Scenario-Driven System Engineering (SDSE) for System of Systems Acquisition

Ray Paul
ASD NII C2 POLICY
Department of Defense
Raymond.Paul@osd.mil

Introduction

- **Knowledge doubles every…**

  = 5 years (Gore, Clinton, Bush)

  = 7 years (Harvard Business Review, 1996)

  $\Rightarrow$ Knowledge grows exponentially

- **Technology growth depends on**
  - ♦ Knowledge Growth
  - ♦ Knowledge to technology transfer rate

Tech Growth $\propto$ F (km(t) $\Downarrow$ Exponential tt(t)) $\Downarrow$ Exponential (Moore, Kozmetsky)

Technology Growth = Exponential (Moore's Law)

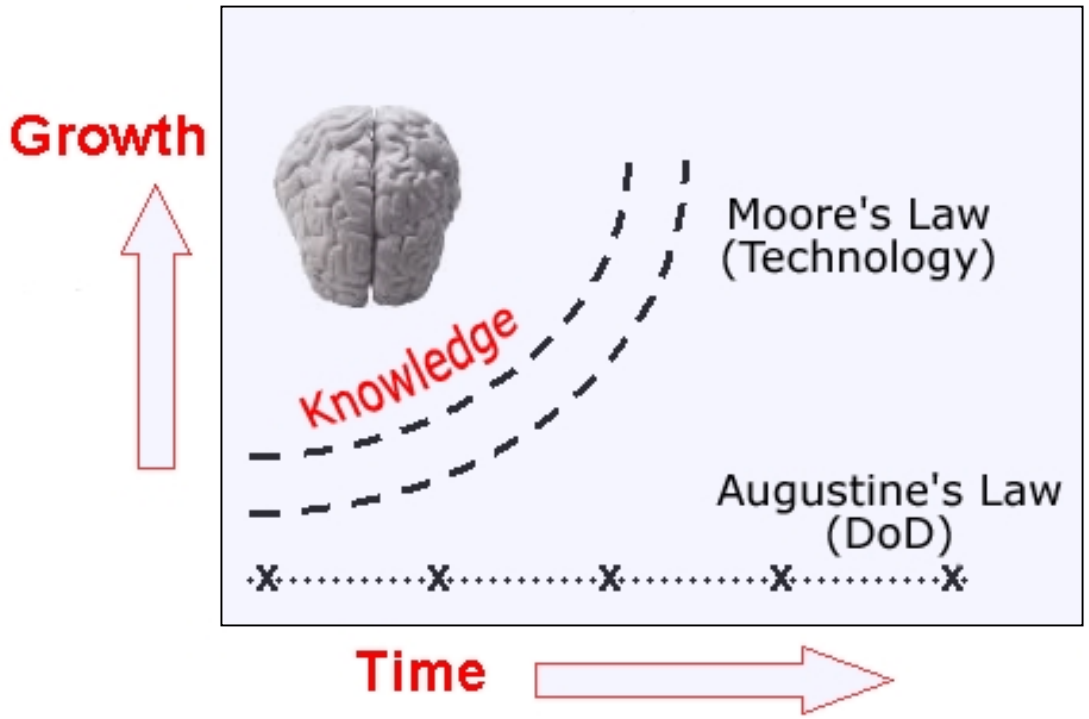
- **Technology growth follows knowledge growth.**

- **Corrigan-Kozmetsky shows that:**
  - ♦ Technology growth follows closely behind knowledge growth
  - ♦ Technology growth curve converges to knowledge growth in mature technologies (Technologies that are moving their physical theoretical limits)

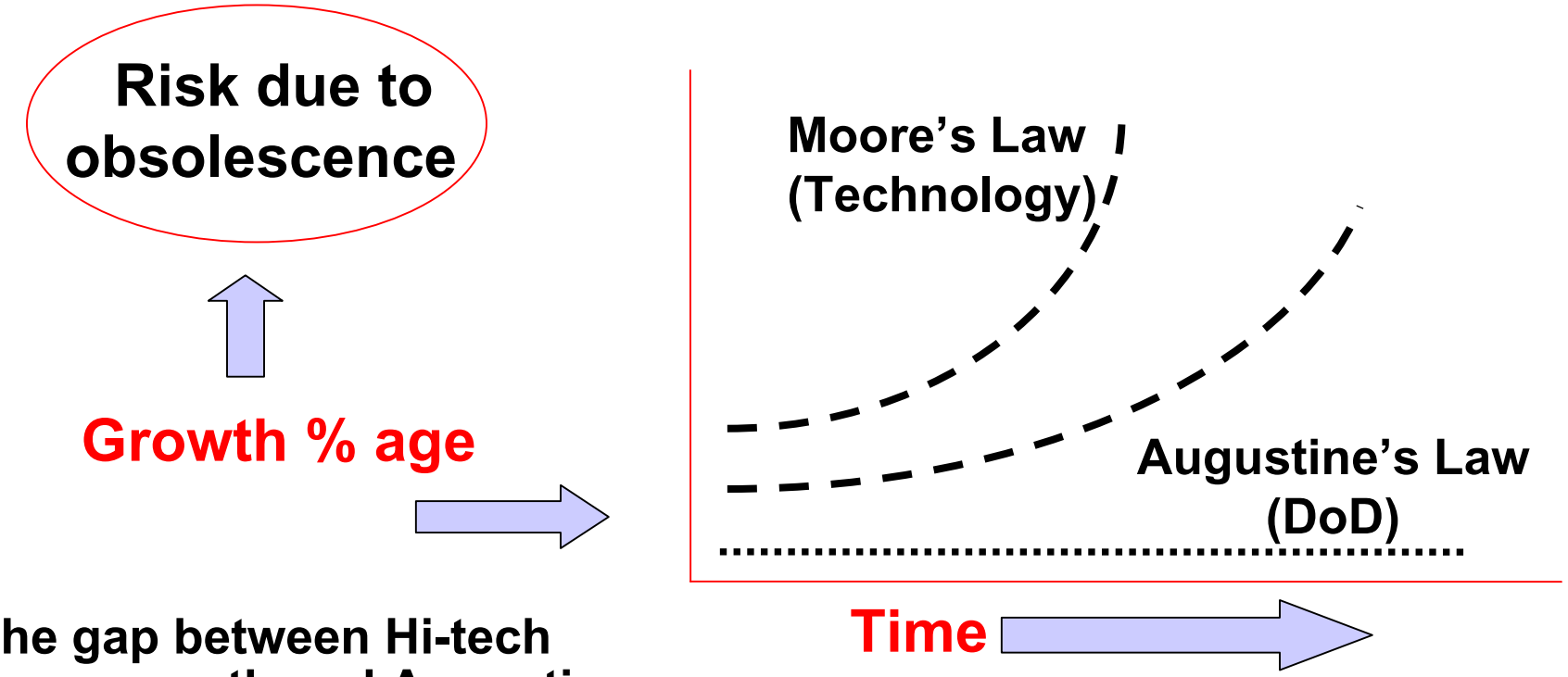
- **The above observation is substantiated by Moore's Law & Andrew Grove's hypothesis of technology-growth "inflection points".**

- **DoD technology developments are constrained by its budget.**

- **DoD technology growth is dependent on its budget growth (Augustine's Law).**
- **Technology Growth** = 67% / year (Moore's Law)
- **Augustine DoD Growth** = 5-7% / year
- **The difference between Hi-tech & DoD:** growth rates = 60%



- This difference represents growth of obsolescence or risk. It is an exponential growth.



- The gap between Hi-tech Moore growth and Augustine DoD growth is an exponentially growing function - We shall call it "The Widening Chasm Effect".

- **We maintain technology and threat are duals.**
  If one changes, the other follows.

- **It is a take-off from the theory of evolution -**
  The "Container-Content" Duality" -
  As the container grows (in capacity), the contents increase to fill it.

- **In general, the container represents the infrastructure and the contents, the systems functions.**

MESSAGE: As technology grows so does the threat, as well as our needs for defense.

- **Technology growth rates between hi-tech (rapidly growing technologies) and lo-tech (stable, static, slow growing) elements produces an integration mismatch, i.e., systems using a combination of hi-tech and lo-tech elements show different parts aging; or obsolescing at different desperate rates.**

This mismatch creates a difficult integration, interoperability, training, maintenance and upgrading problems. Solution may lie in the future system design principles.

Term of Reference

- System of systems (SOS) is defined as a federation of systems, whose context, mission and operational logic are required for handling the allocation and coordination of shared integrated resources.

Thesis

- Today's IT systems development and integration efforts among heterogeneous subsystems over long periods of time are ad-hoc as they are designed. Large scale distributed SoS must be developed from clear, complete and consistent requirements.
- It is essential for rapid development and evaluation of new generations of distributed SOS (embedded and pervasive) be guided by an enterprise level perspective and decision analysis, and understanding. The enterprise perspective provides a coherent and pragmatic framework for designing SOS, acquiring technologies, and dynamic and adaptable deployment.

Modern Combat Systems: The Need for *Rapid* and *Agile* Engineering Development

- We must reduce the cycle time significantly to fully utilize Moore's law.
  - The current development process is too rigid and the cycle time is too long.
  - Proposed an **Income-Tax Acquisition Model** last year to make the DoD acquisition much flexible and adaptable, and empower the system owner to make real-time decisions during IT acquisition.
  - Proposal was bold and if implemented can give DoD IT acquisition the capability to take advantage of Moore's law.
  - However, the model focuses on the acquisition management, it does not cover IT system engineering development. DoD IT acquisition needs **both** agile acquisition management (Income-Tax model) and agile system engineering development.
  - This talk focuses on *agile system engineering*.

System Engineering Tasks

- Requirements definition capturing, analysis, specification, verification, validation, simulation, and documentation.
- System design including architecture, design patterns, concurrency analysis (deadlock and race condition detection, synchronization)
- System analysis include:
  - **Reliability** analysis, estimation, modeling, operational profiling, fault-tolerant computing, dynamic reconfiguration
  - **Safety** analysis including fault tree analysis, event tree analysis, FMEA (failure mode and effect analysis), FMECA (failure mode, effect, and criticality analysis)
  - **Security** analysis including vulnerability analysis, encryption, access control (Bell LaPadula model, Chinese Wall model, Role-based model), firewall, intrusion detection.
  - **Performance** analysis including throughput-delay analysis, simulation, critical path analysis
  - **Timing** analysis including scheduling, runtime verification.
  - **Behavior** analysis including model checking, temporal logic analysis, concurrency control, state model and state analysis.
  - **Verification** and **Validation** including test script/case generation, coverage, completeness and consistency analysis.

DoD Combat Systems

- Many of these systems are real-time mission-critical systems.
  - The consequence of failures is too great.
- Most DoD systems are Systems of Systems (SoS).
  - Interfaces, Integration, Interoperability, and Integration are important.
- Many of these systems are legacy systems.
- How can we develop news systems *rapidly* and *adaptively* in this kind of environment?
  - The goal is to engineer IT cycle time from years to *6 to 9 months*.

Rapid and Adaptive System Engineering

- To reduce IT cycle time from years to 6 to 9 months requires us to change the entire mindset for IT development.
  - We must have a process that can capture requirements rapidly, and verify and validate requirements thoroughly and rapidly.
  - Many of the steps must be automated.
  - The entire development must be based on a fully *integrated* model for system development, not just a hodgepodge or quilt of techniques.

Other Related Work

- Extreme Programming
  - This new process of software development is getting popular and received significant attention
  - It emphasizes the allotment and even encouragement of frequent changes
  - It also emphasizes team work and collaboration, and light on methodology
  - It promotes a test-based development process where testing is an integrated part of software development
  - This process is suitable for commercial software and system development
  - The major problem is that it does **not** focus on system engineering issues
  - Many of the steps in Extreme Programming are performed manually, and thus eventually the speed of development will be limited

Other Related Work

- OMG's MDA is an aggressive program based on UML and related technologies such as XML
  - It emphasizes executable UML
  - Engineers specify their systems using executable UML, and follows two steps:
    - Platform Independent Model (PIM)
    - Platform Specific Model (PSM)
  - Once models are developed, code can be generated. The code generation is partially automated. For example, to run the executable UML, engineers must supply detailed code for the model specified with the code skeleton automatically generated from the UML model.
  - The major issues are:
    - UML essentially is a design model (primarily based on the class hierarchy with additional models such as sequence diagrams and state model), and thus it will take more effort to develop the specification from requirements.
    - MDA addressed the system engineering issues partially and indirectly, many issues such as reliability and security are not addressed or supported as it is.

Scenario-Driven System Engineering (SDSE)

- Rapid and adaptive scenario-driven system engineering instead of traditional documentation driven system engineering. The SDSE include at least:
  - Scenario-driven requirement engineering
  - Scenario-driven design and code generation
  - Scenario-driven verification and validation

Requirements Engineering Process should be Changed

- Instead of traditional complex requirements engineering, which takes significant time, we submit a ***scenario-driven requirement engineering*** method, where system scenarios are specified and evaluated in real time with instant feedback to the developer and user.
  - Scenario language is easier to understand and easy to develop
  - Once system scenarios are available, various static and dynamic analyses should be immediately performed to give the developer and user *instant feedback*.
    - Analyses such as completeness and consistency analysis, sequence analysis, timing analysis, performance, concurrency analysis (such as deadlock analysis), dependency analysis, state analysis, and pattern analysis
  - Scenarios do ***not*** need to be complete or consistent for these analyses to be carried out.

Scenario-Driven Requirements Engineering

- For example, the system can be simulated without any coding or programming, even if the scenarios are inconsistent or incomplete.
  - In fact, the simulation will identify incompleteness and inconsistencies.
  - The simulation will perform behavior validation as well as performance evaluation.
  - The simulation can also generate the system state model for further analysis such as reliability and critical path analysis.
- The key idea is that if system scenarios are changed, the system can be re-simulated and because it is not necessary to develop the simulation code, system can be repeatedly simulated whenever there is a change in system requirements.
- Is such technology feasible?

Scenario Specification Language

- Scenarios can specified using an integrated scenario (ACDATE) model:
  - **A**ctor – the machine or process that performs the action;
  - **C**ondition – the condition that triggers events;
  - **D**ata – the data used in computation or conditions;
  - **A**ction – the computation;
  - **T**iming – this is related to deadlines or delays;
  - **E**vent – an incident or happening.

- This model has been used to model a variety of applications including DoD Command and Control applications and real-time embedded systems.

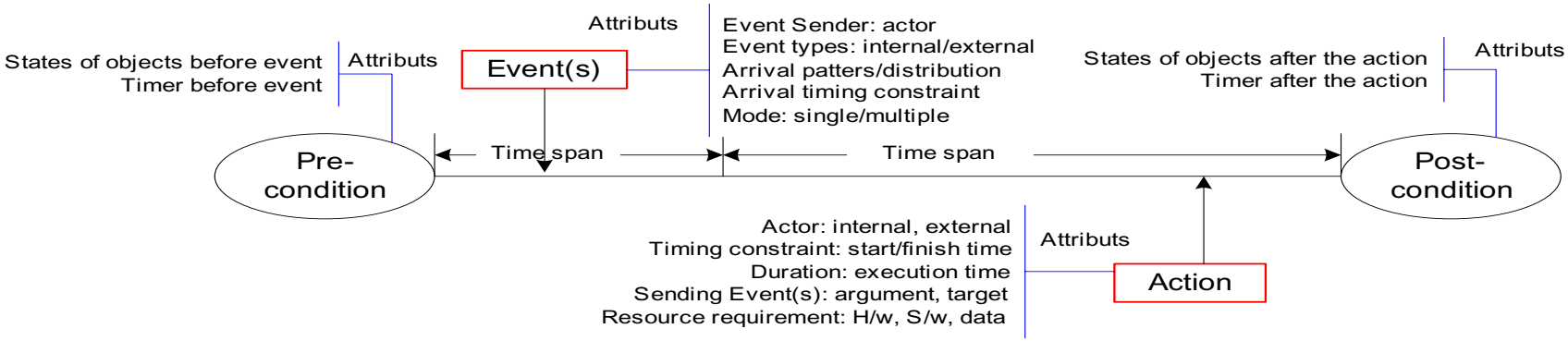
Example Command and Control (C2) Scenario

- A Strike Group (*SG, a actor, consisting of a total of 10 warships*) is rapidly deployed from a forward base in the Gulf in response to a threat. While in transit, the SG conducts Rapid Response Planning (RRP, *an action*), developing Initial Plan of the Battle (*IPB, data*) and Time Critical Strike (*TCS, data*) packages. The SG's shore-to-ship data transmission experiences minimal impact as the Carrier Battle Group (*CBG, a complex actor*) transfers MILCOM connectivity between various MILCOM links (*CAK is a satellite and is an actor*). The SG completes the downloading of weather information files (*data*) over an expanded 10 MB CAK MILCOM circuit via Joint Processing System (JPS, an actor), …….
- This sample scenario highlights only *actors* and *actions* only.

System Analyses Based on Scenarios

- Once system scenarios are available, these analyses can be carried out rapidly:
- Static analyses
  - Dependency analysis
  - Completeness and consistency
  - Control flow analysis
  - Sequence diagram generation
  - Reliability modeling and estimation
  - Pattern analysis
- Dynamic Analyses
  - Simulation
  - State model generation
  - Event tree analysis
  - Failure tree analysis
  - Security analysis
  - Runtime verification
  - Timing analysis

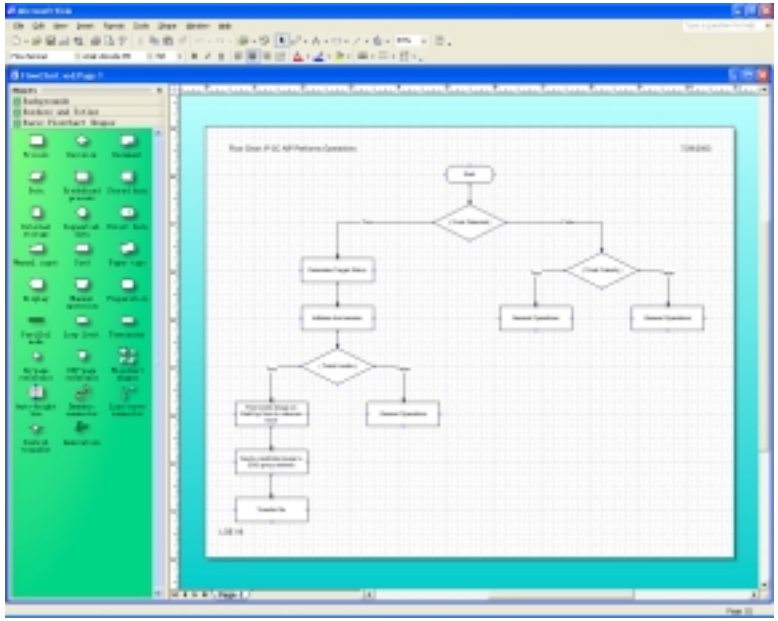
ACDATE Model and Simulation



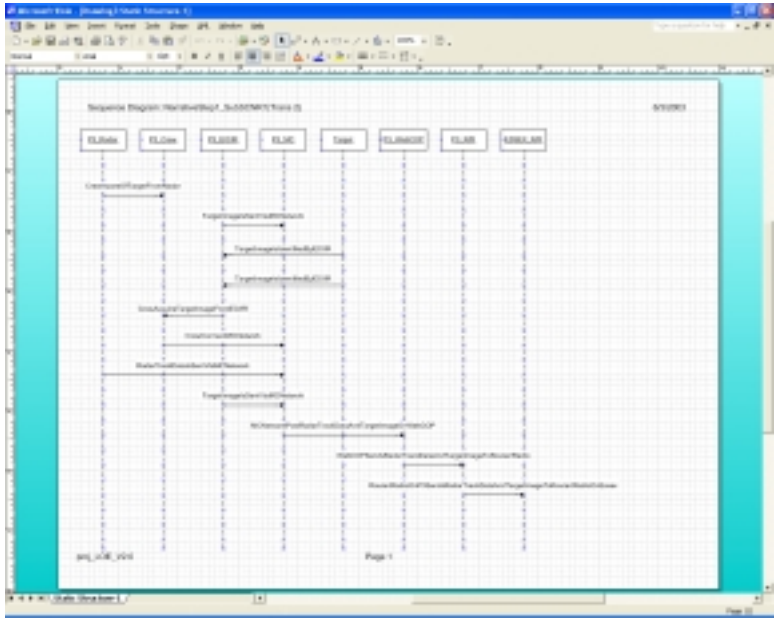
- The semantics of ACDATE can be represented as a state transition (above figure): if an *actor* is in the *pre-condition*, and it receives a triggering *event* that satisfies the guard condition, it will perform an *action* and change its state to the *post-condition*. The *action* performed may generate *events* that can be sent to other *actors*.
- Because the scenario model has a built-in semantic model, once the system scenarios are specified, the system can be simulated right away without any programming.
- If the system requirement is changed, once the scenarios are updated, the system can be simulated again without additional effort. Thus it is suitable for rapid and adaptive system engineering.
- The OMG's MDA program is different. The MDA's model cannot be executable until the detailed code is supplied. And this will take significant time and effort.

Automated Flow Chart and Sequence Diagram Generation from Scenarios

- Flowchart Diagram:

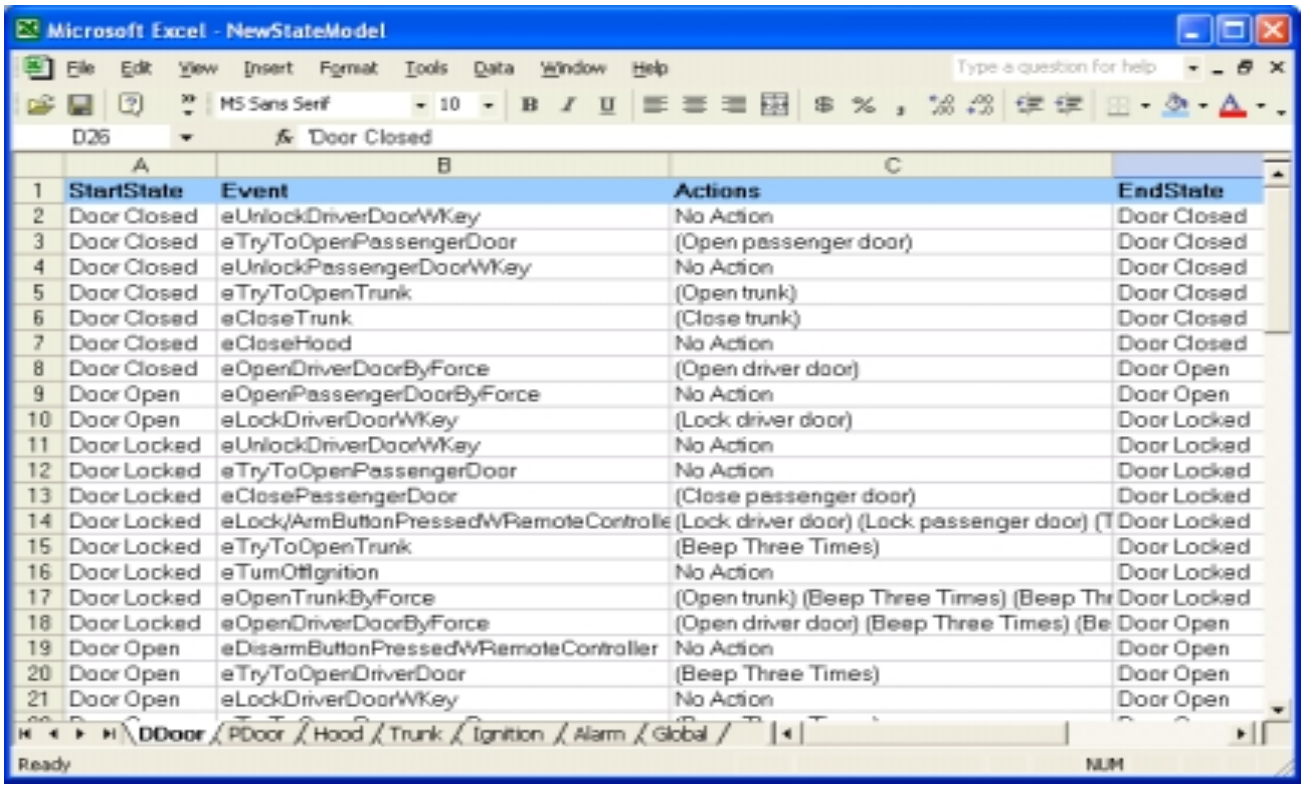


- Sequence Diagram:



Automated State Model Generation from Scenario Simulation

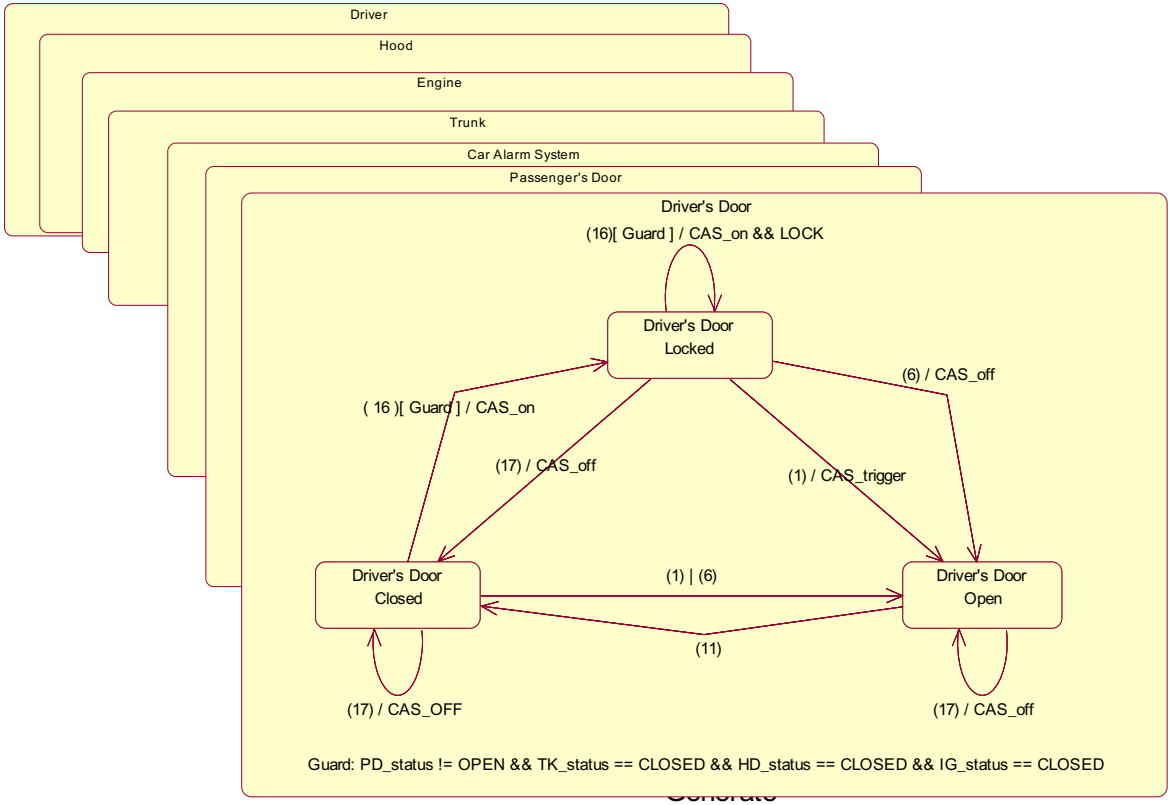
- State diagram in Excel:



	A	B	C	
1	StartState	Event	Actions	EndState
2	Door Closed	eUnlockDriverDoorWKey	No Action	Door Closed
3	Door Closed	eTryToOpenPassengerDoor	(Open passenger door)	Door Closed
4	Door Closed	eUnlockPassengerDoorWKey	No Action	Door Closed
5	Door Closed	eTryToOpenTrunk	(Open trunk)	Door Closed
6	Door Closed	eCloseTrunk	(Close trunk)	Door Closed
7	Door Closed	eCloseHood	No Action	Door Closed
8	Door Closed	eOpenDriverDoorByForce	(Open driver door)	Door Open
9	Door Open	eOpenPassengerDoorByForce	No Action	Door Open
10	Door Open	eLockDriverDoorWKey	(Lock driver door)	Door Locked
11	Door Locked	eUnlockDriverDoorWKey	No Action	Door Locked
12	Door Locked	eTryToOpenPassengerDoor	No Action	Door Locked
13	Door Locked	eClosePassengerDoor	(Close passenger door)	Door Locked
14	Door Locked	eLock/ArmButtonPressedWRemoteControlle	(Lock driver door) (Lock passenger door) (T	Door Locked
15	Door Locked	eTryToOpenTrunk	(Beep Three Times)	Door Locked
16	Door Locked	eTurnOffIgnition	No Action	Door Locked
17	Door Locked	eOpenTrunkByForce	(Open trunk) (Beep Three Times) (Beep Thr	Door Locked
18	Door Locked	eOpenDriverDoorByForce	(Open driver door) (Beep Three Times) (Be	Door Open
19	Door Open	eDisarmButtonPressedWRemoteController	No Action	Door Open
20	Door Open	eTryToOpenDriverDoor	(Beep Three Times)	Door Open
21	Door Open	eLockDriverDoorWKey	No Action	Door Open

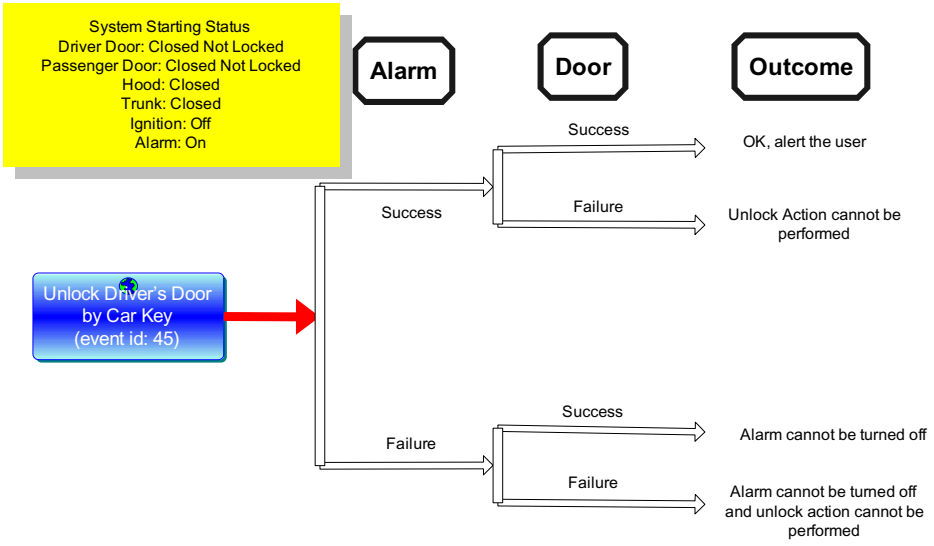
Generating UML State Charts

- Create UML state chart from scenario simulation:



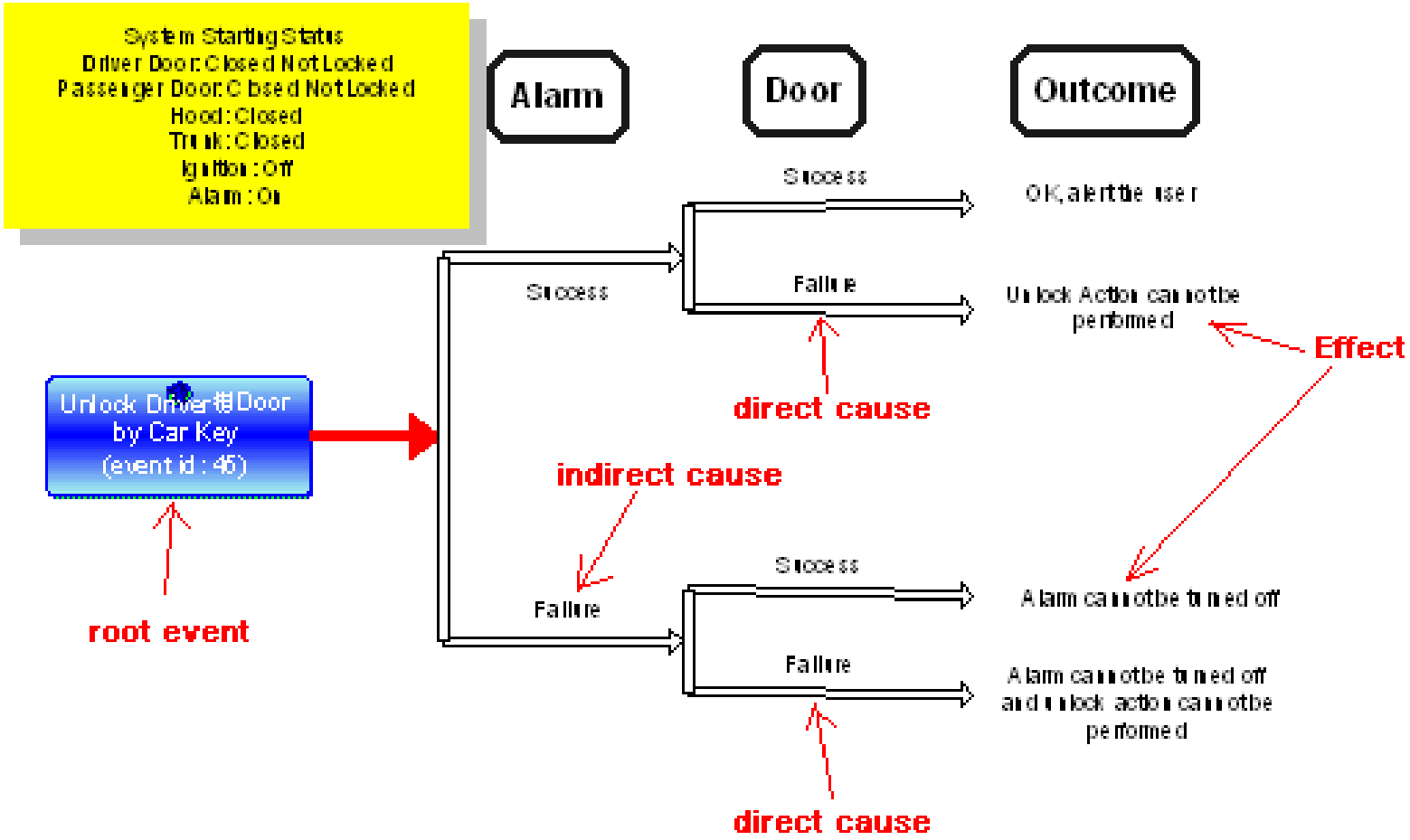
Automated Event Tree Generation

	A	B	C	D	E	F	G	H
1	StartState	Event	Actor_0_Alarm	Actor_1_DriverDoor				
2	(DD:Door Closed Not Locked) (PD:Door Cl	45	Turn off alarm : Success	Unlock driver door : Success				
3	(DD:Door Closed Not Locked) (PD:Door Cl	45	Turn off alarm : Success	Unlock driver door : [Failure Mode: No Response] [Cannot Unlock Door]				
4	(DD:Door Closed Not Locked) (PD:Door Cl	45	Turn off alarm : [Failure Mode: No Respc	Unlock driver door : Success				
5	(DD:Door Closed Not Locked) (PD:Door Cl	45	Turn off alarm : [Failure Mode: No Respc	Unlock driver door : [Failure Mode: No Response] [Cannot Unlock Door]				
6								

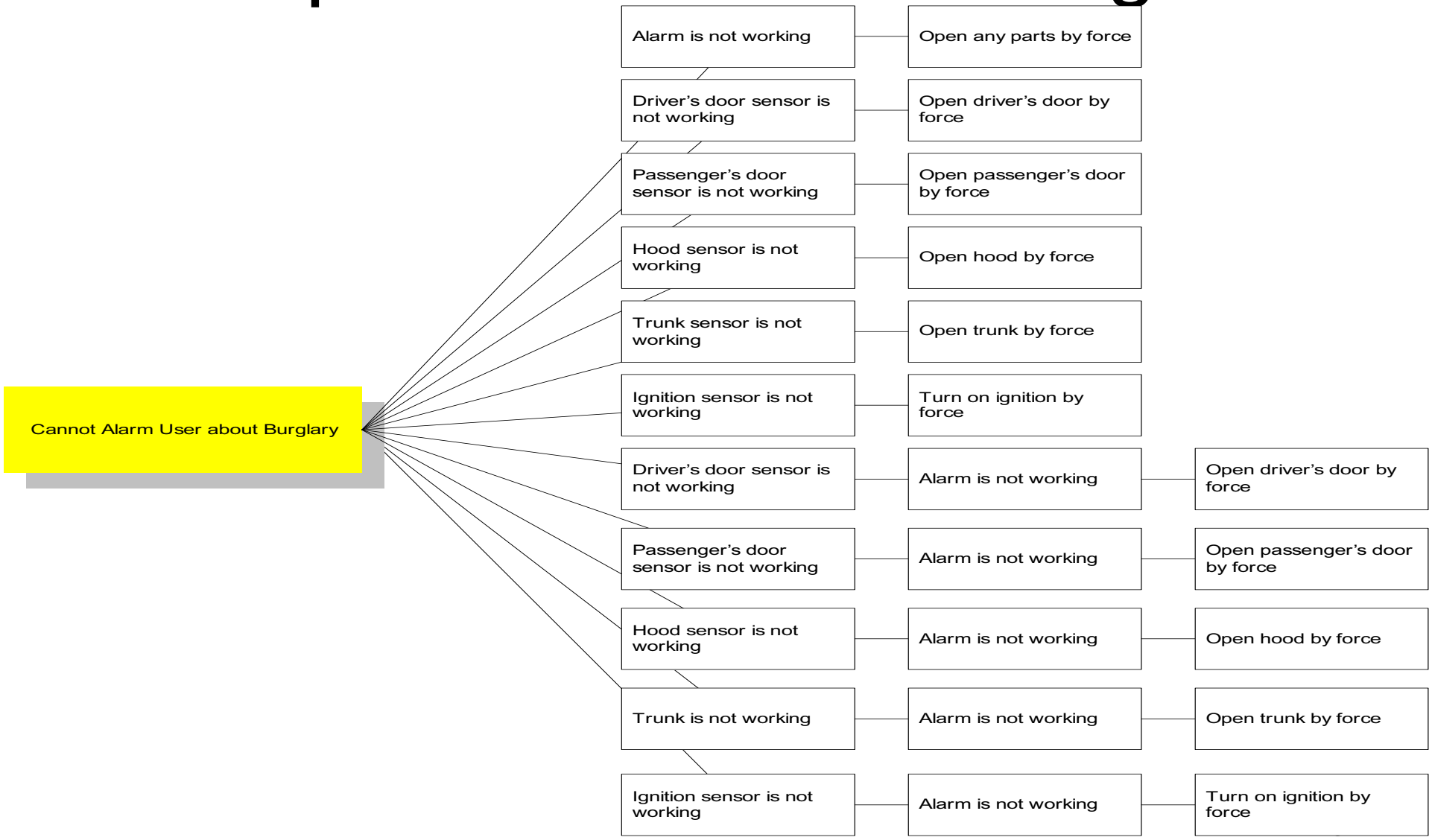


* Event 45 : Unlock Driver's Door with Car Key

Example of Causality Analysis



Sample Effect-Cause Diagram



Reliability Analysis and Estimation

- It is also possible to estimate the system reliability for the system specified using ACDATE/scenario model.
- System performance can be evaluated by simulation, and system reliability can also be estimated by establishing an operational profile to guide the simulation program.
- Different failure distribution functions can be incorporated into the scenario tool.

A rapid, agile and adaptive process is feasible

- Several commercial companies are now developing this set of technologies, and the goal is to reduce the cycle time to few months and in some cases to a few *days* for real-time mission-critical system development.
- The process looks like:
  - Rapid requirement capture and specification using a user-friendly GUI;
  - Rapid system scenario evaluation including various static and dynamic analyses including system simulation;
  - If the results are positive, the system will generate code automatically possibly using reusable components and design patterns;
  - Test scripts are automatically generated from system scenarios to test the generated code, and the testing will be performed in an integrated environment with both legacy code and new code. The testing may be performed on the simulated environment first before trying on the actual environment;
  - If the test results are positive, it will be deployed immediately as the new code already has been tested with the legacy code. The code will be distributed to the remote sites via a network.
  - If the system requirement is changed, the entire process will be repeated. The goal is to have the entire process completed within a few days.

Security Analysis

- By attaching security classification information to the scenario model elements such as actors and data, it is possible to perform security analysis at the system specification level before implementation.
  - For example, Bell and LaPadula model, Chinese Wall model, Role-based Security model can be used to verify the system specified using the ACDATE/scenario model.
- Because the analysis is automated, if the system is changed, the security analysis can be repeated as needed, even at runtime during system reconfiguration. This supports rapid and adaptive system engineering.

Scenario-Driven Design and Code Generation

- Instead of traditional documentation driven manual design, we can have:
  - Automated initial design (such as class/method/algorithm/pattern) generation from system scenarios
  - Automated system design refinement using a catalogue of reusable design components and patterns based on system scenarios
  - Automated design constraint verification
  - Automated code generation from system scenarios

Scenario-Driven Verification and Validation

- Instead of the traditional verification and validation process, which is often complex, expensive and time consuming.
  - Various V&V can be carried out immediately when system scenarios are specified:
    - Simulation (without simulation programming) of system scenarios
    - Test script/case generations based on scenario patterns
    - Automated completeness analysis based on combinatorial analysis on conditions and events
    - Automated model checking for various properties such as deadlock-free, live ness, constraint checker.

Rapid and Adaptive Test Script Generation

- Even though a typical system may have hundreds of thousand scenarios, often only few patterns are needed to cover most of the scenarios. For example,
  - *eight* scenarios patterns are sufficient to cover 95% of system scenarios for an industrial safety-critical implantable medical device
  - *Four* scenario patterns cover 100% of anti-theft car alarm system
- The idea is that a test script template can be used and reused to test all the scenarios belonging to the same pattern. Because we have few scenario patterns only, test scripts can be rapidly developed.
- If a system scenario is changed, once the scenario is updated, its corresponding test script can be rapidly generated by reusing test script template.

Scenario Patterns for a large Industrial Safety-Critical System

Pattern	Coverage (%)
Basic requirement pattern	40
Key-event driven requirement pattern	15
Timed key-event driven requirement pattern	5
Key-event driven time-sliced requirement pattern	7
Command-response requirement pattern	8
Lookback requirement pattern	6
Mode-switch requirement pattern	8
Interleaving requirement pattern	6
Total	95

Rapid, Agile and Adaptive Process; Key Ideas

- Systems must be developed with designers and end user's Joint point of view from the beginning;
- Most of steps in system engineering development are automated;
- Give system designers, developers, and user's instant feedback so that changes can be immediately evaluated and incorporated.

Cycle Time Reduction

- Most of system development time is involved in re-work (to handle changes), verification, t&e and validation.
- The SDSE handles re-work by built-in analysis tools that allow analyses to be repeated as often as needed, and often in real time and at runtime including simulation, safety analysis, security analysis, reliability analysis, and completeness and consistency analysis.
- The SDSE handles V&V and t&e by using a pattern-oriented approach.

Risks

- Do we have all the technologies or tools for 6-9 month cycle time?
- No, even though we have a good start we need to run faster.
- But the risk of not working towards this goal is much greater!
  - This means that DoD will not be able to capture Moore’s law while under the constraint of Augustine's law.

Conclusion

- SDSE can be one of several key enablers for rapid and adaptive system development to reduce DoD IT cycle time from years to 6 to 9 months.
- Other key enablers include an agile and adaptive acquisition process model such as the Income-Tax model.
- It is possible to have SDSE without the Income-Tax model or vice versa, but we need both to achieve the leapfrog that we needed to capture the Moore's law under the Augustine's budget.

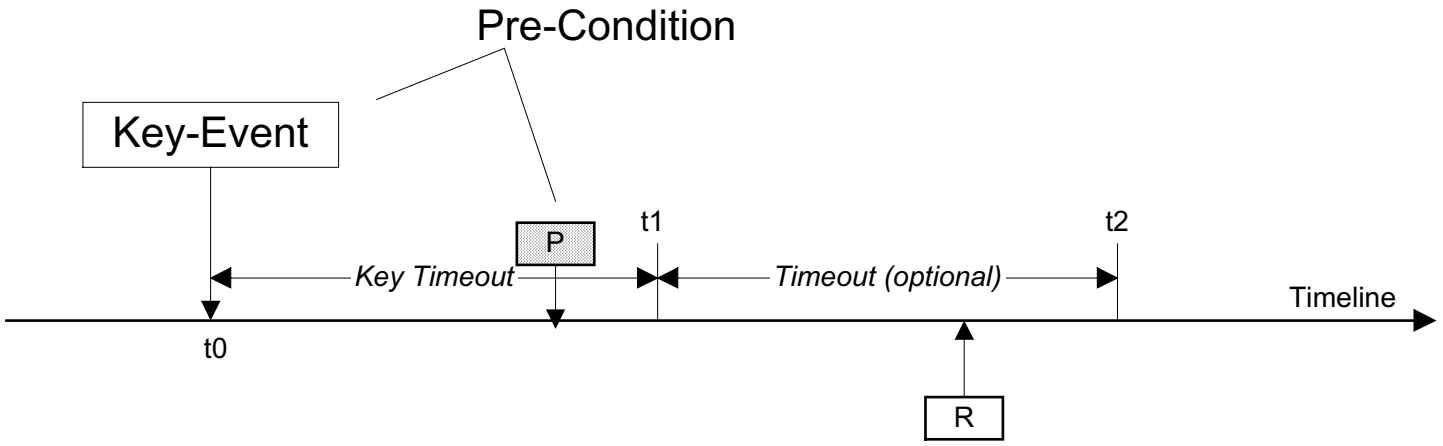
Appendix

Completeness and Consistency Checking of Scenarios

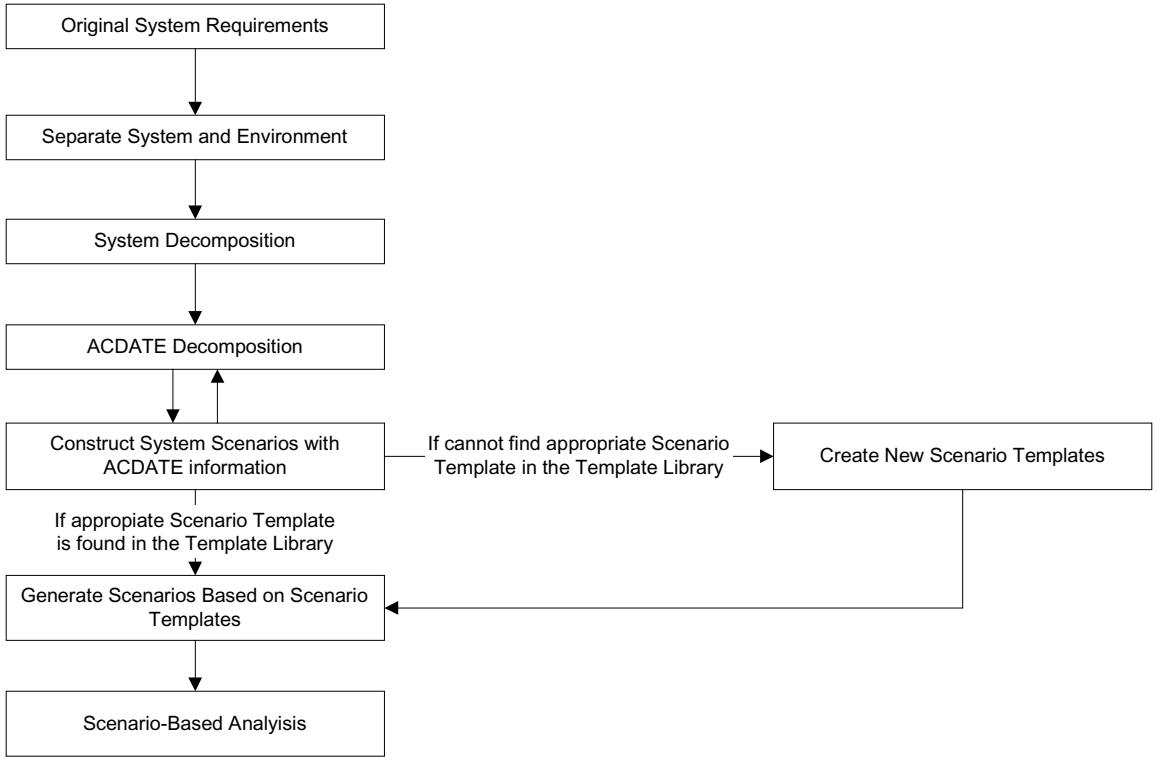
- Scenarios can be checked for completeness and consistency (C&C) with respect to the ACDATE information, the result of C&C can be used to add new scenarios to complete the description, and to generate test scripts using the verification pattern approach.
  - SDSE automatically enumerates various combinations of conditions to ensure complete coverage.
  - Even for a small problem, the completeness analysis will produce numerous cases for examination. This is the well-known state explosion problem in disguise.
  - The key is not to examine all the combination of cases, doing so will require too much effort.
  - The idea is to identify those “don’t care” cases, and delete them from consideration to avoid the state explosion problem, and also use a hierarchical classification algorithm to identify and partition all relevant cases.
  - This process is rather technical but can be automated for *rapid* development.

Scenario Pattern

- Timed Key-Event Driven Pattern
  - Within the duration t1 after the key event, if P then R is expected (before t2).



Scenario Specification Process



Sample Scenario Specification

(**ATOMIC**)
[CAS_RC_concurrently_disarm_unlock_handler_SCNR]
(**Trigger Event** --- eTurnOffAlarmAndUnlockByRC)
using ACTOR:Alarm
using ACTOR:DriverDoor
if (**true**)
then
{

- **do ACTION**: Alarm.TurnOffAlarm
- **do ACTION**: DriverDoor.UnlockDriverDoor

}

Illustration of Actor