URC97042

Integrity Constraint Monitoring in Software Development: Proposed Architectures

Francisco G. Fernandez*
Department of Computer Science
The University of Texas at El Paso 79968
ffernand@cs.utep.edu

1 Introduction

In the development of complex software systems, designers are required to obtain from many sources and manage vast amounts of knowledge of the system being built and communicate this information to personnel with a variety of backgrounds. Knowledge concerning the properties of the system, including the structure of, relationships between and limitations of the data objects in the system, becomes increasingly more vital as the complexity of the system and the number of knowledge sources increases. Ensuring that violations of these properties do not or-cur becomes steadily more challenging. One approach toward managing the enforcement of system properties, called *context monitoring* [2, 3], uses a centralized repository of integrity constraints and a constraint satisfiability mechanism for dynamic verification of property enforcement during program execution.

The focus of this paper is to describe possible software architectures that define a mechanism for dynamically checking tile satisfiability of a set of constraints on a program. The next section describes the context monitoring approach in general. Section 3 gives an overview of the work currently being done toward the addition of an integrity constraint satisfiability mechanism to a high-level program language, SequenceL, and demonstrates how this model is being examined to develop a genera] software architecture. Section 4 describes possible architectures for a general constraint satisfiability mechanism, as well as an alternative approach that, uses embedded database queries in lieu of an external monitor. The paper concludes with a brief summary outlining the, current state of the research and future work.

2 Background

This section provides an overview of the context monitoring approach to software development,. In addition, a detailed examination of a dynamic constraint satisfiability mechanism is provided.

2.1 Context Monitoring: A Brief Overview

Context monitoring [3] is an approach to software development that uses knowledge of the data objects of a software system to ensure program correctness with respect to selected properties, The approach consists of two parts [4]:

- the elicitation and specification of constraints on data or objects being modeled by the system, and
- a constraint satisfiability mechanism that dynamically verifies constraint enforcement during program execution.

Constraints on the data objects of a system can be identified at any stage of the development cycle. Domain experts and end-users identify constraints that define the behavior of the system, System developers may further refine the behavior of the system by making assumptions about the properties of data objects and the environment in which the program will run during the design and implementation of the system. For example, consider a software system for pharmaceutical sales. A government pharmaceutical board may

* THIS WORK WAS SPONSORED BY NASA UNDER CONTRACT NAG-1012 AND NCCW-0089

state that each drug sold at the pharmacy is identified by a unique ID code specified by the board. In turn. the developers of the system may make an assumption about these ID codes, such as assuming that all ID codes are at most 15 characters in length. Such an assumption may not necessarily have been identified by the domain experts in field of pharmaceuticals, but is nonetheless a restriction on the operation of the system imposed by the developers.

Work is currently being done to define methodologies for eliciting system constraints from domain experts and cud-users during the requirements analysis, functional specification and other stages of software development (see, for example, [5]).

2.2 Constraint Satisfiability Mechanism

The constraints are specified as statements in first-order logic and maintained in a central repository. During program execution, this repository is consulted to deter-mine if the system is in fact enforcing the constraints.

An important, concept of the constraint satisfiability mechanism is the *state* of the program: a set of program variable-value pairs that capture a snapshot of memory at a given point of time during program execution. Constraint checks are based on changes in the state of a program.

The idea behind constraint satisfiability is to monitor programs for violations of constraints during execution. The state of the program is monitored and, when a change in state occurs, all constraints relevant to this state change are checked for violations, For example, if the value of variable X changes after an execution step, then only those constraints associated with variable X are checked for violations. If no violations occur, program execution continues. If a constraint violation does occur, the integrity of the program data has been compromised and as a result, program correctness from that point on can no longer be ass u red.

3 Constraint Satisfiability in SequenceL

This section describes the implementation of a constraint satisfiability mechanism in a high level language called SequenceL. A brief description of the language is provided, and the motivation for using it for the initial attempt is described. A description of the implementation is then given.

3.1 The SequenceL Language

SequenceL is a high-level language for processing non-scalar data [I]. In SequenceL, problems are solved by specifying the form and content of the data to processed. The iterative/recursive details of the problem are abstracted from the solution.

The basic data structure of the language is the sequence. From this simple structure, any other data structure can be constructed (i.e., sets, trees, records, etc..). In addition to the basic data structure, there are also basic operations which can be performed on sequences, such as addition, subtraction, multiplication, division, and other more complicated operations. A more detailed description of the language can be found in [1].

The language uses the notion of a *universe*, a collection of variable-value pairings. There are no explicit methods for input and output to a SequenceL program. Instead, program input is provided implicitly as the initial universe, and the final universe is the implicit output. A program begins with a collection of functions and an initial universe. A SequenceL function is defined by the domain arguments it must have in order to execute, the range arguments that arc produced as a result of processing the domain arguments, and the operations to be performed to process the domain and produce the range. SequenceL operates on an event-based model. As a result, function execution is not sequential, but based on the availability of parameter data in the universe. This makes the language non-deterministic, and also open to parallelizability. A. function is executed only if al] of the domain arguments of the functions arc available in the universe. [f a function is eligible, it consumes the domain arguments from the universe, executes the body of the function, and places the range arguments that result, from processing back in the universe.

3.2 Implementation of Constraint Satisfiability

Extending the SequenceL execution model for constraint satisfiability involves two major tasks. First, identifying where in the execution model constraint checks should be made. A state change occurs at a clearly identifiable point, i.e., at the end of execution of a function, The second more difficult task is to extend the

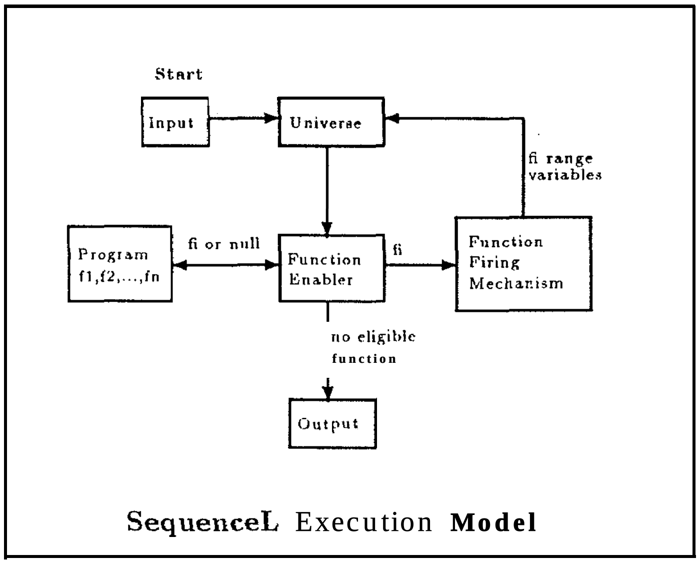

Figure 1: The SequenceL Execution Model.

SequenceL language to define what it means to check integrity constraint satisfiability for a program and a set of constrain. For simplification, the integrity constraint repository is represented by a program composed of boolean functions defining the relations and limitations of data objects in the Sequence], program. These constraint functions are defined as SequenceL functions, and thus the same execution engine is used to check the satisfiability of a constraint as is used to execute the SequenceL program. Figure 1 demonstrates the Sequence], execution model, and figure 2 shows the extended model with constraint satisfiability monitoring.

The development of a constraint satisfiability monitor in SequenceL is more than just an exercise. in fact, the reason for developing the monitor for a specific programming language is to study the interaction between constraint monitoring and program execution. The SequenceL model demonstrates how monitoring occurs in a program, and more importantly, provides a basic architecture model which can be used as the basis for a more general monitor.

4 Generalizing the Satisfiability Mechanism

The ultimate objective of the research is to create a general constraint satisfiability mechanism that can be used to ensure enforcement of data object properties independent of the programming language used to build a software system. This section examines possible software architectures for such a mechanism and alternatives to an external monitor.

4.1 Issues

To understand how to provide a general constraint satisfiability monitoring mechanism, it is important to first identify some of the issues.

Any general mechanism will require recta *knowledge* about the environment it will be monitoring, such as how variables take on values and knowledge that ties constraints to the program. As a simple example, consider a language that allows the usc of array data structures. If the array is being processed within a loop, it might be the case that array should not be considered "changed" until the end of the loop processing. In other words, even though the array is undergoing a change during each iteration of the loop, it is not considered to be processed until the loop terminates. This type of information would be captured in the meta-knowledge.

A second consideration is how the monitor is related to the program it is monitoring. One possibility is that it forms a "wrapper" around the program, In this sense, the monitor is external to the program, and monitors it from the outside. It is somehow made aware that a state change has occurred in the program and that constraint checks must be made. Another possibility is that the monitor uses the meta knowledge

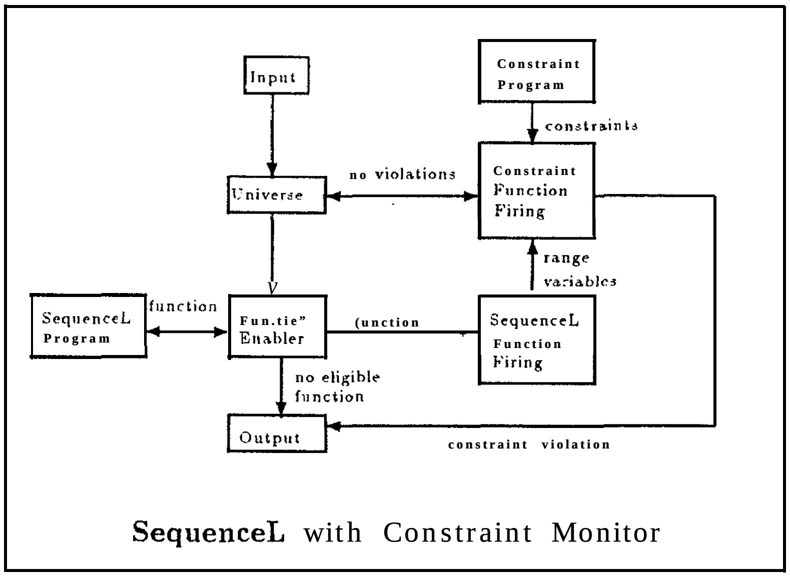

Figure 2: The SequenceL Execution Model with Integrity Constraints.

of the program environment to embed the constraint checks within the program itself. In this sense, the monitor is more of a pre-processor that transforms the program in such a way that it is able to monitor for constraint violations itself.

A third important issue is how constraints will be represented. This issue is dependent upon the implementation of the monitoring system. If the monitor is an external system independent of the program it is monitoring, it is possible to use a tlrst-order logic representation of constraints. If constraints are to be embedded within the program it will be necessary to translate them from their first-order representation into equivalent representations in the language of the program

4.2 Architectures for External Monitoring of Constraint Satisfiability

Two architectures are outlined: the *event model* and the *tagged program model.* In the event model [7], changes in a program state implicitly trigger constraint satisfiability checks. The program broadcasts a change in state. The satisfiability mechanism, an external system monitoring the program, registers this state change broadcast and invokes appropriate constraint, checks. The monitor must use meta-knowledge to identify how the program will broadcast a change in state and how it will have access to the variables it must check for violations. Figure 3 gives a pictoral view of the event model, One advantage to the event, model is that the constraint monitoring can be easily parallelized. The program simply broadcasts that a change in state has occured and continues execution while the monitor checks the satisfiability of the constraints.

A second model is the tagged program model. Here, the monitor uses the recta-knowledge about the programming language to tag the source code for constraint checks. Prior to compilation, the program source code is passed to the satisfiability mechanism. The mechanism uses knowledge about the language and state changes to tag the program at points where checks should be made. During program execution, a constraint check is heralded when a tag is reached. Tags may be at t ached to variables in the program's symbol table, indicating constraints should be checked when the value of that variable changes. Alternatively, tags may be placed after program statements, indicating that constraints should be checked upon completion of that statement. Figure 4 shows an overview of the tagged program model.

4,3 Embedded Constraint Enforcement

An alternative to an external satisfiability monitor is the use of embedded constraint satisfiability checks. Some languages, such as C, provide constructs to embed assertions in programs to check program properties [6]. The main disadvantage to using embedded checks such as these is precisely the fact that the constraint checks arc embedded. With embedded checks, it is not possible to reason about or study constraints outside

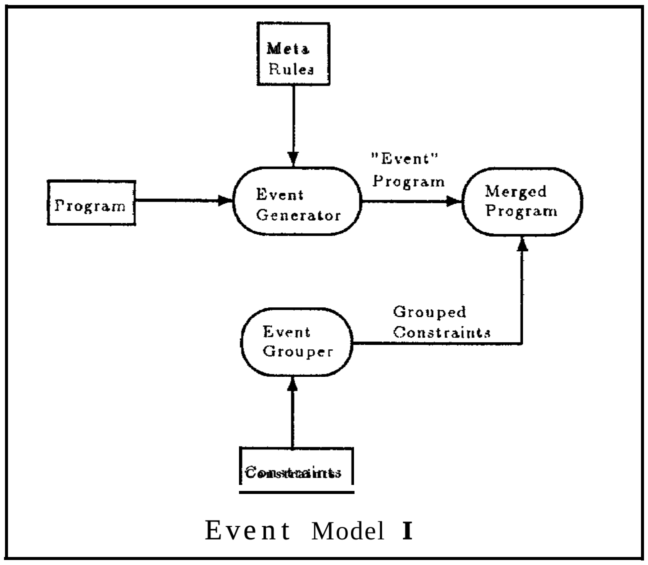

Figure 3: Architecture Diagram for the Event Model.

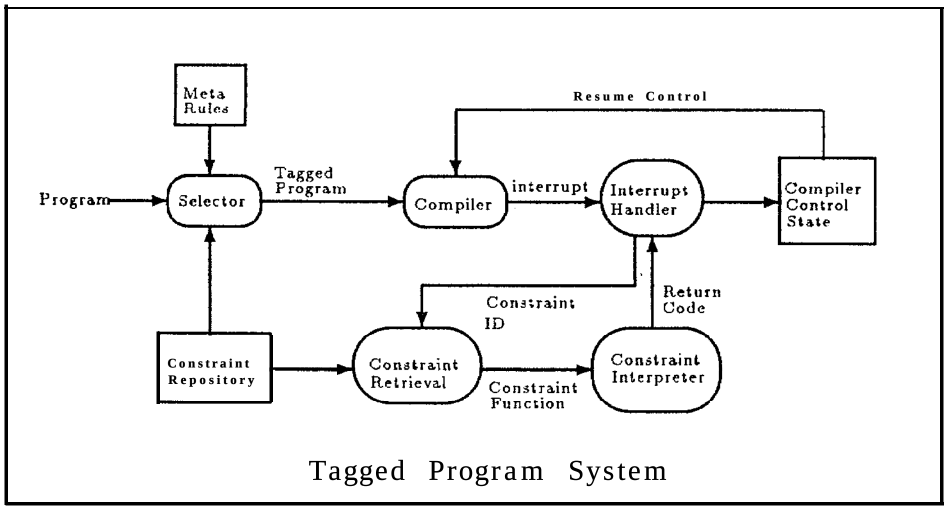

Figure 4: The Tagged Program Architecture Diagram.

of the program. If a change in constraints is required, it is necessary to untangle the embedded checks from within the program, which may be a difficult and tedious process.

Another possibility is to use language extension packages that provide interfaces to external querying systems. An example of such a package is EQUEL (Embedded QUEL), which provides FORTRAN with an interface Lo the INGRES relational database system. The disadvantage here is that every language would have to have an interface package specific to that language. Thus, the constraint satisfiability would not be a general mechanism, hut, a distinct one for each language.

5 Summary

Context monitoring and constraint satisfiability are powerful tools for managing the enforcement of vital system properties. By providing a system where constraints are maintained separately from a program it is possible to study, reason about, and modify them separately from the system. In addition, it may be possible to parallelize the constraint monitoring process to improve overall performance of the system.

Work is currently being be done toward designing a general satisfiability monitor. One issue being addressed is an analysis of programming languages to develop meta-rules about how variables under-go changes and how constraints can be associated to variables within a program. Also, methods for integrating a general constraint satisfiability monitor and different programming languages is being examined.

References

[1] Cooke, D. E., "An Introduction to SequenceL: A Language to Experiment with Constructs for Processing Nonscalars," to appear in *Software Practice and Experience,* 1996.

[2] Gates, A., *Context Monitoring With Integrity Constrain.* Las Cruces, NM: New Mexico State University, 1994 (Ph.D. Dissertation).

[3] Gates, A.Q. and Cooke, D. E., "The Use of Integrity Constraints in Software Engineering", *SEKE '95 Proceedings Software Engineering Knowledge Engineering,* Rockville, MI), 1995. Skokie, IL: Knowledge Systems Institute, 1995, pp. 383-390.

[4] Gates, A. Q., "On Defining a Class of integrity Constraints," *SEKE '96 Proceedings Software Engineering Knowledge Engineering,* Lake Tahoe, NV, 1996. Skokie, IL: Knowledge Systems Institute, 1996, pp. 338-344.

[5] Gates, A. Q., "Building Systems with Integrity Constraints" ,*Proceedings of the Second World Conference on integrated Design and Process Technology,* Austin, 'TX, 1996.

[6] Rosenblum, D. S., "A Practical Apporach to Programming with Assertions," *IEEE Transactions Software Engineering, 21(1),* 1995, pp. 19-31.

[7] Shaw, M. and Garland, D., *Software Architecture: Perspectives* 071 an *Emerging Discipline* Upper Saddle River, New Jersey: Prentice Hall, 1996.