

# An Automated Approach for Goal-driven, Specification-based Testing

Kristina Winbladh, Thomas A. Alspaugh, Hadar Ziv, and Debra J. Richardson  
Donald Bren School of Information and Computer Sciences  
University of California, Irvine  
Irvine, CA 92697-3425 USA  
{awinblad, alspaugh, ziv, djr}@ics.uci.edu

## Abstract

*This paper presents a specification-based approach that addresses several known challenges including false positives and domain knowledge errors. Our approach begins with a goal graph and plans. Source code is annotated with goals and events and precompiled to emit those at run time. Plans are automatically translated into a rule-based recognizer. An oracle is produced from the pre- and postconditions associated with the plan's goals. When the program is executed, goals and events are emitted and automatically tested against plans and oracles. The concept is demonstrated on a small example and a larger publicly available case study.*

## 1. Introduction

Specification-based testing supports the efficient use of project resources by focusing on the most important behavior, i.e., that which the stakeholders specified. This paper presents a specification-based testing approach and tool support that focus on finding mismatches between actual and expected system behavior.

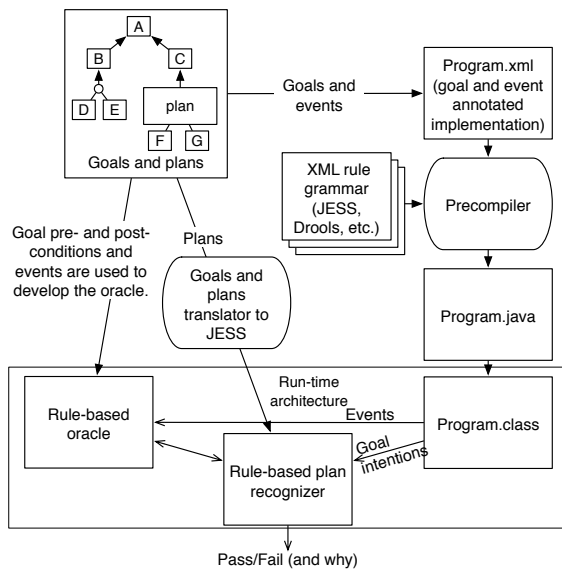
There are two major contributions of our work: First we can test whether an implementation follows an appropriate plan to get the correct results. It is usually impossible to do exhaustive testing, efficient testing should thus focus on whether the system achieves its most important goals. Testing against plans and goals also helps detect false positives. A *false positive* occurs when the program is using an incorrect process, but happens to produce a correct result in a specific case. We verify that the system not only produces correct results but does so by following a plan that can be expected to produce correct results in all similar cases. We can also detect domain knowledge errors – cases when the system follows an intended plan but achieves incorrect results. Second, we provide automated support for this approach that substantially reduces the extra labor.

Similar to Dardenne and van Lamsweerde and Mylopoulos *et al.* [6, 7], we define a *goal* to be the purpose toward which an effort is directed. High-level goals are step-wise refined to derive functional requirements goals, which can be realized as code components. The functional goals are further refined by plans, so that each functional goal is the root of a plan. This refinement of high-level goals to detailed plans is similar to the NFR framework [7] and Cockburn's three leveled use-cases [3] in that the process goes from the requirements level to the design level. A *plan* is an abstract description of how to satisfy some goal by satisfying its subgoals (lowest level goals) in some order. The plans may contain sequences, iterations, and alternations of lowest-level goals. A lowest-level goal is characterised by pre- and postconditions. Each of these conditions represents some state. State transitions from pre- to postcondition can be associated with an abstract operation and identified with a single piece of code which is its implementation. An *event* is an instance of an abstract operation. A lowest-level goal can thus be satisfied by a single event or a sequence of events that alter the system state from the goal's precondition to its postcondition.

The remainder of this paper is organized as follows: Section 2 provides an overview of our approach. Section 3 gives a brief summary of a previous proof-of-concept demonstration, in which we implemented our approach for a specific example, a Tic-Tac-Toe (TTT) program. In Section 4 we apply our approach and tools on a larger example. In Section 5 we summarize related work and finally in Section 6 we present lessons learned and discussion of future work.

## 2. Approach

Figure 1 shows an overview of our testing approach. We determine expected system behavior from the higher-level goals, the lower-level goals they are refined into, and the relationships between those goals. At the upper levels of the goal refinement graph, these relationships are AND/OR



**Figure 1. Process diagram**

refinements. At the lower levels, functional goals are also refined using plans for how each goal is satisfied by a sequence of lowest-level goals. At the lowest goal level, we use oracles to determine whether each goal has been satisfied. The oracles are derived from the pre- and postconditions for each goal as well as events identified during design. At higher levels, we use the plans, or the AND/OR relationships where there are not plans, to determine whether a higher-level goal has been satisfied through the satisfaction of a group of its subgoals; we can also use oracles here as an additional confirmation.

An essential feature of our approach is annotating the system's code with goals from plans and events, so that during program execution these goals and events will be emitted. We precompile the code, transforming the annotations into additional code that will emit statements of intention to satisfy each goal and events the oracle uses to verify that each goal was in fact satisfied, at the appropriate points in the execution. We automatically translate the plans into a plan recognizer that runs with the program and matches its emitted goals with the expected goals. Satisfaction of goals is verified by an oracle, based on the events and the goals' pre- and postconditions. When we run the program with the plan recognizer and oracle, they find mismatches between actual and expected behavior, not just the specific results but also the process by which they are obtained.

Our hypothesis is that goals, goal graphs and plans are an especially advantageous foundation for our approach. A single form, goals, can be used at all levels of abstraction, providing a smooth transition from upper to lower levels that can be validated by stakeholders, both in terms of specific goals and in terms of the relations between them. Goal

graphs with plans allow us to infer satisfaction from the lowest levels up to the highest. Furthermore, plans support a substantial degree of automation.

### 3. Concept Demonstration with Tic-Tac-Toe

In many cases, raw event traces are not sufficient to detect false positive results. For example, consider a raw event trace from a Tic-Tac-Toe (TTT) program being tested to verify the skill level at which the machine is playing against a human player. At the 'expert' level of play, the machine is required to identify fatal second moves and exploit these with a forcing move that leads inevitably to a fork (a situation with two winning moves) and a win. The 'intermediate' level of play requires the machine to identify immediate opportunities to make a winning move or create a fork that leads to a win on the next move, but not to recognize such opportunities one move ahead. Since a raw event trace only shows events, we cannot distinguish whether our program is playing at the expert or intermediate level, i.e. if a move that led to a fork was a forced or random. One approach to testing this would be to perform a large number of test cases, verify that each is consistent with the desired behavior, and use statistics to argue the degree of confidence. Our approach is to produce a goal-annotated event trace that provides more direct evidence.

A goal annotated event trace shows both events and intentions. If the program was playing at the 'expert' level, the goal annotated 'expert' trace would show that the program recognized the player's fatal second move and made a forcing move that led to a fork and a win, by revealing its intentions and actions that produced that result. If the program was instead playing at the 'intermediate' level the goal annotated 'intermediate' trace would reveal that the move that led to the fork was random and not an intentional response to the player's second move. Cases where the program produces correct results but does not follow the intended plan are thus indications of false positives.

Our approach identifies domain knowledge errors by detecting mismatches between expected and actual results even though the intended plan is being followed. If the expert-level strategy uses incorrect definitions of safe and fatal second moves when choosing its moves, the plan recognizer will detect event-traces of games in which the program believed it was making a safe second move, emitted a goal with that intention, and then an event (a move) that the oracle identified as fatal. In our TTT study, we deliberately introduced such errors that our tests detected by finding games in which the events did not satisfy the pre- and postconditions of the current goal of the plan.

## 4. Prototype Application with ATM

### 4.1. Introduction to the ATM Simulation

This exploratory case study shows part of our approach (goals, plans, goal refinement graph, and recognition of plans) applied to a publicly available ATM simulation system developed elsewhere [1]. The ATM program simulates the functionality of a real ATM. An authorized customer is able to perform withdrawals, deposits, transfers, and balance inquiries. The ATM communicates each request to the bank for authorization. If the bank component determines the customer's PIN invalid, the customer is required to re-enter the PIN correctly. Requirements state that if the customer is unable to successfully enter the PIN after three tries, the card will be permanently retained by the machine.

### 4.2. Goal Refinement

Since the ATM project did not provide goals, we re-constructed a goal refinement graph from the requirements and standard banking business rules and goals (see [10] for details). The graph includes high level goals such as **Prevent unauthorized access to accounts**, low level functional goals such as **Withdraw** that map to a single software component, and some goals, such as **ATM authentication** that impact many components of the system. Some parts of the system thus satisfy several goals.

### 4.3. ATM Session Plan

Plans were constructed for the functional goals (see [10]). The `<goal>` elements describe the goals, and their sequence is described by the `<iteration>`, `<sequence>`, and `<alternation>` elements. For example, the alternation of the sequences **Do authenticated transaction** and **Retain unauthenticated card** illustrate valid interactions with the ATM.

### 4.4. Goal-Annotating the Code

We used goals from the plans to manually goal annotate the implementation code for three of the classes in the ATM system (`Transaction.java`, `Session.java`, and `ATMMain.java`).

### 4.5. Precompilation

Our precompiler translated the GoalML-annotated version of the program into a program that emits the goals it is intending to achieve during execution. The GoalML annotations were replaced by program statements (JESS assertions in Java) where each assertion adds a goal as a fact to the working memory of the plan recognizer.

### 4.6. Plan Recognizer

We created JESS rules that together recognize whether the **ATM session** plan is followed. The plan recognizer always knows which goals are expected next, and moves forward through the plan when one of these occurs. The recognizer also contains rules that infer satisfaction of higher-level goals as a result of satisfaction of lower-level goals, by direct AND-OR relations or plans (see [10] for more detail).

### 4.7. Executing Program and Recognizer

When the precompiler-generated program executes, a JESS engine containing the rules of the plan (i.e., the recognizer) is instantiated. As the program executes, goals are asserted into this rule-engine. The first goal asserted by the ATM simulation was **Read card**; this goal was expected by the plan recognizer. The goals **Get PIN**, **Read transaction type**, and **Send transaction request to bank** were then received and matched against the plan. We ran a session using a valid ATM card and PIN, and the result was a sequence of goals accepted by the plan recognizer. Once the plan was matched, the higher-level goal **Provide service** was also matched which in turn inferred the satisfaction of the goal **Provide services**.

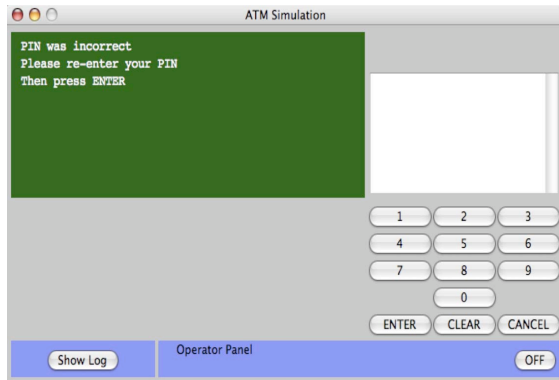
We then ran the simulation using a valid ATM card and invalid PIN. This gave a prefix of the correct goal sequence, but after we entered an incorrect PIN for the third time the plan recognizer detected a mismatch indicating that the program was not following the **ATM session** plan. Figure 2 shows the goal trace with the unexpected fourth (total) **Get PIN** goal; the expected goal at that point was **Permanently retain ATM card**. An incorrect PIN must be entered four times before the ATM card is retained by the simulation, and not three as the requirements stated.

### 4.8. Discussion

Our approach is specifically designed to catch errors manifested as mismatches between actual and expected system behavior. An informal analysis of the error indicates that it can be traced back to problems in the design artifacts, and more specifically to the challenge of moving from one type of software representation to another. This is a likely place to introduce errors which then get propagated through the design and finally manifest themselves in code that does not match the requirements. We believe that following goals and plans reduces the risk of introducing such errors.

## 5. Related Work

In addition to work by van Lamsweerde *et al.* [6], and Mylopoulos *et al.* [7], Uchitel *et al.* provide additional def-



Goal: "Read card" was received when expected.  
 Goal: "Get PIN" was received when expected.  
 Goal: "Read transaction type" was received when expected.  
 Goal: "Send transaction request to bank" was received when expected.  
 Goal: "Get PIN" was received when expected.  
 Goal: "Get PIN" was received when expected.  
 Goal: "Get PIN" was received but not expected.

**Figure 2. Goal trace with mismatch**

initions of goals and scenarios and how they are related to meaningfully describe a system's requirements in terms of intentions and operations [9].

Programming with assertions for discovering program errors has been a topic under investigation for a long time [4, 8]. A challenge with previous techniques is that they do not integrate easily with existing programming environments, an issue our approach addresses. This issue is also addressed by JML, an annotation language that supports runtime assertion checking and testing, as well as other uses [2]. Bandera verifies properties that refer to entities in the program [5] rather than annotating the source code. Although these techniques are effective for a number of types of verification, neither one focuses on verifying that the intended goals of the system are met.

Specification-based assertions have been used as a means for testing [4]. This method involves translating formal specifications to program assertions. During program execution these assertions are checked for violations.

## 6. Lessons Learned and Future Work

Our specification-based approach compares goals and plans against program source code. We evaluated our approach on a TTT game and a larger, publicly-available ATM simulation. The examples exhibit several interesting results. First, in our TTT study, we were successful in finding both false positives and domain knowledge errors. In the ATM simulation case study, we identified a less-clearly characterized mismatch between actual and required system behavior. Second, we were able to significantly automate our

approach, making it substantially quicker and more straightforward to use.

One limitation with our current oracle is that it knows nothing about the state of the world other than what the annotations tell it. In the future we plan oracles that determine the state of the world directly. This would reduce the number of annotations in the code, since event annotations would no longer be needed, and directly determine contributions the environment makes toward goal satisfactions.

Our future goal is to evaluate the efficiency of our testing approach by conducting a more extensive case study in which we measure the time it takes to write plans and annotate code compared to other preparatory work for testing. We also want to evaluate how effective this approach is compared to other kinds of testing. We plan to extend our work to include not only goals, plans, and code, but also other development artifacts such as software architectures, other design representations, requirements, and so on. We hypothesize that with additional artifacts and with traceability among those artifacts, more efficient, purposeful testing can be accomplished. Our ultimate goal is to provide forward mapping of requirements to other software artifacts and to provide automated traceability from those artifacts back to requirements.

## References

- [1] R. C. Bjork. ATM Simulation, 2002. <http://www.math-cs.gordon.edu/local/courses/cs211/ATMexample/>.
- [2] L. Burdy, Y. Cheon, D. R. Cok, M. D. Ernst, J. R. Kiniry, G. T. Leavens, K. R. M. Leino, and E. Poll. An overview of JML tools and applications. *Int. J. Softw. Tools Technol. Transf.*, 7(3):212–232, 2005.
- [3] A. Cockburn. *Writing Effective Use Cases*. Addison-Wesley Longman Publishing Co., Inc., 2000.
- [4] D. Coppit and J. Haddox-Schatz. On the use of specification-based assertions as test oracles. In *29th IEEE/NASA Softw. Eng. Wkp.*, pages 305–314, 2005.
- [5] J. Corbett, M. Dwyer, J. Hatcliff, S. Laubach, C. Pasareanu, and R. Zheng. Bandera: Extracting finite-state models from Java source code. In *22nd Int. Conf. on Softw. Eng.*, 2000.
- [6] A. Dardenne and A. van Lamsweerde. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1–2):14–21, 1993.
- [7] J. Mylopoulos, L. Chung, and B. Nixon. Representing and using non-functional requirements: A process-oriented approach. *IEEE Trans. Softw. Eng.*, 18(6):483–497, 1992.
- [8] D. Rosenblum. A practical approach to programming with assertions. *IEEE Trans. Softw. Eng.*, 21(1):19–31, 1995.
- [9] S. Uchitel, R. Chatley, J. Kramer, and J. Magee. Fluent-based animation: Exploiting the relation between goals and scenarios for requirements validation. In *RE*, 2004.
- [10] K. Winbladh, T. A. Alspaugh, H. Ziv, and D. Richardson. An Automated Approach for Goal-driven, Specification-based Testing. Technical Report UCI-ISR-06-8, 2006.