Extending the Equivalent Transformation Framework to Model Dynamic Interactive Systems

*COURTNEY POWELL, KIYOSHI AKAMA
Information Initiative Center,
Hokkaido University,
Kita 11 Nishi 5, Sapporo, 060-0811,
JAPAN
*kotoni@uva.cims.hokudai.ac.jp, akama@iic.hokudai.ac.jp
http://assam.cims.hokudai.ac.jp/laboe/eti.html

Abstract: - Conceptualizing, visualizing, analyzing, 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 formal framework that most naturally facilitates conceptualization and modelling of DISs is essential. In this paper we propose and explain why extending the Equivalent Transformation Framework to conceptually model DISs satisfies this ideal. The benefits to be derived from using this framework include a simplified and intuitive conceptualization process, mathematically sound models, 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 Modelling, Dynamic Interactive Systems, Equivalent Transformation, Correctness, Formal Methods.

1 Introduction

Dynamic Interactive Systems (DISs) consist of independent objects interacting with each other and changing states dynamically over time. The interactions comprise both object-to-object and environment-to-object interactions. Practical DISs consist of multiple independent objects interacting concurrently.

Even though a variety of techniques such as process algebras and UML can be used to model interactive systems (of which DISs is a subset); they tend to be fraught with various challenges including high-learning curve, high generality, high complexity, and low reasoning facility. In addition considerable planning effort is usually required and expert knowledge of the modelling system assumed. As a result, there is a lack of a coherent framework for systematically developing robust DISs.

The framework required must be able to provide a structured, simple, and intuitive means by which an idea can be easily converted to a model in small increments; eliminating the need to mentally juggle various different aspects of the idea all at once. This would serve to neutralize errors and difficulties due to conceptualization of the idea (widely regarded as the most challenging part of programming) [1, 2]. The model thus created must also conform to Formal Methods [3, 4], providing a logical structure able to facilitate rigorous analysis before decision is taken to write the actual implementation in which it holds.

The DIS model construction challenge can be summed up in one question. How can we get from an idea for a DIS to a rigorously analyzable model in a way that is intuitive, systematic, speedy, minimizes errors, facilitates easy manipulation and modification and allows us to build intellectually manageable systems?

The ET Framework (ETF) [5] has already been used to extend reasoning to UML [6], and XML [7], and also to develop correct databases [8]. It is very close to how humans actually think and, not only supports interaction and dynamism directly, but is also amenable to rigorous mathematical analysis. As a result, we believe that it is the ideal framework for resolving the DIS model construction challenge. This paper describes the ease with which the ETF can be used to create robust and intellectually manageable models of DISs that can be rigorously analyzed.

In this paper we first give an overview of DISs, including relevant features, concepts and the state of the art (section 2). An overview of the ETF is given in section 3 and details of how it is extended for modelling DISs are given in section 4. In section 5 we delve into the declarative side of the ETF, explain its computation method and outline its correctness. In section 6 we explain in detail some

of the reasons why we think the ETF is ideal for modelling DISs. We demonstrate practically how models are constructed in the ETF by using the example of a ping-pong game (section 7), and compare and contrast some other approaches to DIS modelling in section 8.

2 Dynamic Interactive Systems Scope

We think of Dynamic Interactive Systems (DISs) as event-driven systems in which independent entities interact with each other and their environment concurrently, and change states dynamically over time. They are capable of accommodating and processing multiple concurrent inputs, and generating multiple concurrent outputs simultaneously. The complexity of DISs can be quite formidable as a result of the interactive processes occurring concurrently over multiple interaction streams

Independence, concurrency, and dynamism are three major features of DISs and each entity is totally self-contained and has its own domain and sphere of influence. In the traditional design and implementation of DISs, five important concepts are: 1) Object; 2) Event; 3) State; 4) Message passing and; 5) Interaction.

A climate system is an example of a natural DIS; in which there are entities such as land, atmosphere, and oceans interacting. Another example of a DIS is a Web application such as an airline reservation system. These systems usually have a database backend which is accessed, viewed, and modified by multiple concurrent users. As a result of the access and modification by multiple users the data in the database is constantly changing and is thus dynamic. Interaction takes the form of users interacting with the data. In such a system correctness is very essential. A Ping-pong game is a simple example of a recreational DIS.

2.1 DIS – State of the Art

Much research has been done, and is still ongoing, in the modelling of Interactive Systems (ISs) in general. Some of the approaches taken are relevant only to static ISs – ISs whose states change only in response to an input. Others however, can be related to DISs – ISs whose internal states are always in flux. These approaches can be grouped into three basic categories, roughly similar to those outlined in [9], as :1) User-centric ;2) System-centric and ;3) Hybrid.

The User-centric approach is based on Human-Computer Interaction (HCI) and employs devices such as task models and storyboards. User Action Notation (UAN) (a user and task oriented notation) and the fairly recent graphical notation Concur Task Trees (CTT) [10] are some of the notations used. User-centric approaches include [11] and novel approaches such as that done using Stochastic Methods [12]. In the System-centric approach, the focus shifts from the user to the underlying dynamics of the system itself. The focus on status and events in [13] exemplifies this approach. Examples of the hybrid approach include [14] (which provides a framework and a language (IMML) in which models constructed from both HCI and Software Engineering (SE) can be combined) and [15] (in which both HCI and SE are developed synergistically).

3 The Equivalent Transformation Framework

The Equivalent Transformation Framework (ETF) [5] is a rule-based framework in which sets of rewriting rules generated from specifications are used to carry out semantic preserving clause transformations. The ETF has mathematically defined syntax and semantics and is therefore amenable to Formal Methods – which is essential for the creation of models that can be rigorously analyzed.

An ET rule describes methods of rewriting various clauses into other clauses (or sets of clauses), and has the general form:

head, {condition} ==> {execution}, body.

This rule reads: if the pattern of atomic formulas specified by *head* matches the target expression (in the body of a clause) and the condition(s) specified by *condition* is satisfied then the built-in or user-defined operation(s) specified in the *execution* section are performed and the expression(s) specified in *body* replaces the target expression. The actual computation of the solution to a problem is accomplished by the repeated application of ET rules.

There are two types of ET rules. These are: 1) D-Rules (deterministic) and, 2) N-Rules (non-deterministic). Non-deterministic ET rules can have multiple *heads*, *bodies*, and *execution* sections. As a direct result of their non-deterministic nature, N-rules can be used to express several forms of parallel processing [16].

An ET variable is immutable, and exists only in the rule in which it was created; 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. 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.

One of the many features unique to the ETF is the concept of *'Information-attached' ('information-bound')* variables or *'I-vars'*, which allow the specification of information in a variety of ways. This type of variable has the format: **x~(information)* - where *'*x'* represents the information-attached variable; and *(information)* is the attached information.

3.2 Model Creation in the ETF

A model in the ETF consists of a set of equivalent transformation rules and conditionalities for the control of the application of these rules. One ET rule is an independent and executable unit component of an ET model. Thus a model can be made by simply writing each rule separately and then adding them one by one to construct the complete model, as illustrated in Fig. 1. In addition, since all rules are independent of each other, they can be mixed and matched in any combination to give the most efficient output.

In the ETF we take a "top-down" approach to model building. The model is constructed starting from the highest level of abstraction; which is represented by the global ET rule (such as those in section 7). Lower level (second level) ET rules are then written for each section of the global rule that calls a sub-computation. For each sub-computation of these second level rules, more rules are written; and so on until only built-ins are called by a rule.

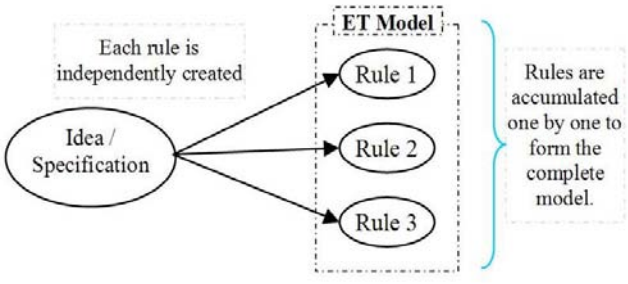

Fig. 1. Model creation in the ETF

4 Modelling DISs in the ETF

In this section we will explain the process of extending the ET Framework (ETF) to model Dynamic Interactive Systems (DISs).

4.1 DIS Representation in the ETF

In the ETF a problem is formulated as a declarative description, represented by a pair of clauses *(D, Q)*. Here *D* represents general knowledge about an application domain and description of particular domain instances, and *Q* specifies a question regarding the content of the definition.

From the definition part *D*, a set of ET rules is prepared. The problem is then solved by transforming the query part *Q* 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 *Q* can be used to represent a DIS.

4.1.1 ET and Event-driven Semantics

ET rules easily satisfy event-driven semantics and also provide intrinsic meaning to the events. We regard events as inputs from the environment (inputs originating outside of a rule). Thus, an ET rule that possesses a condition section is inherently event driven. That is, the condition section has to be satisfied before the rule can be executed. An event-driven rule has the form:

$$atom_1, atom_2, \{event\} ==> \{specialization\}, atom_3, atom_4. \quad (1)$$

4.2 Modelling 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 *information-attached* variables, atoms, rules, and events (installed predicates).

4.2.1 DIS Representation and Manipulation

If we use the query clause *Q* (described in section 4.1) to represent a DIS, the internal components of this DIS will be represented by the body atoms of the definite clause as shown in (2):

$$head \leftarrow atom_1, atom_2, atom_3, \ldots, atom_n. \quad (2)$$

Here, each body atom represents the domain of an object. As a result, object domains (and by

extension objects) can be easily added or deleted by simply adding or removing the relevant body atoms.

Actual objects are represented by information-attached variables (explained in section 3). The attached information can contain any user-defined data relevant to the object, such as object definition, flags, state, etc.

To manipulate these objects in the system and describe their interactions with each other and the environment, rules are used. Typical rules are of the forms depicted in (1) and (3); and both rules may or may not have an *execution* (specialization) section.

$$atom_1, atom_2 ==> \{specialization\},\ atom_3, atom_4. \quad (3)$$

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

4.2.2 ETF Computation and State Transitions

In the 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. Computation takes the form of state transitions, in which each problem is regarded as a state. A final state is a problem that consists of only unit clauses, which is of the form:

$$head \leftarrow .$$

The computation of a program *prg* on a problem *prb* is a nonempty finite or infinite sequence $com = [st_0, st_1, st_2, \ldots]$ 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

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

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.

4.2.3 Analysis of Interaction

In the modelling of interactions in DISs we can divide its domain into two parts – object and environment. Interaction between objects is represented using ordinary rules; with the format of *rule (3)*. Interaction between objects and the environment is expressed by means of event-driven rules; such as *rule (1)*. 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.

4.2.4 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 each 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. In addition, the notation format is intuitively understandable.

5 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.

5.1 Definite Clause Set and its Meaning

In the ETF a problem is represented as a set of definite clauses. 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. If all atoms appearing in *C* are ground then *C* is a ground clause. The set of all definite clauses is denoted by *Gclause*. A declarative description is a set of definite clauses, *P*. Its meaning, *M(P)*, is defined as:

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

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

Here *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.

5.2 Solving the Intersection Problem

Let *D* be a set of definite clauses. Let $\mathbb{Q}$ 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,* $\mathbb{Q}$*)*. Let $\mathbb{S}$ be the set of all sets of definite clauses, where each set consists of *ans-clauses* whose bodies are composed of atoms on *D*. Let *S* and *S′* be arbitrary sets of definite clauses in $\mathbb{S}$. A rule r : $\mathbb{S} \to \mathbb{S}$ on *D* is defined as the relation between *S* and *S′*. This rule *r* is an ET rule on *D*, iff

$$(S, S') \in r \Rightarrow \mathcal{M}(D \cup S) = \mathcal{M}(D \cup S'). \quad (4)$$

In the ETF a program is created by the accumulation of a set of these rules. Thus, a program on *D* is defined as a program comprising rules on *D*.

Given *D* and *R* (a program on *D*), computation in the ETF finds solution set *A* which satisfies *A = M(D)* $\cap$ *rep(q)* for a query $q \in \mathbb{Q}$; where *rep(*α*)* denotes the set of all ground instances of α.

Theorem 1. For an atom *q* representing a query on *D:*

$$\mathcal{M}(D) \cap rep(q) = \{g | ans(g) \in \mathcal{M}(D \cup \{ans(q) \leftarrow q\})\}.$$

Theorem 2. For an arbitrary set *F* of unit clauses, denoted by *ans(b)* ← . , where *b* represents an arbitrary term:

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

Theorem 3. If there exists a set of ET rules *R* on *D* such that under the domain knowledge *D*, the answer set *A* is obtained for query *q* by using transformation rules in the set *R* (given as $D : q \xrightarrow{R} A$) ; then $\mathcal{M}(D) \cap rep(q) = A$.

Theorem 3 follows from the proof outlined as follows. For $Q_0, \ldots, Q_n \in \mathbb{S} (0 \le n < \infty)$, let $Q_0 \to \ldots \to Q_n$ be a transformation sequence within $D : q \xrightarrow{R} A$, where Q_0 = *{ans(q)* ← *q}* and Q_n is a set of unit clauses. Then from Theorems 1, 2, and (4):

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

A program in the ETF is a set of rules which executes equivalent transformation such that:

$Q_0 \to \ldots \to Q_n$

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

starting from Q_0 = *{ans(q)* ← *q}* for all $q \in Q$, and finally computes the solution set *A* [17].

5.3 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 – no more and no less. Proving mathematically that a program is correct goes a long way in guaranteeing absence of program errors.

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 the declarative description $D \cup Q$, 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 $D \cup Q'$ have the same declarative meaning. A rewriting rule is considered to be correct, iff its application always results in a correct transformation step.

5.4 ET Computation and Correctness

A rewriting rule is an ET rule iff after rewriting a declarative description *P* into *P′* the meaning *M(P)* of *P* is equivalent to the meaning *M(P′)* of *P′*, i.e., *M(P) = M(P′)*. A program in the ETF is a set of these rewriting rules and program computation consists of successive rule application.

In the ETF, 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].

6 Reasons why the ET Framework can Effectively Model DISs

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

1. *Clarity* - ET rules are intuitively understandable by humans and the state of the computation can be observed and analyzed. This type of clarity allows us to check whether or not the behavior of an object, such as state changes, etc., is valid.
2. *Rich Expressivity* - The status, properties and interactions associated with an object are richly expressed in the ETF 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 types of parallelism. These are: (1) OR-parallelism; (2) AND-parallelism and; (3) Rule-parallelism (unique to the ETF) [16].
4. *High Level Abstraction* - An abstraction is an idea reduced to its essential form [18]. 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 models. The ETF operates at the conceptual level and so provides a high level of abstraction. As a result an ETF model 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 eases the rule-writing task. Since each rule can be written and focused on exclusively, rules are very easy to write. Additionally, because each rule is a standalone component; it can be executed immediately after being written in order to check that it carries out the intended operation. This enables us to construct highly decomposable models.
6. *Rule Priority* – Each rule can be given an execution priority – with the highest priority rule being the one executed first and so on. This allows the rule selection strategy of the program to be controlled; example for efficiency improvement.
7. *Multi-head Rules* – These rules facilitate easy representation and manipulation of object interactions. Each head atom can be used to represent the domain of an object.
8. *Dynamic Addition and Deletion of Rules* - In the ETF, rules can be dynamically added and deleted. This enables real-time modification of the model at runtime; such as the replacement of one rule by another in order to check its effect on the model, and thereby further improve it. Another benefit is the facilitation of real-time representation of new information and the dynamic addition and deletion of objects.
9. *Standard Treatment of Events* – In the ETF all events are processed in a standardized way. Events are accommodated via the condition section of a rule and processing is transparently carried out by means of lower level rules invoked for the purpose. As in all ET rules, these rules are very easy to write.
10. *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 tables (entities); clauses represent queries and; ET rules carry out query evaluations.
11. *Guaranteed Correctness –"Testing can show the presence of errors, but not their absence"* (E.W. Dijkstra). Thus guaranteeing program correctness goes a long way to relieve uncertainty and program testing time. The structure of the ETF guarantees correct operation of the system. This was outlined in sections 5.3 and 5.4. Detailed proofs can be found in [5].
12. *Declarative Semantics* - DISs 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. The

underlying declarative semantics of the ETF provides us with a means of connecting directly to this underlying nature of DISs and thus enables us to visualize and model all of its various aspects.

13. *Integrated System* - The ETF is an integrated modelling system, i.e., it is able to model the entire DIS spectrum (sections 2 to 4) without the need for any component external to the framework.

7 Analysis of an Example

In this section we will use the example of a simple ping-pong game [19] to illustrate our model construction technique.

The basic idea for the game is: 1) an independent dynamic ball *(Ball)*; 2) one paddle moves in sync with the mouse *(HPM)*; 3) the other paddle moves in sync with the pressing of the up and down arrow keys *(HPK)* and; 4) the interface is updated periodically to display current game status *(Show)*.

We start by first describing the overall system using an ET N-Rule (the global rule) as:

(Game) ==>
(Ball *ball), (HPM *mou), (HPK *key), (Show *disp) (5)

Here *Ball*, *HPM*, *HPK*, and *Show* represent the domains of the *ball*, *paddles*, and *display* entities, respectively. The second level rules are derived by examining the conditions for manipulation in each domain. In the *Ball* domain, the *ball* entity is dynamic and so it should move at specified time intervals. This is represented by the recursive ET rule:

(Ball *ball), {(TimeInt1)} ==>
{(moveBall *ball)}, (Ball *ball) . (6)

In this rule **ball* is an *I-var* representing the actual *ball* entity. *TimeInt1* is the time condition that has to be satisfied before the rule can be applied. The *(moveBall *ball)* atom represents a specialization and will call sub-computations (other rule sets) that will do the actual ball movement.

The *HPM* domain requires that the mouse event occurs before the **mou* entity can move. As a result, the rule is written as:

(HPM *mou), {(OnMouseMove *msg)} ==>
{(moveHPM *mou *msg)}, (HPM *mou) . (7)

In this rule, the **msg* variable is a message from the mouse to the **mou* entity. The actual manipulation of the **mou* entity is carried out by the *moveHPM* ruleset (invoked by the *{(moveHPM *mou *msg)}* specialization section. The rules for the other domains are derived in a similar fashion, resulting in:

(HPK *key), {(OnKeyPress *kmsg)} ==>
{(moveHPK *key *kmsg)}, (HPK *key) . (8)

(Show *disp), {(TimeInt2)} ==>
{(display *disp)}, (Show *disp) . (9)

We can write a multi-headed rule to describe the interactions of our entities of interest by writing each entity domain as one head and one body atom of the rule, to give *rule (10)*. The specialization section *{(doInteract *ball *mou *key)}* invokes the rules that will carry out the actual interactions.

(Ball *ball), (HPM *mou), (HPK *key) ==>
{(doInteract *ball *mou *key)},
(Ball *ball), (HPM *mou), (HPK *key) . (10)

The advantage of using a multi-headed rule such as (10) to carry out entity interactions is that representation of additional entities' interactions (e.g. more balls and paddles) can be done by simply adding the domains of those entities to the head and body of the rule, then inserting the entities into the specialization section (as arguments). For example, if we wanted another ball entity (let's call it *ball2*), we would write the rule for this ball as:

(Ball2 *ball2), {(TimeInt3)} ==>
{(moveBall *ball2)}, (Ball2 *ball2) . (11)

We would then modify the global and interaction rules, (5) and (10), by adding the ball domain to the rules; resulting in rules (12) and (13).

(Game) ==>
(Ball *ball), (Ball2 *ball2), (HPM *mou),
(HPK *key), (Show *disp) . (12)

(Ball *ball), (Ball2 *ball2), (HPM *mou), (HPK *key) ==>
{(doInteract *ball *ball2 *mou *key)},
(Ball *ball), (Ball2 *ball2), (HPM *mou),
(HPK *key) . (13)

The rules for the manipulation of the entities are written separately (as in (6), (7), (8), (9), and (11)), then added to the model. Rules (6), (7), (8), (9), (11), the new global rule (12), and the new

interaction rule (13) represent the global and second level rules for the ping-pong game system and are gathered together, along with their respective event-processing and lower level (specialization) rule sets to form the complete model for the system.

An example of the rule set for capturing events is shown in Fig. 2. This particular rule set represents the event section of rule (7). The rules are all ET D-Rules. The first rule gets a mouse message **m* which is then processed by the *mouseHandle* ruleset. Variables **mx* and **my* specify the *x* and *y-coordinates* of the mouse. As we are only interested in the y-coordinate, only that value is passed to the **msg* variable.

```
(onMouseMove *msg) -->
        (eg:GetMessage *m), (mouseHandle *m *msg).

(mouseHandle (WM_MOUSEMOVE *mx *my) *msg) -->
        (= *msg *my).

(mouseHandle (WM_MOUSEMOVE *mx *my ?) *msg) -->
        (= *my *my).
```

Fig. 2. The *onMouseMove* rule set

The *execution* (specialization) section of rule (7) calls the sub-computations shown in Fig. 3. The *moveHPM* rule first retrieves the information attached to the **mou* entity *(i-var)*, then calls another sub-computation, *(checkLimits *msg *NewHPy)*, to process the value received from the mouse against predefined constraints imposed on the movement of the paddle; and to generate a new y-coordinate for the paddle. The changed information is then stored as an *i-var* and reattached to the **mou* entity. It should be noted that the difference in variable names of the arguments used for the *checkLimits* atom in the *moveHPM* rule and the actual *checkLimits* ruleset is of no significance since the ETF uses pattern matching to transfer values across rules.

```
(moveHPM *mou *msg) -->
        (getInfo *mou (*HPx *HPy *HPw *HPh)),
        (checkLimits *msg *NewHPy),
        (putInfo *mou  (*HPx *NewHPy *HPw *HPh)).

(checkLimits *paddley *Npaddley), {(< *paddley 25)} -->
        (= *Npaddley 25).

(checkLimits *paddley *Npaddley), {(> *paddley 280)} -->
        (= *Npaddley 280).

(checkLimits *paddley *Npaddley) -->
        (= *Npaddley *paddley).
```

Fig. 3. The *HP* and *checkLimits* rule sets

The events and sub-computations for the other rules are written in a similar manner. The entire model is constructed incrementally in this way.

8 Related Work

A lot of research has been done on the modelling of Interactive Systems (ISs) in general. However in this section we will take a look specifically at those works which we feel can actually apply to the modelling of DISs (i.e. DISs as defined earlier by us - your humble authors).

In [11] hand-drawn sketches on an electronic board are used to develop the design cycle then converted into an XML-based CTT specification. This method is highly intuitive but it concentrates only on task modelling aspects, as opposed to integration with system aspects. In addition, the method and language used do not support Formal Methods.

The method outlined in [20] uses a language based on UML 2.0 to model interactive multimedia applications. This method is non-intuitive and also not amenable to Formal Methods. Integration of User-centric and System-centric methods is the main focus of [14]. However the foundation of the models is not mathematical and the framework does not support concurrency. In addition, the semantics of the language used (IMML) can be prone to ambiguity.

The novel approach of [12] uses stochastic techniques, is amenable to Formal Methods, and integrates both approaches. However the notation may be a bit too complex for non-expert users.

Our method may be considered closest to [15] and [21]. The method of [15] is based on a form of equivalent transformation called Petri-Nets. It uses a formal framework to integrate both task and system

activities in an iterative process. However the notation and model creation process are complex and non-intuitive. Geared specifically towards data-driven Web applications, [21] uses a form of first order logic to model ISs. As in several of the others cited so far, the language may be too complicated for anyone less than a seasoned mathematician.

One major fundamental difference between our methodology and the works cited lies in the fact that in our method all development and transformation (from idea to model construction) are carried out using one language - called ETL, and in an integrated framework – the ETF. As a result, the foundation of the model can be guaranteed correct, reliable, and intellectually manageable since all transformation steps can be certified. Another major difference lies in the highly decomposable nature of the ETF. In addition, the notation system used in the ETF is simple and intuitive enough to be quickly understood by even novice programmers.

A major obstacle to the wide-scale adoption of Formal Methods in system development has been the complicated proofs and notations required. We believe that a simple, intuitive, mathematically based framework such as the ETF will help to remove this obstacle. The advantage of the ETF is that it provides a simple, yet mathematically powerful framework, in which a formal model of a DIS can be intuitively constructed in increments. This model is easily comprehensible to both expert and novice programmers alike.

9 Conclusion

In this paper we looked at some of the problems that currently obtain in the construction of formal, rigorously analyzeable models for Dynamic Interactive Systems (DISs); and examined why and showed how the ET framework (ETF) can be extended to overcome these difficulties. We also explained how the ETF can give a comprehensive conceptual model for DISs, which is intuitive, robust, correct, and intellectually manageable. In addition we demonstrated our concept of model creation using a simple ping-pong game.

References:

[1] S. McConnell, *Code Complete, 2nd Edition*, Microsoft Press, 2004.

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

[3] M. Huth and M. Ryan, *Logic in Computer Science: Modelling and Reasoning about Systems, 2nd Edition*, Cambridge University Press, 2004.

[4] J.M. Wing, A Specifier's Introduction to Formal Methods, *Computer*, Vol. 23, No. 9, 1990, pp. 8-23.

[5] K. Akama and E. Nantajeewarawat, Formalization of the Equivalent Transformation Computation Model, *Journal of Advanced Computational Intelligence and Intelligent Informatics*, Vol. 10, No. 3, 2006, pp. 245-259.

[6] E. Nantajeewarawat et al., Toward Reasoning with Unified Modelling Language Diagrams Based on Extensible Markup Language Declarative Description Theory, *International Journal of Intelligent Systems*, Vol. 19, 2004, pp. 89-98.

[7] C. Anutariya et al., XML Declarative Description with First-Order Logical Constraints, *Computational Intelligence*, Vol. 21, No. 2, 2005, pp. 130-156.

[8] V. Wuwongse et al., A Data Model for XML Databases, *Journal of Intelligent Information Systems*, Vol. 20, No. 1, 2003, pp. 63-80.

[9] M.D. Harrison and D.J. Duke, A Review of Formalisms for Describing Interactive Behaviour, *ICSE Workshop on SE-HCI*, 1994, pp. 49-75.

[10] F. Paternò et al., ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models, *Proceedings of IFIP TC13 International Conference on Human-Computer Interaction*, 1997, pp. 362-369.

[11] F. Paternò and M. Volpe1, Natural Modelling of Interactive Applications, *Interactive Systems, Lecture Notes in Computer Science*, Vol. 3941/2006, 2006, pp. 67-77.

[12] G.J. Doherty et al., Reasoning about Interactive Systems with Stochastic Models, *Proceedings of the 8th International Workshop on Interactive Systems: Design, Specification, and Verification-Revised Papers, Lecture Notes In Computer Science*, Vol. 2220, 2001, pp. 144-163.

[13] A. Dix and G. Abowd, Modelling Status and Event Behaviour of Interactive Systems, *Software Engineering Journal*, vol. 11, No. 6, 1996, pp. 334 -346.

[14] J.C. Leite, A Model-based Approach to Develop Interactive System using IMML, *Task Models and Diagrams for Users Interface Design, Lecture Notes in Computer Science*, Vol. 4385/2007, 2007, pp. 68-81.

[15] P. Palanque and R. Bastide, Synergistic Modelling of Tasks, Users and Systems Using

Formal Specification Techniques, *Interacting with Computers*, Vol. 9, No. 2, 1997, pp. 129-153.

[16] K. Akama et al., Generation of Correct Parallel Programs Based on Specializer Generation Transformations, *Proceedings of the 7th international Conference on Intelligent Technologies (InTech'06)*, pp.90-99, 2006

[17] S. Miyajima et al., Detecting Incorrect Rules Automatically in Equivalent Transformation Programs, *Proceedings of the 2nd International Conference on Innovative Computing, Information and Control (ICICIC 2007)*.

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

[19] 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)*.

[20] A. Pleuß and H. Hußmann, Integrating Authoring Tools into Model-driven Development of Interactive Multimedia Applications, *Human-Computer Interaction. Interaction Design and Usability, Lecture Notes in Computer Science*, Vol. 4550/2007, 2007, pp. 1168-1177.

[21] A. Deutsch et al., A System for Specification and Verification of Interactive, Data-driven Web Applications, *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, 2006, pp. 772-774.