A Complex Biological Database Querying Method

Jake Yue Chen
School of Informatics and School of Science CIS Department
Indiana University – Purdue University Indianapolis
Indianapolis, IN 46202, USA
Tel: (01) 317-278-7604
jakechen@iupui.edu

John V. Carlis
Computer Science Department
University of Minnesota
Minneapolis, MN 55455, USA
Tel: (01) 612-625-6092
carlis@cs.umn.edu

Ning Gao
School of Engineering and Technology ECE Department
Indiana University – Purdue University Indianapolis
Indianapolis, IN 46202, USA
ngao@iupui.edu

ABSTRACT

Many biological information systems rely on relational database management systems (RDBMS) to manage high-throughput biological data. While keeping these data well archived, organized, and integrated in a common repository is still a challenging task, performing complex queries, i.e., explorative and abstract *ad hoc* user questions in biology, is an even formidable task often substituted by writing complicated software programs. In this work, we propose a "complex query modeling" method to address the challenge of complex querying in biological domains. Query modeling consists of four distinct but interdependent phases of activities: representation of high-level problems, transformation of problems into connected query interfaces, designing database query structures, and translating query plans into high-performing SQL statements. At each stage, we use different notations and query modeling practices. Using gene indexing project as a case study, we show that query modeling enables prototypical development of high-quality SQL solutions to an inherently abstract and vague user query question, which requires GeneChip designers to sift through millions of database records, process data in dozens of steps, and make myriads of intermediate decisions. We believe our "complex query modeling" method is applicable to other bioinformatics domains with needs for complex database queries.

Keywords

Database Management System (DBMS), Complex Queries, Query Modeling

1. INTRODUCTION

1.1 Background

Biological data analysis tasks have become increasingly complex. An accelerated number of completed genome sequences (more than 1,000 as reported by NCBI in August 2004) and hundreds of gigabytes of microarray experimental data sets, for example, have created an information overload for biologists [1]. On one hand, there is a great need for new biological information systems to support data management and data mining efforts in these large data sets. On the other hand, there is an even greater need for these systems to support *ad hoc* scientific data explorations and hypothesis-driven biological discovery research. It remains a grand challenge for large biological information systems to support the above combined goals of biological data miners and data explorers, due to the inherently different nature of two types of activities—one emphasizing automated reasoning and the other emphasizing manual explorations.

A practical scenario of such a grand challenge for existing biological information systems can be described as follows. A user, a computer scientist or a biologists or a research team, usually does not know in advance the detailed cost of an open-ended complex biological question. Examples of such questions are "can we define the role of protein X in Wnt signaling pathways from existing pathway data?" and "find all the gene expression regulation models of gene family *Y* from publicly available microarray experiments". If a biological information system attempts to make its own assumptions and present its automated solutions to the end user, the results will be seldom significant—since the structure of novel scientific questions always requires guidance of the innovative minds of domain users (biologists). In contrast, if the system let the domain user take complete control and follow the conventional software development route of solving these questions, it will be too time-consuming and programming-intensive to accomplish. By the time a software program solution is fully developed, the initial question from the user may have evolved into a related one that requires a different software solution. Therefore, existing biological information systems, in essence, discourage discovery-minded users from asking data-intensive complex biological questions.

1.2 Concepts

In this research, we present a method called **complex query modeling** to address the challenge of conquering complex biological questions in the database context. By complex query modeling, we refer to the collaborative database query design process among a team of domain users and database developers to answer open-ended, explorative, and abstract questions in a relational database management system (RDBMS). Complex query modeling consists of four distinct but interdependent phases of query design activities, which include:

1) Representation of high-level problems, using the *task-action analysis model* (Section III-c);
2) Transformation of problems into connected query interfaces, using the *query interface model* (Section III-d);
3) Designing database query structures, using the *query structure model* (Section III-e);

SAC'05, March 13-17, 2005, Santa Fe, New Mexico, USA.

4) Translating query plans into high-performing *SQL statements* (Section III-f).

Note that we choose to address complex querying problems in the RDBMS due to two observations in biology. First, most existing biological information systems are built on RDBMS. Second, even for specific biological tasks such as BLAST, which has to be performed outside of the RDBMS, many well-implemented biological information systems keeps the input/output data of BLAST in database tables. In either observation, we can make an abstraction of the user-to-system query environment as database operators or "virtual database operators" (such as BLAST) manipulating on data stored in regular or temporary relational tables.

Representing the structures of complex user questions as different types of query models graphically characterizes the main advantage of complex query modeling over other related approaches, including extended SQL scripting, custom software development, and workflow management systems [2]. **Extended SQL scripting** refers to the use of extended SQL operators and extended SQL syntax in a specialized RDBMS to write expressive SQL statements that would otherwise be difficult to write. Although this querying approach is popular in data integration, temporal databases, and spatial database applications [3-5], it cannot help biology users to decompose initially highly abstract scientific questions in general. **Custom software development** supports the documentation of user requirements based on software engineering principles. However, the development cost and performance cost (round-trip movement of large amounts of data from the RDBMS to the software system) are prohibitively more expensive than developing and running queries in an RDBMS. Some biological information systems incorporate **workflow management system** concepts into their components, for example, SRS[6], SIBIOS[7], and TAMBIS [8]. These systems alleviate user custom software development by providing user customized control flow and data flow design/mapping support, which is particularly useful in addressing challenges for information integration from disparate data sources and software tools. However, none of the workflow-based systems were developed with solving complex database queries as a specific goal. We summarize the differences between complex query modeling and workflow-based system as the following:

1) Complex query modeling provides a formal framework to analyze, decompose, and translate high-level abstract biological questions into computationally efficient database queries, whereas a workflow-based system is a software platform that uses control flow or data flow models to enable pipelined execution of complex tasks as "computational subtasks".
2) Complex query modeling supports a hierarchical set of query models at different abstraction levels, while workflow-based models are represented in just one level.
3) In complex query modeling, query designers ultimately think of all data sets as "relational tables" in a RDBMS; they also think of query design as structuring database operators to work on these "relational tables". In workflow modeling, application developers determine a common data format (e.g., flat file or XML), and use/write software programs to process these data outside the RDBMS.

For the rest of the paper, we present the complex query modeling method using the following structure. First, we will present a practical biological complex query, "how to index all human genes from public cDNA databases to design microarrays?" Second, we show a data model design used to support finding solutions to this complex query. Third, we present four distinct stages of complex query modeling, each using a different set of notations to show solutions at different abstraction levels to the above complex querying problem. Lastly, we discuss the future development of a database query modeling tool, based on our prototypical experimentation, to facilitate the adoption of complex query modeling method in practice.

2. A COMPLEX QUERY EXAMPLE

"How to index all the genes in a target genome from publicly available DNA/mRNA sequence databases?" is a complex biological query, which the designers of spotted cDNA microarrays [9] or high-density microarrays (also called "GeneChips") must answer.

The goal of this complex query is to perform **gene indexing**, or **sequence selections**, i.e., to derive a complete set of high-quality, non-redundant representative sequences to put (called "tile") on microarrays by sifting through millions of mRNA, genes, and expressed sequence tags (ESTs).

To GeneChip designers at Affymetrix, this querying task consists of these complex steps. [10] First, GeneChip designers collect gene "superclusters" to be downloaded from public databases. Then, they "clean out" poor-quality sequences and split initial superclusters into refined clusters called "subclusters"—each corresponding approximately to a gene transcript. Next, they perform a multiple sequence alignment for all sequences within each subcluster to generate an "assembly". Then, they determine a "consensus" (virtual sequence) or an "exemplar" (real representative sequence) for each subcluster assembly. Lastly, they keep most representative design sequences (consensi or exemplars) as the final "design sequences".

This querying task is challenging, because GeneChip designers (primary users) must deal with unknown genome biology issues and large-scale computing issues altogether. On the biology side, users need to address discovery surprises that arise daily. For example, UniGene database was found to have inherited many poor sequence quality problems from GenBank and dbEST databases, contrary to what UniGene database providers had claimed. Therefore, gene indexing cannot proceed unless GeneChip designers developed additional sequence cleaning procedures such as ribosomal RNAs filtering and low-complexity region masking. Sometimes, these added quality-guarding data processing procedures, e.g., subclustering and consensus sequence determination, introduce additional concerns such as the introduction of ambiguity nucleotides into designed consensi or the assembly of wrong sequences into "chimeric consensi". Therefore, manual user examination of intermediate query results and trial-and-error type of experimentation are necessary. On the computing side, sequence selection involves managing millions of sequences and moving them through intricate data processing steps. The starting and ending data sets must be managed by a RDBMS, which also provides intermediate data tracking capabilities during the complex querying process. There are significant challenges for maneuvering large data sets effectively, because the entire querying process will likely run for days on

high-end computer hardware even after the software program are optimized.

In the next section, we show that complex query modeling makes it possible to conquer complex biological database queries.

3. COMPLEX QUERY MODELING

3.1 Preparation: Data Model Design

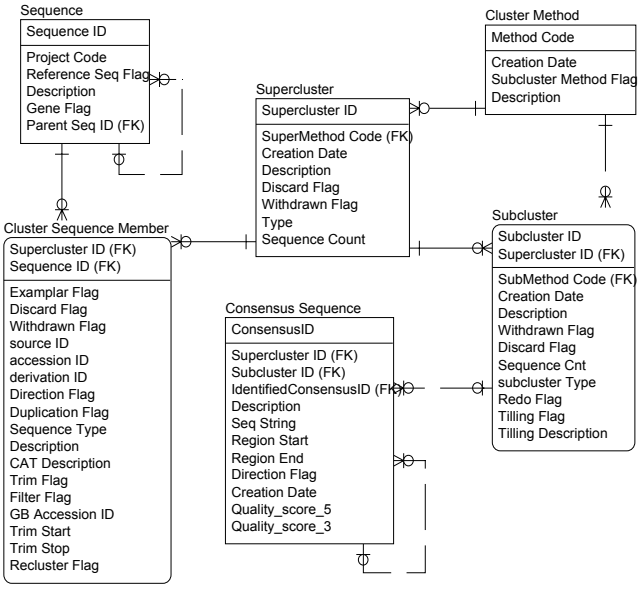

Figure 1. A relational data model (schema fragment) to support the complex querying of GeneChip sequence selections.

In Figure 1, we show a data model for the GeneChip sequence selection system, using a relational data modeling notation described in [11]. A description of how to develop the data model for the GeneChip sequence selection complex query is beyond the scope of this paper. Note, though, that a well-developed data model preludes complex query model development. These relational entities are used directly in the construction of query interface model (as "*virtual schema entities*") and query structure model (as "*relations*"), which we will describe later.

3.2 Overview: Complex Query Modeling

We can divide complex query modeling into four distinct but interdependent phases of design activities (Table 1). At each phase, team members with different roles participate in the modeling process using specific notation and addressing specific concerns. For example, in the second phase, the *architect* and the *query designer* develop "*query interface models*" together; they must transform a maze of *tasks* and *actions* (detailed/structured sub-tasks) produced from the first phase into connected *database query interfaces*, and map data sources to "*virtual schema entities*" (explained later) derived from entities in the data model. As complex query modeling progresses from the first phase to the fourth phase, uncertainty becomes gradually exposed and addressed in increasing levels of details. At the last phase, query models are replaced by high-quality SQL scripts that are optimized for execution in tested RDBMS environments.

Compared to traditional *ad hoc* database query design process using precedence chart [12], complex query modeling can be characterized as the following. First, complex query modeling is too complicated to be completed in a few hours or even a few days, and requires successful collaborations among a whole database query development team. Second, the scale of complex tasks involved in the complex query modeling design requires writing database queries measured by dozens of pages of SQL scripts, which could equate to hundreds of thousands of lines of traditional C/Java codes. Third, complex query modeling requires problem-solving skills at different abstraction levels, from high-level task decomposition to database query optimizations. Fourth, complex query modeling also supports query design outside of the RDBMS environment, as long as the input/output data are tracked in relational tables and the external program can be regarded as a "virtual database operator".

Table 1. An overview of query modeling

		Query Modeling Details		
		Notation	**Concerns**	**Roles Involved**
Query Modeling Phases	**1. Represent High-Level Problem Solutions**	Task-Action Analysis Model	Tasks, actions, and available data stores	Manager, Biologist, Architect
	2. Transform Solutions Into Connected DB Query Interfaces	Query Interface Model	Query interfaces and virtual schema entities	Architect, Query designer
	3. Design Complex Database Query Structures	Query Structure Model	Relational operators and relations	Query designer, SQL programmer
	4. Improve SQL Query Implementation	SQL blocks and stored procedures	Database objects and SQL query scripts	SQL programmer, DBA

3.3 Query Modeling: Phase One

At the first phase of complex query modeling, the goal of a query design team is to engage domain users (biologists) in finding the detailed structures of high-level "*tasks*" (phrases shown in Fig. 1). Complex query modeling at this phase is not much different from designing control flow and data flow models during software development requirement analysis phases. These "*tasks*" become detailed structured subtasks which we call "*actions*", connecting to one another within control flow blocks, and taking available data sources as either the input or the output. The "task-action analysis model" notation and model design concepts are not unique to complex query modeling. For a detailed description of complex query modeling in this level, refer to [13].

3.4 Query Modeling: Phase Two

At the second phase of complex query modeling, the goal of a query designer is to develop "query interface models" with the data architect, who can map the workflow-based "task-action analysis models" onto a design that establishes the connection between user questions and database query vocabularies, i.e., as "*database query interfaces*" and "*virtual schema entities*". For "*database query interface*", we refer to the specification of a

database query using unambiguous English terms. For "*virtual schema entity*", we refer to an entity that is represented in the data model either directly or indirectly (by joining database relations found in the data model).

In Figure 2, we show a query interface model that addresses the "backward compatibility check" *action* found in the last *task* ("to design/pick sequences") of a GeneChip sequence selection complex query design. We use oval or circular boxes as a notation for *query interfaces*, whose titles are always underlined. We use rectangular boxes as a notation for *virtual schema entities*, whose names appear in the top row with a dark background and whose primary keys appear in the top rows of the same box with a light background. We also use thick horizontal lines to represent filters with filtering conditions specified in brackets nearby. The four "*query interfaces*" in the model has been transformed from "*actions*" in the "task-action analysis model", and will be translated into SQL queries in subsequent query modeling phases. The "*virtual schema entities*"—*Design Sequence*, *Old Design Sequence*, *Design Subcluster*, and *Subcluster Match Decision Matrix*—are derived from corresponding entities in the data model (in Fig. 2), which include "*Cluster Sequence Member*", "*Sequence*", "*Subcluster*", and "*Cluster Method*". This query interface model provides query designers with an unambiguous plan how to classify subclusters into several different types (using the shown "*subcluster Maptype*"), which suggest the lineage between new a**nd old GeneChip designs.**

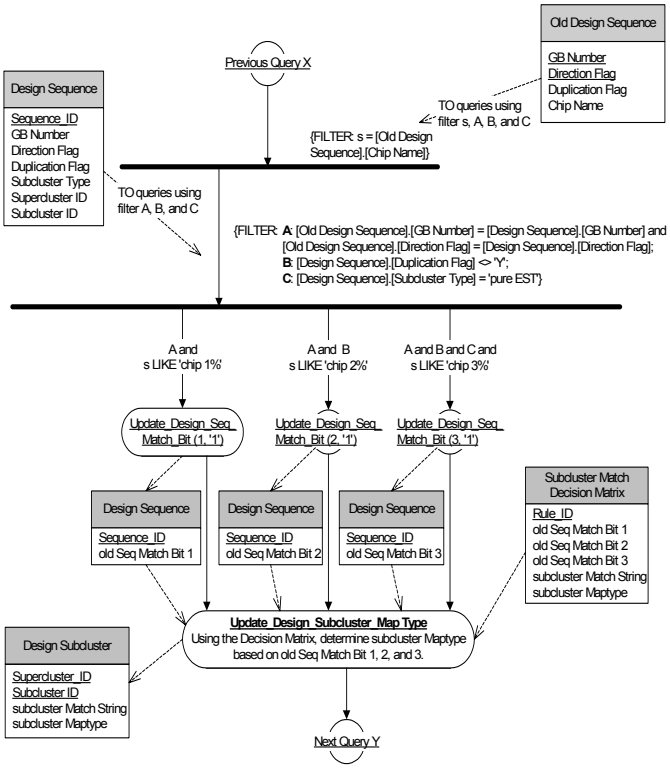

Figure 2. A query interface model to perform "backward compatibility check".

We believe that the query interface model notation can encourage query designers to find complex query solutions using relational database constructs. Compared with the "*task-action analysis model*", query interface model can provide users with notational hints that data should be processed one data set at a time instead of one data item at a time—by using a "filtering line" with guarding filtering conditions instead of using a "branching point" with branching conditions. Furthermore, it liberates the predominant use of looping constructs on data items because of the implied set-oriented processing mechanism.

3.5 Query Modeling: Phase Three

At the third phase of complex query modeling, the goal of a query design team is to further define structures of complex queries from previous query interface models. Each query structure model unravels a query statement by exposing how interleaving relational operators manipulate on relations. Figure 3 shows an example of the query structure model, which shows how to unravel the following query specifications seen in a query structure model:

"Update the consensus sequence type to 'low', if any of the three conditions apply: the sequence contains more than 20% ambiguous bases; the sequence contains less than 3% ambiguous bases, but derives from a subcluster entirely consisting of 'ESTs'; the sequence contains between 3% and 20% ambiguous bases, but derives from a subcluster not entirely consisting of 'mRNA' or 'CDS' sequences".

In this model, we use ovals to represent relational database operators, including "*Filter*", "*Union*", "*Join*", and "*Update*" (not a "true" one). We use rectangles to represent database relations as we did for "*virtual schema entities*". Note that there is a slight distinction among temporary relations (relations labeled as "temp"), intermediate relations (temporary relations with name labels other than "temp"), and permanent relations (relations with relational table names in a dark background). We use directed solid lines pointing towards or away from a database operator to distinguish the query operation's input or output relations.

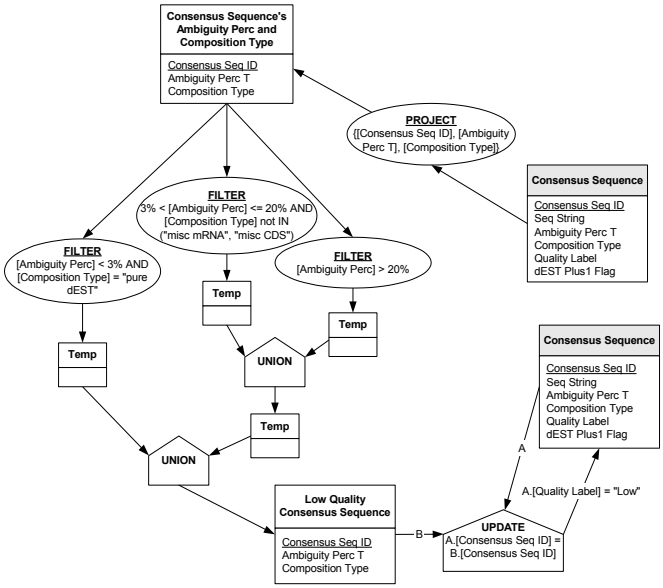

Figure 3. A query structure model to identify low quality consensus sequences.

3.6 Query Modeling: Phase Four

At the fourth and last phase of complex query modeling, the goal of a query design team is to turn fully-developed query structure

models into high-performance SQL scripts. Along with the database administrator (DBA), SQL programmers need to consider how to take advantage of a specific RDBMS indexing and/or SQL query features in order to write efficient SQL statements. The query design team also need to consider adding database transaction control blocks into the SQL scripts to ensure smooth execution of queries that can be restarted in case of hardware failures. For example, the following final SQL scripts are a sample translation of the query structure model (in Fig. 4):

```
/* LOW QUALITY CONSENSUS SEQ SELECTION*/
UPDATE
/*+ PARALLEL (CONSENSUS_SEQUENCE, 6) */
CONSENSUS_SEQUENCE       A
SET              A.Quality_Label = 'Low'
WHERE     A.Consensus_Seq_ID =
(SELECT   B.Consensus_Seq_ID
 FROM     CONSEQ_AMBPERC_AND_COMPTYPE B
 WHERE    B.t > 0.2
   UNION
 SELECT   B.Consensus_Seq_ID
 FROM     CONSEQ_AMBPERC_AND_COMPTYPE B
 WHERE    B.t > 0.03 AND B.t <= 0.2 AND B.c
NOT IN ('misc mRNA', 'misc CDS')
   UNION
 SELECT   B.Consensus_Seq_ID
 FROM     CONSEQ_AMBPERC_AND_COMPTYPE B
 WHERE    B.t < 0.03 AND B.c = 'dEST'
);
```

Note that in the comment line of the above SQL script, the content between /*+ and */ is a Oracle 9i RDBMS hint to the SQL query engine, indicating that this query should be performed on the *Consensus_Sequence* table using 6 threads.

4. RESULTS

Complex query modeling has been successfully applied to the gene indexing complex querying process at Affymetrix during the GeneChip design of the rat, human, and mouse genome microarrays [10]. We take the design of the human genome microarray for example. The query design team, which included one of us authors, began the complex querying with more than 1.3 million UniGene sequences from 88,703 superclusters. Using Microsoft Visio as the complex query modeling tool, the team developed a complete task/action analysis model, a complete query interface model, and a set of query structure models for query specifications that are challenging to translate into SQL directly. The SQL scripts, which the team developed over the course of approximately 6 months, were several thousand lines long and took a Sun E10000 machine (with 11 CPUs and 16GB of memory) more than 10 CPU hours over a period of 2 weeks to complete its execution. The end result of this complex querying is a collection of 84,395 consensus sequences (each likely corresponding to a transcript variant form of human mRNA) from 139,250 intermediate subclusters. During the six months of complex querying, our team developed increasingly detailed query models with hundreds of decomposed query specifications.

5. DISCUSSION

Complex query modeling is a novel method that we developed to support complex database querying. Through a biological querying example, we have found that evolving querying solution over time while engaging the entire query design team at different phases of the query development stage to be critical to our success. Complex query modeling method enables the user and the query design team to spend time together on communicating scientific discovery needs or results, instead of on software implementation details. The final SQL scripts, although long and complicated, becomes easily maintainable, since these scripts have been developed using modular approaches—each module corresponding to the decomposed query specification from the query interface model as query structures captured in the query structure model. Note that even though we use biological complex queries in this work, the significance of our method may not be limited to the biology domain only; in fact, our method may be applicable to any knowledge application domains with similar high-level open-ended explorative querying needs.

Future work in complex query modeling will be both exciting and challenging. Major research issues such as "Are there common patterns in query models, which may help beginning query designers acquire query modeling skills quickly?" and "What are the query modeling principles?" still remain open. To expand the impact of complex query modeling in practice, researchers need to consider developing practical complex query modeling design tools as the next step.

6. REFERENCES

1. Persidis, A., *Bioinformatics.* Nature biotechnology, 1999. **17**: p. 828-30.
2. Lawrence, P., *Workflow handbook 1997*. 1997, Chichester; New York: John Wiley. xxiii, 508.
3. Lakshmanan, L.V.S., F. Sadri, and S.N. Subramanian, *SchemaSQL: An extension to SQL for multidatabase interoperability.* ACM Transactions on Database Systems, 2001. **26**(4).
4. Chen, C.X., J. Kong, and C. Zaniolo. *Design and Implementation of a Temporal Extension of SQL*. in *19th International Conference on Data Engineering*. 2003.
5. Egenhofer, M.J., *Spatial SQL: A Query and Presentation Language.* IEEE Transactions on Knowledge and Data Engineering, 1994: p. 86-95.
6. Etzold, T., A. Ulyanov, and P. Argos, *SRS: information retrieval system for molecular biology data banks.* Methods Enzymol, 1996. **266**: p. 114-28.
7. Miled, Z.B., et al. *SIBIOS: A System for the Integration of Bioinformatics Services*. in *Second International Workshop on Challenges of Large Applications in Distributed Environments*. 2004.
8. Stevens, R., et al., *TAMBIS: transparent access to multiple bioinformatics information sources.* Bioinformatics, 2000. **16**(2): p. 184-5.
9. Schena, M., et al., *Quantitative monitoring of gene expression patterns with a complementary DNA microarray.* Science, 1995. **270**(5235): p. 467-70.
10. Chen, J.Y. and J.V. Carlis. *Managing Bioinformatics Challenges in Expression Microarray Sequence Selection Projects*. in *Proceedings of the Second Chinese Conference on Bioinformatics*. 2002. Beijing, China.
11. Chen, J.Y. and J.V. Carlis, *Genomic Data Modeling.* Information Systems, 2003. **28**(4): p. 287-310.
12. Carlis, J.V. and S. Krieger, *Mastering Database Analysis*. 2004, (to be published): Addison-Wesley.
13. Chen, J.Y., *PhD Thesis: A Bioinformatics Discovery-oriented Framework*. 2001, University of Minnesota: Minneapolis.