# Monitoring and Diagnosing Software Requirements

*(Wang, McIlraith, Yu, Mylopoulos)*

presented by Michalis Famelis

January 8, 2009

A framework for monitoring and diagnosing failures of a software system to fulfil its requirements.

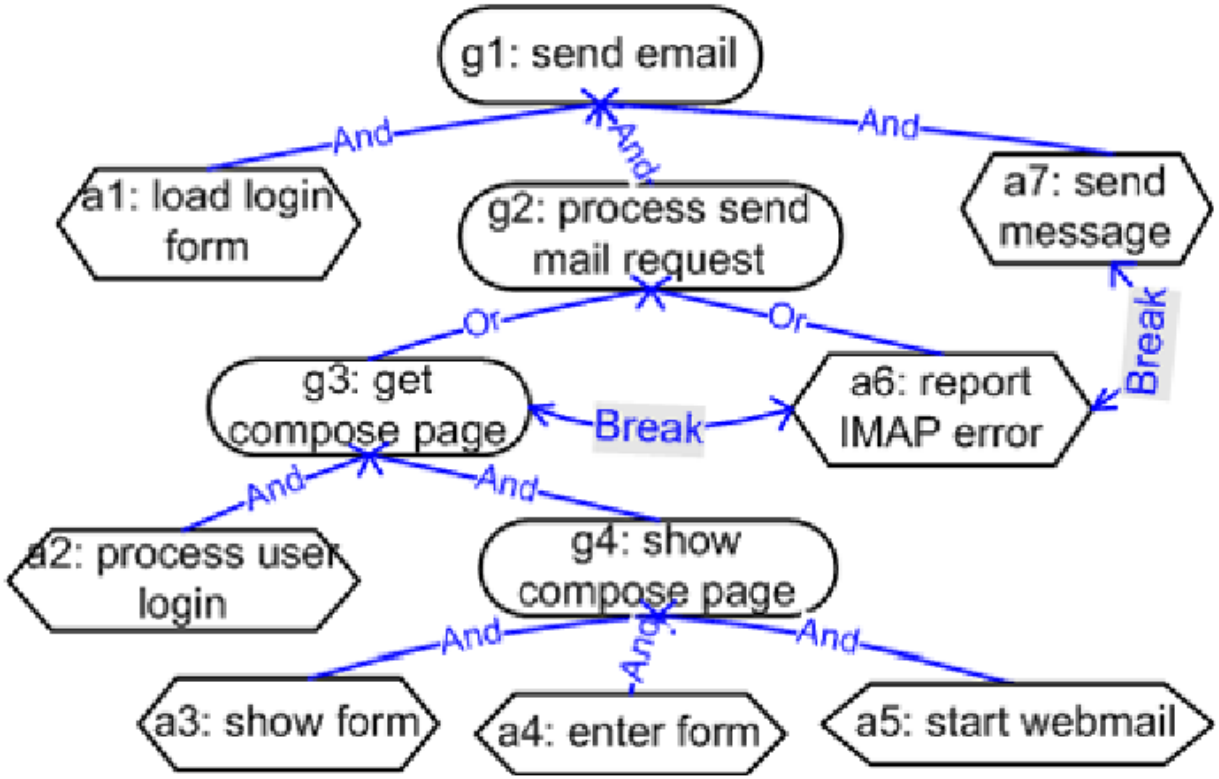
- Monitor software for requirements compliance
- Logs
- Goal models
- Denial of goals
- Diagnosis as a SAT problem
- Application to SOA systems

# Goal Models

# Goal models

- Obtained by requirements analysis or by reverse engineering
- We assume the existence of traceability links
- Graphs with AND and OR decompositions of goals into subgoals and tasks
- Also contain, additional contribution links $(++S, --S, ++D, --D, ++, --)$
- Annotated with preconditions, effects (postconditions)
- and monitoring switches

# Example: Squirrel Mail



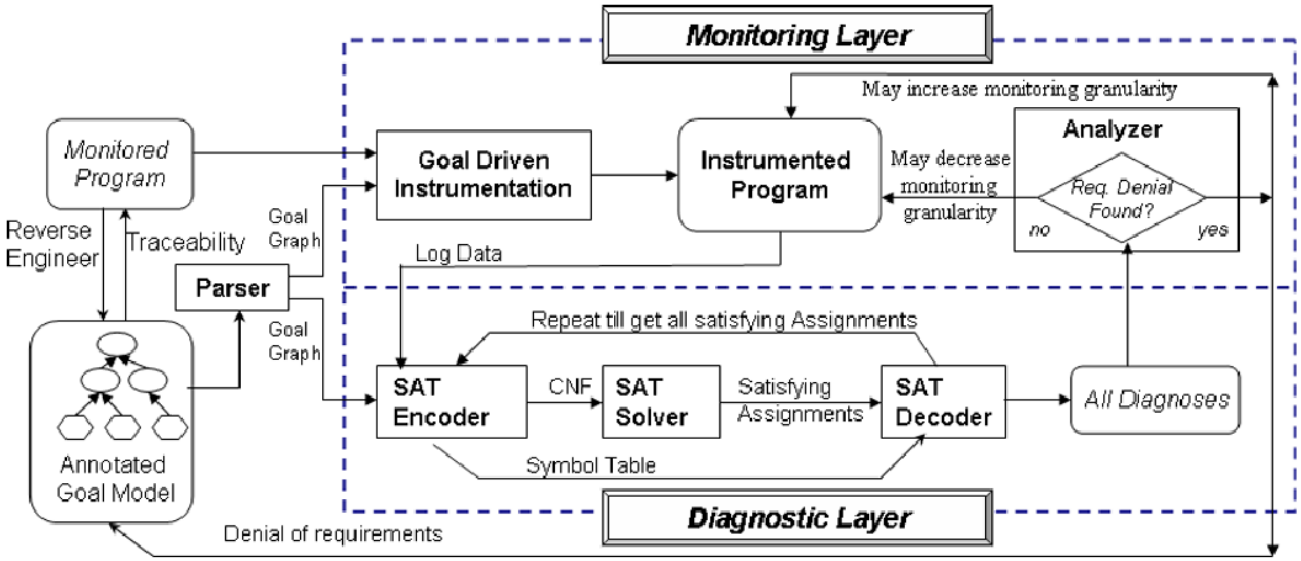
# Squirrel Mail Annotations

**Table 1** Squirrel mail annotated goal model

| Goal/ task | Monitor switch | Precondition | Effect |
|---|---|---|---|
| $a1$ | On | URL entered | Correct form loaded |
| $a2$ | On | $\neg$wrongIMAP $\wedge$ correct form loaded | Correct key entered |
| $a3$ | Off | Correct key entered | Form shown |
| $a4$ | Off | Form shown | form entered |
| $a5$ | Off | Form entered | Webmail started |
| $a6$ | On | WrongIMAP | Error reported |
| $a7$ | On | Webmail started | Email sent |
| $g1$ | Off | URL entered | Email sent $\vee$ error reported |
| $g2$ | Off | Correct form loaded $\vee$ wrongIMAP | Webmail started $\vee$ error reported |
| $g3$ | Off | Correct form loaded $\wedge$ $\neg$wrongIMAP | Webmail started |
| $g4$ | On | Correct key entered | Webmail started |

# Framework Overview

# Overview

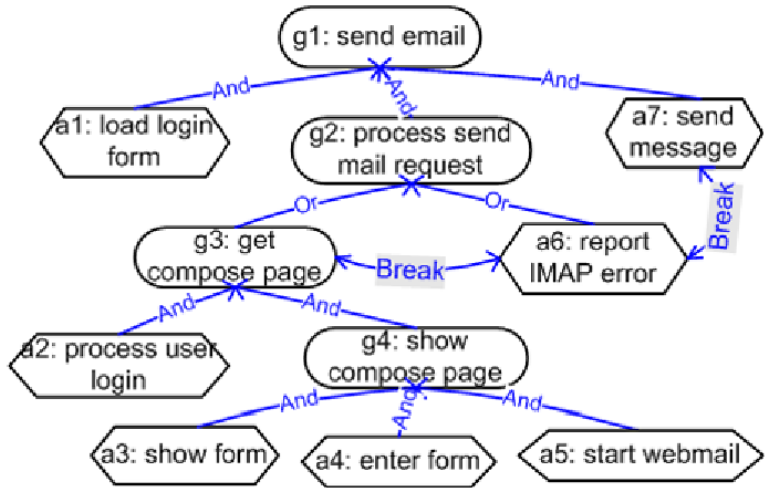



Adaptive monitoring: Can adjust monitoring granularity according to diagnosis using the monitoring switches

# Monitoring

- Monitoring of an instrumented system for a number of sessions
- Logging of the truth values of preconditions and effects
- Logging the occurence of a task (predicate $occ_a(\alpha_i, t)$)

# Example log for Squirrel Mail




$$URLEntered(1), occ_a(a_1, 2), correctFormLoaded(3),$$
$$\neg wrongIMAP(4), occ_a(a_2, 5), correctKeyEntered(6),$$
$$occ_a(a_3, 7), occ_a(a_4, 8), occ_a(a_5, 9),$$
$$\neg webmailStarted(10), occ_a(a_7, 11), \neg emailSent(12).$$

# The formula

- Diagnosis reduced to the satisfiability of a propositional formula $\Phi$
- $\Phi = \Phi_{log} \wedge \Phi_{deniability} \wedge \Phi_{goalModel}$
- $\Phi_{log}$ encodes the log data as mentioned earlier
- $\Phi_{goalModel}$ encodes the goal model (decompositions and contribution links)
- $\Phi_{deniability}$ encodes how tasks or goals are denied with regard to preconditions and effects
- The formula can be extended with $\Phi_{domainConstraints}$

# Encoding Deniability

- Task/Goal Denial
  "If a task/goal occurence was logged and either its precondition didn't hold before it or its effect didn't happen after it, it is denied"
- Task/Goal Session Denial
  "If a task/goal is denied once, it is denied for the whole session"
- Explanation closure (*fluents*)
  "If the truth value of a literal is changed at some point, then some task/goal that has it as an effect must have occured"

# About Diagnoses

- **Diagnosis**: a set $D$ of deniability predicates over all the *goals and tasks* in the goal model, such that $D \cup \Phi$ is satisfiable
- **Core Diagnosis**: a set $CD$ of deniability predicates over all the *tasks* in the goal model, such that $CD \cup \Phi$ is satisfiable
- **Participating Diagnostic Component** (PDC): a deniability predicate over some task in the goal model such that $PDC \cup \Phi$

# Algorithms

- 2 alternative algorithms to encode an annotated goal model into a propositional formula, with (A2) and without (A1) log preprocessing
  - Log preprocessing is a scalability optimization to avoid exponential growth of $\Phi$ by encoding goal/task denial axioms only for timesteps that the goal/task actually occurs
- 2 alternative algorithms for doing diagnosing, using a SAT solver
  - The first algorithm (A3) finds all Core Diagnoses
  - The other (A4) is more scalable because it only bothers to find all Participating Diagnostic Components

# The Framework in Action

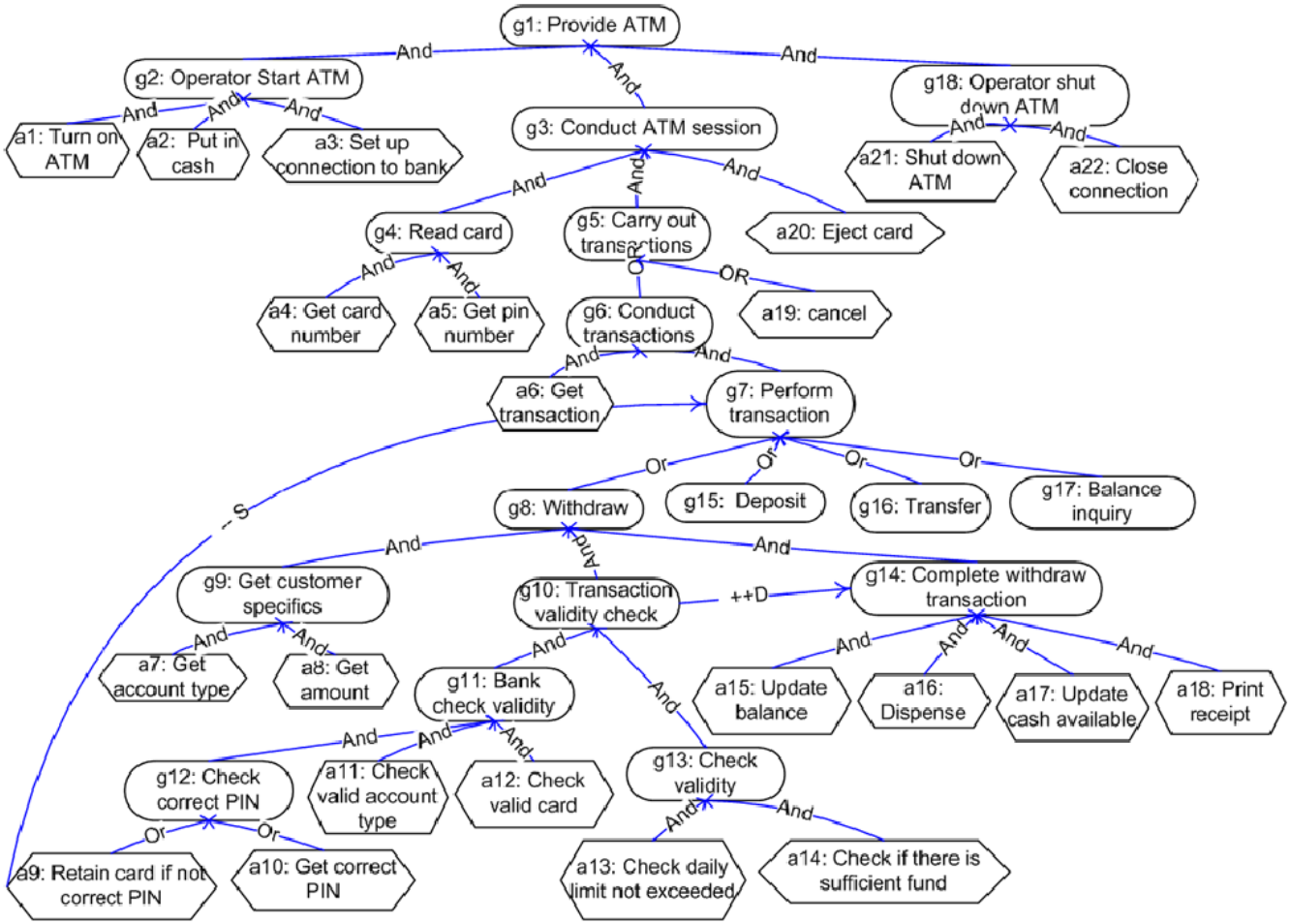
# Notes on Implementation

- Implemented in Java (11 classes, roughly 5 KLOC)
- SAT4J solver as a java library
- Implementation of monitoring specification and instrumentation code with Aspect Oriented Programming (AspectJ)
- Goal Driven Instrumentation:
  - "what" to monitor from the goal model
  - "where" to insert the monitors from the traceability links

# Contribution Links Experiment

- Aim to compare the performance of A3 and A4
- For a denied goal made of $n$ tasks, A3 can return up to $2^n$ Core Diagnoses, A4 only up to $n$ Participating Diagnostic Components
- But the presence of contribution links, as they act as constraints that imporve the efficiency of the SAT solver
- It was confirmed by experiment that with more contribution links the performance gap between A3 and A4 became smaller

# The ATM simulation



# Experimenting with the ATM simulation

- A first set of experiments dealt with the tradeoff between monitoring granularity and diagnostic precision
- A second set explored the issues of scalability of the framework for large goal models
- For both sets, A4 (PDC) rather A3 was used
- The optimized encoding algorithm (A2) was used for the first set and both encoding algorithms for the second set

# First set of ATM experiments

- 5 experiments to monitor increasingly many elements of the goal model, ie increasing monitoring granularity.
- The experiment showed that the number of PDC returned by the diagnostic component is inversely proportional to monitoring granularity
- Also, with greater granularirty the overall load is increased
- But the overall time to diagnose may not increase (more fine grained, less PDCs to find, whereas more coarse, more PDCs to report)
- Benefit by being able to make better diagnoses

# Second set of ATM experiments

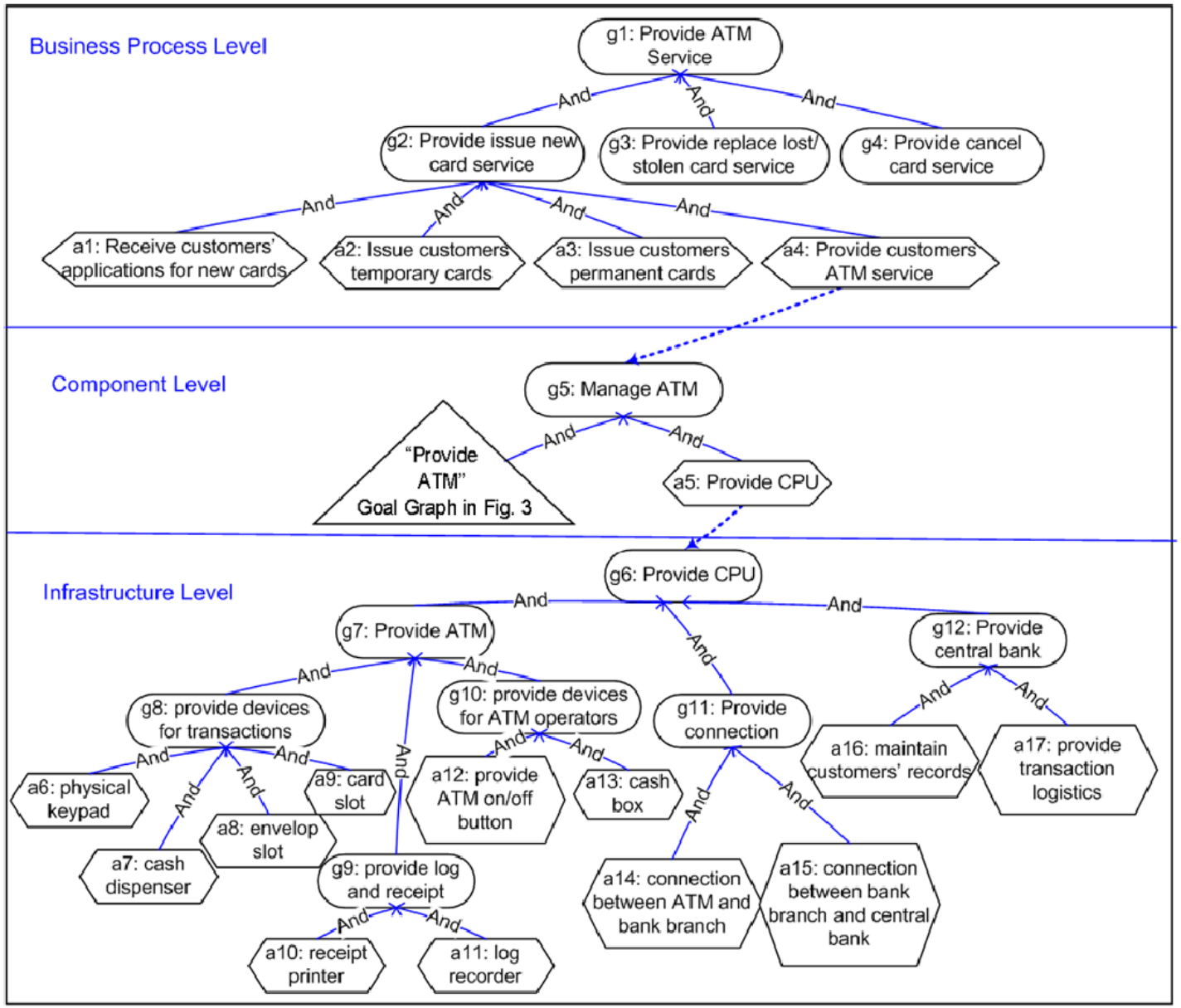
- 20 experiments to assess the scalability of the framework
- Larger models were obtained by cloning the ATM goal model
- The experiments were run with and without log preprocessing for increasingly large goal models.
- Without log preprocessing, the time to encode and diagnose increased exponentially to the size of the GM
- With log preprocessing, the framework scaled well, (almost linearly)

# Application to SOA

# Service Oriented Architecture

- A three layer software architecture
- Business Process Layer (top): treats services as black boxes
- Component Layer (middle): gives the business logic of the BPL services
- Infrastructure Layer (bottom): contains the back-end infrastructure

# ATM SOA global goal model



# Monitoring and Diagnosing Multi-layer systems

- Hierarchical monitoring and diagnosis
- We can monitor on each isolated layer or as a global model
- As higher level depend on the correct functioning of lower leverls, we can treat leafs as black boxes that decompose to root level goals in lower levels
- Tradeoff between monitor granularity and diagnostic precision as noted before

# Evaluation

- 20 experiment sets of 3 experiments (for the 3 SOA levels)
- Each set dealt with a larger global model (more Business Processes)
- Used algorithms A2 and A4
- The experiments showed that the framework scaled well (almost linearly)
- Also, the experiments confirmed again the granularity/precision tradeoff: the framework was most efficient at the BPL and less efficient at the IL

# Conclusion

# Conclusions

- Robust, scalable framework for monitoring and diagnosis of the fulfullment of the requirements of a system, based on its goal model
- A prototype that can be extended to be used for industrial-scale software systems
- The framework can be extended to also handle non-functional requirements, failures by wrong domain assumptions and malicious attacks and different levels of traceability granularity, or the non-existence of traceability

Questions?