# Towards A Theory of Application for QSTR Systems

**Carl Schultz**[1], **Robert Amor**[1], **Hans Guesgen**[2]

[1] Department of Computer Science, The University of Auckland
Private Bag 92019, Auckland, New Zealand

[2] Institute of Information Sciences and Technology, Massey University
Private Bag 11222, Palmerston North, New Zealand

**Abstract**

A wide variety of qualitative spatial and temporal reasoning systems have now been developed that formalise various commonsense aspects of space and time. Despite this, relatively few applications have made use of the reasoning tasks provided by these systems. We address this in a novel way by adopting the perspective of application designers. We present an outline of QSTR application theory, and use this to develop methodologies that support designers in creating suitable qualitative models, implementing metrics for analysing QSTR applications, and conducting application-level QSTR logic validation.

## Introduction

While a variety of sophisticated QSTR methods have been developed over the last 25 years, relatively few significant applications have made use of these systems [1]. This is despite spatial and temporal information playing a critical role in many reasoning tasks across a range of disciplines. Moreover, there is a lack of QSTR applications within the AI community, where a more cognitively motivated, flexible qualitative approach to reasoning [2] is seen as highly desirable. The related area of qualitative reasoning about scalar quantities does not have this problem [3]. It must also be noted that representations provided by QSTR systems have been found to be particularly useful in a number of applications such as [4]. Thus, the main issue is that the reasoning tasks provided by QSTR systems have not been widely applied.

We are taking a significantly novel approach for developing methodologies that support the application of QSTR systems. The standard approach is to focus on the analysis of QSTR systems to determine the reasoning services they can provide, and methods for efficiently providing these services. Alternatively, we consider the problem from the perspective of the application designer. Our view is that the QSTR community has not adequately addressed or acknowledged the unique issues of designing and developing QSTR applications compared to more traditional systems (e.g. numerical models). In particular, standard QSTR services check consistency, identify consistent scenarios and perform deductive closure. However, from an application perspective, a task on average involves probability distributions over input premise information and required outputs. That is, a typical application task has target information that reasoning must provide. This suggests that completing partially specified constraint networks may be an inefficient approach for calculating the necessary target information, and moreover will not scale well as the number of objects increases. Other reasoning methods that operate on an application level such as backward chaining may be more appropriate. Furthermore, the additional information from the application requirements specification, such as the operational profile that includes likelihoods over certain inputs and necessary outputs, can be used to define metrics that assess the performance of QSTR components. The results of these metrics can then be used to enhance reasoning for more probable application tasks.

The primary concerns for an application designer are meeting the specific requirements set out by the end users and the cost of development. This requires methodologies for developing suitable qualitative models that address the required tasks, and methodologies for QSTR logic testing and application validation. Note that theorem proving for application level 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 [5]). These issues of application level qualitative modelling and validation have not been adequately addressed. For example, consider the following qualitative reasoning that resulted from seminal lighting studies [6]:

| |
|---|
| **Visual clarity:** Bright, cool, uniform lighting mode in the occupancy area. Some peripheral emphasis, such as high reflectance walls or wall lighting. |
| **Spaciousness:** Uniform, peripheral (wall) lighting. Brightness in the occupancy area is a reinforcing factor, but not a decisive one. |
| **Relaxation:** Warm, non-uniform lighting mode. Peripheral (wall) emphasis, rather than overhead lighting. |
| **Intimacy:** Non-uniform lighting mode. Tendency toward low light intensities in the immediate locale of the user, with higher brightnesses remote from the user. Peripheral (wall) emphasis is a reinforcing factor, but not a decisive one. |

The reasoning describes lighting effects that directly result from spatial arrangements of light sources and surfaces, and with the order in which rooms are visited (a temporal component). Other studies [7] bridge the gap between physical lighting arrangements and subjective lighting concepts. There is no obvious, immediate correlation in terms of reasoning methods between this qualitative reasoning and the standard Allen-based form, although clearly an application designer implementing this reasoning could benefit from the QSTR community's research. Specifically, these are real-world QSTR systems, however, standard operators of composition and conversion are not defined, there are no JEPD relation sets, and it is not clear that a constraint-satisfaction approach is the only suitable implementation.

We address the issues of application level qualitative modelling and validation with a theory of application for QSTR systems. Specifically we develop QSTR model design and validation methodologies for qualitative applications that solve well defined problems. In the following section we present a basic theoretical framework for QSTR applications. We then present metrics for analysing the behaviour of qualities in the context of a required task. In the subsequent sections we present methodologies for logic validation, and a methodology for structuring the available options for modelling ambiguity. In the final section we present our conclusions.

## QSTR Application Theory

In many cases, the reasoning services provided by an existing QSTR method will not directly meet the needs of a task. As a result, applications often need to mix and combine different QSTR systems. Furthermore, many applications have context specific rules that, while often relatively simple, need to integrate at a deep level with standard QSTR systems. This is necessary so that QSTR theory and tools can be applied over the entire application (e.g. metrics, methods for reasoning, and so on) for complete and uniform application analysis and validation. From an application perspective this calls for a fundamental QSTR system format that is sufficiently general to represent a wide range of QSTR systems in a uniform way. In practice, specialised reasoners such as SparQ [1] are employed to process particular well-defined sub-components, and unique, custom components are processed using more general reasoning methods (supported by QSTR specific design patterns) e.g. using declarative languages such as Prolog or SQL.

We begin with a standard definition for QSTR systems by Ligozat et al. [8]. Given a non-empty universe $U$ a QSTR system partitions $U\times U$ into a finite set $I$ of binary relations $R_i$

$$U\times U = \bigcup_{i\in I} R_i.$$

In general relations can be of any arity,

$$U^d = \bigcup_{i\in I} R_{d,i} ,$$

where the integer $d\geq 1$ is the dimension. Thus a quality (qualitative relation) is a set that contains tuples of objects from $U$ of some fixed arity.

A QSTR standard is to encode ambiguity as *possibly holds* with the disjunction of basic relation types between two objects. *Definitely not holds* is represented as the absence of a basic relation between two objects, and *definitely holds* is implicitly inferred when the set of relations that *possibly hold* is a singleton. In the general case an application may need to explicitly represent *definitely holds*, *definitely not holds* and *ambiguous*. A further case *not applicable* can also occur when qualities require preconditions. So for any quality $q$ we require four mutually exclusive sets $q^+$ (definitely holds), $q^-$ (definitely not holds), $q^?$ (ambiguous), $q^\sim$ (not applicable) such that

$$U^d = \Delta_{\alpha\in\{+,-,\sim,?\}} q^{\alpha}{}_d$$

for integers $d\geq 1$ where $\Delta$ is symmetric difference (mutual exclusion).

In general we cannot assume that all relations are JEPD. Furthermore we cannot assume that the conversion and composition operators have been defined. In general the structure of a QSTR system is defined by constraints between qualities of the form

$$X\,\delta\,Y,$$

where $\delta\in\{\subset,\subseteq,\not\subset,\neq,=\}$ is a set comparison, and $X$, $Y$ are set expressions. Set expressions are either sets (e.g. *before*$^-$), the result of set operations (e.g. *before*$^-\cap$*during*$^-$) or sets defined using builder notation (e.g. $\{y|(x,y)\in \mathit{before}^-\}$). For example, the conversion constraint as given in [8] is

$$R_{2,i}\hat{} = \{(x,y)\in U\times U|(y,x)\in R_{2,i}\}$$

where the function ^ returns the designated converse set (e.g. *before*^ returns *after*). Another example is a constraint used to specify when the subjective colour temperature of a room is cool. If all lights $x$ in the room $y$ have a cool colour temperature, then the room $y$ also has a cool colour temperature,

$$\{y \mid (x,y)\in in^+ \rightarrow x\in cool^+\} \subseteq cool^+. \qquad (1)$$

The domain of a constraint is the set of qualities that occur in the constraint, for example the domain of (1) is $\{in, cool\}$.

We can now represent a wide range of QSTR systems. Allen-based systems can be represented as a set of relations $R_i$ using a definition of $U$ and constraints for conversion, composition, and identity.

We can also represent systems that do not necessarily adhere to the standard Allen-based form. For example, in [9] Qayyum and Cohn present a method for image retrieval using qualitative spatial representations. Each image is divided into cells that are annotated with a semantic concept such as *sky* or *grass* [10]. Qualitative relationships between all pairs of semantic concepts are calculated for an image, e.g. *greater-than* (meaning that more cells of one concept are present in an image compared to some other concept). Thus $U$ is the set of semantic concepts, and each image is a different scenario $s$ defined by the way the relations $R^s_i$ (e.g. *greater-than*, *equal-to*, *less-than*) divide up $U\times U$ such that $\Delta_{i\in I}R^s_i = U\times U$. Images are compared by determining the differences between equivalent relations e.g. if $R^x_i=R^y_i$ for $i\in I$ then two images are equivalent.

The same principle applies to dynamic and multi-granular QSTR systems. Each snapshot of a scene is represented by a new scenario $s_t$, where the contents of the relations in $s_{t+1}$ are constrained by the relations in $s_t$. For example, the continuity assumption means that an object with relation $R_x$ in scenario $s_t$ can take $R_x$ or relations adjacent to $R_x$ in the neighbourhood graph during scenario $s_{t+1}$. Note that, from an application perspective, this is not an axiom, as the constraint can be broken during valid, normal operation. For example, an application that uses sensor data as premise information for constructing scenarios may find that a pair of scenarios $s_i$ and $s_{i+1}$ break the continuity constraint; the application can then respond by increasing its sampling rate.

In the following sections we use this definition to develop methodologies for application level metrics, validation, and modelling.

## QSTR Metrics

The application designer has task specific information in the form of a requirements specification. This includes

- desired functionality of the application
- a source of real (but mathematically informal) QSTR content to be automated in software (e.g. client preferences, formal scientific studies, industry consensuses, and so on)
- use cases that reflect the operational profile

Using this additional information we can define metrics that assess the performance of components in QSTR applications. Firstly we note that qualities group tuples of objects, and constraints assign certain qualities (categories) to objects based on the state of other qualities for those objects (attributes). This is analogous to attributes being used to describe the category of an object. We will now present the metrics cohesion, coupling, and entropy.

Cohesion is a measure of how effectively a set of attributes defines a category. It measures the strength of a category based on a comparison of the attributes between each pair of objects in that category. The comparison is a distance function that sums the differences between each attribute value of two objects, for example, based on weighted conceptual neighbourhoods of the attributes. The algorithm accepts a category $C$ and a set of attributes $\boldsymbol{A}$ as input and calculates statistics on the distances such as the mean and variance.

```
cohesion ($C$, A) |
  for all pairs of objects $i,j\in C$
    calculate $d$=distance($i,j,$A)
    store $d$ in an array
  calculate statistics over stored distances
```

A category is cohesive if the distances have a low mean and a low variance. Ideally categories should be cohesive, as this indicates that all objects in the same category share a similar set of attribute values. This metric can be used in a number of ways. For example, cohesion can be calculated over real-world study data as evidence for including a constraint in the qualitative model that binds the category to the attributes. Alternatively, categories may not be precisely related to the attributes, but more as a rule of thumb. Using cohesion we can assess the performance of the distance function and our attribute selection as a heuristic definition of a category.

Coupling is a measure of how effectively a set of attributes distinguishes between two categories based on a comparison of the attributes of objects in each category. The algorithm accepts two categories $C_1$, $C_2$ and a set of attributes $\boldsymbol{A}$ as input and calculates statistics on the distances such as the mean and variance.

```
coupling ($C_1$, $C_2$, A) |
  for all pairs of objects $i\in C_1$ , $j\in C_2$
    calculate $d$=distance($i,j,$A)
    store $d$
  calculate statistics over stored distances
```

Two categories have low coupling if the distances have a high mean and a low variance. Ideally two supposedly unrelated categories should have low coupling. As with cohesion, coupling can be used check constraints, distance functions and attribute selections. Furthermore, it can be used to assess the accuracy of a conceptual neighbourhood graph by checking that immediate neighbours of a category have higher coupling than more distant categories. A full neighbourhood consistency test would ensure that, from the perspective of each category, all partial orderings of the other categories based on their coupling values (with a tolerance for equal coupling) approximates the neighbourhood graph. Coupling can also be used to assign weights to the conceptual neighbourhood graph, where a low coupling yields a high neighbourhood weight.

Entropy is a measure of the information content provided by constraints. That is, reasoning generates data at every inferential step and, according to information theory [11], the amount of information gain is inversely proportional to the end-user's prior knowledge about the inferred data. If the end-user is very confident about what datum will be provided before an inference step has been taken, then the inference step provides little or no information. Thus, the application designer 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 quality entirely from the model. Information content $I$ for a quality $q$, based on entropy, can be calculated as [11],

$$I(q) = \sum_{\alpha\in\{+,-,\sim,?\}} -\mathrm{p}(q^{\alpha}) \log_2 \mathrm{p}(q^{\alpha}) \quad ,$$

where $\mathrm{p}(x)$ is the probability that a randomly chosen object in a randomly chosen scenario will have the quality $x$ (calculated with use case data from the operational profile). Relationships between qualities can be measured using mutual entropy [11],

$$I(q;v) = \sum_{\alpha\in \mathrm{N}} \sum_{\beta\in \mathrm{N}} \mathrm{p}(q^{\alpha},v^{\beta}) \log_2 \frac{\mathrm{p}(q^{\alpha},v^{\beta})}{\mathrm{p_a}(q^{\alpha}) \cdot \mathrm{p_b}(v^{\beta})}$$

where N={+,–,~,?}, $\mathrm{p}(x,y)$ is the probability that $x$ and $y$ occur together, and $\mathrm{p_a}(x)$, $\mathrm{p_b}(y)$ are marginal probabilities of $x$ and $y$ respectively. Entropy can be used to reduce the ambiguity from inference (e.g. composition) by removing qualities that are improbable based on other factors in the scenario. For example, composition may reveal that a cup's relation to a table is either *meets above* (resting on the table) or just *above* (floating above the table). The QSTR system, having evaluated a number of use cases may determine that the conditional entropy for physical objects with the *above* relation is very low when the object has no *meets above* relations; that is, by analysing the conditional entropy it decides that floating objects are unlikely. In this case entropy is used for induction.

Another example of how entropy can be used is the automatic construction of decision trees to respond to queries about whether objects have certain qualities e.g. "is the kitchen warm and cosy?". Given a set of known attribute values for the query object (e.g. the available premise information about the kitchen) and entropies that relate the attributes to the query quality (e.g. entropies for warmth and cosiness in relation to furniture arrangement, light source parameters, and so on) a decision tree can be produced that infers the query answer with a minimum number of intermediate operations. In general decision trees may also improve reasoning performance by ordering the constraints according to entropy.

## Validation

The aim of program validation in software engineering is to determine if the system is fit for purpose [12], explicitly evaluating the program in terms of its application context. QSTR application validation aims to provide a level of confidence that the system logic, specifically the constraints and relations, provides the intended function by ensuring that the highly used and critical components have been adequately tested.

The test space is defined based on system inputs and outputs, and the system structure such as decisions and control paths. Covering the entire, often infinite test space is clearly impractical and thus software engineers employ methods that isolate key subsets such as boundary checking, equivalence class partitioning, and cause-effect graphs [12]. However, by the very design of a QSTR application, only salient inputs, outputs, and constraints are modelled. That is, qualitative concepts already represent behavioural classes and their delimiting values, and so traditional approaches are not useful for reducing the QSTR application test space. The following sections analyse the characteristics of QSTR applications to derive methodologies for validation.

### Unit Testing

In QSTR applications, the units for testing are the constraints. In the following analysis we identify two critical boundary classes for constraint testing.

Firstly, we note that the sets in constraints can be defined using set builder notation, for example $R_1 \cap R_2 / R_3$ becomes $\{x \mid x\in R_1 \wedge x\in R_2 \wedge x\notin R_3\}$. The conditions in set-builder notation can be divided into two categories, (a) conditions that must hold (e.g. $x\in R_1$) and (b) conditions that must not hold (e.g. $x\notin R_3$). Thus, each quality in a constraint's domain either

- *triggers* the constraint (the constraint will not apply if these conditions are not satisfied)
- *overrides* the constraint (even if the constraint has been triggered, if these conditions hold then they will stop the constraint from applying) or,
- is *independent* (i.e. has no effect e.g. *before*$^{+}$ participates in a constraint which means *before* is in the constraint's domain, but *before*$^{-}$ may not be involved).

For a constraint to apply, it must have one or more triggers and no overrides. We can use this insight as a basis for defining test classes for unit testing. To illustrate this, consider the room colour temperature constraint (1) on page 2. For a given room $y$, the relevant cases for different objects are expressed in the constraint matrix in Table 1.

TABLE 1: Constraint matrix of colour temperature constraint (1)

| $x\in in_y$ / $x\in cool$ | **holds** | **not holds** |
|---|---|---|
| **holds** | trigger | independent |
| **not holds** | override | independent |

We will now derive two critical boundary concepts based on equivalence assumptions for QSTR logic testing.

An infinite number of cases exist for each combination of cells in the constraint matrix. For example, one cool light in a room will activate the same cell as one hundred cool lights in a room. The first equivalence assumption is that cases resulting in the same combination of cells are equivalent and thus only one case for each cell combination needs to be tested. This yields a finite critical test set size of

$$\Sigma_{i=1\ldots n}\ \mathrm{C}^{n}_{i},$$

where $n$ is the number of cells $2^{|D|}$ (where $D$ is the domain) and $\mathrm{C}^{n}_{k}$ is the number of $k$-sized combinations out of $n$.

We reduce the critical test set further by only testing the interaction of different condition classes. Thus, the second equivalence assumption is that, if a set of cases each result in a single cell of the same condition class (such as *trigger*) then any case that results in some combination of these cells is equivalent to the set of individual cases. For example after testing all *trigger* cells individually we can ignore pairwise (and higher) comparisons of *trigger* cells. The critical test set size is now

$$(t{+}o{+}i) + (t{\cdot}i + t{\cdot}o + o{\cdot}i) + (t{\cdot}o{\cdot}i)$$

where $t$, $i$, $o$ are the number of *trigger*, *independent* and *override* cells respectively.

This defines complete boundary case unit testing for a constraint and thus provides an ideal target for logic validation. Furthermore, the development cost is relatively low as these test cases can be easily generated automatically.

## Test Coverage

Integration and system testing is used to ensure that components interact as the designer has intended. As with unit testing, there are often an infinite number of possible cases in which components interact, and so we require approaches that isolate relevant subsets of the test space. For this we use test coverage analysis to guide integration testing by identifying component interactions that have not been adequately assessed.

In standard software engineering, test coverage is a measure of the proportion of components exercised during testing, and typically refers to the control flow graph (CFG) where vertices represent program statements and directed edges represent control flow. In order to adapt coverage measures to QSTR, we define the qualitative constraint graph (QCG) as an analog to the CFG, with vertices representing qualitative constraints and undirected edges representing a shared quality that occurs in the domains of two constraints. The edges in the QCG indicate (necessarily mutual) constraint dependencies; if the constraint $c_1$ modifies a relation that appears in constraint $c_2$ then $c_2$ must be revisited (although it may not require any further scenario modifications). For example, the QCG of the lighting rules given in the Introduction section is illustrated in Figure 1.

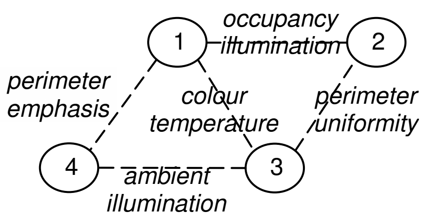

**Figure 1**. Qualitative constraint graph (QCG) of a selection of domain constraints for architectural lighting [6], where vertices represent constraints and edges represent related qualities.

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

- vertex (*statement*) coverage is the percentage of vertices executed in at least one test
- edge (*branch*) coverage is the percentage of edges executed in at least one test
- $k$-path (*path*) coverage is the percentage of paths of length $k$ executed in at least one test (for $k$>2)
- condition (*decision*) coverage (applied to either vertices, edges, or paths) is the percentage of modification combinations resulting in execution, exercised in at least one test

Condition coverage refers to the observation that when a constraint involves a number of relations there are multiple ways in which the vertex could be executed. Similarly, when a constraint involves multiple relations of the same type there exist multiple ways for its edges to execute.

Figure 2 illustrates the partial ordering of the coverage measures in terms of strength. While stronger coverage may increase the chance of detecting logical defects, it requires more test cases and resources to be achieved.

$$\begin{tikzcd}
\text{vertex} \arrow[r] \arrow[d] & \text{vertex condition} \arrow[d] \\
\text{edge} \arrow[r] \arrow[d] & \text{edge condition} \arrow[d] \\
\text{k-path} \arrow[r] & \text{k-path condition}
\end{tikzcd}$$

**Figure 2**. Partial ordering of coverage criteria from weakest to strongest (if full coverage is achieved).

## Modelling Ambiguity

Reasoning is the process of using definite premise information to infer values for indefinite or ambiguous components. Thus the application designer must necessarily specify how their constraints should behave in scenarios where domain qualities are ambiguous. In this section we present a methodology that defines ambiguity in a constraint, allowing the designer to easily consider alternative modelling options.

Each constraint has a number of different versions depending on how ambiguity is handled. For example, Table 2 is the constraint matrix for the colour temperature rule (1) with additional cells for ambiguity; the designer has decided that if a light is not in the room then it is disregarded for this rule.

TABLE 2: Constraint matrix for the neutral form of (1)

| $x \in in_y$ / $x \in cool$ | **holds** | **?** | **not holds** |
|---|---|---|---|
| **holds** | trigger | | independent |
| **?** | | | independent |
| **not holds** | override | | independent |

Table 2 is the neutral (or definite) form of the constraint where ambiguity is left unspecified. During reasoning the neutral form will default to Table 3 where unspecified ambiguous cells become independent conditions.

The designer can modify the behaviour of the constraint by classing the unspecified cells in different ways. We define categories of constraint versions by adapting the concepts of aggressive and conservative used in the

TABLE 3: Default semantics for the neutral form of (1)

| $x\in in_y$ / $x\in cool$ | **holds** | **?** | **not holds** |
|---|---|---|---|
| **holds** | trigger | independent | independent |
| **?** | independent | independent | independent |
| **not holds** | override | independent | independent |

ConQuer database system [14]. The conservative form requires that all possible scenarios satisfy the constraint. Ambiguity is resolved towards overriding the constraint as shown in Table 4.

TABLE 4: Maximally conservative form of (1)

| $x\in in_y$ / $x\in cool$ | **holds** | **?** | **not holds** |
|---|---|---|---|
| **holds** | trigger | independent | independent |
| **?** | override | override | independent |
| **not holds** | override | override | independent |

The aggressive form requires that at least one possible scenario satisfies the constraint. Ambiguity is resolved towards triggering the constraint as shown in Table 5.

TABLE 5: Maximally aggressive form of (1)

| $x\in in_y$ / $x\in cool$ | **holds** | **?** | **not holds** |
|---|---|---|---|
| **holds** | trigger | trigger | independent |
| **?** | trigger | trigger | independent |
| **not holds** | override | independent | independent |

Thus each constraint has a lattice of different versions between these two extremes, based on the following partial ordering: rule $\theta_x$ is more conservative than rule $\theta_y$ if

$$(t_x+1) / (o_x+1) < (t_y+1) / (o_y+1)$$

where $t_i$, $o_i$ are the number of *trigger* and *override* cells in rule $\theta_i$ respectively. This methodology makes the versions of a constraint readily accessible to the application designer. Once the designer has specified their conservative and aggressive constraint versions, a number of variations on reasoning can be executed. For example, a conservative query will reason using conservative forms of constraints that infer *triggers* and aggressive forms for constraints that infer *overrides*.

## Conclusions

In this paper we have presented several novel methodologies that support the application of QSTR systems. These methodologies support application designers in developing suitable qualitative models and conducting application-level QSTR logic validation. We have outlined a theory of application that provides a broad, general representation for QSTR systems. This allows validation methodologies and metrics to be developed that uniformly assess QSTR applications, regardless of the component QSTR systems or semantics. We presented three application level metrics of cohesion, coupling and entropy for analysing the behaviour of qualities in the context of the required task. Furthermore, we presented methodologies for validation that identify criteria for determining logical equivalence as a target for unit testing, and adapt test coverage criteria to assist in integration and system level testing. Finally, we presented a design pattern for modelling ambiguity that assists the application designer by structuring the available modelling options. Specifically, we classified variants of a constraint according to the way the constraint behaves under ambiguity by adapting the concepts of conservative and aggressive queries. Thus, by adopting a novel application design perspective of QSTR systems, we have developed a set of methodologies to aid application designers in producing real-world QSTR applications.

## References

[1] F. Dylla, L. Frommberger, J.O. Wallgrün, D. Wolter (2006). SparQ: A Toolbox for Qualitative Spatial Representation and Reasoning. In Proceedings of the Workshop on Qualitative Constraint Calculi: Application and Integration.

[2] C. Freksa (1991) Qualitative Spatial Reasoning. In D.M. Mark & A.U. Frank (eds.), Cognitive and Linguistic Aspects of Geographic Space, 361-372, Kluwer Academic Publishers.

[3] Y. Iwasaki (1997) Real-World Applications of Qualitative Reasoning. IEEE Expert 12(3), 16-21.

[4] J. Fernyhough, A.G. Cohn, D.C. Hogg (2000) Constructing qualitative event models automatically from video input. Image and Vision Computing 18 (2000) 81–103. Elsevier.

[5] A.G. Cohn, B. Bennett, J. Gooday, N.M. Gotts (1997) Qualitative spatial representation and reasoning with the region connection calculus. GeoInformatica 1 (3) 275-316. Springer, Berlin.

[6] J.E. Flynn (1977) A study of subjective responses to low energy and nonuniform lighting systems. Lighting Design & Application 7 (2) 6–15.

[7] C. Cuttle (2003) Lighting by Design, Elsevier, Amsterdam.

[8] G. Ligozat, D. Mitra, J. Condotta (2004) Spatial and temporal reasoning: beyond Allen's calculus. AI Communications 17 (4) 223–233.

[9] Z.U. Qayyum, A.G. Cohn (2007) Image retrieval through qualitative representations over semantic features. Proceedings of the 18th British Machine Vision Conference (BMVC2007), 610-619.

[10] J. Vogel, B. Schiele (2007) Semantic Scene Modeling and Retrieval for Content-Based Image Retrieval. International Journal of Computer Vision 72 (2) 133-157.

[11] C.E. Shannon (1948) A Mathematical Theory of Communication. Bell System Technical Journal, 27, 379-423.

[12] I. Burnstein (2003) Practical software testing : a process-oriented approach. Springer, New York.

[13] A. Fuxman, E. Fazli, R. J. Miller (2005) ConQuer: efficient management of inconsistent databases. Proceedings of the 2005 ACM SIGMOD, International Conference on Management of Data, Baltimore, Maryland. 155-166.