A PREDICTIVE ENGINE FOR THE QUALITATIVE SIMULATION OF CONTINUOUS DYNAMIC SYSTEMS

Mark WIEGAND and Roy LEITCH

Intelligent Automation Laboratory,
Department of Electrical and Electronic Engineering,
Heriot-Watt University,
Edinburgh
Scotland

1. Introduction

This paper describes the development of a general architecture for a reasoning mechanism that is able to simulate the dynamic evolution of a physical system utilising qualitative and quantitative information about the variables in the system. The "predictive engine" that results forms a tool component in the general-purpose toolkit being developed under ESPRIT project P820 whose remit is to develop a set of high-level tools for a range of tasks within the process industries. This set of tools now forms the QUIC (Qualitative Industrial Control) toolkit [Leitch1989a].

Much work has been done by various researchers in trying to develop algorithms which perform qualitative dynamic reasoning and/or which deal with the problems inherent in using qualitative values (e.g. ambiguity). In attempting to exploit this new technology, decisions have to be made about which ideas to include in the system, which ideas are redundant or superfluous, and which are already catered for in another guise. In the development of the "predictive engine", an attempt has been made to keep the architecture as 'general' as possible. Rather than construct a series of systems, each applicable to a few small and artificial examples, the remit for a tool component must include such attributes as generality and coherence. This work is an attempt to move from conceptualising to implementation without discarding these attributes. In doing this, various sources are drawn upon and the results placed in a context where they can be seen to perform a specific function. Whilst the "predictive engine" is perhaps sub-optimal for any specific application, it is general and flexible, and reflects well the principles involved in its development.

The architecture of the "predictive engine" is layered and strictly modular, each module having a well-defined functionality; see Figure 1. Each module is completely independent and communicates with the lower-level module via a 'Tell-and-Ask' type interface. At the 'core' of the "predictive engine" lie the graph-based representations that hold the Quantity Space (the allowable qualitative values for each system variable) and the Time Box (a record of the temporal relationships between the changing values of the system variables). Above this core, an Event Map collates the values of system variables with their temporal extent and presents these 'tuples' to the Prediction Module which manages the inference of the system behaviour. The results of the inference process are passed back down to the core where they may contribute to

further inferences.

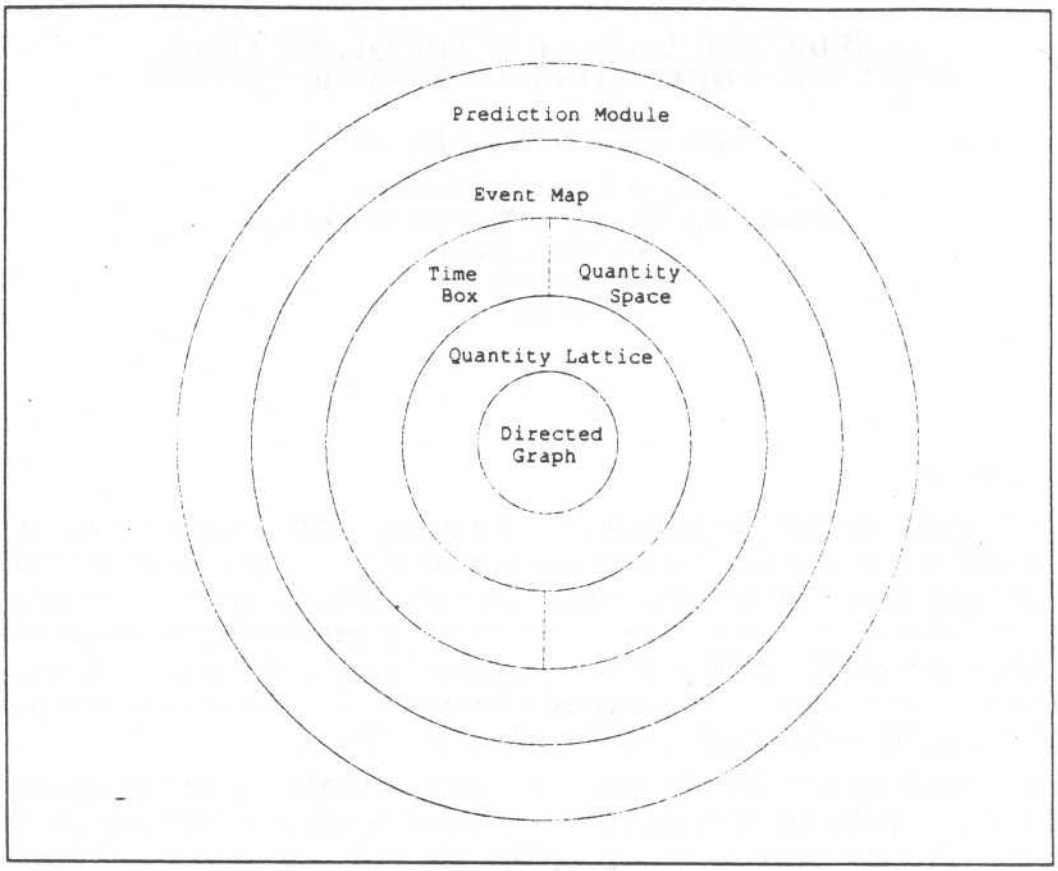

Figure 1 Architecture of the Predictive Engine

This design is strongly motivated by the work of Williams, who advocates a 'rule-based' approach in his Temporal Qualitative Analysis (TQA) [Williams1984a] and an 'event-based' approach in his Temporal Constraint Propagation (TCP) [Williams1986a]. The "predictive engine" reflects a generalised combination of TQA and TCP. In his work on reasoning in dynamic domains, Williams applied TQA to the analysis of MOS circuits. The "predictive engine" takes Williams' classification of feedback characteristics in electrical systems, and applies similar ideas to canonical forms of physical phenomena in other domains with continuous variables, notably process control. The "predictive engine" also investigates Williams' use of an 'event-based' architecture and TCP for qualitative reasoning as a way of handling pure-time delay more efficiently than in a 'state-based' approach, e.g. [Kuipers1985a]. This particular ability is very important for at least one of the demonstrators in the ESPRIT project P820, where a lumped parameter approximation has been used in order to avoid consideration of a partial spatial derivative.

An 'event-based' approach needs to be able to reason with partially-ordered temporal intervals. Following the work of Vilain and Kautz [Vilain1986a] which suggested that an interval-based implementation may be computationally intractable, we utilise a point-based implementation which is based on Simmons' Quantity Lattice

[Simmons1986a]. The Quantity Lattice actually belongs to a general class of systems which may be termed "inequality reasoners". It manages the integration of real and symbolic values, allowing the reasoning mechanism to make use of quantitative information when this is available (or necessary).

2. Directed Graph

The Directed Graph constitutes the basic information storage module of the system. Directed graphs hold both the Time Box and the Quantity Space in the "predictive engine", albeit through a Quantity Lattice interface which enables the integration of real with symbolic values.

Digraph nodes are used to represent either time points (the end points of periods/events) or landmarks in the quantity space: the actual semantics of these nodes will depend on the interface that the digraph is viewed through (Time Box or Quantity Space). In Simmons' original description of his Quantity Lattice, nodes were also used to hold real numbers. However, in the implementation of the "predictive engine", real numbers are not stored as nodes in the digraph but are generated by the Directed Graph module interface as requested. In the large-scale systems for which it is envisaged that the "predictive engine" will be used, inclusion of reals as proper nodes would clutter the graph. Of course, real values may be given an explicit status when they form an important part of the reasoning process; this can be done by assigning a symbolic point and giving this point the real value, for example, the 'zero' of the Quantity Space. Then, the symbolic point is available for the reasoning process and its real value is available when required.

Nodes can also be expressions. Simmons' use of arithmetic expressions was quite extensive in his application domain. At the moment, only binary subtraction and binary addition are being considered; binary subtraction is required in order to handle information about the duration of time periods (the difference between two time points), and binary addition is then required as a consequence.

The nodes of the digraph are connected with labelled directed arcs. Simmons used six labels, $<$, $=$, $>$,$\leq$,$\geq$ and$\neq$. However, use of$\neq$ is problematic. In the "predictive engine" only the labels $<$, $=$, and $>$ are used (with simple extensions to$\geq$ and$\leq$). The digraph arcs may be used to express partial orderings. Additional arc labels, allowing for the representation of 'order of magnitude' relations, have been included in the full implementation of this prototype. These operators will facilitate the inclusion of some form of 'order of magnitude' reasoning in the Prediction Module, enabling the use of temporal hierarchies in the Time Box digraph, and increased (selective) granularity in the Quantity Space digraph.

To enable the smooth integration of real and symbolic values in the "predictive engine", the Directed Graph also holds information about the real value of each node. Associated with each node is a real interval which represents what is currently known about the real value of that node. By default, the interval is $(-\infty,+\infty)$, or $[0,+\infty)$ if it is known beforehand that each node is non-negative. As the value of each node becomes further constrained during use of the "predictive engine", either as a result of inferences made or supplemental data, it is the Quantity Lattice that is responsible for maintaining the consistency of these interval bounds across the directed arcs that connect the nodes. A relationship between two nodes might be inferred from the intervals

associated with them, even though no labelled arc exists.

3. Quantity Lattice

The Quantity Lattice module is responsible for maintaining the consistency of information in the Directed Graph, and for servicing requests from the Time Box and Quantity Space modules. A major requirement is to 'hide' the mechanism of the Quantity Lattice from the Time Box and Quantity Space by providing a set of general purpose functions which control access to the graphs: the Time Box and Quantity Space have somewhat different requirements of the interface.

The Quantity Lattice in the "predictive engine" is based on that of Simmons. However, as has already been stated, problems were found in using Simmons' choice of arc labels, i.e. $<$, $=$, $>$,$\leq$,$\geq$,$\neq$. These problems can best be illustrated with an example; they centre around the use of$\neq$. Consider Figure 2: it seems quite possible to create this graph in Simmons' system. If we ask for the relationship between q_1 and q_4, the Quantity Lattice performs a breadth-first search along paths which contain an entry in a transitivity table for ordinal relationships. The result of this is:

q_1 (=)
q_2 ($\leq$)
q_4 ($\leq$)

Note that the path q_2 to q_3 is not searched because$\leq$ and$\neq$ in series do not form a valid relationship. However, if we ask for the relationship between q_2 and q_4, the result is:

q_2 (=)
q_4 ($\leq$) and q_3 ($\neq$)
q_4 ($\leq$) and q_4 ($\neq$)
q_4 ($<$)

Then, combining q_1$\leq$ q_2 with q_2 $<$ q_4, we should get

q_1 $<$ q_4

whereas Simmons' system gave us

q_1$\leq$ q_4

This problem lies in the use of$\neq$, and in the fact that the search is linear (local). Several attempts have been made to solve this problem, but it seems that the only way is to make the search non-linear (non-local) and much less efficient. For this reason it was decided to use only $<$, $=$, and $>$ from Simmons' original label set and for these relationships the above problem does not occur.

Recent work by Nokel [Nokel1989a] has extended the work of Vilain and Kautz [Vilain1986a] by further classifying the subalgebras of Allen's [Allen1983a] full relation algebra (based on thirteen primitive relations) where the global consistency check can be carried out in polynomial time. Each subalgebra is formed by adopting a subset of the thirteen primitive relations, say N where $| N | = n < 13$, and allowing relations formed by disjunctions of these adopted primitives, of which there are then a total of

2^n. Polynomial time consistency checking within the subalgebra depends on a property of "convexity" among the disjunctive relations. Interpreting these results in a point-based implementation, we find that the only "non-convex" relation that can be formed from the primitives $<, =, >$ is that of ($<$ or $>$), i.e. $\neq$. This essentially explains the problems we found in using the $\neq$ relation in Simmons' Quantity Lattice.

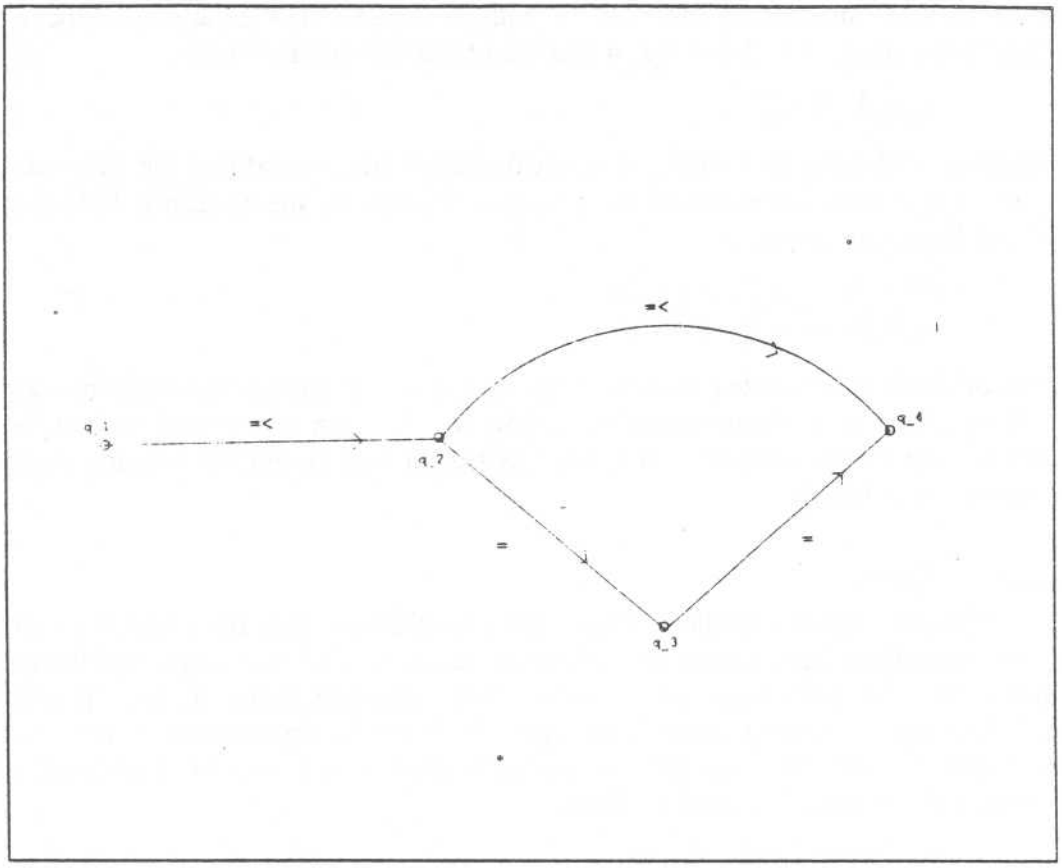

Figure 2 Problems with Simmons' Quantity Lattice

It is important that the Quantity Lattice module employs breadth-first search, as opposed to depth-first. In particular in the Time Box digraph it is possible for very long paths to be created during use, and depth-first search could prove very inefficient. Following Simmons, the "predictive engine" caches the results of the breadth-first search, though it is not clear what the pay-off for this is in efficiency terms.

The Time Box and Quantity Space require different functionality in the Quantity Lattice. In particular, access to the graphs is not the same for the higher level modules. In the Time Box, the inference mechanism may consider relations that span more than one time point. However, in the Quantity Space, the continuity rules in the Prediction Module may only require one-step search to the next largest landmark. Obviously different functions are required in the Quantity Lattice.

The Quantity Lattice maintains the consistency of the real intervals associated with nodes by numeric constraint propagation along the search paths in the graph. This is straightforward, but the issue is complicated when we include arithmetic

expressions as nodes in the graph for expressing period durations. Information constraining the end points of a period will also constrain the duration of a period, and vice versa. The Quantity Lattice employs 'interval arithmetic' and 'relational arithmetic' to move information about intervals between the expression node and its argument nodes [Simmons1986a]. To facilitate this propagation, whenever a duration node is created in the Time Box graph, additional expression nodes are created to yield each argument so that information can flow in both directions. For example, suppose we have time point nodes t_a_3 and t_a_4 and we create the duration node:

(t_a_4 - t_a_3)

This duration will have an associated interval, and if this is modified the intervals for t_a_3 and t_a_4 may be modified as a result. Therefore, the Quantity Lattice also creates the following nodes:

((t_a_4 - t_a_3) + t_a_3)
(t_a_4 - (t_a_4 - t_a_3))

The first of these is connected with an = arc to t_a_4 and the second with an = arc to t_a_3, then information about intervals can flow in either the time points to duration or duration to time points direction. It is for this reason that the binary addition operator must also be introduced.

4. Quantity Space

The Quantity Space module defines what a qualitative value for a variable can be. Different researchers have expressed qualitative values in different ways, and the quantity space has not been used consistently. The approach taken in the "predictive engine" is to say: "whatever the quantity space is, it can be represented as some set of directed graphs". So for example, (+,0,-) qualitative values can be expressed as a graph with three nodes, +, 0, and -. Then,

+	has interval	$(0,+\infty)$
0	" "	[0,0]
-	" "	$(-\infty,0)$

Also, Kuipers' totally ordered sets of landmark values [Kuipers1985a] may be expressed by having a graph for each variable. Other researchers have chosen to use a 'global' quantity space, rather than have a separate one for each variable. It may be desirable to use a common space for variables with a common meaning, e.g. the levels of two coupled tanks.

The Quantity Space should 'hide' the digraph implementation from the reasoning mechanism. So, for example, if (+,0,-) values are used for the higher order derivatives, as in [Williams1984a], then the Quantity Space should handle the semantics of this use. This means servicing requests from the reasoning mechanism about how values may change and how values combine in constraints.

The Quantity Space module must include functions for the creation of landmark points (especially if landmark discovery is employed in the reasoning mechanism). When a group of variables connected by a single arithmetic constraint/equation in the

model must all reach landmarks in their respective quantity spaces simultaneously, these landmarks are called 'corresponding values' [Kuipers1985a]; 'corresponding values' may be seen to constitute arithmetic relations between nodes appearing in the Quantity Space digraph.

5. Time Box

The term 'time box' was originally coined by Williams [Williams1986a]. However, it is not completely clear from this reference what the 'time box' actually is. Williams attempts to characterise it in terms of its required functionality:

(i) What questions will be asked?
(ii) What temporal information is available?
(iii) What inference is needed to answer these questions?

It is clear that the 'time box' is considered with respect to some proposed application. Williams represented the 'time box' using Simmons' Quantity Lattice; from this we can deduce an initial architecture for the module and suggest a functionality.

The Time Box uses a digraph to hold time points (the beginning and end of time periods) as nodes, and what is known about the order in which these time points occur is expressed by labelled (<,=,>) arcs between the nodes. Time points are either observed (if a system variable is being treated as an input to the "predictive engine") or they are generated by the Prediction Module. A time point marks the place where there was a qualitative change of value in some variable. Requests to create a new time point for a variable will come to the Time Box, and this module is responsible for managing the Quantity Lattice in updating the Directed Graph that holds the temporal information. Any new information about when a time point occurred with respect to other time points will be represented by connecting arcs to it. Note that the Time Box is using symbolic time (just as the Quantity Space is symbolic). Any information about the actual (real) time at which time points occur is stored in the interval associated with each node.

As the "predictive engine" runs, the Time Box constructs a history map showing how the time periods are ordered. Two points arise from this. Firstly, depending on the applications, there may come a time during the execution when information before a certain time point could not possibly help in predicting behaviour, and it may be more efficient to delete it (or at least take it out of the graph search space). For this reason, history deletion functions are provided in the Time Box. Secondly, it is not absolutely clear that this module is what Williams meant by 'time box'. It is possible that Williams intended the variables' values to be held with the relevant time periods. This is the function of the Event Map module; the Time Box only holds information about time periods, not value/period tuples (i.e. events).

To those familiar with temporal logics, the Time Box may appear as a 'point-based' implementation. The background to our adopting this approach has already been explained. A set of rules in the Time Box allow queries to make use of Allen's 'period-based' temporal logic [Allen1983a], based on a subalgebra of the 13 possible relationships, by translating to 'point-based' relations.

As with the Quantity Space, an important function of the Time Box is to 'hide' implementation details from the higher level modules of the system. As part of this remit, the Time Box handles period durations smoothly by interfacing to their implementation as arithmetic expression nodes in the graph. The inclusion of arithmetic expression nodes is a source of great inefficiency in the Quantity Lattice. If arithmetic expression nodes are not absolutely necessary, it must be considered whether the functionality for handling durations and corresponding values can be moved from the Quantity Lattice to the Time Box and Quantity Space modules respectively.

6. Event Map

The job of the Event Map is to present 'events' as propagation units to the Prediction Module, and to relay requests for information to the core of the system and pass back results. An 'event' in this architecture is a qualitative value/temporal period tuple; it represents the fact that a particular variable held a particular value for a particular period or moment of time. There are two types of event in the system. The first kind are termed 'period events'; they express that a variable holds a value between two distinct time points. The other kind are 'moment events' which express a variable's value at a time point. In much work on temporal logic the term 'interval' is used to refer to a 'period', and the term 'point' to refer to a 'moment'. However, in this paper, 'period' and 'moment' are used to refer specifically to temporal concepts, hopefully avoiding confusion. The 'event history' for a variable will consist of a sequence of alternating 'period events' and 'moment events', showing how that variable's qualitative value changes over time. For example, assuming a variable 'a' uses (+,0,-) semantics in its quantity space, the following might represent a section of its event history:

```
event(a,+,(2,3))
event(a,0,3)
event(a,-,(3,4))
```

The first is a period event, expressing 'a''s value between time points 2 and 3. The second is a moment event expressing the value at time point 2. The third is a period event. Note that the event history is 'concise' [Williams1986a]; this means that a new event is only encountered when the variable's qualitative value changes. For example, the following event history is not concise:

```
event(a,+,(2,3))
event(a,+,3)
event(a,+,(3,4))
```

The Event Map presents events for propagation by the Prediction Module, and decomposes events into their constituent factors for expression in the relevant core modules. We may view this process as a 'mapping' from events (the external appearance) to 'pseudo-events' (the internal representation), as follows:

event(a,+,(2,3))→ pseudo_event(variable(a),
(quantity(zero),quantity(+∞))
(timepoint(a,2),timepoint(a,3)))

event(a,0,3)→ pseudo_event(variable(a),
(quantity(zero),quantity(zero))
(timepoint(a,3),timepoint(a,3)))

7. Prediction Module

The modules so far described play only a supporting role in the overall "predictive engine". The main inference mechanism resides in the Prediction Module. This module embodies a synthesis of qualitative reasoning techniques for continuous dynamic systems. The techniques currently being used include those found in systems such as Kuipers' QSIM [Kuipers1985a] and those that handle feedback effects (and pure time delay) [Williams1986a, Williams1984a].

The Prediction Module takes as input a set of equations or constraints representing the assumed model, and generates a prediction of the qualitative behaviour of the system. These constraints currently include the standard arithmetic relations, and the functional operations of monotonic relations and integration/differentiation, with primitives for handling pure-time delay under development.

The interpretation of these constraints is encoded declaratively in a set of Predictive Rules for handling arithmetic relations, qualitative integration, etc. The Predictive Rules are employed by a Predictive Algorithm. This is a procedural encoding of what might be called 'meta-level inference', expressing when and how certain rules should be applied. The algorithm is event-based and it is constructive (as opposed to the generate-and-test algorithm of QSIM). The algorithm attempts to determine how a system evolves qualitatively in response to an input function expressed in qualitative terms. The input variable(s) gives us a notion of exogenous variable which is used to determine a dynamic causal ordering using the algorithm of Iwasaki [Iwasaki1988a] (or rather a polynomial-time implementation developed in the ESPRIT project P820 [Porte1988a]). The results of causal ordering allow the Prediction Module to determine a unique direction of propagation of information through the system model.

The system model can be differentiated (up to two times, depending on the modelling primitives used) and information about higher-order derivatives propagated through the model. In this way, the input function can be more fully characterised. Though the Predictive Rules are based on the properties of continuous and differentiable functions of time, the Predictive Algorithm can also manage jump discontinuities in the input function; these are propagated as discontinuities through the system model. Note the difference between this scheme and QSIM: whereas QSIM considers a qualitative value to be a magnitude/derivative tuple, the Prediction Module only considers magnitude and expresses derivatives as variables in higher-order models of the system.

The constraints are partitioned into two classes depending on whether they imply the passage of time. The two sets of constraints are applied separately and sequentially by the Predictive Algorithm in stages called 'causal propagation' and 'qualitative

integration/transition analysis'. This partitioning is directly analogous to the structure of conventional numerical simulations of continuous dynamic systems. Static relations (which exclude integration/differentiation and relations involving pure time delay) propagate instantaneously (i.e. during the same moment). Dynamic relations (involving integration/differentiation) require some time to elapse during propagation. The nature of the qualitative change taking place determines the type of delay that occurs during integration: point-to-interval transitions occur in infinitesimal time (though not instantaneously), whereas interval-to-point transitions require some non-infinitesimal time to pass. If there are several interval-to-point transitions that may occur, then transition ordering is required (though we use a heuristic that seems to avoid unnecessary generate-and-testing). Finally, some of these interval-to-point transitions may not occur at all (i.e. they may take infinite time), and a scheme for asymptotic reasoning is under investigation.

Also under investigation is a system for utilising information about pure time delays expressed in the model. Such rules will enable inferences to be made when the system being modelled is subject to transport delays. It is to be hoped that one of the advantages of using an event-based scheme is that pure time delay can be handled in a natural way. The possibility of using 'order of magnitude' relations on delay parameters to express time-scale abstraction hierarchies [Kuipers1987a] is being pursued within one of the demonstrators of the ESPRIT project P820 where a lumped parameter model is used to avoid consideration of spatial derivatives.

In addition to this equational form of representation, the issue of interfacing to other forms of knowledge is being examined within the work on the "predictive engine". In particular, techniques and a methodology for overcoming qualitative ambiguity that involve the use of more detailed "empirical" knowledge of functional relationships between specific system variables are under investigation. Such ambiguity is bound to be a problem where there is a summation point that it not part of a feedback loop, or where the feedback loop is cross-coupled with others. This issue is considered paramount for the realistic application of qualitative techniques.

The Prediction Module will eventually contain interfacing functions that use information held in the QUIC toolkit knowledge representations, particularly the Component Based Language (CBL) "dynamic" domain. Empirical sources of knowledge in the toolkit will also supply the Prediction Module (as mentioned above). Current work is only focusing on the information content that must be present in order to make useful deductions. Once a methodology has been completely determined the appropriate interfaces will be implemented.

8. Conclusion

The architecture of a general-purpose "predictive engine" for reasoning about the dynamic evolution of physical systems has been described.

The design of the "predictive engine" makes use of the work of a number of Qualitative Physics researchers, notably Williams, and it should be noted that this work is continually evolving. As new algorithms and approaches are evaluated and deemed fit for inclusion in the "predictive engine", it is to be hoped that the generality of the architecture will allow this to be done without substantial re-coding.

We argue for the usefulness of a conceptual separation such as this as opposed to the development of more restrictive implementations perhaps only illustrating one technique. Current work is focussed on the development of a Prediction Module capable of operating on laboratory-scale equipment. The prototype "predictive engine" is developed in Prolog, and work is now underway to re-implement it in Lisp for inclusion in the QUIC toolkit.

Acknowledgement

This paper describes developments undertaken in the ESPRIT project P820, partly funded by the Commission of the European Communities within the frame of the ESPRIT programme. Project P820 consists of a consortium composed of CISE, Aerospatiale, Ansaldo, CAP Sogeti-Innovation, F.L. Smidth, Framentec, and Heriot-Watt University. The authors want to acknowledge here the contribution of all the members of the project team to the ideas expressed in this paper, while taking full responsibility for the form in which these ideas are expressed.

References

Allen1983a.

J.F. Allen, ''Maintaining Knowledge about Temporal Intervals,'' *Communications of the ACM*, vol. 26, no. 11, pp. 832-843, 1983.

Iwasaki1988a.

Y. Iwasaki, ''Causal Ordering in a Mixed Structure,'' in *Proceedings of Seventh National Conference on Artificial Intelligence (AAAI-88)*, vol. 1, pp. 313-318, Saint Paul, Minnesota, U.S.A., 1988.

Kuipers1985a.

B. Kuipers, ''The Limits of Qualitative Simulation,'' in *Proceedings of Ninth International Joint Conference on Artificial Intelligence (IJCAI 9)*, vol. 1, pp. 128-136, Los Angeles, U.S.A., 1985.

Kuipers1987a.

B. Kuipers, ''Abstraction by Time-Scale in Qualitative Simulation,'' in *Proceedings of Sixth National Conference on Artificial Intelligence (AAAI-87)*, vol. 2, pp. 621-625, Seattle, Washington, U.S.A., 1987.

Leitch1989a.

R.R. Leitch and A. Stefanini, ''High-Level Tools for Intelligent Automation,'' *to appear in International Journal for Artificial Intelligence in Engineering*, 1989.

Nokel1989a.

K. Nokel, ''Convex Relations Between Time Intervals,'' in *Proceedings of 5. Osterreichische Artificial-Intelligence-Tagung*, ed. K. Leidlmair, Informatik-Fachberichte 208, pp. 298-302, Springer-Verlag, Berlin, 1989.

Porte1988a.

N. Porte, S. Boucheron, J. Sallantin, and F. Arlabosse, ''An Algorithmic View at Causal Reasoning,'' in *Proceedings of Second Workshop on Qualitative Physics*, ed. F. Gardin, IBM Paris Scientific Centre, Paris, France, 1988.

Simmons1986a.
R. Simmons, "Commonsense" Arithmetic Reasoning," in *Proceedings of Fifth National Conference on Artificial Intelligence (AAAI-86)*, vol. 1, pp. 118-124, Philadelphia, U.S.A., 1986.

Vilain1986a.
M. Vilain and H. Kautz, "Constraint Propagation Algorithms for Temporal Reasoning," in *Proceedings of Fifth National Conference on Artificial Intelligence (AAAI-86)*, vol. 1, pp. 377-382, Philadelphia, U.S.A., 1986.

Williams1984a.
B.C. Williams, "Qualitative Analysis of MOS Circuits," *Artificial Intelligence*, vol. 24, no. 1, pp. 281-346, North-Holland/Elsevier, Amsterdam, 1984.

Williams1986a.
B.C. Williams, "Doing Time: Putting Qualitative Reasoning on Firmer Ground," in *Proceedings of Fifth National Conference on Artificial Intelligence (AAAI-86)*, vol. 1, pp. 105-112, Philadelphia, U.S.A., 1986.