Practical Issues with Formal Specifications

Lessons Learned from an Industrial Case Study

Michael Altenhofen *Achim D. Brucker*
{michael.altenhofen, achim.brucker}@sap.com

SAP RESEARCH
Vincenz-Priessnitz-Str. 1, 76131 Karlsruhe, Germany

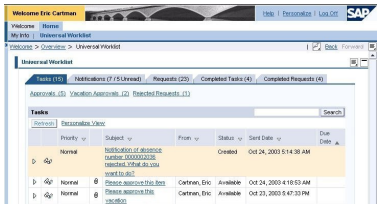
15th International Workshop on
Formal Methods for Industrial Critical Systems
(FMICS 2010)
Antwerp, Belgium, September 20-21, 2010

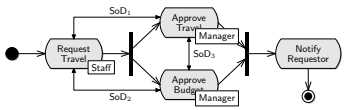
Agenda

Modern Business Software

A Target for Formal Methods?

- Modern business software is
  - complex
    (large code base, various programming languages)
  - parallel
    (distributed, service-oriented implementation, multi-tenancy)

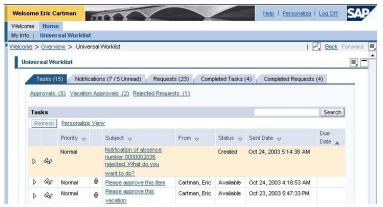


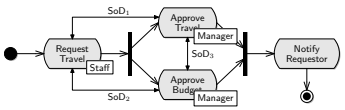


Modern Business Software

A Target for Formal Methods?

- Modern business software is
  - complex
    (large code base, various programming languages)
  - parallel
    (distributed, service-oriented implementation, multi-tenancy)
- Modern business software has to be
  - correct,
    (correct and compliant processes)
  - reliable, and
    (no crashes, guaranteed availability)
  - fast
    (response times, virtualization)

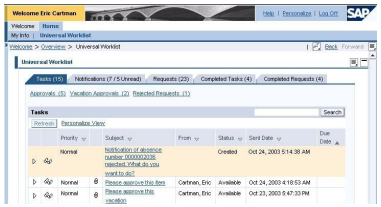


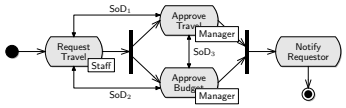


Modern Business Software

A Target for Formal Methods?

- Modern business software is
  - complex
    (large code base, various programming languages)
  - parallel
    (distributed, service-oriented implementation, multi-tenancy)
- Modern business software has to be
  - correct,
    (correct and compliant processes)
  - reliable, and
    (no crashes, guaranteed availability)
  - fast
    (response times, virtualization)
- Can formal methods/specifications help?





Agenda

The Problem:

Consistent Distributed Object Management

> Ensure the *correctness* of our rule-engine.

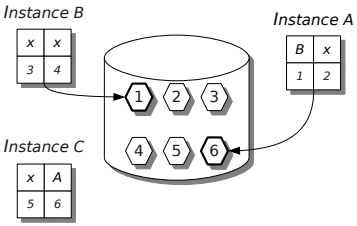
- Implementation of an event-condition-action rule engine:
  - events are represented as object state changes,
  - conditions are formulated as expressions on object, attributes
  - actions may change the object state.
- The implementation is multi-threaded and multi-clustered:
  - multiple application instances are running on different server nodes
  - application instances may start or stop during the life-time of the cluster
- Thus, we need to
  - coordinate object access across different application instances
  - deal with variations in the cluster topology, especially in the case of unexpected changes due to application or cluster node failures.

The Solution:

Object Ownership and Cluster Failover Management

- Guarantee exclusive object ownership
- Each application instance
  - manages the ownership of some objects
  - authoritative indexer for these objects
- Acquiring ownership of an object:
  - contacts the indexer of that object
  - only two messages required
  - indexer can be computed locally
- Topology changes requires updates
- Ownership via restructuring protocol
  1. all instances agree on the same view
  2. each instance knows the ownership
- A dedicated *master* instance provides the current view to joining instances.

- Instance *A*
  - indexer for object 1 and 2
  - owns object 6
- Instance *B*
  - indexer for object 3 and 4
  - owns object 1
- Instance *C*
  - indexer for object 5 and 6



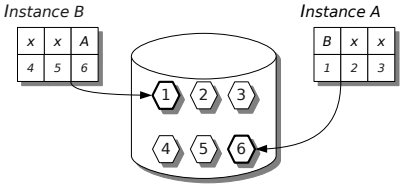
The Solution:

Object Ownership and Cluster Failover Management

- Guarantee exclusive object ownership
- Each application instance
  - manages the ownership of some objects
  - authoritative indexer for these objects
- Acquiring ownership of an object:
  - contacts the indexer of that object
  - only two messages required
  - indexer can be computed locally
- Topology changes requires updates
- Ownership via restructuring protocol
  1. all instances agree on the same view
  2. each instance knows the ownership
- A dedicated *master* instance provides the current view to joining instances.

After instance *C* has left

- Instance *A* and *B* agree
  - on cluster size 2
  - *A* indexer for $1, 2, 3$
  - *B* indexer for $4, 5, 6$
- *B* is informed that *A* owns 6



Implementation Constraints

Industrial software development does not happen in isolation

- Avoid additional functionality by reusing existing frameworks
  - no specialized runtime
  - named communication channels using JNDI
- Minimize central knowledge while avoiding redundancy
  - existing infrastructure does not provide redundancy/replication
  - only local meta information (i. e.„ object ownership) per instance
  - needs to be synchronized whenever the cluster topology changes
- Global synchronization via locks
  - one global lock for all instances
  - master election: ability to acquire global master lock
- Synchronous mode of operation
  - synchronous communication via RMI
- Continuous operation during restructuring
  - restructuring should not block regular operation

Writing Formal Specifications

- Formalization of an existing implementation
  - reverse-engineer the implementation into an executable specification
  - focus on robustness of the protocol against communication failures
- We followed a two-staged approach:
  - A high-level abstract specification on paper:
    - initial starting point
    - primary communication and discussion medium with the development team
    - clarification of the overall system architecture
  - An executable specification
    - as soon as possible, we refined the abstract specification
    - using *asynchronous multi-agent ASM* in CoreASM
  - Both specifications updated in parallel
  - At the end: ca. 3200 lines of formal specification

Cluster Protocol Invariants

Stable Cluster States

```
derived IndicesInSync =
  forall node in RunningNodes() holds IndexInSync(node)

derived IndicesAreValid =
  forall node in RunningNodes() holds IndexIsValid(node)

derived IndexInSync(node) =
  forall oid in [1..OID_MAX] holds SlotInSync(node, oid)

derived SlotInSync(node, oid) =
  node = authIndexer(oid, node) implies
    OWNER(OWNER(node, oid), oid) = OWNER(node, oid)

derived IndexIsValid(node) =
  forall oid in [1..OID_MAX] holds SlotIsValid(node, oid)

derived SlotIsValid(node, oid) =
  node = authIndexer(oid, node) implies
    OWNER(node, oid) memberof RunningNodes()
```


Simulation Results

- The original implementation was based on a wrong assumption:
  - notifications in the case of failure are always sent immediately
  - this should be ensured by the runtime environment
- Assume a delayed notification while a new node is starting up:
  - the new node becomes master of new cluster
  - if delayed notification is received, the old nodes will
    - try to become master and fail
    - do nothing and wait for restructuring
  - resulting in an inconsistent state (`IndicesInSync` does not hold)
- Initialize cluster with two active nodes and an inactive one:

```
nodeList = ["N1", "N2"]
newMasterNode = "N3"
```


Simulation Results

- The original implementation was based on a wrong assumption:
  - notifications in the case of failure are always sent immediately
  - this should be ensured by the runtime environment
- Assume a delayed notification while a new node is starting up:
  - the new node becomes master of new cluster
  - if delayed notification is received, the old nodes will
    - try to become master and fail
    - do nothing and wait for restructuring
  - resulting in an inconsistent state (`IndicesInSync` does not hold)
- Deactivate node failure detection

```
SuspendNodeFailureHandlers()
clusterIsStable := true
```


Simulation Results

- The original implementation was based on a wrong assumption:
  - notifications in the case of failure are always sent immediately
  - this should be ensured by the runtime environment
- Assume a delayed notification while a new node is starting up:
  - the new node becomes master of new cluster
  - if delayed notification is received, the old nodes will
    - try to become master and fail
    - do nothing and wait for restructuring
  - resulting in an inconsistent state (`IndicesInSync` does not hold)
- Shutdown the current master node

```
killedMaster := MasterNode()
remove NodeID(MasterNode()) from nodeList
SignalNodeShutdown(MasterNode(), true)
clusterIsStable := false
```


Simulation Results

- The original implementation was based on a wrong assumption:
  - notifications in the case of failure are always sent immediately
  - this should be ensured by the runtime environment
- Assume a delayed notification while a new node is starting up:
  - the new node becomes master of new cluster
  - if delayed notification is received, the old nodes will
    - try to become master and fail
    - do nothing and wait for restructuring
  - resulting in an inconsistent state (`IndicesInSync` does not hold)
- Start inactive node

```
if (NodeIsDown(killedMaster)) then {
  AddNode()
  add newMasterID to nodeList
}
```


Simulation Results

- The original implementation was based on a wrong assumption:
  - notifications in the case of failure are always sent immediately
  - this should be ensured by the runtime environment
- Assume a delayed notification while a new node is starting up:
  - the new node becomes master of new cluster
  - if delayed notification is received, the old nodes will
    - try to become master and fail
    - do nothing and wait for restructuring
  - resulting in an inconsistent state (`IndicesInSync` does not hold)
- Re-activate handling of node failures

```
if (MasterNode() != undef
    and HasJoinedCluster(newMasterID)) then {
  ResumeElemLossHandlers()
}
```


Agenda

Lessons Learned

- Missing scope for locations
- Missing tool support for “Refinements”
- Reusables specification modules
- Insufficient support for literate specifications
- Debugging support
- Combining formal and semi-formal development processes
- Lack of commercially applicable tools
- Think twice before you decide for a specific (set of) method(s)
- There are places for interactive and for automated Methods

Conclusion

- Well-know facts:
  - already writing a formal specification can reveal problems
  - Fully automated tools can be used by non-experts in formal methods
- We used the “consultancy model” also in other formal method projects
  - requiring an formal methods exports for the (interactive) analysis is fine
  - non-experts in formal methods should be able to
    - write specifications
    - type-check and animate (execute) specifications
    - write (simple) properties that should be verified
- Business applications can be worthwhile targets for formal methods
- Formal methods are applied, if there is there is a business case

Thank you!

Bibliography I

Michael Altenhofen and Achim D. Brucker.
Practical issues with formal specifications: Lessons learned from an industrial case study.
In Stefan Kowalewski and Marco Roveri, editors, *International Workshop on Formal Methods for Industrial Critical Systems (FMICS)*, number 6371 in Lecture Notes in Computer Science, pages 17–32. Springer-Verlag, 2010.

Specification

Table: An overview of the modules of the ASM specification

Module	Lines	Rules	Functions
Control ASM States	50	0	1
Cluster Master	161	12	10
Protocol Messages	138	0	25
Cluster Membership and Object Management	1 796	114	159
Object Requests	128	10	3
Cluster Environment (Notification)	328	19	31
Lock Management	141	7	19
Message Passing	362	12	56
Control Flow	88	10	5
Control State Handling	63	5	9
Total	3 255	189	318

© Copyright 2010 SAP AG All Rights Reserved