A Mode-Based Pattern for Feature Requirements, and a Generic Feature Interface

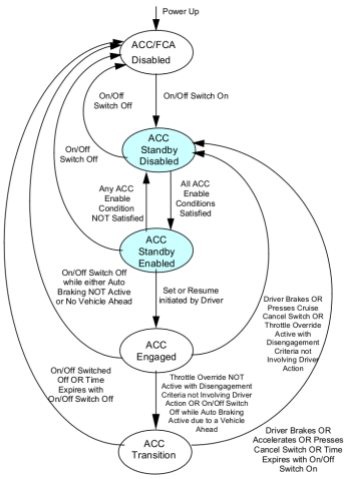
David Dietrich
Joanne M. Atlee
David R. Cheriton School of Computer Science

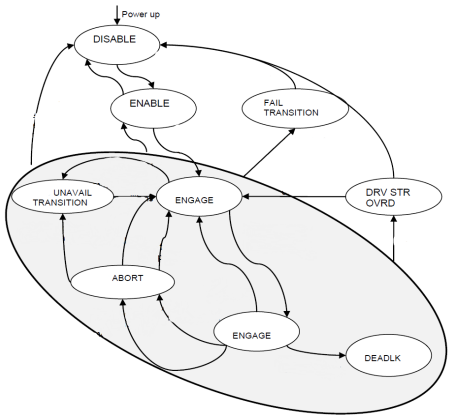
UNIVERSITY OF WATERLOO

necsis

1

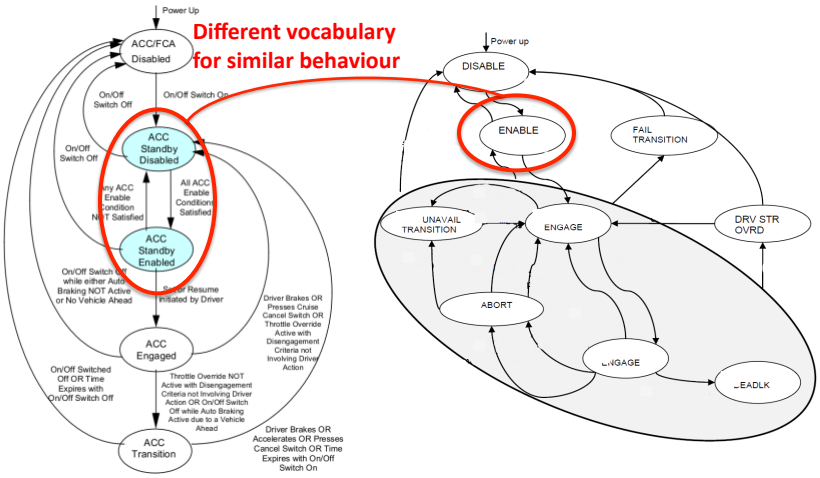
Problem:
State-machine models are hard to write





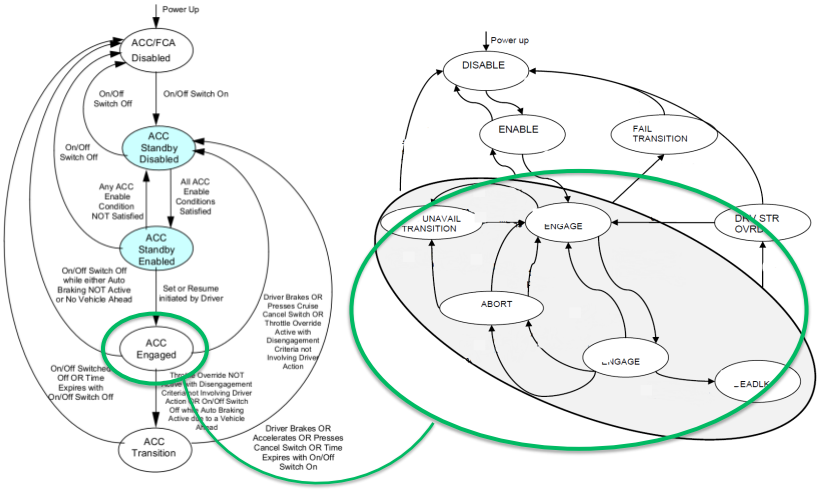
2

Potential for a Requirements Pattern



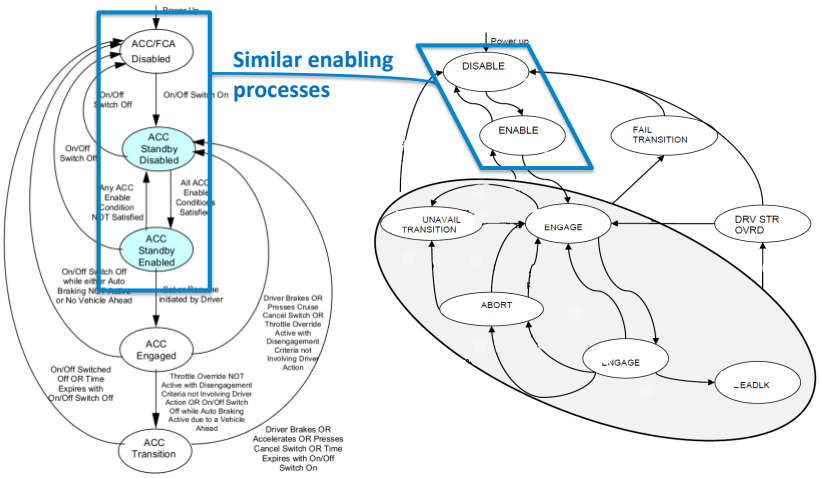
3

Potential for a Requirements Pattern



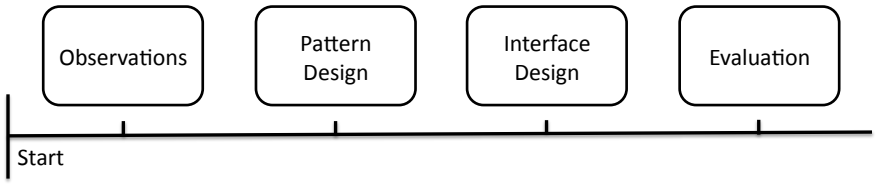
4

Potential for a Requirements Pattern



5

Research Timeline



6

Requirements Patterns

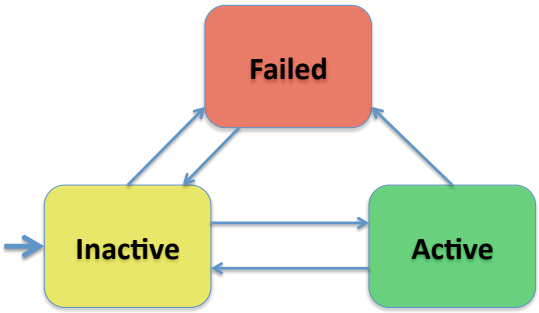
- Domain-specific patterns
  - The majority of patterns are domain specific [RePa '11, '12, '13]
- Patterns for easing requirements elicitation
  - Cliches [Rubenstein and Waters, TSE '91]
  - Domain models [Sutcliffe and Maiden, TSE '98]
- Feature interfaces
  - Everything about a feature is revealed [Jackson and Zave, TSE '98]
  - Features interact solely through a shared context [Apel et. al., ICMT '09]
  - Aspect-aware interfaces provide an interface for aspect modules [Aldrich, ECOOP '05]

7

THE PATTERN

8

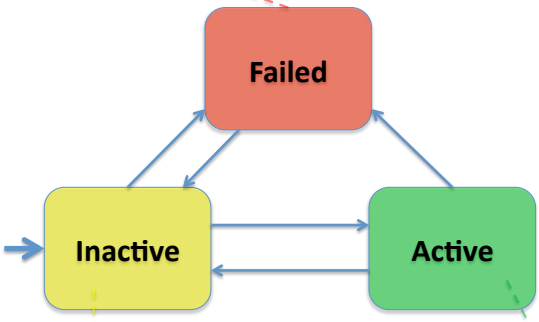
Pattern



9

Pattern

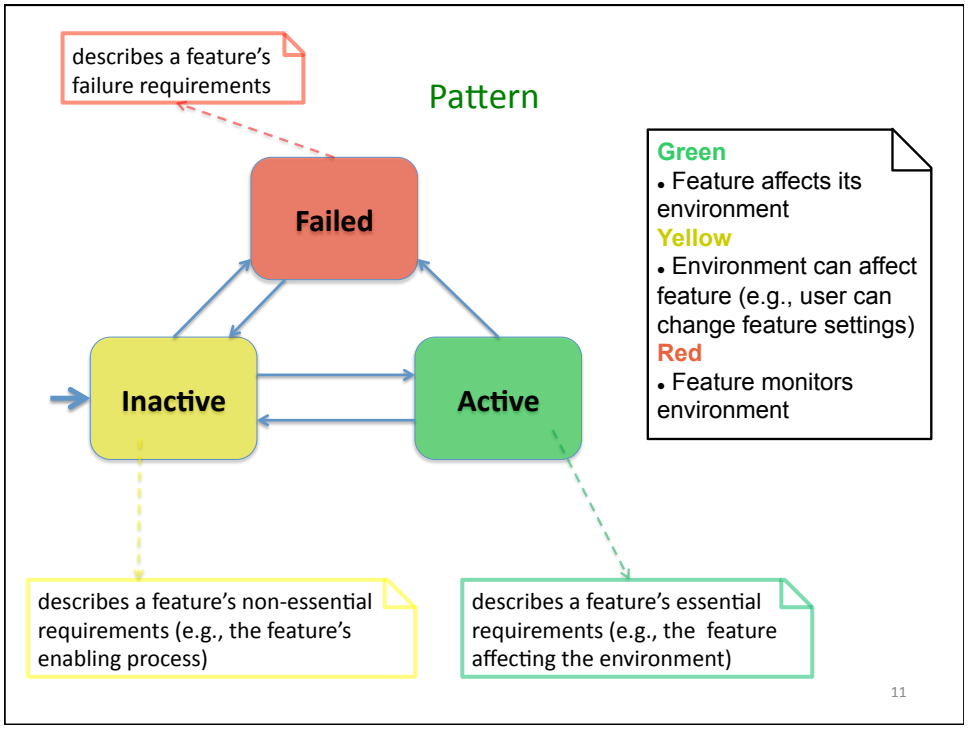
describes a feature's failure requirements

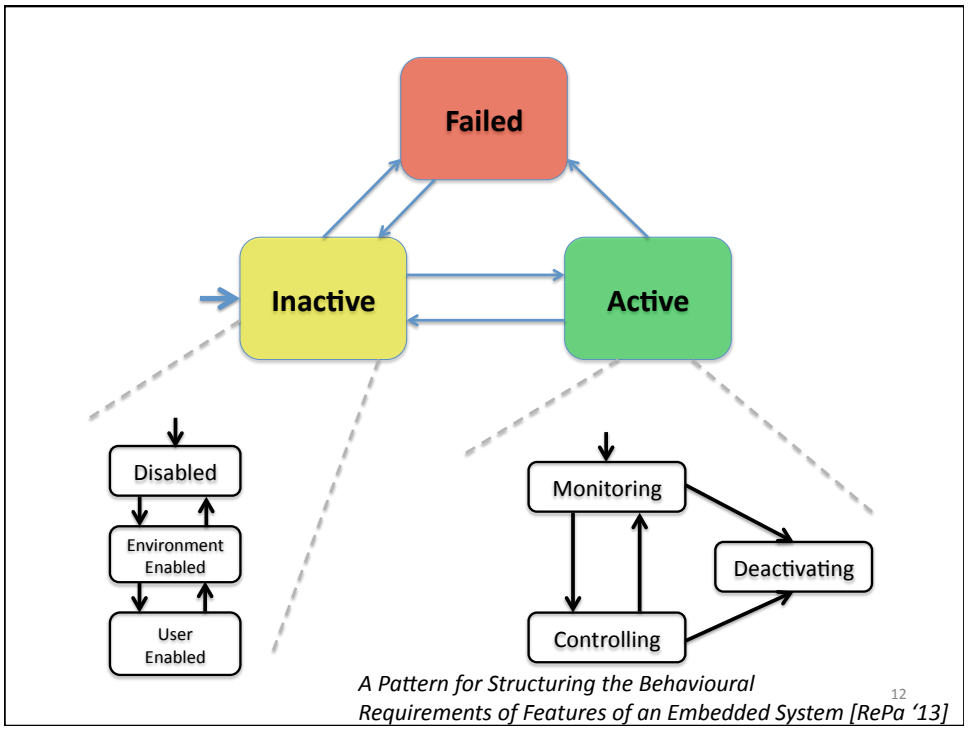


describes a feature's non-essential requirements (e.g., the feature's enabling process)

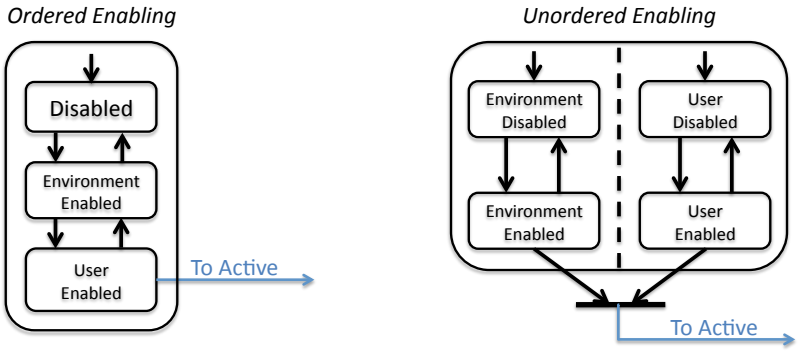
describes a feature's essential requirements (e.g., the feature affecting the environment)

10


describes a feature's failure requirements
Pattern
Failed
Inactive
Active
Green
• Feature affects its environment
Yellow
• Environment can affect feature (e.g., user can change feature settings)
Red
• Feature monitors environment
describes a feature's non-essential requirements (e.g., the feature's enabling process)
describes a feature's essential requirements (e.g., the feature affecting the environment)
11


Failed
Inactive
Active
Disabled
Environment Enabled
User Enabled
Monitoring
Deactivating
Controlling
A Pattern for Structuring the Behavioural Requirements of Features of an Embedded System [RePa '13]
12

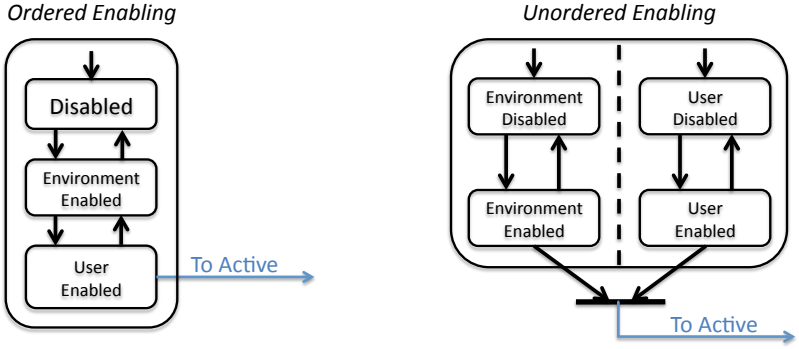
Inactive State Extensions



1. Differentiate between user enabling actions and environment enabling conditions

13

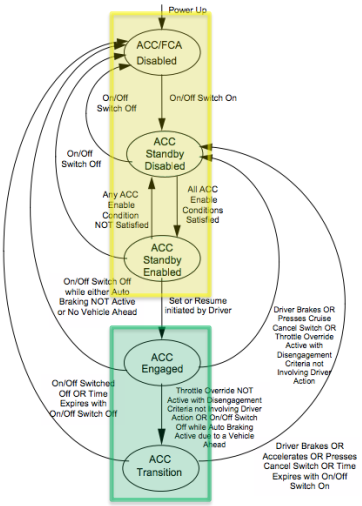
Inactive State Extensions

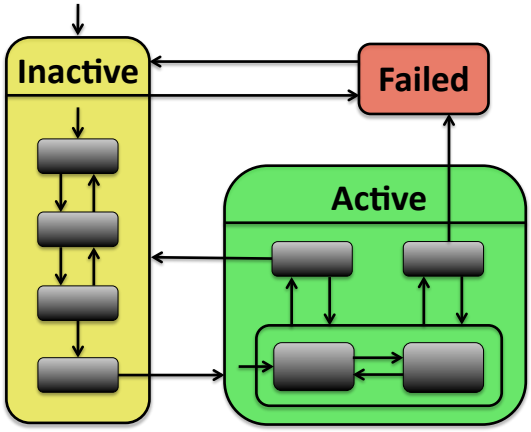


1. Differentiate between user enabling actions and environment enabling conditions
2. Distinguish between ordered enabling processes and processes where ordering does not matter

14

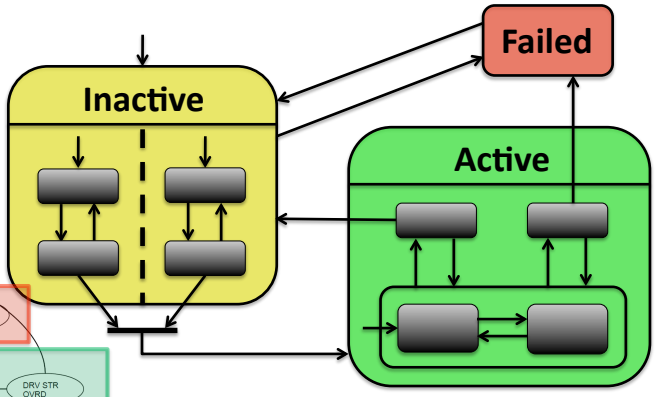
Example 1

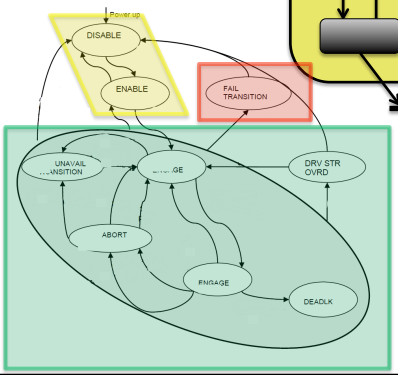




15

Example 2

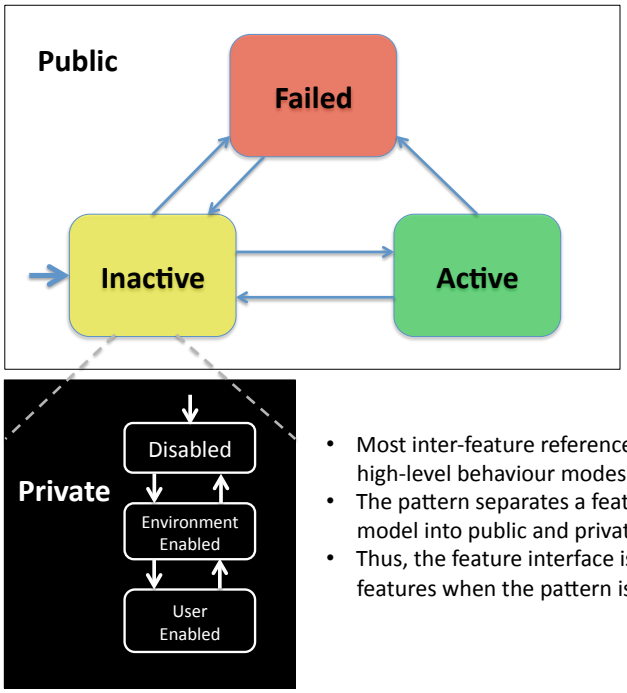




16

THE INTERFACE

17

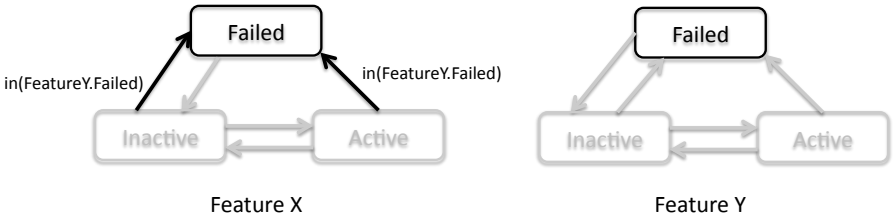


- Most inter-feature references are to the high-level behaviour modes
- The pattern separates a feature's behaviour model into public and private components
- Thus, the feature interface is generic to all features when the pattern is widely used

18

Example

Existing Text: *[FeatureX_Fail] flag shall be set to true when FeatureY is in fail state…*



19

EVALUATION

20

Case Study Methodology

- The pattern was designed by examining five production-grade features
- Two rounds of verification and refinement were performed using an additional 16 features

21

Case Study

- Examined the complete requirements of features to determine if they could be modelled using the pattern
- Also looked at inter-feature references and determined if they reference only public information or private information

21/21 features can be modelled using the pattern

50/57 inter-feature references use the public interface

22

User Study
(not reported in paper)

Performed a user study with **12** participants with varying levels of experience with state-machine modelling

- Three participant groups: **Control (C), Pattern (P), Pattern +Interface (PI)**
- Provided each participant with a tutorial
- Asked to answer questions about a provided model with pattern (for **P** and **PI** groups) or without pattern (**C** group)
- Asked to create a model from a textual description

23

User Study: Reviewing Models

	Participants with correct answers (#/4)		
Model Comprehension Questions	**PI**	**P**	**C**
List all environmental conditions to activate feature	0	2	0
List all user actions to activate feature	4	4	1
List all references to other features	4	3	1
List all states in which the feature affects environment	4	3	2
Describe the failure process	4	3	3
What is the name of the initial state	4	3	3
Average	3.3	3	1.7

24

User Study: Specifying Models

Correctness of written model:

	Participants with correct behaviour (#/4)		
Model behaviour	**PI**	**P**	**C**
References related feature correctly	3	0	1
Includes all enabling conditions	4	4	2
Correct inactive behaviour	4	4	2
Correct deactivation conditions	3	1	0
Correct deactivation behaviour	4	3	2
Correct controlling/monitoring behaviour	3	3	2
Correct failing sequence	4	3	2
Average	3.6	2.7	1.58

25

Conclusion

- We propose a **pattern** for modelling state-machine-based feature requirements and an **interface** to features
- The pattern seems to provide several benefits:
  - General enough to be applied to many features
  - The interface provides a generic method to reference features
  - The pattern and interface improve the readability of state-machine models
  - The pattern and interface improve the correctness of written models

26

References

[1] H. Reubenstein and R. Waters, "The requirements apprentice: automated assistance for requirements acquisition," *IEEE Transactions on Software Engineering*, vol. 17, no. 3, pp. 226-240, March 1991.

[2] A. Sutcliffe and N. Maiden, "The domain theory for requirements engineering," *IEEE Transactions on Software Engineering*, vol. 24, no. 3, pp. 174 –196, March 1998.

[3] M. Jackson and P. Zave, "Distributed feature composition: A virtual architecture for telecommunications services," *IEEE Transactions on Software Engineering*, vol. 24, no. 10, pp. 831–847, Oct. 1998.

[4] S. Apel, F. Janda, S. Trujillo, and C. Kastner, "Model superimposition in software product lines," in *Proceedings of the 2nd International Conference on Theory and Practice of Model Transformations (ICMT'09),* 2009, pp. 4–19.

27

Evaluation: Threats to Validity

- Case Study
  - All features were gathered from one domain in one company
  - The pattern creator performed the case study
- User Study
  - Only 12 participants, split into 3 groups

28