Conceptual Modeling of Dynamic Interactive Systems Using the Equivalent Transformation Framework

*COURTNEY POWELL, KIYOSHI AKAMA
Information Initiative Center,
Hokkaido University,
Kita 11 Nishi 5, Sapporo, 060-0811,
JAPAN

http://assam.cims.hokudai.ac.jp/laboe/eti.html

Abstract: - Conceptualizing, visualizing, reasoning about and implementing Dynamic Interactive Systems (DISs) are difficult and error-prone activities. To conceptualize and reason about the sorts of properties expected of any DIS, a framework that most naturally models DISs is essential. The declarative paradigm is closer than any other to the abstract behavior of DISs. In this paper we propose and explain why the Equivalent Transformation Framework (with its declarative roots) is an ideal framework for conceptually modeling DIS. The benefits to be derived from using this framework include guaranteed system correctness, high level abstraction, clarity, granular modularity, and an integrated framework for reasoning about, manipulating, and optimizing the various aspects of DISs.

Key-Words: - Conceptual Modeling, Dynamic Interactive Systems, Equivalent Transformation, Correctness, Abstraction.

1 Introduction

Dynamic Interactive Systems (DISs) consist of independent objects interacting with each other and changing states dynamically over time. The interaction involves both events which occur at specific moments and more persistent status phenomena which can be observed any time. Practical DIS systems consist of multiple objects operating concurrently (i.e. parallel and communicating).

Conceptualizing, reasoning about, and implementing a DIS are difficult and error-prone activities. In addition, as argued in [1], the cause of many other additional difficulties in DIS system construction is the fact that traditional programming languages are algorithmic, and thus best suited to writing programs that acquire all their inputs before execution and produce a result only on termination. In contrast, DISs by their nature obtain inputs and output results throughout the life of the program. As a result DISs are incompatible with algorithmic languages. Current methods used to overcome this incompatibility add to system complexity.

Even though a variety of abstractions have been developed for modeling and reasoning about interactive systems (and by extension DISs), there is a lack of a coherent and robust paradigm for building robust DISs. Algorithmic languages treat programs as black boxes which produce final values on termination [1]. However, DISs are open to observation and influence from outside and must be able to adjust their internal states in response to each interaction in order to maintain the consistency of the computation.

Humans are incapable of reasoning properly using the technical logic used in computers; doing this only results in innumerable mistakes [3]. Therefore it is of vital importance that rules be developed for human reasoning. For DISs, modeling provides the means of doing this. Good models result in natural flow of programming from idea to implementation; manageable modules and a means of modification without major reorganization of the software. On the flip side, bad models can result in a "series of nasty surprises" [2]. These include interfaces that become clumsy because they are forced to accommodate unexpected interactions, and difficulty making even the simplest changes. The many difficulties involved when such a bad abstraction is used oftentimes can only be corrected by starting over from scratch (i.e. from the idea stage). Thus, we can see that using the right model is of paramount importance.

The ET Framework [4, 6] not only supports interaction and dynamism directly (stemming from its declarative foundation) but can also efficiently adjust its computation to input changes. As a result it can be used to model DISs in abstract and then transform the optimized program into the target language. In addition, the ET Framework is very close to how humans actually think and so, we believe that by using the ET Framework to model systems we can bridge the gap between the type of reasoning that comes naturally to humans and the type of technical logic required by computers with regards to DISs.

2 Dynamic Interactive Systems Scope

Four of the most important concepts in DISs are:1) objects; 2) events; 3) state; and 4) interaction. An object can be regarded as any item that can be individually selected or manipulated, e.g. a data file, or piece of text. An event is something that occurs in a specific place at a particular instance in time, or interval of time. As a result, it is regarded as an atomic, non-persistent occurrence. A state, on the other hand, persists and has a measurable value at any given moment. Interaction takes the form of both object to object communication and environment to object communication.

Databases are a form of DIS. In database systems multiple concurrent users access, view and may modify data (i.e. the attributes of entities). As a result of the access and modification by multiple users the data in the database is constantly changing and can thus be viewed as being dynamic. Interaction takes the form of users interacting with the data.

Another form of DIS is Web based systems. These systems must also have the ability to accommodate access by multiple concurrent users.

DISs range from the very simple (e.g. a Ping-Pong game) [7] to the very complex (e.g. a climate system).

2.1 Features of Dynamic Interactive Systems

Two of the major features of DISs are: (1) Independence and; (2) parallelism (concurrency). All objects in the system operate independently of each other. As a result, the state of, and the standard actions carried out by an object are not dependent on the other objects in the system. Parallelism refers to the fact that all objects can carry out various (similar and/or dissimilar) actions concurrently.

DISs consist of two sides: (1) Procedural and; (2) declarative. The procedural side includes simple procedural oriented applications, while the declarative side includes database systems and takes into consideration such attributes as correctness.

3 Modeling Dynamic Interactive Systems using ET

Equivalent Transformation (ET) [4, 6] is a new computational paradigm that is based on semantic preserving clause transformations carried out by sets of rewriting rules generated from specifications. In the ET Framework (ETF), a given complex problem is transformed successively and equivalently into a simpler problem until a problem from which answers can be directly or easily obtained is reached.

3.1 Representation and Computation with ET Rules

An ET rule describes methods of rewriting various clauses into other clauses (or sets of clauses). A rule specifies, in its left-hand side, a pattern of atomic formulas to which it can be applied, and defines the result of its application by specifying, in its right-hand side, one or more patterns of replacement atoms. The rule is applicable to a definite clause when the pattern in the left-hand side matches atoms contained in the body of the clause. When applied, the rule rewrites the clause into a number of clauses, resulting from replacing the matched body atoms with instances of the patterns in the right-hand side of the rule. The actual computation of the solution to a problem is accomplished by the repeated application of equivalent transform rules.

There are two types of ET rules. These are: 1) D-Rules (deterministic) and, 2) N-Rules (non-deterministic).

3.1.1 ET Variables

The ETF consists of named and anonymous (unnamed) variables. A named variable begins with the asterisk symbol, '*' and is followed by a letter, numeral, etc. (e.g. *X, *X1, *5). An anonymous variable begins with the question sign symbol '?' and may or may not be followed by a letter or numeral. (e.g. ?, ?X, ?5). Wherever a named variable changes, other variables having the same name will change simultaneously. In contrast to this, anonymous

variables, even if they have the same name, are each treated as different variables.

Objects can be richly expressed using information-attached variables (variables to which information has been attached). This type of variable has the format: **x~(information)*. Here '**x*' represents the information-attached variable; *(information)* is the attached information; and the sign tilde (~) ties the information to the variable.

An ET variable exists only in the rule in which it was created, so it is limited in scope. For example, a variable, **var1*, created in a rule, *rule1*, will be different from a variable with the same name, **var1*, in rule, *rule2*. As a result, variable communication takes place only within the rule in which the variable was created. Since ET uses pattern matching, instead of determination of input clauses by unification, inter-rule communication is in the form of values, not names.

ET variables are immutable, i.e., once an ET variable is assigned a value that value does not change. After a particular rule has been applied to a clause, the variables (and by extension the values they contain) in that rule, are destroyed. In the next computation cycle, new variables are created and new values assigned. In this way, values are changed in ET rules.

3.1.2 Object Creation and Termination

In the ETF, a problem is formulated as a declarative description, represented by the union of two sets of definite clauses, one of which is called the definition part, and the other the query part. The definition part provides general knowledge about the problem domain and descriptions of some specific problem instances. The query part specifies a question regarding the content of the definition part. From the definition part, a set of ET rules is prepared. The problem is then solved by transforming the query part successively, using the ET rules, into another set of definite clauses from which the answers to the specified question can be obtained easily and directly.

The query part described above can be used to represent an object. This object is expressed as a definite clause in the format:

$$\text{head} \leftarrow \text{atom}_1, \text{atom}_2, \text{atom}_3, \text{atom}_4, \text{atom}_5.$$

Where *head* represents the object and the body atoms represent the internal components of the object.

In the ETF computation of a program takes the form of state transitions, where problems are regarded as states. A final state is a problem that consists of only unit clauses, which is of the form:

$$\text{head} \leftarrow.$$

The computation of a program *prg* on a problem *prb* is a nonempty finite or infinite sequence $com = [st_0, st_1, st_2, ...]$ of states such that $st_0 = prb$ and the following conditions are satisfied: (1) for any two successive states st_i, st_{i+1} in *com*, st_i is not a final state and *prg* transforms st_i into st_{i+1} in one step; (2) if *com* is finite, then *last(com)* is the final state or *prg* is not applicable to *last(com)*, where *last(com)* denotes the last element of *com*.

If *com* is finite and *last(com)* is the final state, then the answer set obtained from *com* is the set

$$\{g \mid ((a \leftarrow) \in \text{last(com)}) \,\&\, (g \text{ is a ground instance of } a)\},$$

and is undefined otherwise.

As a result, each state transition carried out by the ET program will result in a change in state of the object (an instance of the object). This is represented by the state of the definite clause at each successive transformation stage during the application of ET rules. The unit clause that remains at the end of computation represents the terminated object.

3.1.3 ET and Event-driven Semantics

The notion of events plays a central role in the construction of most software that involves interaction or simulation. However, in these systems, the events are often just symbols with no intrinsic meaning. ET rules easily satisfy event-driven semantics and also provide intrinsic meaning to the events. If we regard events as inputs from the environment (inputs originating outside of the rule), then an ET rule that possesses a condition section will be inherently event driven. That is, the condition section has to be satisfied before the rule can be executed. The condition to be satisfied is dependent on forces external to the rule.

3.2 Modeling using ET

DISs comprise a number of objects, interaction between objects and, interaction between objects and their environment. In the ETF, these are realized by

means of atoms, rules, and events (installed predicates).

3.2.1 Object Representation and Manipulation

If we use an atom to represent an object, the internal state of the object (atom) can be represented by a definite clause of the form:

$$\text{atom} \leftarrow \text{atom}_1, \text{atom}_2, \text{atom}_3, \text{atom}_4, \text{atom}_5.$$

To move this atom and describe its interactions with other atoms and the environment, rules are used. A typical rule is of the form:

$$\text{atom}_1, \text{atom}_2, \{\text{event}\} \Longrightarrow \{\text{specialization}\}, \text{atom}_3, \text{atom}_4.$$

Some of the ways in which this rule can be used are: (1) Changes in atoms appearing in the rule are expressed by means of atom replacement and specialization; (2) atoms that are changed without the changes appearing in the rule (i.e. changes are not transparent to the rule) are expressed by means of specialization; (3) changes influenced by the environment are received and examined via events.

3.2.2 Advantages of the Description Methods

Some of the advantages of these description methods are: (1) The model used is a combination of the clause and rule models, which are both well known models; as a result, it has clarity and significance and is general purpose; (2) the modification being expressed by the rule is localized so efficient execution is possible; (3) as a result of their high level of independence, rules are very easy to write and; (4) events are handled uniformly as installed predicates.

3.2.3 Analysis of Interaction

In the modeling of DISs we can divide its domain into two parts – object and environment. Interaction between objects is represented using ordinary rules. Interaction between objects and the environment is expressed by means of event-driven rules. In this case interaction is achieved through: (1) Change based on replacement with rule (head and body); (2) change based on specialization with rules (through the medium of variables, or even other atoms); (3) through the use of *getContext* it is possible to obtain the status of atoms that are not head atoms and; (4) through the use of events influences from the environment can be accommodated.

4 The Declarative Side: Correct Problem Solving

In this section we will explain some of the features of the declarative side as it relates to ET and DISs.

4.1 Definite Clause Set and its Meaning

A definite clause, C, is an expression of the form $h \leftarrow b_1, b_2, \ldots, b_n$, where $n \geq 0$. h is called the head of C and is denoted by $head(C)$. The set $\{b_1, b_2, \ldots, b_n\}$ is called the body of C and is denoted by $body(C)$. When $n = 0$, C is said to be a unit clause. When all atoms appearing in the definite clause, C, are ground atoms, C is said to be a ground clause. The set of all definite clauses is denoted by *Gclause*. A declarative description is a set of definite clauses.

A set of ground atoms represented by a declarative description, P, is called the meaning of P, denoted by $M(P)$, and is defined as:

$$\mathcal{M}(P) \stackrel{\text{def}}{=} \bigcup_{n=1}^{\infty} [T_P]^n(\emptyset),$$

Where:

$$T_P(x) = \{head(C\theta) \mid C \in P,\ \theta \in \mathcal{S},\ C\theta \in Gclause,\ body(C\theta) \subseteq x\}.$$

x is an arbitrary set of ground atoms, θ a specialization, S a set of specializations. $M(P)$ is a least fix point of T_P , and agrees with a least model when a definite clause is regarded as a logical formula.

4.2 The Intersection Problem

Let D be a set of definite clauses. Let Q be the set of atoms which represent the set of all queries for D. Then a problem is given in the form of the pair (D, Q). For any atom α, let $rep(\alpha)$ denote the set of all ground instances of α. Consider computing the solution set A which satisfies $A = M(D) \cap rep(q)$ for a query $q \in Q$.

Theorem 1. For a query q, let q' be the atom obtained from q by changing its predicate into one that does not appear in D. Then:

$$M(D) \cap rep(q) = \{q \mid q' \in M(D \cup \{q' \leftarrow q\})\}.$$

Theorem 2. For a set of unit clauses, denoted by F,

$$\mathcal{M}(D \cup F) = \bigcup_{(a\leftarrow)\in F} \mathrm{rep}(a).$$

A rule r is an ET rule in D, iff

$$(S, S') \in r \Rightarrow M(D \cup S) = M(D \cup S') \quad (1)$$

for arbitrary sets S and S' of definite clauses. Let Q_0 be a set of definite clauses that is defined by $Q_0 = \{ans(q) \leftarrow q\}$. Let also Q_n $(0 \leq n < \infty)$ be a set of unit clauses. Assume that we have a transformation sequence $Q_0 \rightarrow \ldots \rightarrow Q_n$ obtained by applying ET rules successively. Then from Theorems 1, 2, and (1):

$$\begin{aligned} \mathcal{M}(D) \cap \mathrm{rep}(q) &= \mathcal{M}(D \cup Q_0) \\ &\vdots \\ &= \mathcal{M}(D \cup Q_n) \\ &= \bigcup_{(a\leftarrow)\in Q_n} \mathrm{rep}(a). \end{aligned}$$

$$A = \bigcup_{(a\leftarrow)\in Q_n} \mathrm{rep}(a).$$

A program in the ET model is a set of rules which executes ET such that:

$Q_0 \rightarrow \ldots \rightarrow Q_n$
$M(D \cup Q_0) = \ldots = M(D \cup Q_n)$

starting from $Q_0 = \{ans(q) \leftarrow q\}$ for all $q \in Q$, and finally computes A [5].

4.3 Correctness

Discussions of correctness must take into consideration the intended meaning of a program. An intended meaning of a program is a set of ground goals. A program P is correct with respect to an intended meaning M iff $M(P)$ is contained in M. That is, the program should do only what we intended it to.

In a Rule-based Equivalent Transformation (RBET) framework, such as the ETF, the correctness of computation relies solely on the correctness of each transformation step. Given a declarative description $D \cup Q$, where D and Q represent the definition and query parts of a problem respectively. The query part Q is said to be transformed correctly in one step into a new query part Q', by the application of a rewriting rule, iff the declarative descriptions $D \cup Q$ and $D \cup Q'$ are equivalent, i.e., they have the same declarative meaning. A rewriting rule is considered to be correct, iff its application always results in a correct transformation step.

4.4 ET Computation and Correctness

In the ET model a problem is represented as a set of definite clauses, and a specification is a pair (D, Q), where D is a set of clauses representing background information (i.e. general knowledge about an application domain and description of particular domain instances), and Q is a set of problems, each of which is also a set of definite clauses. It is required that for each problem $q \in Q$, the predicates occurring in the heads of clauses in q occur neither in D nor in the bodies of the clauses in q.

A program in the ET model is a set of rewriting rules and program computation consists of successive rule application. A program *prg* is partially correct with respect to a specification $S = (D, Q)$ iff for each $q \in Q$, *prg* yields the correct answer set to q whenever it transforms q into a set of unit clauses in a finite number of transformation steps. It is totally correct with respect to S iff it is partially correct with respect to S and it always terminates with a set of unit clauses when executing each problem in Q.

5 Why the ET Framework can Effectively Model DISs

The following features of the ETF are among the many reasons why it is ideal for modeling DISs:

1. *Clarity* - ET rules are intuitively understandable and, in the ETF the states of objects are clearly discernable as the state of its computation is clearly shown. This type of clarity allows us to check whether or not a change of state is valid.
2. *Rich Expressivity* - The status, properties and interactions associated with an object can be richly expressed in ET using information-attached variables and ET rules.
3. *Nondeterminism and Parallelism* – In order to reliably model independent concurrent objects and their interactions the ability to simulate parallel processes is invaluable. The inherent nondeterministic nature of the ETF gives us the ability to simulate either of three (3) types of parallelism. These are: (1) OR-parallelism; (2) AND-parallelism and; (3) Rule-parallelism (unique to the ETF).

4. *High Level Abstraction* - An abstraction is an idea reduced to its essential form [2]. It provides us with the ability to focus on a concept while safely ignoring some of its details (i.e. different details are handled at different levels). The best abstractions capture their underlying ideas naturally and convincingly and provide a means of visualizing, expressing, analyzing, manipulating, and optimizing an idea before commitment to code. Without abstractions systems tend to be overly complex and intellectually hard to manipulate. Languages that support abstraction are needed in order to create intellectually manageable programs. The ETF operates at the conceptual level and so it both supports and provides a high level of abstraction. As a result the system can be freely manipulated and optimized without the restrictions associated with concrete implementation details such as type declaration, memory allocation, etc.
5. *Independent Rules* - The highly independent nature of ET rules further strengthens the capability of the ETF to provide powerful abstractions. As each rule can be written and focused on exclusively, we are able to use different rules to safely and independently represent differing types and levels of details in an abstraction.
6. *Dynamic Addition and Deletion of Rules* - In the ETF rules can be dynamically added and deleted. This gives additional versatility in the real-time representation of new information and the dynamic addition and deletion of objects.
7. *Natural Connection to Aspects of Database Systems* - The ETF connects naturally to the semantics and reasoning underlying database systems. Atoms can be used to represent entities. The relation between entities and their interactions can be represented by rules.
8. *Guaranteed Correctness* - The structure of the ETF guarantees correct operation of the system. This was explained fully in sections 4.3 and 4.4.
9. *Declarative Semantics* - DISs inherently satisfy a declarative model of computation [1]. They are required to accommodate new information at random points in time, while maintaining the consistency of their computations. This is easily done in the declarative paradigm while in the algorithmic paradigm new information may render the algorithm completely useless. The underlying declarative semantics of the ETF provides us with a means of connecting directly to the underlying nature of DISs and thus enables us to visualize and model all of its various aspects.
10. *Integrated System* - The ETF is an integrated modeling system, i.e., it is able to model the entire DISs' spectrum without the need for any component external to the framework.

6 Conclusion

In this paper we looked at some of the problems that currently obtain in the construction of Dynamic Interactive Systems (DISs); and examined why and showed how the ET framework (ETF) can be used to overcome these difficulties. We also explained how the ETF can give a comprehensive conceptual model for DISs, which is intuitive, coherent, robust and correct. This model of DISs in the ETF, and the benefits contained therein, can be easily implemented in the language(s) of choice by transformation.

References:

[1] X1. R. Perera, Programming Languages for Interactive Computing, Dynamic Aspects, http://dynamicaspects.com/papers/FInCo2007.pdf, March 2007.

[2] X2. D. Jackson, *Software Abstractions: Logic, Language, and Analysis*, The MIT Press, 2006.

[3] X3. B. Mills, *Theoretical Introduction to Programming*, Springer-Verlag, 2006.

[4] X4. H. Koike, K. Akama, and H. Mabuchi, A Programming Language Interpreter System Based on Equivalent Transformation, *Proceedings of Intelligent Engineering Systems* (INES '05), 2005, pp. 283 – 288.

[5] X5. S. Miyajima, K. Akama, H. Mabuchi, and Y. Wakamatsu, Detecting Incorrect Rules Automatically in Equivalent Transformation Programs, *Proceedings of the 2nd International Conference on Innovative Computing, Information and Control* (ICICIC 2007).

[6] X6. The ET Framework http://assam.cims.hokudai.ac.jp/laboe/eti.html.

[7] X7. C. Powell and K. Akama, Structured Development of DHTML Programs from Abstract Ideas Based on the Equivalent Transformation Framework, *Proceedings of the 2nd International Conference on Innovative Computing, Information and Control* (ICICIC 2007).