requirements.

* Corresponding author.
*E-mail address:* ozkaya@sei.cmu.edu (I. Ozkaya).

0926-5805/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.autcon.2006.11.006

2. Bottlenecks resulting from lack of tool support for switching between problem specification and design exploration can be circumvented.
3. Design errors can be traced and tracked as design progresses, enabling efficient use of requirement information elicited and better management of change during design.

We followed a process-based approach for identifying how to integrate form generation aspects of design with information management aspects through requirement traceability. Computational Hybrid Assistance for Requirement Management process framework, CHARM in short, describes a process whereby a designer, in the context of this work an architect, needs to be aware of the requirement decisions as she explores design solutions. The process defines tasks where the designer can query for the requirement information of a given solution, or track emerging data by interacting with the computational system. The core phases are requirement *definition*, *structuring* and *maintenance* (Fig. 1).

CHARM is not designed to capture processes involved in client interactions. Its input is based on the output of any client-based requirements elicitation activity. Definition is the phase where requirements are specified. This specification ideally should support both natural language specifications, for example "classrooms for 9th graders should have an area of 300 SF" and formal specifications, for example "classroom type = 9th grader, classroom size = 300 SF". Structuring of requirements is the phase in which the relationships between specific requirements of the given design project are formed. Lastly, maintenance is the phase where consequences of the designer's actions are kept intact within the requirement structures defined in the definition and structuring phases. Refs. [6] and [7] describe the process framework in further detail.

In order to investigate how to create requirement traceability capability within computer-aided architectural design based on the process phases CHARM describes, we created a prototype application, *DesignTrack*. We evaluated the capabilities of *DesignTrack* by applying them to Leadership in Energy and Environmental Design (LEED) for new construction and major renovation, version 2.1 standard requirements [8]. We parsed the guidelines given in LEED and defined them according to *DesignTrack*'s internal requirement traceability data structure. In particular, we examined the representation capabilities of *DesignTrack* for quantitative and qualitative requirement information. In this paper, we present these findings. Specifically, Section 2 introduces the conceptual overview of *DesignTrack*. Experiences in using *DesignTrack* for specifying and tracking LEED requirements follow. We describe the system design and present guidance for extending *DesignTrack* with a building information model focusing on Industry Foundation Classes (IFC) in Section 4. The paper concludes with a discussion of observed advantages of requirement traceability enabled design computing along with viable future research directions in the area of computer-aided support for requirement management.

## 2. *DesignTrack:* conceptual overview

*DesignTrack* is an interactive requirement modeling and form exploration medium. The system can be used by an architect who would like to manage her design tasks along with the requirements of the design problem at hand. This higher level goal may need the creation of specific shared design libraries, project repositories and project files. The interaction of the system with the user does not change, although different domain users may be at task such as a project manager, a draftsman, or an architect. The primary goals of the system are:

- Provide an integrated medium where designers can manage requirement information along with form exploration.
- Give ability to designers to reuse previous requirement information as complete body of requirement sets or in chunks.

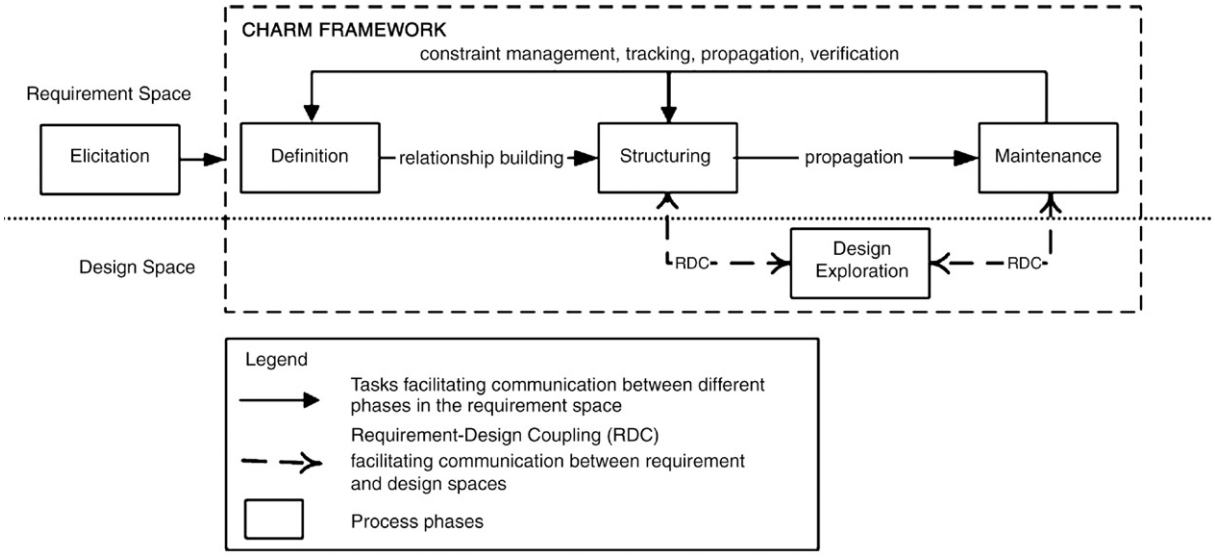


Fig. 1. CHARM process components.

- Allow designers to track the changes as they go through requirement specification or form exploration tasks.

Providing support for traceability in design computing, in other words allowing for a relationship-based view of the architectural design world via requirements can lead to a new mental model for architects in computer-aided architectural design. *DesignTrack* investigates the *integrated design environment* metaphor as a possible mental model where the architect can have one-stop tool availability for activities, which she otherwise often performs at the back of her mind. An integrated design environment provides a computer programming environment as a single application bundling a compiler, debugger, and graphical user interface builder. To her end, the interaction between requirement and design spaces in *DesignTrack* is very similar to how a computer-aided software engineering (CASE) tool operates. In general, tools to assist development of software are referred to as CASE tools [10]. CASE tools aim to support all facets of the software development process, from requirement definition such as use cases, to code generation, such as that observed in IBM Rational Rose. Given a representation in a CASE tool, for example an object-oriented class diagram, the user can generate code; edit code independently; or select the code and generate a class diagram. The user can choose to separate the diagrammatic representation, e.g. the class diagram, and the code by manually creating them as well.

*DesignTrack* perceives architectural design in a similar way, bundling a requirement modeler, a geometry explorer and the relationships between them (Fig. 2). This approach intends to provide the users with greater capability and increased options for accessing design information. Management and storing of the information happen in the same design session. Ability to investigate what those relationships are, especially in a scenario where the creator and consumer of requirement models are not the same user, provides a mechanism whereby the user can unveil more information about the data that is available through the system.

In order to support the primary goals for requirement management and traceability, we group functionalities that need to be satisfied under two high-level areas: 1) design information management, which includes requirement and design data use along with requirement-design coupling (RDC) strategies and 2) supportive functionalities, which includes session management, library use, and navigation. Next we will describe these high-level areas.

### 2.1. Design information management: requirements, designs, RDC

Requirement and design management and how they relate to each other, i.e. requirement-design coupling (RDC), carry the most significant functionality of *DesignTrack*. In *DesignTrack*, as the designer inputs requirements, she can switch to the design space to specify a solution and manually link it to the requirements it satisfies. Or she can select a set of requirements and choose to generate a design satisfying the requirements. The designer can query a requirement for where it is satisfied in the design space, which requirements it depends on or is derived from, the impact of making changes in its assigned values, and its creation history. Alternatively, the designer can query a specific design element and see to which requirements it corresponds. RDC functionalities bring together design domain and requirement domain features by adding functionality that facilitates using them simultaneously in conjunction with each other.

In *DesignTrack,* the mapping between form exploration and requirement specification activities is treated as a transaction, which is a single operation of computation. Based on the information received from the user, transactions are created atop related requirement and design data (Fig 3). There are two types of transactions; *requirement transactions* and *geometry*

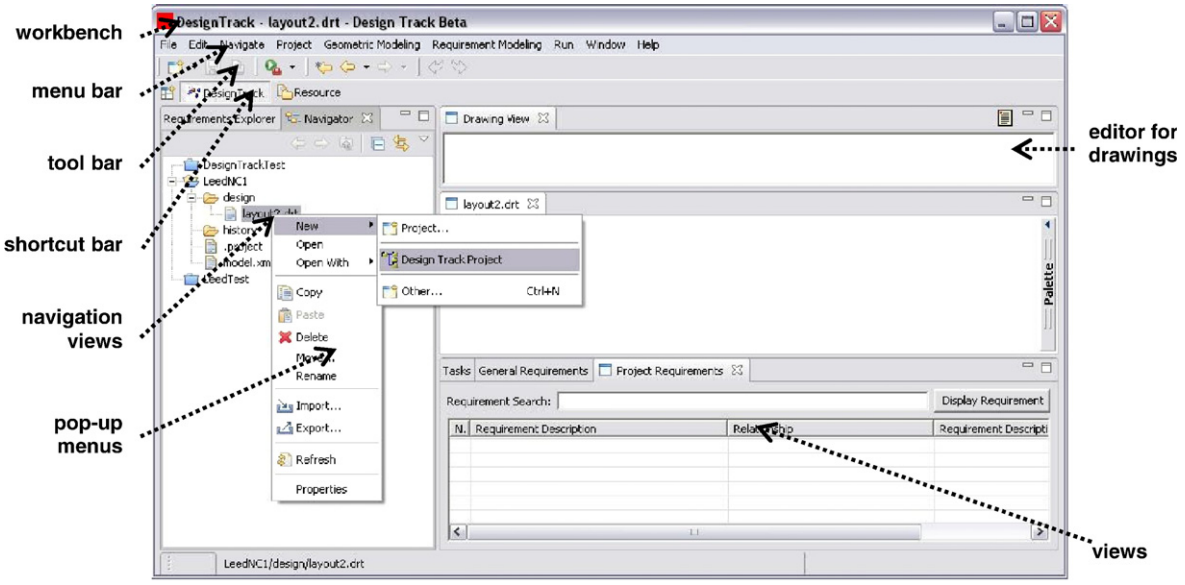


Fig. 2. The main window of *DesignTrack*.

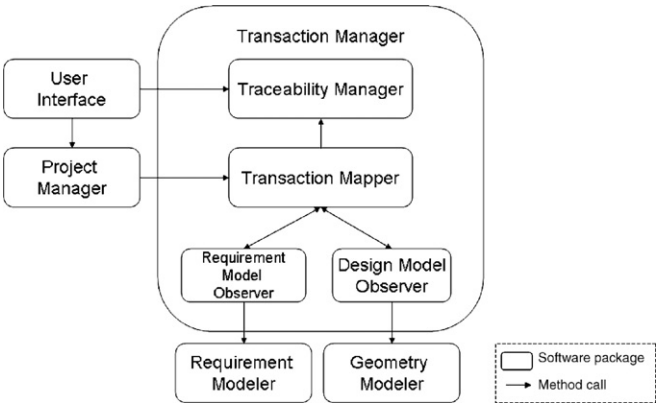


Fig. 3. Transaction Manager package in *DesignTrack*.

*transactions*. Editing, placing and deleting geometric components are the geometric transactions. Creating a new relationship, creating a new requirement, editing a requirement, deleting a requirement and propagating the changes are requirement transactions. Geometry transactions invoke requirement transactions because associations between requirements and designs are built as originating from requirements. Once a user associates a requirement with a geometric structure the requirement and geometry transactions are coupled, allowing for using traceability relationships. For example, if an edit operation is done on a geometric form, the system invokes propagation and requirement editing transactions for completeness of operation based on the geometries and requirements related to each other. The transactions execute based on the global identifier of the geometry that the operation acts upon.

The desired links in *DesignTrack* between requirement space and the design space necessitates dynamic restructuring of the requirement space. This task is easier with requirements that specify the structure, such as "Space A" is adjacent to "Space B", or "Space A is *x* square feet" for this translates to a one-to-one mapping between the design structure and the requirement space. For behavioral requirements, this task is harder to isolate. For example electrical wiring has implication both in an isolated space level and also as a cross cutting concern that may be applied to all the spaces. In its current version *DesignTrack* handles tracking requirements between design and requirement modeling if they are geometrically representable. Performance requirements can be tracked within the requirement modeling space.

The community of building product data modeling in CAD research has been struggling with this problem of many-to-many relationships of design at an instance level for over a decade now [11]. There are other research prototypes that also tackle requirement modeling problem as a dynamic data modeling process, creating requirement objects as it is done within *DesignTrack*. SP-II, Seed-Pro [12], and RaBBiT [13] all instantiate a requirement object. SP-II defines a user-initiated, flexible set of attributes on that requirement object. RaBBiT takes a further step and allows for dependencies between requirements by formula editing. *DesignTrack* similarly instantiates a requirement object; however, the attributes on the given requirement object are partitioned as sets of relationship and other requirements.

*DesignTrack*'s requirement definition is based on parsing a given specification into requirement-relationship streams. *DesignTrack* makes the distinction between composite-requirements and atomic-requirements by the use of relationships. An *atomic-requirement* is a requirement that cannot further be broken down into other requirements in a requirement statement. It may imply other requirements, but that represents a relationship with other requirements. A requirement defined as an atomic-requirement can later participate in generation of a new requirement by means of relationships. A *composite-requirement* is a requirement statement made of atomic-requirements and the relationships between them. Since designers cannot refer to a complete list of requirements at each instance of the solution path, the tendency is to try to think of general relationships between them. *Hierarchy*, *indexing*, *classification*, *association*, *dependency*, and *implication* relationships are used in generating complex requirements.

Whether requirements are statically defined, to the extent that they become space requirements, equipment requirements and the like (as exemplified in Seed-Pro), or very flexibly left as a requirement (as exemplified in *DesignTrack*, RaBBiT, SP-II), a requirement is best handled as an object. The most cumbersome implementation aspect of this is when relating requirements to designs. One requirement object in one design session may instantiate a design component; in another session the same requirement object may instantiate multiple design components. Since in *DesignTrack* attributes are specified as relating objects, the many-to-many relationships can also be built at an instance level.

RDC takes advantage of requirement states. Until the user associates a defined requirement with a geometric entity, a requirement is in the *Awaiting* state. These states are not apparent to the user, but are used to assist the computation of traceability. If the architect associates a requirement with a geometric entity, the state of the requirement changes to *Satisfied* state. If *DesignTrack* cannot detect changes, the state of the requirement changes to either *Undetermined* or *Unsatisfied* when deletions or changes occur.

In computing requirement traceability between the requirement and geometry spaces there are two viable high-level scenarios. Generative design is the first of those. In generative design the system needs to be implemented with a capability to generate a design from requirements and recall from which requirements the design was generated. *DesignTrack* pursues the second of the possible scenarios where the architect indicates the dependencies (Fig. 4). The advantage of this approach is that both the requirement and geometry exploration can expand independent of the existing generation capabilities. The drawback is that it may grant more control to the architect than desirable, with possible user interaction overhead.

Fig. 4 demonstrates a case where the association of the requirement with the geometry is at space level only. Since this particular requirement only identifies the need for the space, rather than any geometric attributes, even if the size of the space changes the requirement is still going to be satisfied. If the geometric entity is deleted the coupling is broken. *DesignTrack*

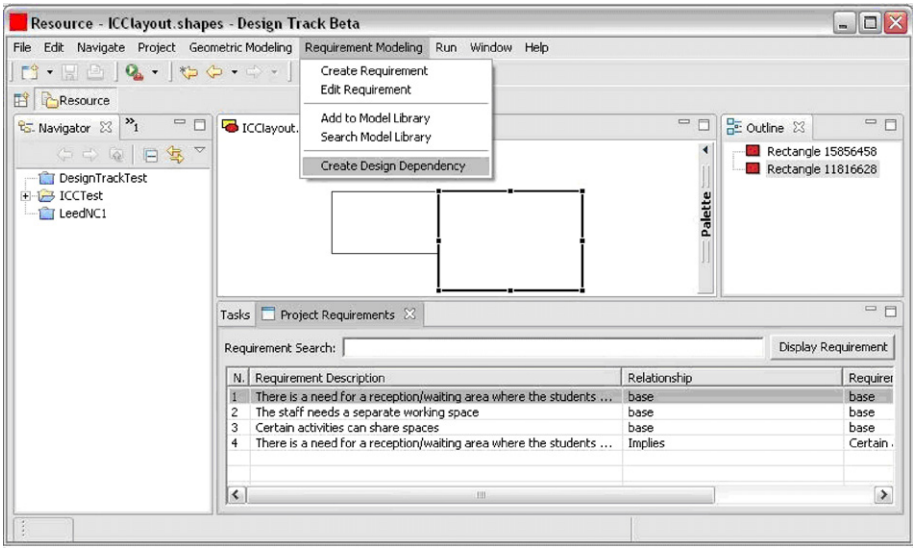


Fig. 4. Associating requirements with designs.

changes the state of the requirement back to *Awaiting*. This is reported with attributes as requirement details as well (Fig. 5A).

*DesignTrack* abstracts geometric manipulation in a two dimensional space level. The architect can create and manipulate rectangular spaces interactively. A space in the context of *DesignTrack* has height, width, space id, space function, and location attributes (Fig. 5B). Using these attributes, namely space descriptions and space sizes, adjacencies, proximities, and accessibility requirements can be related to designs in *DesignTrack*.

In associating requirements and geometries, the user must specify the values to be associated between requirement views and geometric constructs. Fig. 5B shows the screen values in a scenario where the minimum value of a tutoring room is taken from the classrooms and is specified by selecting the relevant space geometry's specification dialog.

### 2.2. Supportive functionalities: session management, library use, navigation

Session management, library use, and navigation provide supportive utilities. These are intended to ease interaction with the tool and enhance the integrated design environment metaphor. Fig. 6 is an example snapshot where project navigator view, tasks view, and project pop-up menus are captured.

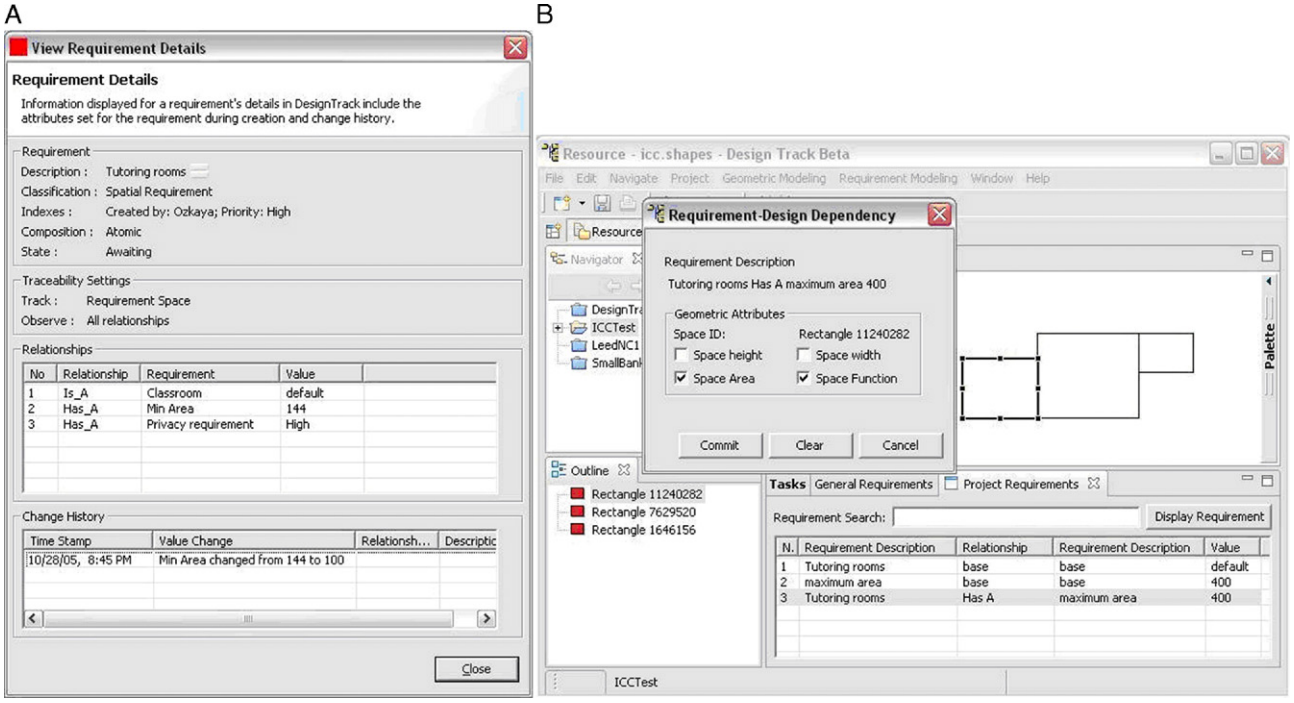


Fig. 5. A) Inspecting the details of a requirement in *DesignTrack*, B) Associating geometries with requirements.

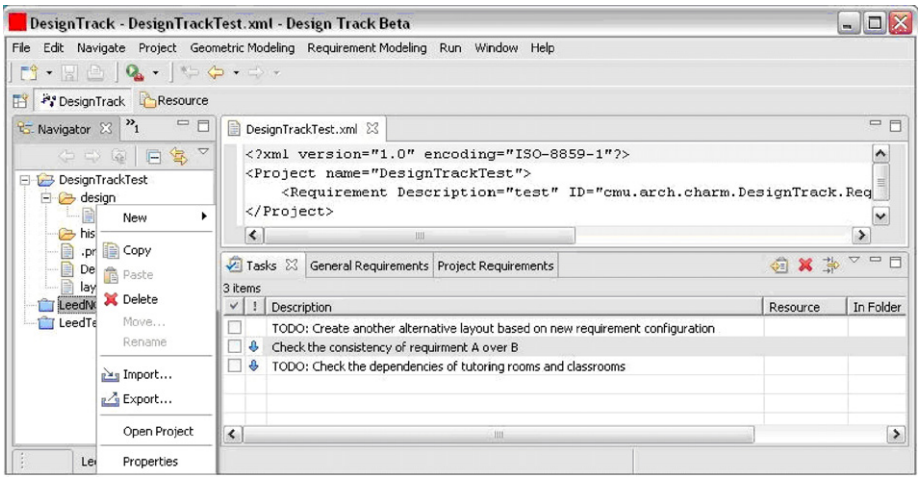


Fig. 6. Project Navigator view, Tasks view, and Project pop-up menus.

Session management related functionalities group together user goals related to managing the current design session, such as creating a project or setting project settings.

Library use related functionalities address cases where a user may need to maintain a requirement library. The user can maintain a requirement schema. This includes defining, changing and deleting a set of requirements that are related to each other as an xml file. Lastly, navigation-related functionalities are those that offer tracking capabilities of the requirements. Invoking the functionalities in this group helps the user inspect individual requirement details, its change history, and dependencies between requirements along with the design forms associated with them. For example the outline view provides an added mechanism to track spaces. As spaces are added to the editor they also appear in the outline view (Figs. 4 and 5B). Propagation of change is handled as a navigation strategy. When a change occurs either in design or requirement space the parts that are dependent on the change are presented to the user to assist verification of change. The relationships built assists in querying and extracting the affected parts. Assistance to browse a history on a requirement object is another example to navigation support.

## 3. An evaluative study: defining, structuring and tracking LEED guidelines in *DesignTrack*

### 3.1. Defining and structuring

Any architectural design project begins with a set of given requirements. These requirements can be as well-defined as a pre-prepared program or as fuzzy as a list of spaces and soft expectations. Designers acquire new information about the project, based on the initial given set of requirements and local design decisions. However, developing a successful design does not end there. There are various sets of external factors that apply to the design project as global (or external) requirements and expectations. Satisfying LEED criteria following green and sustainable design principles is an example of global expectations that may apply to designs.

In a design context where such bi-polar requirement sets are applicable to the design decisions, various design alternatives may be used to satisfy different requirement sets. *DesignTrack* provides distinct viewers for general versus project specific requirements. In *DesignTrack*, a general requirement is a requirement that has a global impact on every project, for example "U.S. General Service Administrations building guidelines" could be defined as general requirements. Any given project must follow them, in addition to specific project-based requirements. A project requirement then would be a requirement that is applicable only in the context of the given project. Using a generic requirement model in *DesignTrack* assists categorization of requirements. A generic requirement model in principle is added to the requirements that a user wants to abide by, therefore specifying one in a given design session is not permitted in *DesignTrack*. However, in *DesignTrack* emerging requirement models can be saved in the library and can be used in future design sessions. Therefore, specifying and using LEED requirements were done in two separate sessions; one session to define the LEED requirements, and another session to use them in conjunction within a design project.

Defining natural language requirements such as those in LEED takes advantage of the requirement-relationship mechanism we presented in Section 2.1. LEED as a requirement model is a set of guidelines rather than a set of standards with specific quantifiers. Modeling this in *DesignTrack* uses defining the requirements as atomic-requirements. Given general guidelines such as those suggested in LEED, the relationship mechanism in *DesignTrack* allows for modeling the multitude of paths in this standard, partially formalizing otherwise hard to formulate semantic dependencies between requirements.

Modeling the erosion and sediment control requirement is an example. The erosion and sediment control requirements in LEED provide the following implementation strategies [8].

> Adopt an erosion and sediment control plan for the project site during construction. Consider employing strategies such as temporary and permanent seeding, mulching, earth dikes, silt fencing, sediment traps and sediment basins.

There are several requirements that can be extracted from this description as atomic-requirements. These are shown in Fig. 7. *DesignTrack* allows for the user to define her own

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <Project name="LeedNC1">
- <Model Type="General">
  <Requirement Description="temporary seeding" Classification="Erosion/Sedimentation
    Control"
    ID="cmu.arch.charm.DesignTrack.RequirementModeler.RequirementDataStructure.
    Requirement@1970991" />
  <Requirement Description="sediment traps" Classification="Erosion/Sedimentation
    Control"
    ID="cmu.arch.charm.DesignTrack.RequirementModeler.RequirementDataStructure.
    Requirement@1262bf4" />
  <Requirement Description="mulching" Classification="Erosion/Sedimentation Control"
    ID="cmu.arch.charm.DesignTrack.RequirementModeler.RequirementDataStructure.
    Requirement@1a5f230" />
  <Requirement Description="Prevent sedimentation of storm sewer or receiving
    streams." Classification="Erosion/Sedimentation Control"
    ID="cmu.arch.charm.DesignTrack.RequirementModeler.RequirementDataStructure.
    Requirement@19d0cf0" />
  <Requirement Description="earth dikes" Classification="Erosion/Sedimentation Control"
    ID="cmu.arch.charm.DesignTrack.RequirementModeler.RequirementDataStructure.
    Requirement@1bf6ec8" />
  <Requirement Description="Prevent polluting the air with dust and particulate matter."
    Classification="Erosion/Sedimentation Control"
    ID="cmu.arch.charm.DesignTrack.RequirementModeler.RequirementDataStructure.
    Requirement@1bc93a7" />
  <Requirement Description="sediment basins" Classification="Erosion/Sedimentation
    Control"
    ID="cmu.arch.charm.DesignTrack.RequirementModeler.RequirementDataStructure.
    Requirement@10fba68" />
  <Requirement Description="permanent seeding" Classification="Erosion/Sedimentation
    Control"
    ID="cmu.arch.charm.DesignTrack.RequirementModeler.RequirementDataStructure.
    Requirement@181e7fe" />
  <Requirement Description="Prevent loss of soil during construction by stormwater
    runoff and/or wind erosion, including protecting topsoil by stockpiling for reuse."
    Classification="Erosion/Sedimentation Control"
    ID="cmu.arch.charm.DesignTrack.RequirementModeler.RequirementDataStructure.
    Requirement@1ca7841" />
  <Requirement Description="silt fencing" Classification="Erosion/Sedimentation Control"
    ID="cmu.arch.charm.DesignTrack.RequirementModeler.RequirementDataStructure.
    Requirement@10241ae" />
  </Model>
  </Project>
```

Fig. 7. Base requirements for erosion and sediment control.

classification and index schemas. This feature permits easier grouping of the LEED requirement categories, such as identifying this set of requirements as erosion and sedimentation control class requirements. In this particular example, the user classifies these requirements as “Erosion/Sedimentation Control” to structure the requirement set further.

Building on the erosion and sediment control requirements example, while the guidelines are generic the user can further decide with which strategies to address LEED concerns in the project view. Fig. 8 has an XML snippet exemplifying this; notice that not all base LEED requirements are included here. This allows for managing choice of strategy to address for project specific concerns via requirements. This transition from general requirements guidance to a specific one also exemplifies an area where the borders between requirements and solutions start disappearing as well since selecting a strategy is a solution approach taken in addressing the general requirement.

The existence of multiple models is powerful, especially when the generic model is global in nature, where the requirements may lead to unexpected design decisions. The advantage of keeping them separate is that multiple requirement data structures can co-exist in the design space and a given design structure can conform to several specified requirements.

The classification and indexing mechanism also assist modeling requirement prerequisites. An example of such a requirement is the HVAC system guidelines in LEED, in which CFC-based refrigerants should not be used. After this requirement is satisfied, then there are several levels of energy optimization requirements to which a design should adhere. The traceability relationship mechanism allows for understanding these dependencies. The indices help in grouping such requirements further. For example it is quite easy to distinguish the indispensable or prerequisite requirements from the optional ones in the modeling stage can be distinguished by indexing them and reuse these guidelines as needed in future projects as general requirements.

In LEED, requirements are provided as guidelines that should be taken into consideration; however, the quantifiable aspects are not provided. Relating the general guidelines to a specific quantifiable mechanism becomes possible when a given project requirement uses the general requirements. When a general requirements model is specified as part of a project in *DesignTrack*, the requirements of that model becomes part of the requirement model to be tracked. They are not editable as generic requirements in the project session, since they are part of a standard. However, as project requirements they can be further

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <Project name="SmallBank">
- <Model Type="Project">
- <Requirement Description="Prevent loss of soil during construction by stormwater
runoff and/or wind erosion, including protecting topsoil by stockpiling for reuse."
Classification="Erosion/Sedimentation Control"
ID="cmu.arch.charm.DesignTrack.RequirementModeler.RequirementDataStructure.
Requirement@1ca7841">
- <Relationships No="3">
<Relationship Type="Has_A" RelatingRequirement="sediment basins"
Classification="Erosion/Sedimentation Control" value="5"
ID="cmu.arch.charm.DesignTrack.RequirementModeler.RequirementDataStructure.
Requirement@10fba68" />
<Relationship Type="Has_A" RelatingRequirement="permanent seeding"
Classification="Erosion/Sedimentation Control" value="default"
ID="cmu.arch.charm.DesignTrack.RequirementModeler.RequirementDataStructure.
Requirement@181e7fe" />
<Relationship Type="Implies" RelatingRequirement="silt fencing"
Classification="Erosion/Sedimentation Control" value="around perimeter"
ID="cmu.arch.charm.DesignTrack.RequirementModeler.RequirementDataStructure.
Requirement@10241ae" />
</Relationships>
</Requirement>
</Model>
</Project>
```

Fig. 8. Transitioning general requirements to project specific context.

quantified and related. Since the generic requirements become sub-models of the project requirements, it also becomes possible to use them in relationships. Here the strength of *DesignTrack* is that a user can start with a base model and expand on it.

### 3.2. Tracking

*DesignTrack* looks at the design world from the perspective of requirements by subsuming geometric explorations. The state of entire design at any given time can be answered in *DesignTrack* by analyzing the state of the existing requirements. Therefore, *DesignTrack* does not give an absolute answer, but provides a relative one by tallying the requirements based on each state. Using the LEED requirements revealed an interesting aspect of this issue. Often design solutions do not represent 100% quantitative compliance with given requirements, but they represent a point between trade-offs. Moreover, in the context of LEED, there are many requirements that are action-based rather than design-based, such as “hire a commissioning team” or “submit LEED letter template signed by architect or owner that the recycle collection area is accessible”. Similar requirement examples that affect the design, but are not representable with drawings, frequently occur in design. While it is still possible to identify such instances as parts of the requirement model in *DesignTrack*, this approach creates a potential correctness problem when *DesignTrack* tallies design states. Project actions and requirements that can be tracked with geometric representation should be separated from each other.

When working through LEED requirements, there were several requirements, often in the form of performance requirements that could not be represented geometrically. The lighting requirement is a typical example. While calculating the lighting values require the position of windows, evaluating the state of design with respect to satisfying that requirement necessitates performance analysis. In that respect, the state of design that would be reported is the *Undetermined* state.

Currently, *DesignTrack* reports the state of a requirement when it cannot track or propagate changes. However, the *Tasks* view allows the architect to report soft aspects, requirements or constraints (Fig. 6). In the context of the erosion and sedimentation control requirements in LEED, a task may be “ask for the United States Environmental Protection Agency (EPA) Document No. EPA 832/R-92–005 (September 1992) forms from US Green Building Council”. The capability to model such soft aspects of the requirement modeling problem enables the tool to subsume simple project management tasks as well.

## 4. System design

*DesignTrack* is implemented in the Eclipse platform as a plug-in application, making use of the extensions mechanism [9]. A plug-in is a computer program that can, or must, interact with another program to provide a certain, usually very specific, function. Extensions, on the other hand, modify or add to existing functionality. The main difference is that plug-ins generally rely on the main program’s user interface, and have a well-defined boundary to their possible set of actions. Extensions generally have fewer restrictions on their actions, and may provide their own user interfaces.

The Eclipse runtime is responsible for managing the lifecycle of a plug-in within a workbench. A workbench in Eclipse is the overarching container for all windows. All of the plug-ins for a particular environment are located in a plug-in folder within the directory structure of rich client platform application. Fig. 9 shows how *DesignTrack* plugs into the Eclipse environment. Upon execution, the Eclipse runtime will discover all of the available plug-ins and use this information to create a global plug-in registry. The high-level system architecture is composed of *User Interface*, *Project Manager*, *Transaction Manager*, *Requirements Modeler*, and *Geometry Modeler* packages (Fig. 10). A package in this context is a collection of logically related classes, other packages and functionalities.

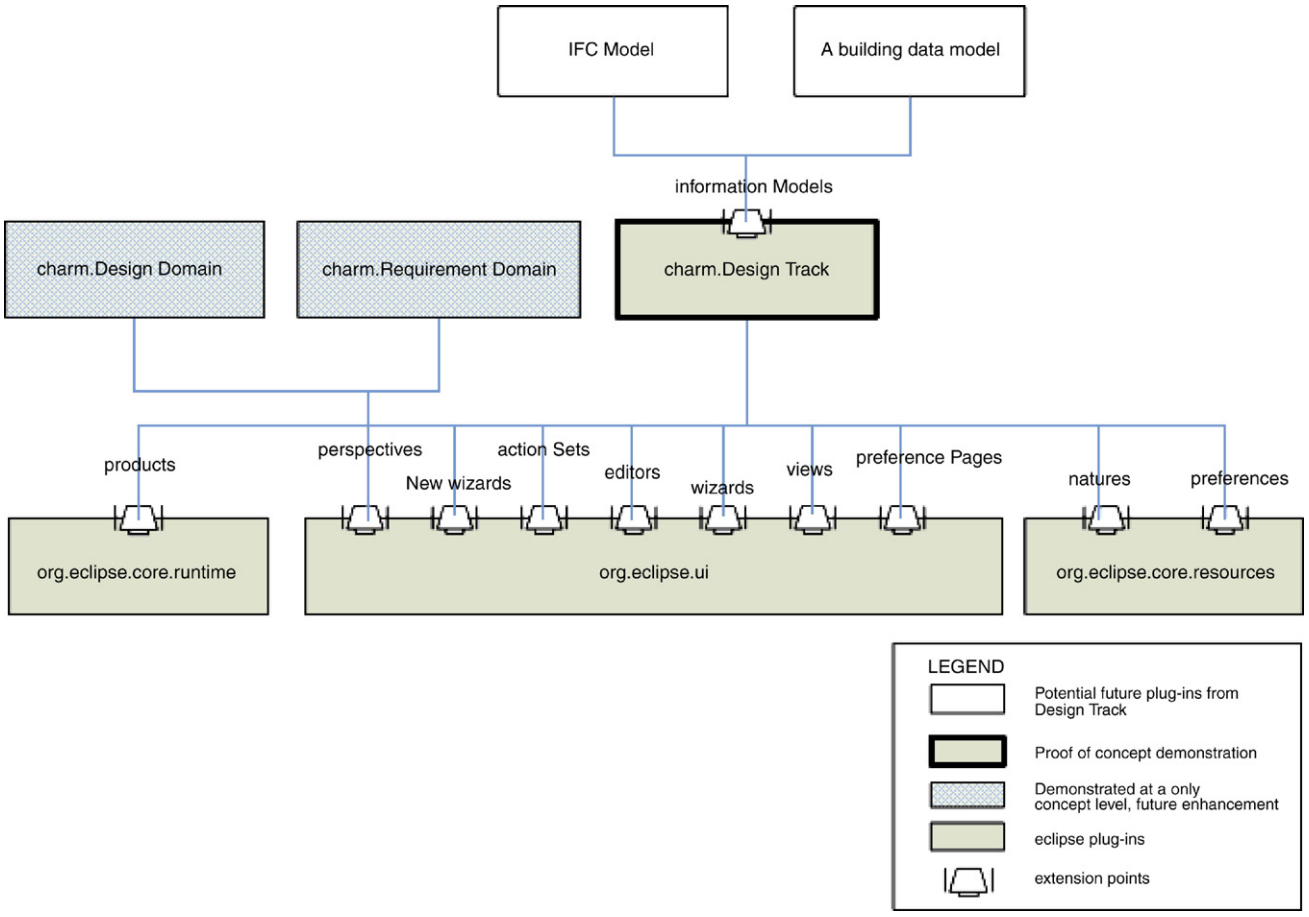


Fig. 9. Plug-ins *DesignTrack* utilizes within Eclipse.

The advantage of a platform such as Eclipse can be observed in handling a building information model with a system such as *DesignTrack*. The challenge of building data models in addressing information traceability – including the most widely used building product data model both in industry and academia, IFC – is the lack of reliable and computable protocols, which can help trace the information back and forth between different stages of design. In order to observe the conformity issues of building information models to requirements data, we modeled example relationships using IFC. IFC are incomplete in requirement data modeling; however, more critically the requirement information is dispersed in the building information model. For example, the requirement "The staff needs a separate working space" can be represented with the participation of ifcOccupant, ifcTask, ifcSpace classes, moving between geometric- and information-based objects. This examination supports the observation that simultaneous development of geometric and requirement data is often called for. Moreover, this simultaneous development is in an instance level. IFC is a static representation; its run-time instances do not exist for the data classes.

A promising and potentially successful approach to make *DesignTrack* "building information model" intelligent is based on how *DesignTrack* separates the definition of classifications and indices and allows for a file-based definition of those, rather than a session-based definition. Currently, the classification and indices capability takes care of only Boolean and text-based indices and classifications. Extending the classification and indexing components into handling complex object-based classes and indices would enable a requirement to be classified as a given IFC-based class as well.

If IFC compatibility or manipulation is desired a developer could best take the following approach in extending *DesignTrack*:

- Create an extension project to the current *DesignTrack*. Creating this as an extension will cause this model to be loaded only when used.
- Extend the Classification class of *DesignTrack* to specialize an IFCBasedClassification class.
- In the IFCBasedClassificationClass identify which classes and types to be utilized as classes.

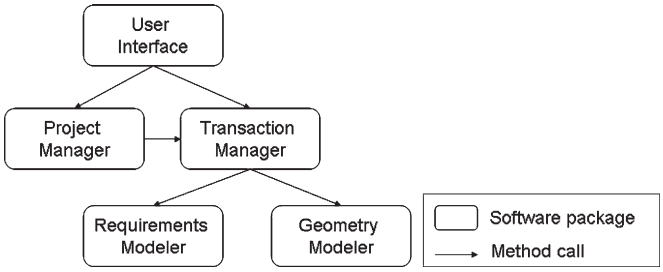


Fig. 10. High-level system design within *DesignTrack*.

- Add a new dialog to the charm.*DesignTrack*.dialogs that is capable of showing the object-based classifications with attributes from IFC, such as that of IfcSpaceProgram.
- Depending on what is required to be displayed create a new view in charm.*DesignTrack*.views, if more attributes are required to be displayed.

Building information models take the standardization view where the universe of discourse in a given building domain can be represented in its totality. *DesignTrack* takes the approach where the user can get by representing only what she needs in it. In the world of modeling this idea creates a potential mismatch because often slicing what you need from a building information model is not as straight forward. Especially when there is a complex type-based system, it is quite easy to come up with bulkier representations of models than needed. For example, to model a space program with IFC the system will also need to be aware of IfcIdentifier, IfcAreaMeasure, IfcOccupant and IfcTask classes. The multiple model perspective of *DesignTrack* can also be beneficial in this regard as well.

This approach is in a way slicing a view from the IFC model which is useful only for requirement purposes without differentiating between requirement or space objects. Typical approaches, such as Kiviniemi's [14], extend a building information model or create a new one with all possible requirement attributes. While extending a model is an attempt in creating a universal view, it ignores the fact that in architectural design semantic differences occur very often in representing design attributes. Extending *DesignTrack* to enable use of IFC as a classification superset provides a flexibility for users to take advantage of modeling their own world. This approach makes the building information model transparent to the user and allows it to be used with other specific models.

## 5. Conclusions and future work

We observed several advantages in following a plug-in based prototype development environment which can assist enhancing CAD tool usage metaphors. The benefits of an extensible system design as in *DesignTrack* for computational support for requirement traceability are:

1) *DesignTrack* can also have extension points. A possible extension for *DesignTrack* is the possibility to integrate it with building data models as we describe in Section 4 and follow the data attributes that exist in those models. Several research projects discuss the shortcomings of incompleteness of these models, the inability to traverse the models for completeness of information and the bottlenecks that occur when the models are extended with new functionality. A plug-in based development environment allows for switching selected old functionality on and off without computing overhead.
2) Plug-in based extensible environment supports modifiability by ease of integrating new functionality onto *DesignTrack* through defining new extensions. It also makes the possibility of realizing the scenario allowing the architect to switch between multiple design phases by switching domains in one application.
3) Reusability is achieved. A new software application, which aims to integrate traceability functionalities as they are exercised in *DesignTrack*, provided that they choose Eclipse as a development framework, only needs to include the plug-in into their environment. They can develop their added functionalities by not having to be constrained by the design of *DesignTrack* or compile its code, but still use it.
4) Plug-in and extension mechanisms support ease of user interface implementation. Usability is an important aspect of requirement traceability research. Eclipse comes with its user interface plug-ins which allows for demonstrating *DesignTrack* with better incorporated user interface in the beta version. This is both a usability and work redesign problem [15,16]. *DesignTrack* provides some basic mechanisms to assist the added design tasks; such as navigation, tabular views of requirements, a Task view to take notes and some obvious warning and error messages (Fig. 6).

Traceability is not only a computation task, but it is also a very challenging usability problem. The prototype provides a promising direction by introducing the integrated design environment metaphor to managing multiple design phases in one session. For example, the ability to track geometric components and their attributes both graphically and as lists of components in another view is a useful feature. While in small examples these prove to be helpful, in large design problems even navigating through that space may prove to be challenging. The integrated design environment metaphor can help by providing hot-key links, through which selecting one object in one view would cause the selection of it in other views, and so on.

The examples in LEED showed that dependencies between requirements and designs often are more complex than that can be captured with a predefined model, they are instance-based. The one-to-many and many-to-many relationships between requirements and designs are observed during design sessions rather than during data modeling. This work introduces a design session-based view to treat these relationships. The strategies to compute requirement–design relationships include state-based and transaction-based computations. Transaction-based computation allows for capturing information moves both in design and requirement spaces depending on the design action. Transaction-based modeling also allows for managing multiple models and tracking both design and requirement spaces. This approach is a "design move" approach in contrast to the "model the world" approach that is favored in building information modeling.

One requirement may be represented by one or many design structures in a given design session. *DesignTrack* by incorporating state as part of a requirement object enables computation of design compliance as well, hence advancing the treatment of requirement–design relationships in design computing. State-based computation enables cognitive assistance to architects such as tallying the state of the design and reporting inconsistencies based on state.

An important question to ask is whether the integrated design environment approach is compatible with the natural flow of design and what it takes for the users to adjust to such a mental model. Addressing this requires understanding the gap an integrated architectural computational design medium would fill. The nature of design activities that can be handled more easily with an integrated design environment would be those that take a performance view and worry about systems design, such as for HVAC, lighting and acoustical performance. These requirements are hard to track and tally; and for such concerns, architects often have to manage multiple incompatible application environments. Requirement activities are often handled in the background of the designers mind. *DesignTrack*, the outcome of investigating with CHARM, embodies such a metaphor for a new design environment.

Automation is a challenging task in design computing. There are numerous existing approaches such as rule-based systems, expert systems and other artificial intelligence techniques [17]. For requirement-driven design understanding automation first requires to understand the requirement space well. One of the initial steps this current work provides is in understanding repeating requirement model structures that can be candidates for automation. This requires modeling as many different examples as possible and capturing repeating patterns to generate automation techniques. A maintainable and traceable specification of early design information as demonstrated by *DesignTrack* has several potential applications in design. Building commissioning, performance analysis and generative design are a few potential domains that can utilize such flexible requirement information and multiple model use. Understanding the types and nature of repeating requirement information and most commonly used relationships between them is a promising future research direction based on the work we describe here. Understanding these patterns would allow focus automation tasks in the areas most beneficial to the practice that would be immediately useful in these domains.

## References

[1] R. Hershberger, Architectural Programming and Predesign Manager, McGrawHill, New York, 1999.
[2] E. Cherry, Programming for Design, John Willey and Sons Inc., NY, 1999.
[3] W. Pena, S. Parshall, Problem Seeking: An Architectural Programming Premier 4th ed, Washington AIA Press, 2001.
[4] J.M. Kamara, C.J. Anumba, N.F.O. Evbuomwan, Capturing Client Requirements in Construction Projects, Thomas Telford, 2002.
[5] I. Ozkaya, Ö. Akin, Electronic requirements management: a framework for requirement engineering for iterative design, EIA9: E-activities and Intelligent, Support in Design and the Built Environment 9th EuropIA International ConferenceIstanbul, Turkey 8–10 October, 2003, pp. 173–183.
[6] I. Ozkaya, Ö. Akin, Requirement-driven design: assistance for information traceability in design computing, Design Studies Vol 27 (No 3) (2006) May 2006.
[7] I. Ozkaya, Ö. Akin, Requirement traceability in collaborative design environments, Collaboration in Design Journal (2005) September 2005.
[8] LEED (2002) Leadership in energy and environmental design green building rating system for new construction & major renovations (LEED). Version 2.1 https://www.usgbc.org/Docs/LEEDdocs/LEED_RS_v2-1.pdf Last viewed Feb. 17, 2006.
[9] Eclipse (2006). http://www.eclipse.org/. Last viewed on May 2006.
[10] Computer-aided software engineering http://en.wikipedia.org/wiki/CASE_tool Last viewed on October 31, 2006.
[11] R. Amor, I. Faraj, Misconceptions About Integrated Project Databases Vol. 6 (2001) 57–68 ITcon journal, http://www.itcon.org/2001/5/, ISSN 1400–6529.
[12] Donia M (1998), Computational Modeling of Design Requirements for Buildings, Ph.D. Dissertation, Department of Architecture, Carnegie Mellon University, Pittsburgh, PA.
[13] H.I. Erhan, U. Flemming, User–system interaction design for requirements modeling, Proceedings of 10th Annual Conference of CAADRIA 2005. New Delhi, India, 2005.
[14] Kiviniemi, A. (2005), Requirements Management Interface to Building Product Models, PhD Dissertation, Department of Civil and Environmental Engineering, Stanford University.
[15] D. Norman, The design of every things,, Currency (1988) (New York).
[16] Hugh Beyer, Karen Holtzblatt, Apprenticing with the customer: a collaborative approach to requirements definition, Communications of the ACM (1995) May 1995.
[17] J.H. Garrett Jr., Applying knowledge-based expert system techniques to standards representation and usage, Building and Environment Journal Vol 25 (No. 3) (1990) 241–251.