

Specifying Graceful Degradation is Exponentially Complex:

Must rank 2^N system configurations of N software components, sensors, and actuators

Our Model Achieves Scalable Specification for Data Flow-Centric Embedded Systems:

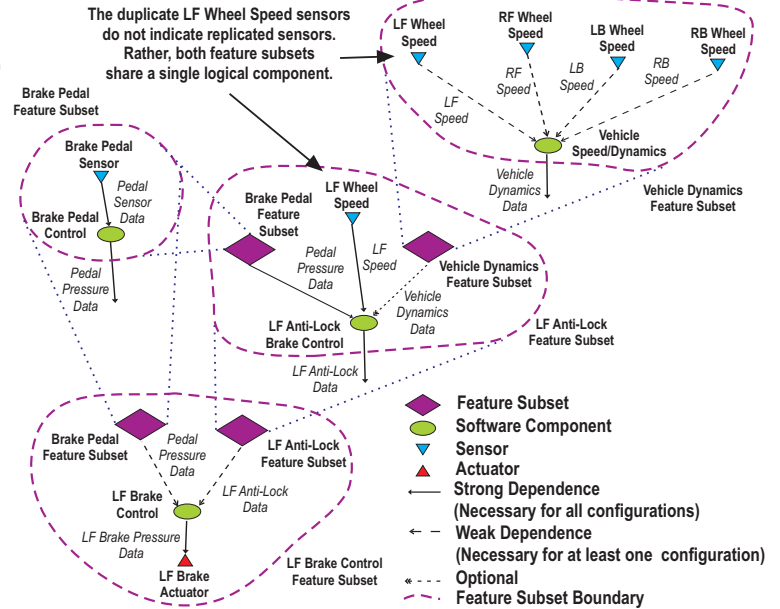
Analysis reduced from $O(2^N)$ to $O(N*2^k)$; k = number of components per subsystem

1. Separate system into orthogonal software and hardware views; focus on software configs
2. Partition system into *feature subsets* based on component input and output interfaces
3. Allow feature subsets to overlap and share components
4. Rank relative utility of configurations of subsystems, not all configurations of entire system
5. Determine system configuration utility as a composition of working subsystem utilities

Proof of Concept:

Scalable Specification of an example Brake-By-Wire system

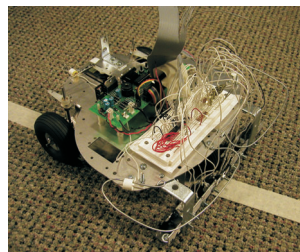
- Real-world electromechanical anti-lock braking system transformed to hypothetical embedded network control system
- 10 Software Components, 5 Sensors, 4 Actuators = 19 System Components
- $2^{19} = 524,288$ possible system configurations; 89,600 valid with positive utility
- In our model: 10 defined subsystems, max 5 components per subsystem
- Specify utility of only 52 subsystem configurations for complete utility function for all system configurations**



Application to Real Distributed Embedded Systems:

Autonomous Robot Navigation:

- Combine line following sensor system with location tracking and navigation Algorithms
- Run robot with combination of failed sensors and software to observe graceful degradation



Graceful Degradation Validation:

Relative utility differences of system configurations in the model match relative performance differences of system configuration implementations

