DYNAMIC SYSTEM SPECIFICATION

by
Dr. Robert M. Balzer
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, California 90291

INTRODUCTION

Our group is perhaps unique in that our approach to modeling dynamic systems has a strong heritage from all three fields represented at this workshop. Each of the fields has contributed four major concepts to our modeling approach.

From the Data Base field, we have adopted the idea of a single global model (data base) which defines the state of the system. Fully associative relations among typed atomic objects (no internal structure) constitute the entire state description. An information model (query language) is defined for extracting information from the state (such as one or all objects satisfying some description, or the truthfulness of some proposition). Finally, the model is updated via transactions, so that either the entire operation is completed successfully, or the state is unchanged.

From the Programming Languages field, we have adopted the notion of operational semantics for abstract operations which enables these specifications to be used as a "simulation" model of the specified system. Non-determinism has been incorporated to provide additional specificational power. Concurrent processes are used to model the interacting active participants of the specified system. Finally, an exception handling mechanism is employed to simplify operation definitions by separating the description of normal and exceptional cases, and by defining universal reverse information flow mechanisms.

From Artificial Intelligence, we have incorporated an inferencing capability which enriches the information model by hiding the distinction between explicit and derived data and enables its definition to be abstractly specified. Arbitrary constraints are included in our model to define, by behavior, which non-deterministic executions are acceptable. Demons, which are spontaneously activated when their patterns are satisfied by the state of the global model, are used to enable the active participants (processes) to react to a stimulus in their environment (the global model) and respond by altering it (via abstract operations). Finally, a type lattice (rather than hierarchy) is used to organize the objects being modeled, and all the components of the system description (the participants, operations, inferencing, constraints, etc.) are part of the model and are defined in the relational mechanism (so that self examination is possible).

PRINCIPLES OF GOOD SPECIFICATION

Principle #1: Separate functionality from implementation.

First, by definition, a specification is a description of WHAT is desired, rather than HOW it is to be realized (implemented). Specifications can adopt two quite different forms. The first form is that of mathematical functions: Given some set of input, produce a particular set of outputs. The general form of such specifications is find [A/THE/ALL] result such that P(input), where P represents an arbitrary predicate. In such specifications, the result to be obtained has been entirely expressed in a WHAT (rather than HOW) form. In part this is because the result is a mathematical function of the input (the operation has well defined starting and stopping points) and is unaffected by any surrounding environment.

Principle #2: A process-oriented systems specification language is required.

Consider instead a situation in which the environment is dynamic and its changes affect the behavior of some entity interacting with that environment (as in an "Embedded Computer System"). Its behavior cannot be expressed as a mathematical function of its input. Rather, a process-oriented

This research was supported by DARPA contract DAHC 15 72 C 0308.

* The views expressed are those of the author.

description must be employed, in which the WHAT specification is achieved by specifying a model of the desired behavior in terms of functional responses to various stimuli from the environment.

Such process-oriented specifications, presenting a model of system behavior, have normally been excluded from formal specification languages, but they are essential if more complex dynamic situations are to be specified. In fact, it must be recognized that in such situations both the process to be automated and the environment in which it exists must be described formally. That is, the entire system of interacting parts must be specified, rather than just one component.

Principle #3: A specification must encompass the system of which the software is a component.

A system is composed of interacting components. Only within the context of the entire system and the interaction among its parts can the behavior of a specific component be defined. In general, a system can be modeled as a collection of passive and active objects. These objects are interrelated and over time the relationships among the objects change. These dynamic relationships provide the stimulus to which the active objects, called agents, respond. The responses may cause further changes and, hence, additional stimuli to which the agents might respond.

Principle #4: A specification must encompass the environment in which system operates.

Similarly, the environment in which the system operates and with which it interacts must be specified.

Fortunately, this merely necessitates recognizing that the environment is itself a system composed of interacting objects, both passive and active, of which the specified system is one agent. The other agents, which are by definition unalterable because they are part of the environment, limit the scope of the subsequent design and implementation. In fact, the only difference between the system and its environment is that the subsequent design and implementation effort will operate exclusively on the specification of the system. The environment specification enables the system "interface" to be specified in the same way as the system itself rather than introducing another formalism.

It should be noted that the picture of system specification presented here is that of a highly intertwined collection of agents reacting to stimuli in the environment (changes to objects) produced those agents. Only through the coordinated actions of the agents are the goals of the system achieved. Such mutual dependence violates the principle of separability (isolation from other parts of the system and environment). But this is a DESIGN principle, not one of specification. Design follows specification and is concerned with decomposing a specification into nearly separable pieces in preparation for implementation. The specification, however, must accurately portray the system and its environment as perceived by its user community in as much detail as required by the design and implementation phases. Since this level of required detail is difficult, if not impossible, to foresee in advance, specification, design, and implementation must be recognized as an iterative activity. It is therefore critical that technology exist for recovering as much of this activity as possible as the specification is elaborated and modified (during both initial development and later maintenance) [3].

Principle #5: A system specification must be a cognitive model.

The system specification must be a cognitive model rather than a design or implementation model. It must describe a system as perceived by its user community. The objects it manipulates must correspond to the real objects of that domain; the agents must model the individuals, organizations, and equipment in that domain; and the actions they perform must model those actually occurring in the domain. Furthermore, it must be possible to incorporate into the specification the rules or laws which govern the objects of the domain. Some of these laws proscribe certain states of the system (such as "two objects cannot be at the same place at the same time"), and hence limit the behavior of the agents or indicate the need for further elaboration to prevent these states from arising. Other laws describe how objects respond when acted upon (e.g., Newton's laws of motion). These laws, which represent a "physics" of the domain, are an inherent part of the system specification.

Principle #6: A specification must be operational.

The specification must be complete and formal enough that it can be used to determine if a proposed implementation satisfies the specification for arbitrarily chosen test cases. That is, given the results of an implementation on some arbitrarily chosen set of data, it must be possible to use the specification to validate those results. This implies that the specification, though not a complete specification of HOW, can act as a generator of possible behaviors among which must be the proposed implementation. Hence, in an extended sense, the specification must be operational.

This operationality may exist only in a theoretical sense, since it involves replacing existentially and universally quantified objects in the specification by brute force generation and testing (the British Museum algorithm) of all possibilities (which may be infinite).

One way in which the specification can be used to partially test an implementation without relying on the British Museum Algorithm is to use the possibilities generated by the implementation on a specific run in place of the search required by the British Museum Algorithm. The specification then becomes a validation filter for the generated possibilities (it does not, however, guarantee that all valid possibilities were generated by the implementation, only that those generated are valid).

Principle #7: The system specification must be tolerant of incompleteness, and augmentable.

No specification can ever be totally complete. The environment in which it exists is too complex for that. A specification is always a model--an abstraction--of some real (or envisioned) situation. Hence, it will be incomplete. Furthermore, as it is being formulated it will exist at many levels of detail. The operationality required above must not necessitate completeness. The analysis tools employed to aid specifiers and to test specifications must be capable of dealing with incompleteness. Naturally this weakens the analysis which can be performed by widening the range of acceptable behaviors which satisfy the specification, but such degradation must mirror the remaining levels of uncertainty.

Principle #8: A specification must be localized and loosely coupled.

The previous principles deal with the specification as a static entity. This one arises from the dynamics of the specification. It must be recognized that although the main purpose of a specification is to serve as the basis for design and implementation of some system, it is not a precomposed static object, but a dynamic object which undergoes considerable modification. Such modification occurs in three main activities: formulation, when an initial specification is being created; development, when the specification is elaborated during the iterative process of design and implementation; and maintenance, when the specification is changed to reflect a modified environment and/or additional functional requirements.

With so much change occurring to the specification, it is critical that its content and structure be chosen to accommodate this activity. The main requirements for such accommodations are that information within the specification must be localized so that only a single piece (ideally) need be modified when information changes, and that the specification is loosely structured (coupled) so that pieces can be added or removed easily, and the structure automatically readjusted.

REALIZATION

The above principles have been used to determine 18 requirements of good specification languages, and these in turn have formed the basis for a new formal specification language. This language has attempted to include formal versions of the high level constructs used to informally describe systems rather than provability criteria.

This language uses a single global data base to model the current state of a domain through fully associative relations between objects.

Objects have neither a "value" nor structure. They are atomic units which gain definition from their relationships with other objects. The uniformity of this relational modeling suppresses representational issues, and the associativity of the relations suppresses access path issues.

The global data base contains a powerful inference capability so that the distinction between explicit and derived information is hidden. Thus, ANY computation which does not involve a state change in the modelled domain is a "self-organizing" sequence of the needed inferences.

Only the CURRENT STATE of the domain is modelled. All previous states, and the sequence of states itself is automatically part of the information space which can be interrogated, through historical reference, just like the current state. The domain specification is thus freed from having to explicitly provide a memory component which retains all needed information from the past and instead just models the current state of the objects in the domain.

Constraints are used to define required or prohibited states (or through historical reference, sequences of states). These constraints are neither declarative comments nor dynamic consistency checks which cause errors to be raised. Rather, they interact with the non-deterministic constructs (such as "exists x such that P[x]") to determine which of the possible behaviors are acceptable. Only those behaviors which don't violate any constraints can occur. This provides a powerful declarative method for controlling a process oriented specification in terms of acceptable behaviors without having to define how choices are made. It is the implementation task and not the specifier's to determine what criteria, testable in the current state, guarantees the acceptability of the result.

The domain is modelled as a set of autonomous interacting agents that respond to situations in the world and react by performing actions which modify the world. Normally, a subset of these agents would constitute a system to be implemented.