Advanced Engineering Informatics xxx (2008) xxx–xxx

Contents lists available at ScienceDirect

Advanced Engineering Informatics

journal homepage: www.elsevier.com/locate/aei

# Qualitative design support for engineering and architecture

C.P.L. Schultz [a,*], R. Amor [a], B. Lobb [b], H.W. Guesgen [c]

[a] Department of Computer Science, The University of Auckland, Private Bag 92019, Auckland, New Zealand
[b] Department of Psychology, The University of Auckland, Private Bag 92019, Auckland, New Zealand
[c] School of Engineering and Advanced Technology, Massey University, Private Bag 11222, Palmerston North, New Zealand

ARTICLE INFO

*Article history:*
Received 1 November 2007
Received in revised form 3 July 2008
Accepted 9 July 2008
Available online xxxx

*Keywords:*
Qualitative Spatial reasoning
Qualitative temporal reasoning
Engineering and architecture design support

ABSTRACT

Conventional design support software tools cannot effectively manage the complex, heterogeneous information used in engineering and architecture (EA) tasks. Crucially, despite uncertainty being an inherent quality of EA information particularly in the early stages of a design project, current tools solely rely on numerical approaches which do not support such incomplete and vague information. In this paper, we establish a complete framework for developing qualitative support tools that directly address these shortcomings. Our framework is application oriented and addresses the broader issues surrounding the actual use of qualitative methods. It provides design principles and strategies that allow a software engineer to develop custom qualitative software tools according to their specific EA task specifications. Our framework also provides the engineer with practical theory and guidelines for implementing their custom qualitative model and validating their system using context specific test data. We demonstrate the validity of our framework by presenting a case study in architectural lighting in which a prototype qualitative reasoning engine successfully automates qualitative logic about the subjective impressions of a lighting installation.



## 1. Introduction

The disciplines of engineering and architecture (EA) involve more than simply meeting well-defined technical specifications. The information that EA practitioners must deal with is extremely complicated, not only in terms of volume, but in terms of the heterogeneity of the information, and the range of abstraction levels involved. The design process itself naturally involves incomplete information and an engineer or architect may only have general and vague guidelines from which to develop initial ideas and designs. For example, as an architect is designing a building, the exact materials may not be specified, certain dimensions may not be available, the client may still be deciding on whether or not to have an open-plan living space, and so on. Furthermore, the design process involves information that is inherently uncertain and subjective, such as a client requiring the impression or atmosphere of an environment to be dramatic and sophisticated. This information is not only ambiguous, but relies heavily on an individual's aesthetic disposition. Moreover, while official performance-based codes now offer the architect more freedom in the design process, verifying that goal-oriented specifications have been met is more difficult compared to prescriptive codes [1].

Despite this enormous variety of information present within EA, software tools are often very limited in the type of information that they can process, and, crucially, do not support engineers and architects when reasoning about vague and incomplete information. The central issue is that software tools rely solely on numerical approaches for modeling and reasoning about spatial and temporal phenomena, and cannot directly implement and process more qualitative information. For example, the focus of many tools has been on providing computationally expensive simulations such as ray tracing to render or visualise an environment or to calculate luminance distributions across a space, e.g. [2]. Three significant problems arise from using numerical approaches in this context.

First, numerical methods cannot effectively deal with incomplete information. Even if information about the uncertainty of attribute values is available, such as range restrictions or probability distributions, the complex interaction of many numerical formulae can make the resulting value estimates unpredictably sensitive to small changes in the initial parameters. For example, ray tracing simulation results can be irrelevant if the architect is uncertain about the building materials or physical dimensions of the room; they may decide to only slightly extend the length of a wall, causing it to occlude light from entering a large portion of the room and thus completely change the luminance distribution.

Second, numerical methods often employ an inappropriate level of precision. Many simulators use a high level of precision and

* Corresponding author.
*E-mail address:* carl.schultz@gmail.com (C.P.L. Schultz).

1474-0346/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.aei.2008.07.003

complexity, resulting in very computationally expensive processing that may take minutes or even hours to complete, thus preventing the architect from simply experimenting with lots of different subtle design variations. Especially for the early stages of a design, the architect may spend a considerable amount of time executing very precise simulations, whereas only rough, approximate and general information supplied by a simpler processing method is required. Alternatively, despite the significant resources taken to execute a simulation, the level of modelling precision can obviously never be sufficient to guarantee complete real-world accuracy. Engineers and architects are therefore interested in the relative differences between the performance of various design options, rather than absolute simulation results. A more efficient design support approach is required that directly reasons about relative information.

Lastly, numerical methods have even more fundamental limitations in their ability to model vague and subjective information. The problem is that many concepts within architecture that involve vagueness and ambiguity cannot be immediately expressed in numerical units. For instance, a numerical description of *dramatic* or *tense* impressions of a space would be meaningless as the notion of drama cannot be captured by relationships between numerical units.

In response to these limitations, computer scientists have developed alternative reasoning methods used to represent and reason about vague and incomplete information. Rather than the numerical bottom-up approach to modelling, where fundamental numerical units are composed to realise higher-level concepts, these methods use models built from qualitative values that take a top-down approach, by incrementally introducing more refined qualitative concepts into the model until the exact distinctions necessary to accomplish the task within the domain have been made [3]. For example, rather than defining distance as a quantity of unit regions such as metres, the architect considers the impact that different distances have on the task. If the task requires a significantly different result when two objects are considered *near* as opposed to *far*, then the architect will incorporate exactly this distinction in the model. Moreover, qualitative methods offer a more computationally efficient and human-intuitive approach to working with information, as they explicitly represent causality, the nature of interaction and everyday terms that automatically capture imprecision and vagueness [4] such as *very bright* and *fairly dim* compared to "356 lux".

In this paper, we establish a framework that supports a software engineer in developing qualitative design support tools for EA applications. Such software tools can assist engineers and architects during the design process by supporting higher-level concepts and reasoning about information that is not immediately expressed in units. Our framework addresses the broader issues of design, implementation and validation that surround the actual development and use of qualitative EA software tools.

This paper is organised as follows. Section 2 presents an overview of the qualitative spatial and temporal reasoning field. Section 3 presents our framework for developing qualitative design support tools, covering design, implementation and validation. Section 4 presents a prototype architectural lighting software tool that we have developed that demonstrates the effectiveness of our framework. Section 5 presents the conclusions of this paper.

## 2. Qualitative spatial and temporal reasoning

People reason very effectively with qualitative information to accomplish a wide variety of tasks. This has led to the development of a field in artificial intelligence called qualitative reasoning [4,5] that provides methods for reasoning with coarse and uncertain information about physical phenomena. The intention of qualitative information is to only describe the key aspects that are relevant to the task at hand. This paper focuses on two important subfields called qualitative spatial and qualitative temporal reasoning (QSTR), as architectural design tasks naturally involve spatial and temporal considerations, such as the physical layout of an environment, functional support and psychological effects based on a spatial configuration, change within an environment, and perceived flow as a person moves through a space.

Methods for qualitative temporal reasoning have been developed over the past three decades [6] that allow software applications to manage coarse-grained causality, action and change (e.g. [7]). A notable and highly influential example is Allen's interval calculus [8], which defines a set of thirteen atomic relations between time intervals; Fig. 1 (left) illustrates a subset of these. Allen provides an algorithm for reasoning about networks of relations. The algorithm is based on a composition table containing the possible temporal relations that can exist between the intervals $t1$ and $t3$ given relations for ($t1$, $t2$) and relations for ($t2$, $t3$). Two examples follow, where is the composition operator:

$t1$ *before* $t2 \cdot t2$ *contains* $t3 = t1$ *before* $t3$.
$t1$ *overlaps* $t2 \cdot t2$ *during* $t3 = t1$ (*overlaps*, or *during*, or *starts*) $t3$.

Allen's interval calculus has motivated a number of methods for reasoning about spatial objects and relationships in the area of qualitative spatial reasoning [10]. Guesgen [9] introduces a cognitively motivated one-dimensional spatial logic directly based on Allen's original temporal logic. The central idea is to represent relative spatial relationships between objects rather than using absolute object positions. Fig. 1 illustrates an extract of the basic atomic relationships that are defined. Guesgen provides a composition table and constraint satisfaction algorithm for constructing locally consistent networks of spatial relationships. Guesgen then extends this method to reason about higher spatial dimensions by using an $n$-tuple of spatial relationships between each pair of objects, where each component of the tuple represents a different dimension of the modelled scene. For example, the three dimensional scene illustrated in Fig. 2 can be described with the spatial relations below, if each component of the tuple represents the $x$, $y$ and $z$ axes, respectively:

$O1 <$ *inside*, *attached to*, *inside* $> O2$.
$O2 <$ *left of*, *inside*, *overlapping* $> O3$.

The reasoning method then infers the possible relationships that can hold between objects $O1$ and $O3$ by applying the composition table to the relation components.

A significant amount of research has focused on analysing the computational properties of the algorithms used to reason about qualitative information. It has been shown that the (complete) deductive closure algorithms inspired by Allen's temporal calculus are NP complete [11] and thus in the worst case intractable. However, the exact restrictions which can be applied to Allen's approach have been identified in order to remain in the tractable portion of the problem domain, requiring algorithms with only polynomial complexity [12].

Other research focuses on combining and generalising these approaches to provide a unified and systematic qualitative treatment of spatial and temporal information [13–16], although no single QSTR method has proven to be fundamental and generally applicable to a wide variety of tasks. The QSTR research community has now developed an off-the-shelf software package called the SparQ toolbox [13]. It provides a library of ready-to-use QSTR systems allowing a software developer to easily integrate these qualitative methods into their own software applications.

Please cite this article in press as: C.P.L. Schultz et al., Qualitative design support for engineering and architecture, Adv. Eng. Informat. (2008), doi:10.1016/j.aei.2008.07.003

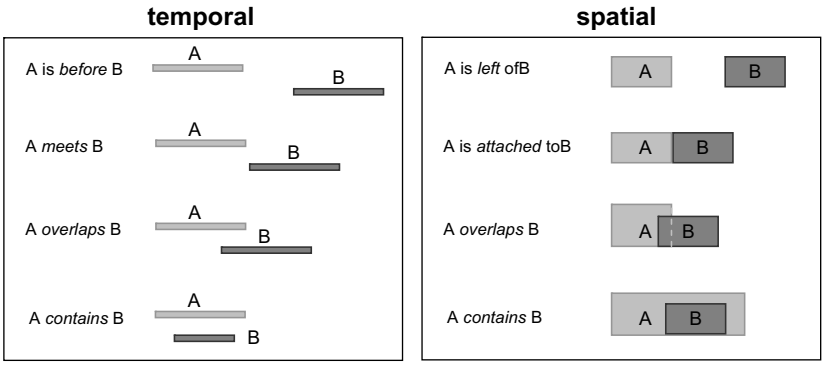


**Fig. 1.** An extract of Allen's [8] qualitative relations between temporal intervals (left) and an extract of Guesgen's [9] one-dimensional qualitative spatial relations between two objects.

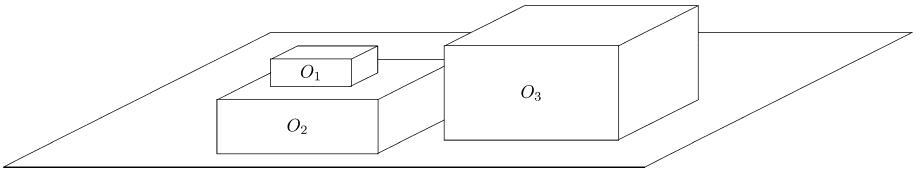


**Fig. 2.** The relative orientations of the blocks in this three-dimensional scene can be expressed in Guesgen's spatial reasoning method by assessing each dimension independently, and then combining the results (reproduced from [9]).

However, there are fundamental restrictions on QSTR applicability when a developer is limited to using pre-made methods. Foremost is that, in many cases, an existing QSTR method will not directly meet the needs of a task due to two properties of qualitative models:

(1) *highly context specific*: Qualitative terms are vague, and rely on context for disambiguation and meaning e.g. *near* can be a function of financial cost, time travelled or distance covered, and can vary depending on the user being a pedestrian, a car owner or a city planner.
(2) *highly task specific*: Qualitative models apply the exact degree of concept refinement necessary, where it is necessary, in order to address the task. It is therefore difficult to gauge the usefulness of a particular qualitative model component apriori without knowing the requirements of the task.

Thus, qualitative formalisms often need to be extended and modified by adding and removing qualitative relations, changing some part of an inference rule or creating entirely new custom QSTR systems. Unfortunately, when a developer modifies one of the established and well-understood formalisms, such as Allen's temporal calculus, they cannot easily identify the impact that this change will have on the computational properties of the method [15]. We are therefore taking a broader approach to QSTR application by providing a framework that supports engineers in developing completely custom QSTR methods and then integrating these methods into software tools.

## 3. Framework for developing qualitative engineering and architecture tools

Our framework supports an engineer in designing, implementing and validating general QSTR systems that are customised for specific EA task requirements. Fig. 3 illustrates an overview. It must be noted that our framework is not software based (i.e. it does not consist of code fragments) but is instead a methodology of developing QSTR systems from specific task information. In particular it details the necessary processes involved in the development lifecycle of a QSTR-based software tool and analyses effective practices that engineers can use in each lifecycle phase.

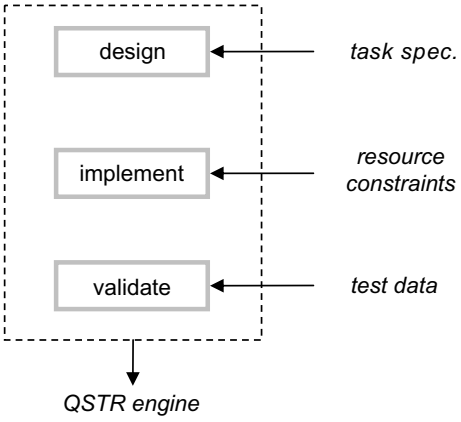


**Fig. 3.** Outline of our framework for supporting the development of QSTR-based architecture tools. Solid boxes within the dotted box represent key framework processes, and italicised labels with arrows represent external components that function as framework input and output.

The design process supports engineers in formulating a customised QSTR engine according to task information and specifications. The implementation process supports the engineer in producing software versions of their QSTR design along with principles and strategies for modifying the design to accommodate memory and execution time constraints. The validation process supports the evaluation of QSTR methods in relation to the intended application environment by applying context specific test data. The framework makes no assumptions about the ordering of the processes or the engineer's software lifecycle model employed. Each process in the framework consists of principles that are used to drive practical strategies for applying QSTR. The following sections present the framework's design principles and practices, implementation principles and a validation system.

Please cite this article in press as: C.P.L. Schultz et al., Qualitative design support for engineering and architecture, Adv. Eng. Informat. (2008), doi:10.1016/j.aei.2008.07.003

### 3.1. Design principles

The framework provides design principles and strategies in order to equip an engineer with the necessary information to design a QSTR-based software application that addresses their task requirements. The following sections describe our four fundamental principles of designing qualitative models that provide a foundation for developing custom qualitative modelling systems.

#### 3.1.1. Qualitative modelling

The first principle relates to modelling using qualitative information. Qualitative approaches require the definition of "qualities" used to describe objects and their relationships. We identify the way in which a quality describes the world by considering the term's use in everyday language: a "quality" is an inherent or distinguishing property [17]. From this we determine that, first, qualitative models consist of objects and qualities (i.e. qualitative relations), and secondly that the primary function of a quality is to provide a distinction between the objects that are being modelled.

We now consider how a quality can provide a distinction between objects. To identify the essential characteristics of qualitative modelling we restrict ourselves to the simplest possible type of distinction that a quality can provide, i.e. a single point of difference where the quality either (a) holds for an object in the model, or (b) does not hold.

We can also introduce qualities into our model that are not applicable for all modelled objects due to preconditions e.g. a light source is *occluded* from a surface if an obstacle interferes with the light beam; however if the light source is not *directed at* the surface, then the quality of *occluded* can be neither true nor false, but simply not applicable.

From this we derive a basic principle of qualitative modelling:

**Modelling principle**. *'Qualities" are distinctions between objects based on whether the quality holds for the object, does not hold, or is not applicable.*

This concept naturally maps to mathematical structures thus providing a formal definition for analysis. Each quality can be formalised as a predicate in first-order logic (FOL), where predicate arguments represent objects being modelled. Predicates evaluate to true (holds), false (not holds) or null (not applicable) depending on the argument variables given. The underlying definition of a qualitative model is classical set theory, where each quality is represented by three mutually exclusive crisp sets representing that a quality holds, does not hold or is not applicable respectively, and all objects in a model must appear in exactly one of these sets.

#### 3.1.2. Qualitative reasoning

In most design situations there will be parts of the scenario that the architect is modelling that are indefinite or unknown, i.e. some part of the architect's model will be incomplete. This represents the information that the software application will calculate or derive through reasoning in order to support the architectural task at hand.

In general, reasoning applies domain constraints to the available information to infer plausible values for the incomplete fragments of the model. In the context of qualitative modelling, domain constraints specify relationships between qualities, so that, given a certain combination of quality values (holding and not holding) for an object, the constraint specifies which further qualities must hold or must not hold for that object. Reasoning then exploits these constraints when given premise information, thus inferring whether or not other qualities hold.

Following is an example of a domain constraint between time intervals *a*, *b* and *c*:

> "if *a* happened *before b*, and *c* happened *during b*,
> then *a* must also have happened *before c*"

Fig. 4(left) illustrates a schedule for which this rule holds. This qualitative inference rule can be expressed using FOL, where predicates represent qualitative relations, and $I$ is the set of time intervals:

$$\forall x, y, z \in \mathbf{I} \cdot before(x, y) \wedge during(z, y) \rightarrow before(x, z).$$

The underlying set theoretic implementation of the inference rule is expressed as follows, where $before_y$, $during_y$ and $before_z$ are sets of time intervals:

$$\forall y \in \boldsymbol{I} \cdot before_{\mathbf{y}} \subseteq \cap_{\mathbf{z} \in during\,\mathbf{y}} before_{\mathbf{z}}.$$

To make the above expression evaluate to true, the set $before_y$ must be a subset of each $before_z$ set, for all $z$ in the set $during_y$. Fig. 4(right) illustrates the underlying set theoretic implementation of the qualitative schedule. The above inference rule is satisfied, as the sets $before_b$ and $before_c$ only contain a, and are thus equal (where $c$ is the only time interval that is in the $during_b$ set).

Inference rules can also refer to qualities at different levels of abstraction, for example:

> "if a *room* has *warm, bright ambient illumination* then it will evoke a *relaxing* impression"

Again, this inference rule can be expressed using FOL, where $R$ is a set of room objects:

$$\forall r \in R \cdot warm\_ambient\_illumination(r) \wedge bright\_ambient\_illumination(r) \rightarrow relaxing(r).$$

The corresponding set theoretic implementation of the rule is expressed as follows, where each term (e.g. *warm_ambient_illumination*) is a set of room objects:

$$warm\_ambient\_illumination \cap bright\_ambient\_illumination \subseteq relaxing.$$

Fig. 5 illustrates a set theoretic implementation of a qualitative model of the scene where $r$ is a room object. Note that, because the subset constraint is applied rather than equality, the above expression does not restrict the addition of further rules containing

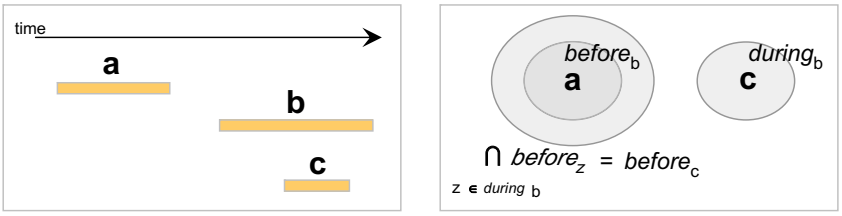


**Fig. 4.** *a*, *b*, and *c* are time intervals. (left) Schedule illustrates that *a* is before *c*, given that *a* is before *b*, and *c* is during *b*. (right) Set theoretic implementation where *c* is in the set $during_b$, and *a* is in the sets $before_b$ and $before_c$ (satisfying the inference rule).

Please cite this article in press as: C.P.L. Schultz et al., Qualitative design support for engineering and architecture, Adv. Eng. Informat. (2008), doi:10.1016/j.aei.2008.07.003

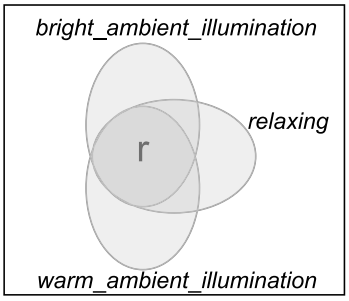


**Fig. 5.** Venn diagram illustrating the qualitative model of a scene with room object *r*, implemented in set theory where *bright_ambient_illumination*, *warm_ambient_illumination*, and *relaxing* are sets of rooms. The inference rule stating that all warm, bright rooms are also relaxing rooms is satisfied.

different, even contradictory criteria for rooms to be considered relaxing.

**Reasoning principle.** *The basis of pure QSTR reasoning is the domain constraints between qualities specifying that certain qualities must hold, and must not hold, for the same object at the same time.*

*3.1.3. Purely qualitative tasks*

Given the above principles for qualitative modelling and reasoning we can determine the type of tasks QSTR methods perform. For this we make two observations. First, applying a quality to a set of objects forms object classes or categories based on whether the quality holds, does not hold, or is not applicable. Secondly, applying a combination of qualities to a set of objects also forms classes. For example a qualitative constraint provided by a client states that "rooms near the kitchen that overlook the lake are idyllic"; this defines the class of *idyllic*rooms based on the combination of qualities *near* the kitchen, and *overlooking* the lake.

The engineer can only introduce qualities or combine qualities in the qualitative model, thus:

**Task principle.** *All pure QSTR tasks are a process of object classification.*

That is, QSTR methods are used to solve tasks by determining the class of certain objects in the model. This provides the extent to which QSTR can be used to address a problem. For example, reasoning to infer information about an incomplete model is the process of specifying the classes that the indefinite objects belong to. Querying, i.e. isolating some relevant parts of the model, requires the user to provide a class description (in the form of qualities holding and not holding) and returns objects that meet the given class criteria.

It must be noted that qualitative and numerical systems do not need to be completely disconnected. Indeed a very common task in the QSTR field is to determine consistent numerical instantiations of a given qualitative model. Numerical interpretations of qualities can be defined, for example, associating the numerical range "100 l× to 300 l×" to *bright* ambient illumination. The instantiation task is to then find suitable numerical values for each variable that are consistent with the qualitative model. This helps to bridge the gap between numerical and qualitative values within a design task, thus relieving the architect from tedious numerical interpretation and allowing the focus to remain on the design objectives. In our framework this can be understood more generally as finding a consistent interpretation of a given qualitative model in some other knowledge representation system (which may be, for example, a numerical system, a fuzzy system or even another qualitative system). A significant benefit of this hybrid design support is that the high level logic can remain unchanged while the numerical mappings are adjusted according to user preferences or variations in the environment. A further advantage is that the qualitative reasoning can be verified in terms of the numerical definitions. However, this is not a pure QSTR task as it involves other information representation systems outside of the qualitative model, and must be managed using different mechanisms (e.g. constraint satisfaction techniques [18]; for specific examples refer to [19] Section 3.1).

*3.1.4. Modelling application behaviour*

The final principle places QSTR systems in the context of software applications. In general, the architect uses models to represent salient aspects of a problem, and by reasoning about the model the architect can acquire a useful problem solution. QSTR performs tasks by applying qualitative models and qualitative reasoning.

Fig. 6 illustrates a process flow diagram describing the behaviour of QSTR during the execution of a software application (we have defined this formally as an automaton in [20]). The process of reasoning may develop or refine the model e.g. by inferring new information based on the given premises (model development). Alternatively reasoning may provide a model querying mechanism, where interesting parts of the model are isolated based on query criteria (model querying).

For example, an architecture tool may allow the client to describe certain qualitative parameters of a museum exhibition during the early stages of design and get qualitative feedback on the likely subjective impressions of the participants. The client describes the lighting configuration ("dim lighting at the entrance", "a bright band of light along the walkway floor with dim lighting along the perimeter") and the spatial configuration of exhibition pieces ("place a large sculpture in the centre of the foyer"). The architect then installs specific qualitative domain constraints according to scientific studies along with client preferences and information based on museum participant surveys from previous exhibitions. The software application processes the qualitative

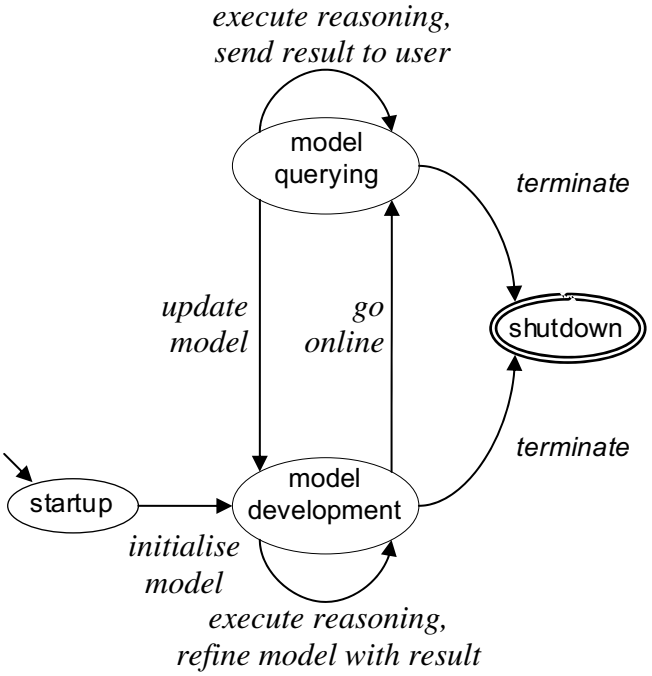


**Fig. 6.** QSTR behaviour during the execution of the software application being developed. Circles represent states, arrows represent possible state transitions, and the arrow annotations describe the effect of the transition on the QSTR system.

Please cite this article in press as: C.P.L. Schultz et al., Qualitative design support for engineering and architecture, Adv. Eng. Informat. (2008), doi:10.1016/j.aei.2008.07.003

model, applies qualitative reasoning and presents the findings, e.g. "the foyer will appear dramatic and tense", "participants will be drawn through the foyer into station 1". The software application then closes. In the process flow diagram (Fig. 6) the client and architect initialise the qualitative model, causing the application to transition from the startup state to model development. The QSTR component applies the domain constraints during reasoning to refine the qualitative model by inferring the subjective impressions, repeatedly looping back to the model development state. Once reasoning has finished the system transitions to the shutdown state.

**Application behaviour principle**. *Properties of the software application's behaviour that impact the design of the QSTR system can be defined based on the process flow diagram in Fig. 6.*

That is, by using this process flow diagram the software engineer can begin to characterise their QSTR system by explicitly incorporating task requirements into the behaviour of the software application.

### 3.2. Design practice

The design practice component of our framework applies the previous principles to motivate strategies for designing a QSTR system based on specific EA task requirements. This section gives an outline of how the design principles can be used to drive design.

#### 3.2.1. QSTR applicability

First, the software engineer must decide whether QSTR is applicable to the architecture problem at hand. The task principle (Section 3.1.3) identifies the limit of a qualitative approach, i.e. if the required task cannot be accomplished by a form of object classification then a qualitative modelling system will not be of any assistance to the architect. Thus the engineer must decide whether the problem can be solved by classifying objects. The three most common tasks are:

- *Reasoning:* Inferring information about incomplete models.
- *Consistency* checking: Ensuring that the model is consistent with domain constraints.
- *Querying:* Isolating interesting fragments of a model.

#### 3.2.2. Characterising application behaviour

The engineer must then characterise the behaviour of their software application by using the process flow diagram described in Section 3.1.4, as system behaviour greatly influences the design of the QSTR model. For example, if the QSTR system must allow substantial querying flexibility then the engineer must include independent qualities (i.e. qualities that are not related by constraints). This is because independent qualities, while being ineffective for reasoning about incomplete models, can be combined in many interesting ways thus providing a richer query language.

Two further examples of task properties that can influence the QSTR system design are as follows:

- *Model lifetime state:* Once initialised the model may never change, or it may change either monotonically or non-monotonically. This can affect the way the engineer designs the interaction between model components e.g. they should loosely couple dynamic and static components to avoid the application having to update the entire model when a change occurs.
- *Model stability:* An unstable model changes frequently, whereas a stable model does not. This can affect the way the engineer organises the model components. For example, the application should invest additional time and effort organising stable model components during initialisation (e.g. sorting, clustering and so on) to improve runtime efficiency, i.e. incur an initial overhead cost to improve all future queries. However, unstable components may change too frequently to justify the additionally time used for organisation.

#### 3.2.3. Qualitative model construction

The engineer must then decide on the objects and qualities that will be modelled in the architecture software tool. Following from the qualitative modelling principle (Section 3.1.1) the engineer should only include a quality if the distinction the quality provides is relevant for the architecture task at hand. As a guide, the engineer should review the task specifications for the expected typical use of their software tool, particularly the likely premise information (or application input) and the desired reasoning output.

While the core of a qualitative concept is well-defined, the boundary between neighbouring concepts may be vague or incomplete [21–23]. If the design task only requires coarse models to approximate a scenario then the engineer can simply introduce further qualitative distinctions closer to the boundary e.g. *moderately bright*. In other cases the engineer may need to directly model this vagueness by referring to the membership that an object has in a quality. Membership represents the nature in which a quality holds, and can be applied using a plethora of sophisticated techniques such as fuzzy logic and rough sets. In these cases the engineer must modify the reasoning operators to work with the membership information. These extensions, however, require more input information such as membership functions and probability distributions, and have more complex processing algorithms.

Note that the fundamental problem of constructing a QSTR model is to define appropriate distinguishing qualities, and to constrain these so that reasoning accurately performs a useful qualitative task. In particular, the engineer's aim is not to model concepts accurately in general, but to construct a model that performs a useful qualitative task as described in Section 3.2.1. For example, rather than constructing a definitive model of nearness based on first principles, the engineer only considers qualities and constraints that accommodate the relevant effect of nearness. From this viewpoint, when an engineer analyses the effectiveness of their QSTR model in terms of how useful it is (e.g. as a guide during design) the semantics of the model components are arbitrary and thus irrelevant (QSTR model validation is discussed in Section 3.4). This is analogous to the semantics of messages being irrelevant to the problem of transmitting a message accurately in Shannon's theory of communication [24]. To bridge this gap, the engineer can map the QSTR model to a suitable ontology. Darlington and Culley [25] provide a framework for supporting engineers in developing suitable ontologies to capture concept semantics involved in a given task. Constructing a QSTR model is one possible implementation of such an ontology.

#### 3.2.4. Qualitative reasoning support

Finally, the engineer must specify the domain constraints that will be used during reasoning. Domain constraints link the input of the model to the output, and may require further intermediate qualities. The engineer should review suitable sources of domain information such as scientific studies, industry consensus, expert knowledge and client preferences (i.e. the intended users of the application being developed).

A great advantage of qualitative reasoning is that, given only vague or incomplete qualitative information at any of the represented levels of abstraction, a qualitative approach will infer as much as possible according to distinctions within the model. In contrast, a numerical approach requires information to be

Please cite this article in press as: C.P.L. Schultz et al., Qualitative design support for engineering and architecture, Adv. Eng. Informat. (2008), doi:10.1016/j.aei.2008.07.003

expressed as unit values, and processing capability is limited if the quantities are not available.

### 3.3. Implementation

Once the engineer has produced a suitable QSTR system design, they must then implement the application in software. The software must represent objects, qualities and scenarios, where a scenario requires representing that each quality either holds, does not hold, is not applicable, or is indefinite for each object (refer to the modelling principle in Section3.1.1). The engineer may also need to implement a reasoning algorithm that can process an incomplete scenario or provide a querying mechanism. This section describes issues relating to the software implementation of qualitative design support tools.

#### 3.3.1. Platforms

Different implementation platforms provide a tradeoff between ease of implementation and control. For example, the FOL specification of qualities and reasoning constraints (as described in the Section 3.1.1) can be implemented using general purpose declarative languages such as Prolog [26]. These languages provide an inbuilt query engine that the engineer can use for their application's qualitative reasoning. Alternatively, the underlying set theoretical basis for qualitative systems (described in Section 3.1.1) leads to a natural implementation in languages that also have a set theoretic definition such as relational databases [27]. The engineer can use these languages by implementing qualities as relations (or tables) and quality values for objects as relation tuples (or table entries). This requires expertise in areas such as database query optimisation and data storage.

#### 3.3.2. Resource constraints

The qualitative model may need to be modified to accommodate resource constraints, such as limitations on the available memory for data storage or the requirement that reasoning executes within a specified amount of time. To accomplish this, the engineer must reduce the amount of information used by the qualitative model when describing a scene. Note that a single datum of information in a qualitative model expresses the state of a single quality for a single object, e.g. expressing that "the kitchen is near the bathroom" in software requires one datum.

The key to reducing the amount of data used in the model is to implicitly maintain certain quality states in cases when they can be inferred by other explicitly represented quality states, i.e. the engineer must avoid expressing redundant information in the model by forming and relating clusters. For example, the quality *directed at* specifies whether a light source is pointing towards a surface and *occluded* specifies whether a light beam is blocked from a surface. The engineer can use these qualities in combination for determining how bright a surface will appear. However, if the light source and the surface are in different rooms then, in the vast majority of scenarios, the value of *occluded* will not be applicable and *directed at* will be irrelevant and never used for reasoning. Thus, without interfering with the QSTR logic that performs the architect's task, the engineer can introduce the quality *in same room*; if a light source and a surface are not together in a room then the value of *occluded* and *directed at* will not be explicitly represented in the model. This vastly reduces the amount of data needed to model the scenario, potentially by orders of magnitude.

In general, clustering introduces layers of intermediate qualitative information by grouping objects that share the value of qualities. The result is that quality values for each object in the cluster do not need to be explicitly enumerated. In terms of implementation, this clustering approach is a form of query pre-processing, where the common aspects of query results are stored and retrieved rather than recomputed.

Note that, from a design perspective, clustering is identical to classing objects using qualities. The difference is that the engineer's motivation for introducing a quality in the initial model design is to satisfy the functional requirements of the task (e.g. modelling the appropriate input and output information described in Section 3.2.3) whereas their motivation for clustering is to satisfy resource constraints.

#### 3.3.3. Implementing pure qualitative reasoning

As discussed previously, the engineer may decide to implement their own customised reasoning engine rather than using inbuilt querying mechanisms in languages such as Prolog. We now outline a deeper principle about pure qualitative reasoning to provide a method for implementation.

All pure QSTR tasks are a sequence of queries and model updates. In particular, qualitative inference (i.e. used for incomplete models) is an extension of a querying task, where the query result is fed back into the model. For example, four database operations are required to satisfy the following rule about temporal intervals (refer to Section 3.1.2):

$$before_{\mathbf{y}} \subseteq \cap_{\mathbf{z} \in during\,\mathbf{y}}\, before_{\mathbf{z}}$$

1. *Query:* Gather all elements in $before_y$.
2. *Query:* Gather all elements in $during_y$.
3. *Query:* Gather all *before* sets associated with each element from result (2).
4. *Update:* Add elements from (1) to each set from (3).

This is significant because, first, all pure QSTR tasks can be implemented as a sequence of queries and updates, and secondly, that query optimisation can greatly influence performance as querying underlies every pure QSTR task performed by a qualitative design support tool.

The aim of query optimisation, given a set of rules, is to determine the optimal rule execution sequence that minimises the number of executed rules in the worst case. Because the execution of a rule can modify the database, previously executed rules may need to be recomputed, i.e. reasoning is often non-linear. For further information the engineer can refer to the related problem of variable and value ordering in constraint satisfaction [28] (chapter 5). The next step is to perform a finer level of optimisation by interleaving inference rule execution to minimise the number of actual database queries required. For this, the engineer can refer to research in database optimisation [27].

The main issues the engineer must consider when implementing their reasoning engine are verifying that the rule base is consistent, or that any inconsistencies are appropriately managed (otherwise reasoning may never terminate), ensuring that interleaving the rules will not interfere with the intended logic, and finally, determining whether reasoning using the given rule base will be suitable in terms of computational complexity and at least identifying cases where the task is intractable.

### 3.4. Validation

The engineer will need confirm that their qualitative model design meets the needs of the EA task at hand. In this section we outline our framework's validation system.

The aim of program validation in general software engineering is to determine if the developed system is fit for purpose [29], explicitly evaluating the program in terms of its application context. QSTR model validation aims to provide a level of confidence that the system logic, specifically the qualities and

Please cite this article in press as: C.P.L. Schultz et al., Qualitative design support for engineering and architecture, Adv. Eng. Informat. (2008), doi:10.1016/j.aei.2008.07.003

constraints, provides the intended function, e.g. by ensuring that the important highly used or critical components have been adequately tested. Note that theorem proving for validation is not practical in general because, firstly, it requires axioms for the logic which in many cases will not be available and secondly, even expert logicians in the QSTR research field find the task non-trivial (e.g. refer to page 292 and Section 6.2 of [30]).

Fig. 7 illustrates the framework's validation system. The qualitative model and the test set are passed to the black-box and white-box testing methods which are described in the following sections. If the black-box pass rate or the white-box information content results do not meet target thresholds, then appropriate changes to the qualitative model must be made. If the white-box coverage results are not sufficient then the test set must be appropriately modified. The engineer's model is validated for the test set if all three results are satisfactory.

Test cases can be generated from the task specification for tool inputs and outputs, analysis of the qualitative model structure, field study data, and other qualitative research in the intended application domain (e.g. [31]).

*3.4.1. Assessing model pass rate*

The black-box approach to validation focuses on system inputs and outputs without considering its internal structure [29]. The engineer can use black-box testing to determine if the qualitative model is providing the intended function, and to assist in identifying any defective or missing functionality. This is accomplished by feeding the test case's premise information into a new scenario, executing the reasoning algorithm and comparing the resulting scenario to the test case's expected output. The qualitative model passes a test case if, after applying the reasoning algorithm, the final scenario meets the expected quality states. After executing a set of test cases, a report is compiled indicating which tests passed and which tests failed. As shown in Fig. 7, if the black-box pass rate is below a target threshold then the engineer must modify the qualitative model until the pass rate is acceptable.

The engineer can apply clustering methods [32] for identifying patterns in the group of failed tests to isolate faulty model components. Classes of failed tests are identified by clustering the tests according to either the similarity of failed test case inputs, similarity of test case expected outputs, or the similarity of the discrepancy between the qualitative model's actual output and expected output. Dense clusters indicate a specific faulty component in the model. Sparse clusters may result from ambient test noise.

*3.4.2. Assessing test set quality*

In standard software engineering, test coverage is a measure of the proportion of components exercised during testing [29]. The engineer can use coverage analysis to evaluate the suitability of the test set by determining how much of the qualitative model is actually assessed, and to guide test generation by identifying components that have not been adequately exercised. We have developed test coverage methods that are motivated by standard software engineering approaches used in program control flow graphs [29]. This is a white-box testing approach because the engineer takes the internal structure of the qualitative model into account during validation.

To determine test coverage the engineer must use a structure called the qualitative constraint graph (QCG) where vertices represent domain constraints and undirected edges represent a shared quality type between two constraints. The edges in the QCG indicate constraint dependencies; if the constraint $c1$ modifies a quality that appears in constraint $c2$ then $c2$ must be revisited.

For example, a selection of qualitative constraints for the subjective impressions from lighting in a room (based on [31]; Section 4 of this paper discusses Flynn's research in more detail) are as follows:

(1) If the room has cool light, bright even work-surface illumination and some perimeter emphasis then the room evokes clarity.
(2) If the room has uniform perimeter emphasis and bright even work-surface illumination then the room is spacious.
(3) If the room has bright, warm ambient illumination, and non-uniform perimeter emphasis then the room is relaxing.
(4) if the room has low ambient lighting in the occupancy area and a bright perimeter then the room is intimate.

The QCG for these constraints is illustrated in Fig. 8.

A vertex is executed in a qualitative scenario if the reasoning algorithm has modified the scenario in order to satisfy the vertex's constraint. An edge between vertices $v_1$ and $v_2$ with quality type $q$ is executed if $v_1$ is executed with the modification of a quality of type $q$, causing $v_2$ to execute (at some future time). Using these definitions we now adapt standard coverage measures for the QCG:

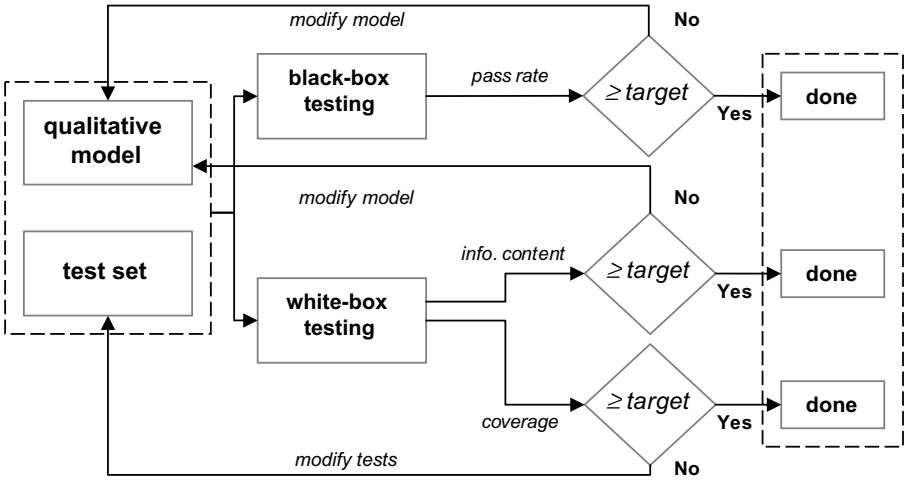


**Fig. 7.** Validation system assessing the logic of the qualitative model according to context specific test data. Solid boxes represent system components, dotted boxes represent system input and output, arrows represent information flow and diamonds represent decisions.

Please cite this article in press as: C.P.L. Schultz et al., Qualitative design support for engineering and architecture, Adv. Eng. Informat. (2008), doi:10.1016/j.aei.2008.07.003

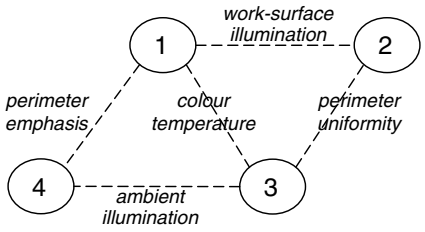


**Fig. 8.** Qualitative constraint graph (QCG) of a selection of domain constraints for architectural lighting [31], where vertices represent constraints and edges represent related qualities.

- Vertex coverage is the percentage of vertices executed in at least one test.
- Edge coverage is the percentage of edges executed in at least one test.
- *k*-Path coverage is the percentage of paths of length *k* executed in at least one test (for $k > 2$).

For example 100% vertex coverage means that reasoning used every domain constraint at least once to modify a scenario in some test.

*3.4.3. Model fine-tuning*

While coverage can be used to determine whether the test set is sufficient, alternatively the engineer may have a test set that statistically reflects the pattern of usage and criticality in the intended environment, i.e. the software application's operational profile. In this case, the engineer can use white-box testing to refine the qualitative model by removing redundant components. For example, in the test set for a town planning software tool the landfill is never *near* the central city region. Thus, reasoning about this quality is redundant and the engineer can remove it from the qualitative model.

Reasoning provides the end-user with data at every inferential step and, according to information theory [24], the amount of information gain is inversely proportional to prior knowledge about the inferred data. If the engineer is very confident about what data will be provided before the inference step has been taken (because the same result appeared in all test cases), then the inference step provides little or no information. Thus, the engineer can improve reasoning efficiency if they make this data an assumption (e.g. by setting it as a default rather than making the reasoning engine infer it) or remove the entire quality from the model.

Information content for a quality, based on entropy, can be calculated as follows. Given a quality $q$, for any object $q$ either holds ($v_q^+$) does not hold ($v_q^-$), is not applicable ($v_q^\sim$), or remains unknown ($v_q^?$). Given $n$ preconditions $v_{q1\ldots qn}$ the probability of the event $v_q^\alpha$ in test $t$ is

$$P_t(v_q^\alpha, v_{q1}, \ldots, v_{qn}) = \frac{|t(q^\alpha) \cap t(q_1) \cap \ldots \cap t(q_n)|}{|t(q_1) \cap \ldots \cap t(q_n)|,}$$

where $\alpha$ is one of $\{+,-,\sim,?\}$, the cardinality operator $|Y|$ returns the number of elements in set $Y$, $\cap$ is the standard set intersection operator and the function $t(q_n)$ returns the quality $q_n$ in the test case $t$. Following from equations in [24] the information content $I_q$ of knowing the object's state for quality $q$ is

$$I_q(v_q^+, v_q^-, v_q^\sim, v_q^?) = -\sum_{\alpha\in\{+,-,\sim,?\}} P(v^\alpha)\log_2 P(v^\alpha),$$

and is generalised by accepting the preconditions $v_{q1\ldots qn}$ and using the probability function $P_t$ above. As shown in Fig. 7, if the information content of a relation is below a target threshold then the engineer must modify the qualitative model by either fixing the relation state as a modelling assumption, or removing it from the model.

## 4. Architectural lighting case study

We have developed a qualitative design support tool for architectural lighting that validates the principles described in Section 3.1 by demonstrating that qualitative logic taken from scientific literature can be accurately automated in a software system. This prototype software tool assists an architect during the early stages of a project by analysing an electrical lighting installation and providing fast qualitative feedback on the subjective impressions that will be evoked. The tool accepts qualitative input describing building components and the lighting configuration, but can also accept numerical input (e.g. dimensions from a preliminary CAD design).

Lighting has both a functional component concerned with the ease and accuracy of visual perception and a subjective component [33]. In the lighting research community it is generally recognised [34–38] that important factors in lighting an environment include luminance, luminance distribution, uniformity and spectral power distribution, however a definition of lighting quality is still being debated [38]. It is now also clear that the lighting designer can achieve a broad range of subjective impressions, such as relaxation, excitement, intimacy, and spaciousness by varying aspects of the lighting installation while remaining within the practical health and safety guidelines established according to the task requirements [39].

Flynn has performed studies on the relationships between luminances, luminous patterns and subjective response [31,40]. Flynn aimed to establish basic guidelines on how to influence a range of non-visual effects with a lighting scheme, and identified the following five key impressions

(i) *Visual clarity (clear to hazy):* A person's subjective impression of how clearly or distinctly interior details, objects, and other peoples' features appear.
(ii) *Spaciousness (spacious to cramped):* Apparent volume of a space.
(iii) *Pleasantness (like to dislike):* Subjective evaluation of the lighting environment.
(iv) *Relaxation (relaxed to tense):* Apparent work intensity.
(v) *Intimacy (private to public):* Feeling of privacy in a space.

The above subjective responses are elicited by a number of intermediate qualitative lighting conditions as shown in Table 1. For example, to create a sense of relaxation the lighting designer could selectively place indirect luminaires around the periphery (e.g. wall sconces or accent lighting on wall art decorations) com-

**Table 1**
The lighting conditions (rows) required to elicit the desired subjective impression (columns)[a]

| | Clarity | Spaciousness | Relaxation | Intimacy | Pleasantness |
|---|---|---|---|---|---|
| Ambient illumination | | | ☑(bright) | ☑(low in occupancy area) | |
| Room colour temperature | ☑(cool) | | ☑(warm) | | ☑(warm) |
| Perimeter emphasis | ☑(some) | ☑(uniform) | ☑(nonuniform) | ☑(high brightness) | ☑ |
| Work surface illumination | ☑(bright, uniform) | ☑(bright, uniform, central) | | | |

[a] Based on research by Flynn [31], and adapted from [41] (pp. 61–72), and [42] (pp. 118–119).

Please cite this article in press as: C.P.L. Schultz et al., Qualitative design support for engineering and architecture, Adv. Eng. Informat. (2008), doi:10.1016/j.aei.2008.07.003

plemented with direct low intensity incandescent lamps placed over the area of occupancy [41].

Cuttle [39] proposes six central factors that influence a person's subjective impression of a lighting environment with the aim of supporting architectural design objectives in the creation of a lighting scheme:

(i) Ambient illumination in a space (bright to dim).
(ii) Illumination hierarchy (emphatic to none) that structures a space with varying degrees of emphasis, taking into account the subjective illuminance differences.
(iii) Flow of light through a space (dramatic to very weak) which strongly impacts on the object modelling quality.
(iv) Sharpness of light affecting surface highlights and shadowing.
(v) Visible presence of luminous elements, involving glare or sparkle.
(vi) Provision for visual performance, defined as the adequate discrimination of colour and detail.

We primarily base our work on research from Cuttle and Flynn.

### 4.1. Qualitative model design

This section formalises the qualitative rationale proposed by Cuttle [39] and Flynn [31] in a suitable manner for software automation. The task is to provide an analysis of the subjective reaction that a person will have to a given lighting installation. The interpretation of qualitative lighting concepts such as "bright uniform light across centrally located work surfaces, with some perimeter emphasis" requires the explanation of each qualitative component.

Table 2 presents a summary of the different qualitative measures. We selected and derived measures 1, 2, 3, 4, 5 and 11 based on the task specifications, and applied measures 6, 7, 8, 9 and 10 based on relevant architectural lighting research. Each measure is a quality representing either an object property or a relationship between a pair of objects.

Standard spatial relations for orientation, distance and topology are used to infer the state of more abstract relations such as uniform perimeter emphasis (Table 1). The key to defining these qualitative spatial relations is domain knowledge (as argued in Section 3.2.3); at an early stage of design, the basic initial decision to apply accent lighting to a particular art piece is far more important than details such as precisely orienting a light source to minimise spillage. The type of light that the architect has chosen (e.g. spot light versus diffuse light) is a qualitatively significant indicator of the architect's intended use of the light. Thus, the type of light provides a basis for approximating the light beam: the beam shape of a *directed* light source (such as a spot light or a small aimable light) is represented as a line, so that the source is only directed at surfaces to which it is purposefully aimed, regardless of beam width.

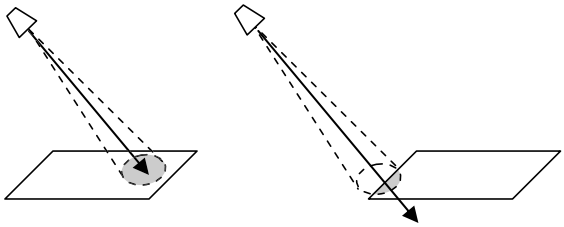

**Fig. 9.** Approximating the beam of a *directed* light source as a line shape, to qualitatively determine whether the source is directed *at* or *away from* a surface. If the line intersects the surface (left) the directed value is *at*, otherwise (right) the directed value is *away* regardless of beam width.

Alternatively, if a light source is *diffuse* then it is directed at every surface in the room (see Fig. 9).

We now describe the modelled qualities in more detail. Source direction indicates whether a light source is directed towards or away from a surface according to the geometry and other basic properties of the scenario. Beam intersection shape approximates the shape of the projected beam on a surface ignoring any occlusion that may occur. Source coverage indicates whether the projected beam area is significantly smaller than the area of the surface according to the qualitative source direction and beam intersection shape. Occlusion indicates whether a beam of light is obstructed from striking a surface, where the value *possibly occluded* means that more information is required and *not applicable* means that the source is not directed towards the surface. Layout indicates the region of the room in which an object is located by making a qualitative distinction between centrally located and perimeter objects based on a volume partition.

We now present qualities and domain constraints introduced into our model based on specific, pertinent research. We have also applied numerical mappings available in the literature. It must be noted that the system does not require numerical input and primarily accepts qualitative input directly. However, this task specification also requires that the architect can supply a combination of qualitative and detailed numerical input describing a lighting configuration (e.g. generated using computer aided design software).

Cuttle [39] defines mean room surface exitance $M_{rs}$ as an approximation of the average eye illuminance in a room by assuming that the lumens are uniformly distributed. In cases where room illuminance irregularity is an issue, Cuttle divides the room into sections and applies $M_{rs}$ to each individual volume, while also suggesting a qualitative approach for scenarios where this is obviously a concern [39]. $M_{rs}$ is calculated using the first-bounce lumens, called the first reflected flux (FRF), and the capacity of the surfaces in a room to absorb light, called the room absorption $A\alpha$,

$$M_{rs} = \mathrm{FRF}/A\alpha.$$

**Table 2**
Summary of the intermediate qualitative measures used by Cuttle [39] and Flynn [31] to infer higher level subjective impressions

| Measure | Type | Applicability | Values |
|---|---|---|---|
| 1. Source direction | Relation | Between source and surface | *at*, *away* |
| 2. Beam intersection geometry | Relation | Between source and surface | a 3D shape |
| 3. Source coverage | Relation | Between source and surface | *none, partial, full* |
| 4. Occlusion | Relation | Between source and surface | *none*, *partial, full, n/a* |
| 5. Layout | Property | Of sources and surfaces | *central*, *perimeter* |
| 6. Perceived illuminance difference | Relation | Between two illuminances | *none*, *noticeable*,*distinct*,*strong*,*emphatic* |
| 7. Colour temperature | Property | Of sources and rooms | *cool*, *intermediate*, *warm* |
| 8. Approx. surface illuminance | Property | Of a surface | a positive real |
| 9. Illuminance pattern | Property | Of surfaces and rooms | *uniform, non-uniform* |
| 10. Ambient illumination | Property | Of a room | *none, very dim, dim, acceptably bright, bright, distinctly bright* |
| 11. Perimeter emphasis | Property | Of a room | *none, some, lots* |

Please cite this article in press as: C.P.L. Schultz et al., Qualitative design support for engineering and architecture, Adv. Eng. Informat. (2008), doi:10.1016/j.aei.2008.07.003

For a room with $n$ surfaces, FRF and room absorption are calculated [39] using

$$\mathrm{FRF} = \sum_{s=1 \text{ to } n} E_{s(d)} A_s \rho_s,$$

$$A\alpha = \sum_{s=1 \text{ to } n} A_s (1 - \rho_s),$$

where $E_{s(d)}$ is the direct surface illuminance, $A_s$ is the area of $s$ and $\rho_s$ is the surface reflectance factor which takes a value between 0 and 1. The indirect illuminance of surface $s$ is approximate using [39]

$$E_s = E_{s(d)} + M_{\mathrm{rs}}.$$

Perceived illuminance difference indicates significant subjective variations in illuminance across a space. Cuttle [39] informally suggests illuminance ratios required to achieve qualitatively significant categories of perceived illuminance difference based on exercises conducted with students as shown in Table 3. The values are based on statistical observations and therefore represent average values rather than a precise numerical mapping to qualitative values.

Colour temperature indicates the apparent subjective temperature evoked by the colour of a room's ambient light. Many literature sources [34,37,39,43] agree on the qualitative impression of the correlated colour temperature values given in Table 4.

Illuminance pattern distinguishes between uniform and non-uniform illuminance across a surface and a room e.g. a bright localised region on a surface can cause non-uniform illuminance. An engineer can approximate uniformity by comparing the ratio between the minimum and average illuminances to a uniformity threshold. For example CIBSE Code (L05 (3). Lighting Legislation II) [35] gives a uniformity threshold of 1:1.25.

Ambient illumination indicates the perceived brightness of a space based on research in [44] that identifies threshold illuminance values correlating with the subjective assessment that people gave for the appearance of a room [39] as shown in Table 5.

Perimeter emphasis indicates whether the perimeter has enhanced lighting that provides visual clarity or draws attention to the edges of a room, such as wall wash lighting, accent lighting on art decorations, ornate wall mounted sconces, or well lit side tables.

### 4.2. Framework for applying qualitative spatial reasoning

Fig. 10 illustrates the flow of information in the developed qualitative design support tool. The architect firstly specifies a qualitative description of a building and lighting configuration (our prototype tool accepts a formatted script in Java code, however a practical application would use a graphical user interface). Additionally, the software tool can take numerical values directly from a building specification file in a format such as the Industry Foundation Classes (IFC) [45] model specification and derive salient qualitative characteristics using the methods described previously. It then stores the building and lighting installation model internally as surfaces and light sources with various qualitative (and some possibly numerical) properties such as dimensions, position and reflectance for surfaces, and beam intensity and beam angle for sources. The qualitative reasoning engine then applies the domain constraints described previously to derive as many qualitative relations and properties as possible, using the low level qualitative measures to determine the high level subjective impressions of the lighting scheme such as clarity or intimacy. It then produces a summary report of the analysis, specifying the inferred subjective responses along with the rationale behind the reasoning process.

**Table 3**
Illuminance ratios required to achieve the qualitative perceived difference, informally presented in [39] and based on exercises conducted with students

| Perceived difference | Illuminance ratio |
|---|---|
| Noticeable | 1.5:1 |
| Distinct | 3:1 |
| Strong | 10:1 |
| Emphatic | 40:1 |

**Table 4**
Correlated colour temperatures required to achieve the qualitative, thermally described impression [39]

| Colour appearance | CCT |
|---|---|
| Cool (bluish white) | ⩾5000 K |
| Intermediate (white) | <5000 K; ⩾3300 K |
| Warm (yellowish white) | <3300 K |

**Table 5**
Threshold eye illuminance values for qualitative ambient illumination appearance [39]

| Ambient illumination | Eye illuminance |
|---|---|
| Lowest level for reasonable colour discrimination | 10 lx |
| Dim appearance | 30 lx |
| Lowest level for 'acceptably bright' appearance | 100 lx |
| Bright appearance | 300 lx |
| Distinctly bright appearance | 1000 lx |

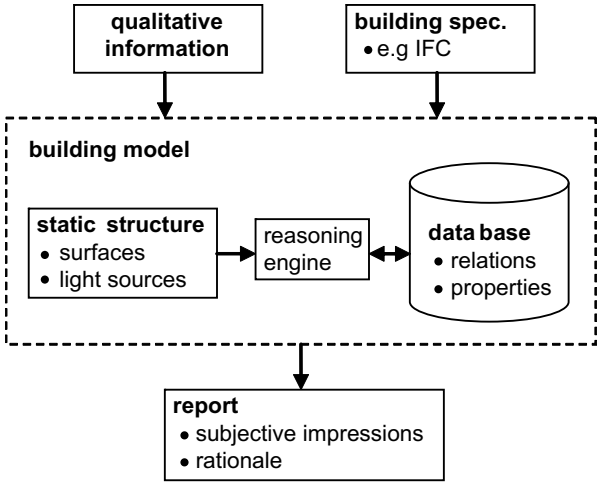


**Fig. 10.** Diagram indicating the flow of information (arrows) between the components (solid and dotted boxes) required to apply qualitative spatial reasoning to architectural lighting.

### 4.3. Experiments and results

The aim of the software tool is to accurately implement the qualitative logic prescribed by Flynn. Thus, we use the results of experiments conducted by Flynn [31,40] to validate the reasoning engine's applicability to lighting design. Note that we are not addressing the question of whether the logic actually captures the concepts of subjective impressions, as this can only be addressed by field psychologists, domain experts, researchers and so on. Instead, our thesis is that, given qualitative logic, an engineer can implement it accurately in a software system.

Flynn presents six different lighting conditions for a meeting room and a summary of the qualitative impressions reported by participants of the study. The study uses four different light sources in various combinations, which are overhead direct

**Table 6**
Six lighting conditions used in experiments by Flynn [31], the qualitative assessments that people gave during the study, and the qualitative assessment given by the prototype qualitative spatial reasoning (QSR) engine

| Cond | o/h direct | o/h indirect | p. indirect (fluor.) | p. indirect (incand.) | Study results | QSR engine analysis |
|---|---|---|---|---|---|---|
| 1 | ☑(low) | | | | Generally hazy, quiet; Strong confinement | *hazy, cramped* |
| 2 | | | ☑(low) | ☑(low) | Neutral clarity; Spacious | *clear, spacious* |
| 3 | | ☑(low) | | | Strongly hazy, quiet; Neutral spaciousness; Tense | *hazy, cramped tense* |
| 4 | ☑(low) | | | ☑(low) | Neutral clarity; Mostly neutral spaciousness; Relaxed | *hazy, cramped, relaxed* |
| 5 | | ☑(high) | | | Strong clarity; Somewhat spacious | *clear, spacious* |
| 6 | ☑(mod) | ☑(mod) | ☑(mod) | ☑(mod) | Strong clarity; Strong spaciousness | *clear, spacious* |

(incandescent), overhead indirect (fluorescent), and peripheral indirect (fluorescent on one wall and incandescent on another). Table 6 presents these conditions along with the derived subjective impressions from our qualitative reasoning engine. This is a black-box approach to validation as described in Section 3.4.1.

The reasoning engine correctly determines subjective impressions when strong responses are reported, that is (i) the clarity rating for conditions three, five, and six, (ii) the spaciousness rating for conditions one, two, and six, and (iii) the relaxation rating for conditions three and four. When the response is only moderate the engine still determines the correct qualitative value (e.g. clarity in condition one is generally hazy, which is qualitatively classed as simply hazy). When the response is neutral the engine does not respond in a consistent manner between different impressions (e.g. in condition two the neutral clarity rating is assessed as clear, whereas in condition three the neutral spaciousness rating is assessed as cramped). The engine requires a further neutral qualitative value to capture this intermediate case or a fuzzy logic approach where membership functions are provided to determine the degree to which a scene is considered clear or hazy. It must be noted that the results are completely deterministic, that is, identical scenarios will always result in an identical qualitative assessment by the qualitative spatial reasoning engine.

These preliminary prototype results are very promising as they demonstrate that the qualitative logic by Flynn can be automated in software and that qualitative design support is applicable to architectural tasks.

## 5. Conclusions

We have established a framework for developing qualitative design support tools for engineering and architecture (EA) applications. Unlike standard numerical systems, qualitative models maintain and reason about both incomplete and vague information very efficiently and thus can be used to drive powerful design tools. Our framework supports a software engineer in the custom design, implementation and validation of a qualitative spatial and temporal reasoning system by explicitly incorporating context specific task requirements. The framework provides design principles and strategies for qualitative modelling and reasoning, performing purely qualitative tasks and characterising the engineer's QSTR system by incorporating task requirements according to our process flow diagram. The framework provides support for implementing the qualitative system design in software with methods for modifying the qualitative model to accommodate task specific resource constraints and selecting the implementation platform. The framework also provides support for validating the engineer's qualitative model according to context specific test data with methods for assessing the model's black-box testing pass rate, assessing the test set's quality based on test coverage metrics and fine-tuning the model by measuring the information content of the model components. We have validated the framework's processes and principles by developing a qualitative design support tool for architectural lighting that demonstrates how an engineer can develop a software system that accurately automates scientifically based qualitative logic.

## Acknowledgement

This research has been funded by the Bright Future Top Achiever Doctoral Scholarship (Tertiary Education Commission, New Zealand). This work was presented in part at the CIB W78 conference on Bringing ITC Knowledge to Work, Maribor, Slovenia, 26–29 June.

## References

[1] G.C. Foliente, Developments in performance-based building codes and standards, Forest Products Journal 50 (7/8) (2000) 12–21.
[2] G.W. Larson, R.A. Shakespeare, Rendering with Radiance: The Art and Science of Lighting Visualization, Morgan Kaufmann, San Francisco, 1998.
[3] J. Renz, Qualitative Spatial Reasoning with Topological Information, Springer, Berlin, 2002.
[4] K.D. Forbus, Qualitative reasoning, in: A.B. Tucker (Ed.), CRC Handbook of Computer Science and Engineering, CRC Press, Boca Raton, FL, 1996, pp. 15–733.
[5] B. Kuipers, Qualitative Reasoning, MIT Press, 1994.
[6] A. Artale, E. Franconi, A Survey of Temporal Extensions of Description Logics, Annals of Mathematics and Artificial Intelligence, Springer, Netherlands 30 (1–4) (2000) 171–210.
[7] J.F. Allen, J.A. Koomen, Planning using a temporal world model, in: Proceedings of the 8th International Joint Conference on Artificial Intelligence, William Kaufmann, Karlsruhe, West Germany, 1983.
[8] J.F. Allen, Maintaining knowledge about temporal intervals, Communications of the ACM 26 (11) (1983) 832–843.
[9] Guesgen HW. Spatial reasoning based on Allen's temporal logic. Technical report TR-89-049, International Computer Science Institute, 1989.
[10] A. Cohn, S.M. Hazarika, Qualitative spatial representation and reasoning: An overview, Fundmenta Informaticae 46 (1–2) (2001) 2–32.
[11] M. Vilain, H. Kautz, P. van Beek, Constraint propagation algorithms for temporal reasoning: a revised report, Readings in Qualitative Reasoning about Physical Systems, Morgan Kaufmann, San Francisco, CA, USA, 1989.
[12] B. Nebel, H.J. Bürckert, Reasoning about temporal relations: a maximal tractable subclass of Allen's Interval Algebra, Journal of the ACM 42 (1) (1995) 43–66.
[13] J.O. Wallgrün, L. Frommberger, F. Dylla, D. Wolter. SparQ User Manual V0.6. No. 007-07/2006, SFB/TR 8 Spatial Cognition, Universität Bremen, 2006.
[14] D. Hernandez, Qualitative Representation of Spatial Knowledge, Springer, New York, 1994.
[15] G. Ligozat, D. Mitra, J. Condotta, Spatial and temporal reasoning: beyond Allen's calculus, AI Communications 17 (4) (2004) 223–233.
[16] M.T. Escrig, F. Toledo, Qualitative Spatial Reasoning: Theory and Practice, Application to Robot Navigation, IOS Press, Amsterdam, 1998.
[17] C.S. Soanes, A. Stevenson, (Ed.), Concise Oxford English Dictionary, 11th ed., Oxford University Press, 2006.
[18] F. Rossi, P. vanBeek, T. Walsh (Eds.), Handbook of Constraint Programming, 1st ed., Elsevier, Boston, 2006.
[19] P. van Beek, Reasoning about qualitative temporal information, in: E.C. Freuder, A.K. Mackworth (Eds.), Special issue of Artificial Intelligence: Constraint-based Reasoning, MIT Press, Cambridge, MA, USA, 1994, pp. 97–326.
[20] C. Schultz, R. Amor, H. Guesgen, A framework for supporting the application of qualitative spatiotemporal reasoning, in: Proceedings of the Spatial and Temporal Reasoning AAAI Workshop, Technical Report WS-07-12, Menlo Park, CA, 2007, pp. 34–39.
[21] A. Cohn, N.M. Gotts, The 'Egg-Yolk' representation of regions with indeterminate boundaries, in: Proceedings of GISDATA Specialist Meeting on Geographical Objects with Undetermined Boundaries, Taylor & Francis, 1995.
[22] H.W. Guesgen, From the egg-yolk to the scrambled-egg theory, FLAIRS (2002) 476–480.

[23] H.W. Guesgen, Fuzzifying spatial relations, Applying soft computing in defining spatial relations, Physica-Verlag, Heidelberg, Germany, 2002, pp. pp. 1–16.
[24] C.E. Shannon, A mathematical theory of communication, SIGMOBILE Mob. Comput, Commun. Rev., ACM, New York, NY, USA 5(1) (2001) 3–55.
[25] M.J. Darlington, S.J. Culley, Investigating ontology development for engineering design support, Special issue of Advanced Engineering Informatics: Intelligent Computing in Engineering and Architecture, Elsevier, Amsterdam 22 (1) (2008) 112–134.
[26] I. Bratko, Prolog Programming for Artificial Intelligence, 3rd ed., Addison-Wesley, New York, 2001.
[27] A. Silberschatz, in: A. Silberschatz, H.F. Korth, S. Sudarshan (Eds.), Database System Concepts, 5th ed., McGraw-Hill Higher Education, Boston, 2006.
[28] S.J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, 2nd ed., Prentice-Hall, Upper Saddle River, NJ, 2003.
[29] I. Burnstein, Practical Software Testing: A Process-oriented Approach, Springer, New York, 2003.
[30] A.G. Cohn, B. Bennett, J. Gooday, N.M. Gotts, Qualitative spatial representation and reasoning with the region connection calculus, GeoInformatica Springer, Berlin 1 (3) (1997) 275–316.
[31] J.E. Flynn, A study of subjective responses to low energy and nonuniform lighting systems, Lighting Design & Application 7 (2) (1977) 6–15.
[32] W. Pedrycz, Knowledge-based Clustering from Data to Information Granules, Wiley, Hoboken, NJ, 2005.
[33] P.A. Jay, Review: Subjective criteria for lighting design, Lighting Research and Technology 34 (2) (2002) 87–99.
[34] M.S. Sanders, E.J. McCormick, Human Factors in Engineering and Design, 7th ed., McGraw-Hill, New York, 1993.
[35] CIBSE Charted Institute of Building Services Engineers, Code for interior lighting, London, UK, 1994.
[36] IESNA, in: M.S. Rea (Ed.), Lighting Handbook – Reference & Application, 9th ed., IESNA, 2000.
[37] R.S. Bridger, Introduction to Ergonomics, 2nd ed., Taylor & Francis, 2003.
[38] J.A. Veitch, Psychological processes influencing lighting quality, Journal of the Illuminating Engineering Society 30 (1) (2001) 124–140.
[39] C. Cuttle, Lighting by Design, Elsevier, Amsterdam, 2003.
[40] J.E. Flynn, T.J. Spencer, O. Martyniuk, C. Hendrick, Interim study of procedures for investigating the effect of light on impression and behaviour, Journal of the Illuminating Engineering Society 3 (2) (1973) 87–94.
[41] G. Steffy, Architectural Lighting Design, Wiley, New York, 2002.
[42] M.D. Egan, Concepts in Architectural Lighting, McGraw-Hill, New York, 1983.
[43] F.H. Rohles, C.A. Bennett, G.A. Milliken, The effects of lighting, color and room decor on thermal comfort, ASHRAE Transactions 3 (1981) 511–527.
[44] D.L. Loe, K. Mansfield, E. Rowlands, A step in quantifying the appearance of a lit scene, International Journal of Lighting Research and Technology 32 (4) (2000) 213–222.
[45] IAI International Alliance for Interoperability, IFC/ifcXML Specifications 2006, Available at <http://www.iai-international.org/Model/IFC(ifcXML)Specs.html> (as of 31 October 2007).

Please cite this article in press as: C.P.L. Schultz et al., Qualitative design support for engineering and architecture, Adv. Eng. Informat. (2008), doi:10.1016/j.aei.2008.07.003