Development Framework for Qualitative Spatial and Temporal Reasoning Systems

Carl Schultz, Robert Amor
Department of Computer Science, The University of Auckland
Private Bag 92019, Auckland, New Zealand
csch050@ec.auckland.ac.nz, trebor@cs.auckland.ac.nz

Hans Guesgen
Institute of Information Sciences and Technology, Massey University
Private Bag 11222, Palmerston North, New Zealand
h.w.guesgen@massey.ac.nz

Abstract

A significant number of qualitative methods for representing and reasoning about spatial and temporal information have now been developed that address limitations of purely numerical approaches. Despite this, qualitative spatial and temporal reasoning (QSTR) is yet to be fully utilised by researchers and software developers outside the QSTR community. In our view, the problem is that in many cases none of the existing pre-made QSTR methods directly meet the needs of a task and instead require modifications. In other cases a completely new and unique QSTR approach may be required. To address this we are developing a framework that supports the application of QSTR by providing a methodology of developing custom QSTR systems from specific task information. In this paper we give an overview of our framework. As an example we also present the design component, one of the key framework processes, in more detail.

Introduction

Purely numerical methods for modelling and reasoning about spatial and temporal information have limitations. Firstly, if critical pieces of numerical information about a model (or scenario) are not available then either the equations simply can not be evaluated and reasoning algorithms break down, or the results are meaningless. Secondly, because numerical models lack explicit causality, the output of large and complex models can be unpredictably sensitive to the initial conditions, i.e. a small change in the input can result in an unexpectedly large change in the output due to many complex, obscured interactions. Finally, numerical methods have inherent restrictions on the type of information that can be expressed, as they fundamentally rely on the definition of a homogenous unit to represent a concept, which is often unavailable or impossible.

Qualitative spatial and temporal reasoning (QSTR) methods address these issues. Firstly, a QSTR method will always apply any available premise information rather than halting on a particular missing piece of information, i.e. reasoning gracefully fails rather than completely breaking down. Secondly, by the design of QSTR systems causality is made explicit. Moreover, only necessary distinctions are modelled so that any change in the initial conditions of a scenario is necessarily significant, thus avoiding the issue of unpredictable sensitivity. Finally, QSTR modelling does not require fundamental homogeneous units to describe concepts. Instead, it is a top-down process where concepts are incrementally refined by the inclusion of increasingly finer qualitative relations.

An array of QSTR techniques have now been developed that focus on fundamental aspects of our daily experience of space and time (Cohn and Hazarika 2001). Despite this extensive body of QSTR research few software applications exist that make significant use of QSTR approaches (excluding prototype systems developed by the QSTR community). QSTR researchers have been addressing this by unifying the numerous QSTR methods (Ligozat and Renz 2004) and developing QSTR software libraries and toolboxes such as SparQ (Wallgrün et al. 2007) that provide implementations of existing QSTR methods in a standard framework with a uniform interface.

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 formalism will not directly meet the needs of a task. This is because, firstly, qualitative terms are vague, and rely on context for disambiguation and meaning, e.g. “near” can be a function of financial cost, time traveled, or distance covered, and can vary depending on the user being a pedestrian, a car owner, or a city planner. Secondly, the qualitative concepts being modelled must be relevant for the task; it is therefore difficult to gauge the usefulness of a

particular qualitative relation within a QSTR method a-priori 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.

We are developing a broader framework for supporting the design and application of QSTR methods based on specific task requirements. The framework is application oriented and considers the theory and practice of applying QSTR in the context of problem solving and software development. It must be noted that the 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 expounds necessary processes involved in the development lifecycle and analyses effective practices that developers can use in each lifecycle phase.

The remainder of this paper is as follows. In the following section we present an overview of our framework. We then present the theoretical and practical components of the design process in the framework as an example. We then present our future work and the conclusions of this paper.

Framework Overview

The framework that we are developing supports the design, implementation and validation of general QSTR systems that are customised for specific task requirements. Fig 1 illustrates an overview of our framework.

The design process supports the developer in formulating a customised QSTR engine according to task information and specifications. The implementation process supports the developer 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 developer in evaluating QSTR methods with respect to the intended application environment by incorporating context specific test data. It must be noted that the framework makes no assumptions about the ordering of the processes or the software lifecycle model employed.

Each process in the framework consists of a theoretical element used to derive principles, and a practical element used to derive strategies for applying QSTR. In the following sections we will focus on the design process of the framework covering both the theoretical and practical elements.

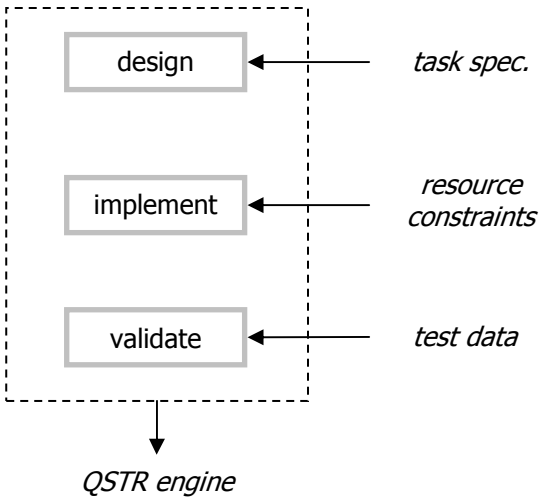

Fig 1. Overview of our framework for supporting the development of QSTR systems. Solid boxes within the dotted box represent key framework processes, and italicised labels with arrows represent external components that function as framework input and output.

QSTR Design Theory

The purpose of detailing the design process in our framework is to equip a developer with the necessary theoretical foundation and practical strategies to design a QSTR based software application that addresses their task requirements. The developer must understand

- the process phases that an actual QSTR system follows when being used to address a problem,
- what a QSTR model is and what it consists of,
- what QSTR reasoning is and what it consists of and
- the type of tasks that can be performed by QSTR according to the modelling and reasoning theory.

The following sections present our design theory for each of these four aspects of applied QSTR.

Problem Solving Process

In general, models are used to represent salient aspects of a problem, and by reasoning about the model we intend to acquire a useful problem solution. QSTR is the problem solving approach that uses qualitative models and qualitative reasoning.

We define an automaton to describe the QSTR problem solving process as illustrated in Fig 2. 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). A formal definition of this automaton is provided in (Schultz, Amor and Guesgen 2007).

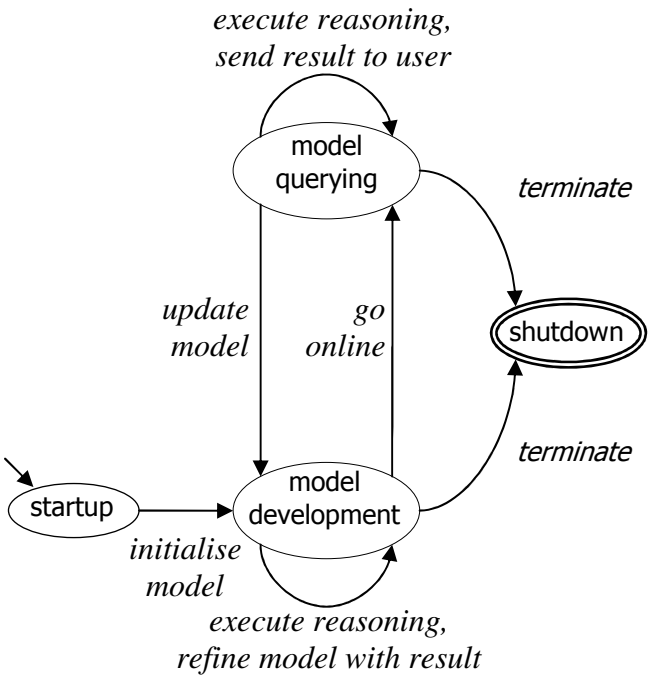

Fig 2. Nondeterministic finite state automaton used to describe the problem solving process. Circles represent states, arrows represent possible state transitions, and the arrow annotations describe the effect of the transition on the QSTR system.

Using this automaton the software developer can begin to characterise their QSTR system by explicitly incorporating task requirements into the problem solving process. Strategies for using this automaton are part of design practice section of our overall framework. 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
- *model stability*: an unstable model changes frequently, whereas a stable model does not
- *model dependency*: a problem may only have initial dependency where all the information required for reasoning is initially available; other problems may have ongoing model dependency where some required information is not initially available

Qualitative Modelling

In order to determine what a qualitative model is, we consider a standard definition of the term "quality" as used in everyday language: a quality is an inherent or distinguishing property (Soanes and Stevenson 2006). From this we determine two things: firstly, 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 in our model.

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: either the quality holds for an object in the model, or it does not hold. 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, or it does not hold.

Qualitative Reasoning

In most cases some part of the model we are using is incomplete. This represents the information that must be calculated or derived through reasoning in order to solve the problem at hand.

In general, reasoning applies domain constraints to the available information to infer plausible values for the incomplete fragments of the model. In our interpretation of a qualitative model, domain constraints specify relationships between qualities; 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. For example the qualitative constraint "if a room appears warm and bright, then it evokes a relaxing impression" states that if a room object is "warm" and "bright" then it must also be "relaxing".

reasoning principle. the basis of 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.

Tasks Performed by Pure QSTR

We now consider the tasks that can be performed by QSTR according to our previous definitions of qualitative modelling and reasoning. The process of applying qualities to a set of objects forms object classes or categories based on which qualities hold and do not hold. For example the qualitative constraint "roads far from the city that overlook the lake are idyllic" defines the class of "idyllic roads" as including road objects that are both "far from" the city and "overlooking" the lake.

task principle. all pure QSTR tasks are a process of object classification

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 a further common task in the QSTR field is to determine consistent numerical instantiations of a given qualitative model. 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). This task is beyond the scope of pure QSTR and will not be addressed in this paper.

QSTR Design Practice

Design practice uses the previous theoretical principles to provide strategies for designing a QSTR system based on specific task requirements. This section gives a brief outline of how each of the design principles can be used to drive design.

Firstly the developer must decide whether QSTR is applicable to the problem at hand. Considering the task principle (refer to the section "Tasks Performed by Pure QSTR"), they must decide whether the problem can be solved by classifying objects. The two most common tasks are:

- reasoning: used to infer information about incomplete models
- querying: used to isolate interesting fragments of models

The developer must then characterise the behaviour of their system by using the problem solving automaton (refer to the section "Problem Solving Process"), as this greatly influences the design of the QSTR model. For example, if the QSTR system must allow substantial querying flexibility then the design theory recommends the inclusion of 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 providing a richer query language.

The developer must then decide on the objects and qualities that will be modelled in the system. A quality should only be included if the distinction it provides is relevant for the task. As a guide, the developer should review the expected typical use of their application, particularly the likely premise information or input and desired reasoning output.

Finally, the developer must specify the domain constraints that will be used for reasoning. This can be viewed as linking the input of the model to the output, and may require the inclusion of further intermediate qualities. The developer must 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).

Further support for refining the design is provided in the framework's implementation and validation processes.

Future Work

We are currently investigating methods for empirically assessing the effectiveness of our framework in supporting the application of QSTR. For example, we anticipate conducting studies involving software developers with no previous experience of QSTR. The participants will be asked to use our framework to develop a number of QSTR based software tools that each address a given domain specific problem, and then report on the results.

Conclusions

We are developing a framework for supporting the design, implementation and validation of QSTR systems based on specific task requirements. The framework is application oriented and considers the theory and practice of applying QSTR in the context of problem solving and software development. Thus, rather than providing a developer with software libraries of pre-made QSTR methods alone, we are taking a broader approach that equips a developer with the necessary theoretical foundation and practical strategies to develop a customised QSTR based software application for addressing their specific task requirements.

We have presented the theory and practice of the design process component of our framework. Our design theory investigates four aspects of applied QSTR: the problem solving process, modelling, reasoning and task applicability. Design practice then uses design theory principles to provide strategies for designing a QSTR system based on specific task requirements.

We are exploring methods for empirically determining how effective our framework is in supporting QSTR system development, for example, conducting studies in which software developers apply our framework to a selection of problems.

Acknowledgements

This work has been funded by the Bright Future Top Achiever Doctoral Scholarship (Tertiary Education Commission, New Zealand).

References

Cohn, A. G. and Hazarika, S. M. 2001. Qualitative Spatial Representation and Reasoning: An Overview. *Fundamenta Informaticae* 46(1-2): 1-29.

Ligozat, G. and Renz, J. 2004. What Is a Qualitative Calculus? A General Framework. In *Proceedings of the Eighth Pacific Rim International Conference on Artificial Intelligence: Trends in Artificial Intelligence, Auckland, New Zealand, August 9-13*, 53-64. Lecture Notes in Computer Science 3157: Springer.

Schultz, C.P.L., Amor, R., and Guesgen, H.W. 2007. A Framework for Supporting the Application of Qualitative Spatiotemporal Reasoning. In *Proceedings of the Workshop on Spatial and Temporal Reasoning at the Twenty-Second Conference of the Association for the Advancement of Artificial Intelligence (AAAI)*, Guesgen, H.W., Ligozat, G., Renz, J., and Rodriguez, R.V. program co-chairs, Technical Report WS-07-12, 34-39. Menlo Park, California: AAAI Press.

Soanes, C.S. and Stevenson A. eds. 2006. *Concise Oxford English Dictionary*. 11th ed. Oxford: Oxford University Press.

Wallgrün, J. O., Frommberger, L., Wolter, D., Dylla, F. and Freksa, C. 2007. Qualitative Spatial Representation and Reasoning in the SparQ-Toolbox. In Barkowsky, T., Knauff, M., Ligozat, G., Montello, D. eds. *Spatial Cognition V: Reasoning, Action, Interaction: International Conference Spatial Cognition 2006, Bremen, Germany, September 24-28*, 39-58. Lecture Notes in Computer Science 4387: Springer.