Visualizing the Analysis of Dynamically Adaptive Systems Using *i** and DSLs

Pete Sawyer, Nelly Bencomo, Danny Hughes,
Paul Grace, Heather J. Goldsby, Betty H. C. Cheng

REV’07,

Agenda

- The problem
- How and why (we think) our visualization helps
- What and who it is for
- How we derived our visualizations
- Our visualizations are …
- How they work
- Practical application
- Show and tell
- Pros, cons, open issues
- Next steps

The problem

- Dynamically Adaptive Systems (DASs)
  - **Challenge:** the need to handle changes to the requirements and corresponding behaviour of a DAS in response to varying environmental conditions.
  - The requirement for dynamic adaptation introduces complexity of a kind not seen in conventional systems where adaptation can happen off-line.

How and why (we think) our visualization helps

- Explicit separation of concerns
  - Identify the global goals and softgoals
  - Identify a discrete set of *domains*
    - ~ stable states of the problem environment
  - Identify the requirements for the system w.r.t. each domain
  - Identify the requirements for adaptation
- Make these concerns explicit using *i**
  - A set of models that correspond to each concern

What and who it is for

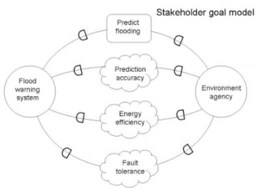
- What:
  - A class of DASs that can be partitioned into a discrete set of domains
    - Problem environments subject to unknown events are (currently) out of scope
- Who:
  - Analysts of DASs
  - System architects
    - Many emerging applications of DASs are subject to technology constraints
    - Often bottom-up, technology-driven
    - … although we want to generalise our approach to anticipate architects having a real choice of s/w infrastructure

How we derived our vizs

- Berry, Cheng & Zhang's paper[1] identified 4 levels of analysis:
  - Level 1: monitoring
    - Conventional analysis per domain
  - Level 2: Decision making
    - Adaptation scenarios
  - Level 3: Adaptation
    - Requirements for adaptation
  - Level 4: Research
    - Requirements on the infrastructure

1. D.M. Berry, B.H.C. Cheng, J. Zhang, "The Four Levels of Requirements Engineering for and in Dynamic Adaptive Systems", *Proc. 11th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'05)*, 2005, Porto, Portugal.

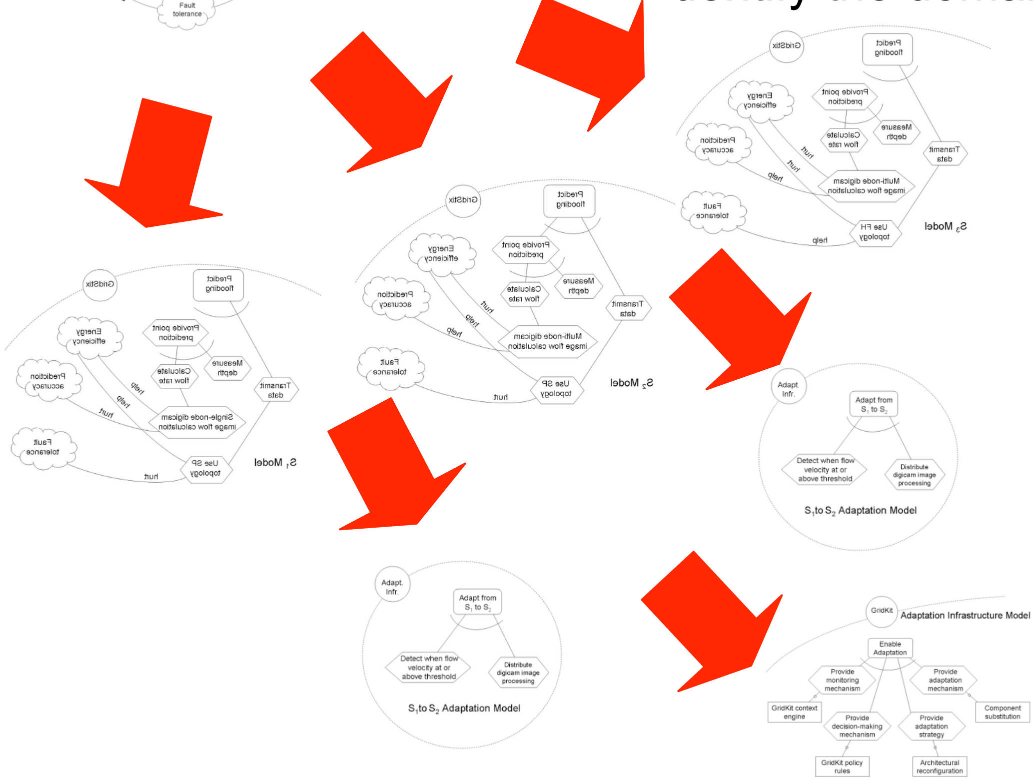
Our visualizations are …



Identify the goals & softgoals

Identify the domains, and …

…. construct one level 1 model for each



Construct a level 2 model that specifies how adaptation to requirements of each domain is accomplished

Specify the adaptation infrastructure

How they work

- Each level one model must:
  - Specify how each goal can be satisfied
  - Specify how each softgoal can be satisfied
    - Expose the trade-offs among the softgoals
- Each level two model must:
  - Satisfy the adaptation goal
    - *adapt from* <<domain a> *to* <<domain b>>
- Each level three model must:
  - Satisfy the goal
    - *enable adaptation* and set of 4 derived tasks

Practical application (1)

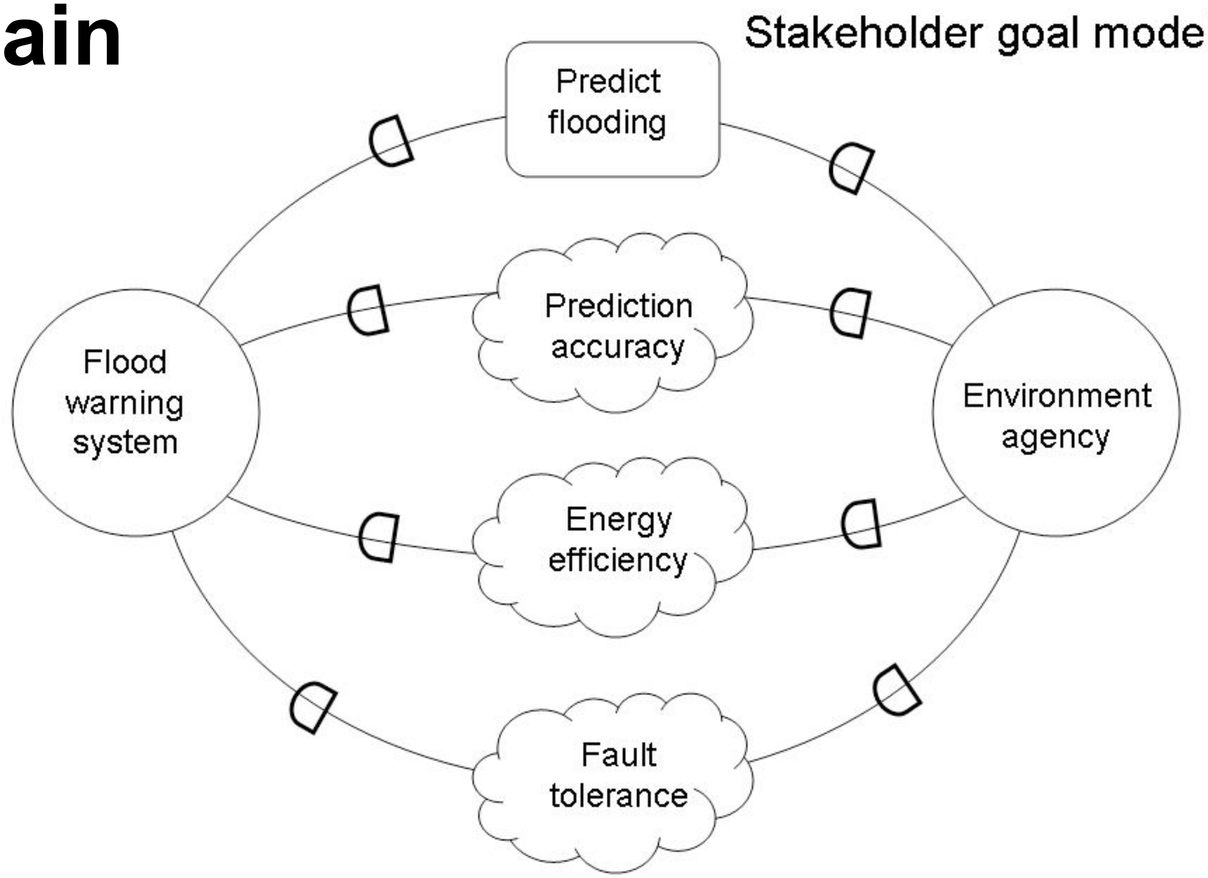
- GridStix is an experimental flood warning system on the River Ribble

Practical application (2)

- GridStix
  - Remote location, no local power infrastructure, but a cellphone network
  - Cheap components
    - Smart sensors
    - Wavelan and Bluetooth
    - Digicams
  - Sensor network
    - Smart nodes configured as a computational grid
    - On-site execution of flood prediction models
  - Adaptation infrastructure
    - The GridKit adaptive middleware framework

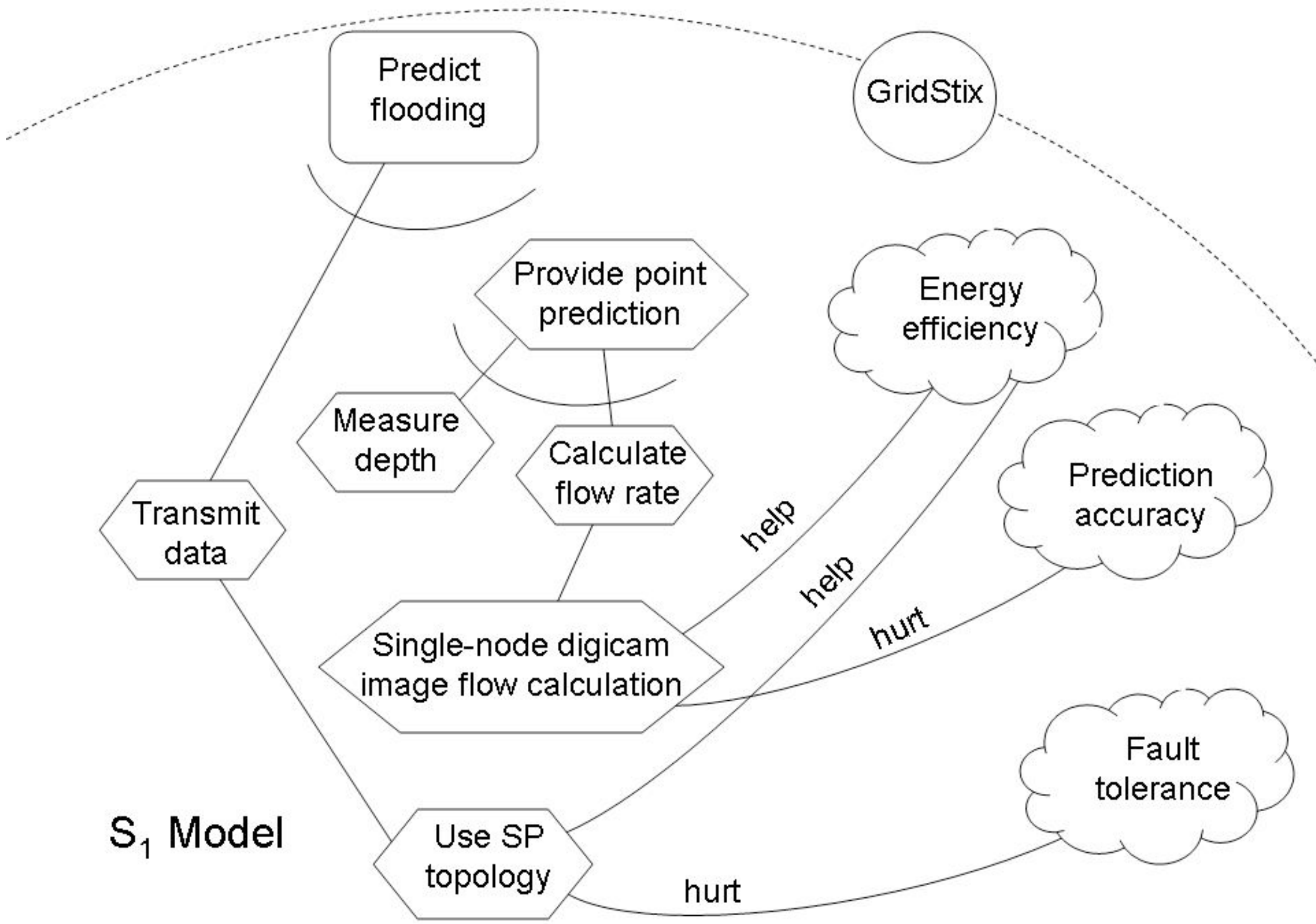
Show and tell (1):

- **Strategic dependencies in the GridStix domain**



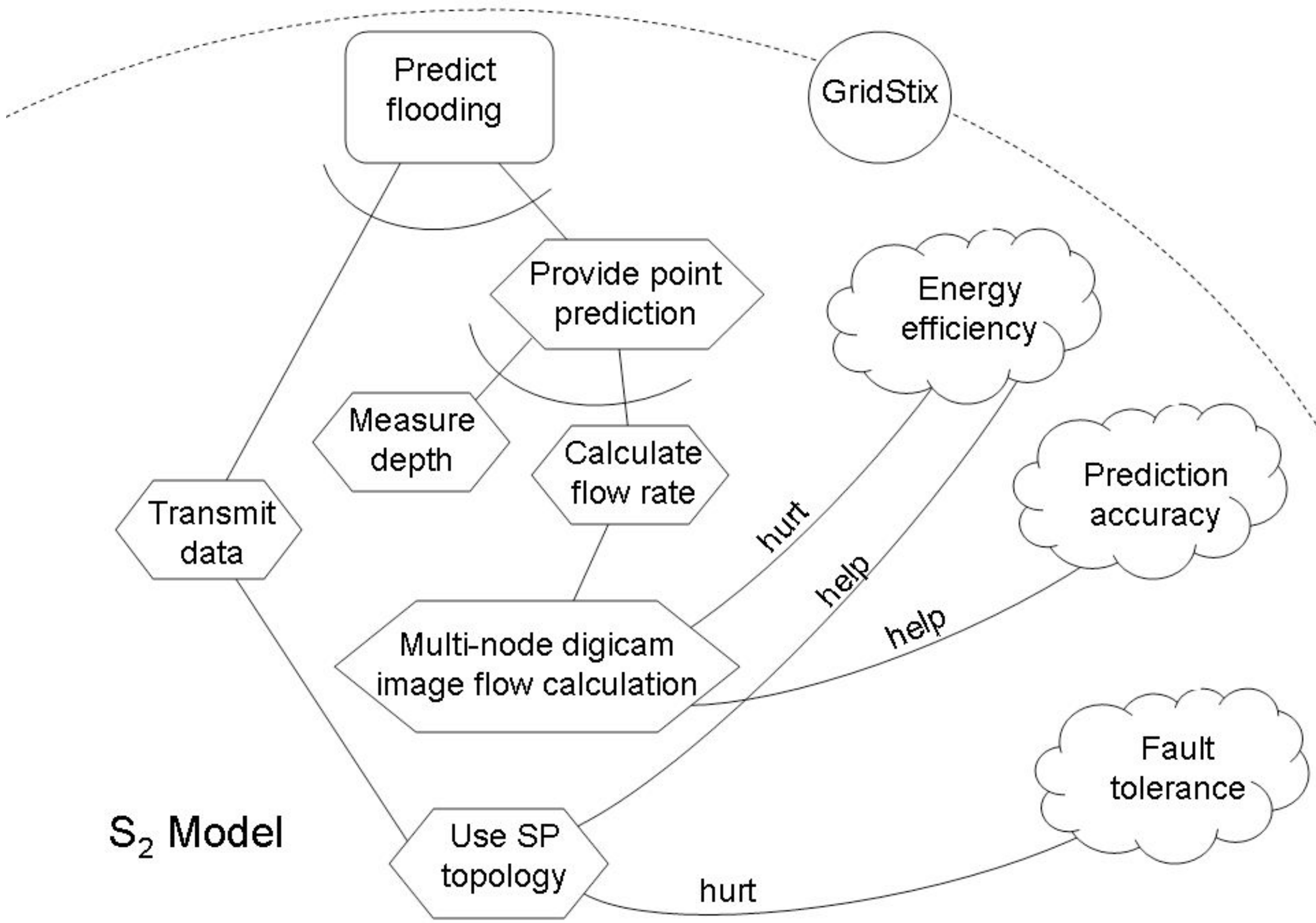
Show and tell (2):

- **Level 1: S1: normal operation**



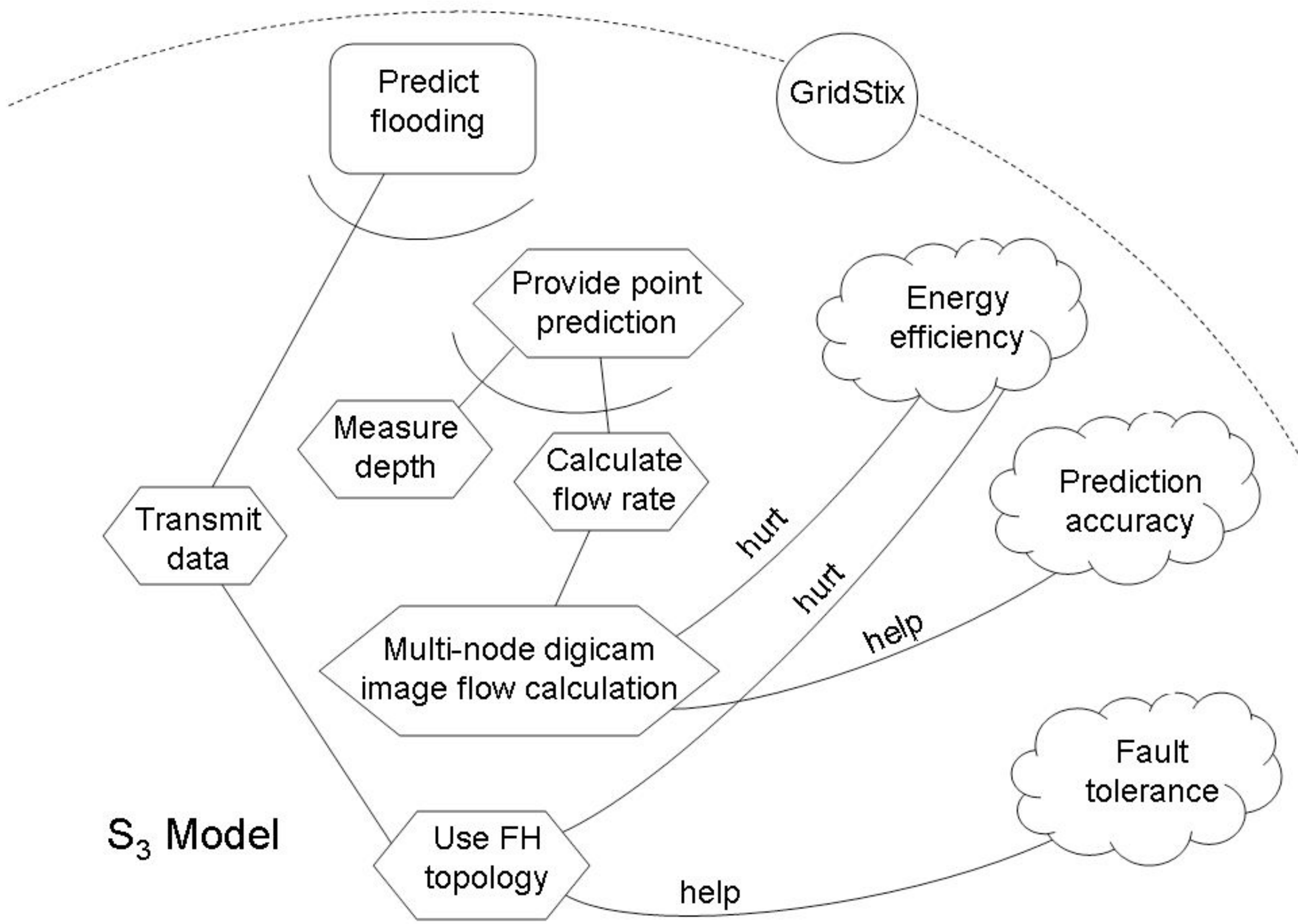
Show and tell (3):

- **Level 1: S2: flow increase**



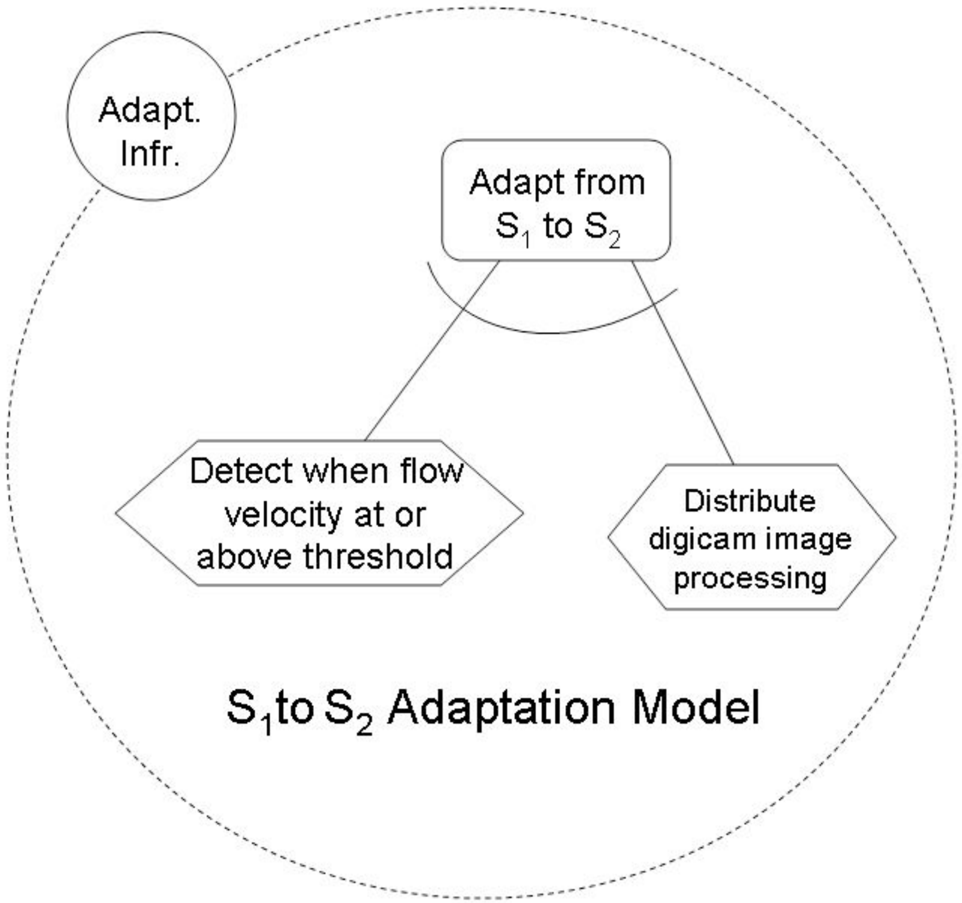
Show and tell (4):

- **Level 1: S3: depth increase**



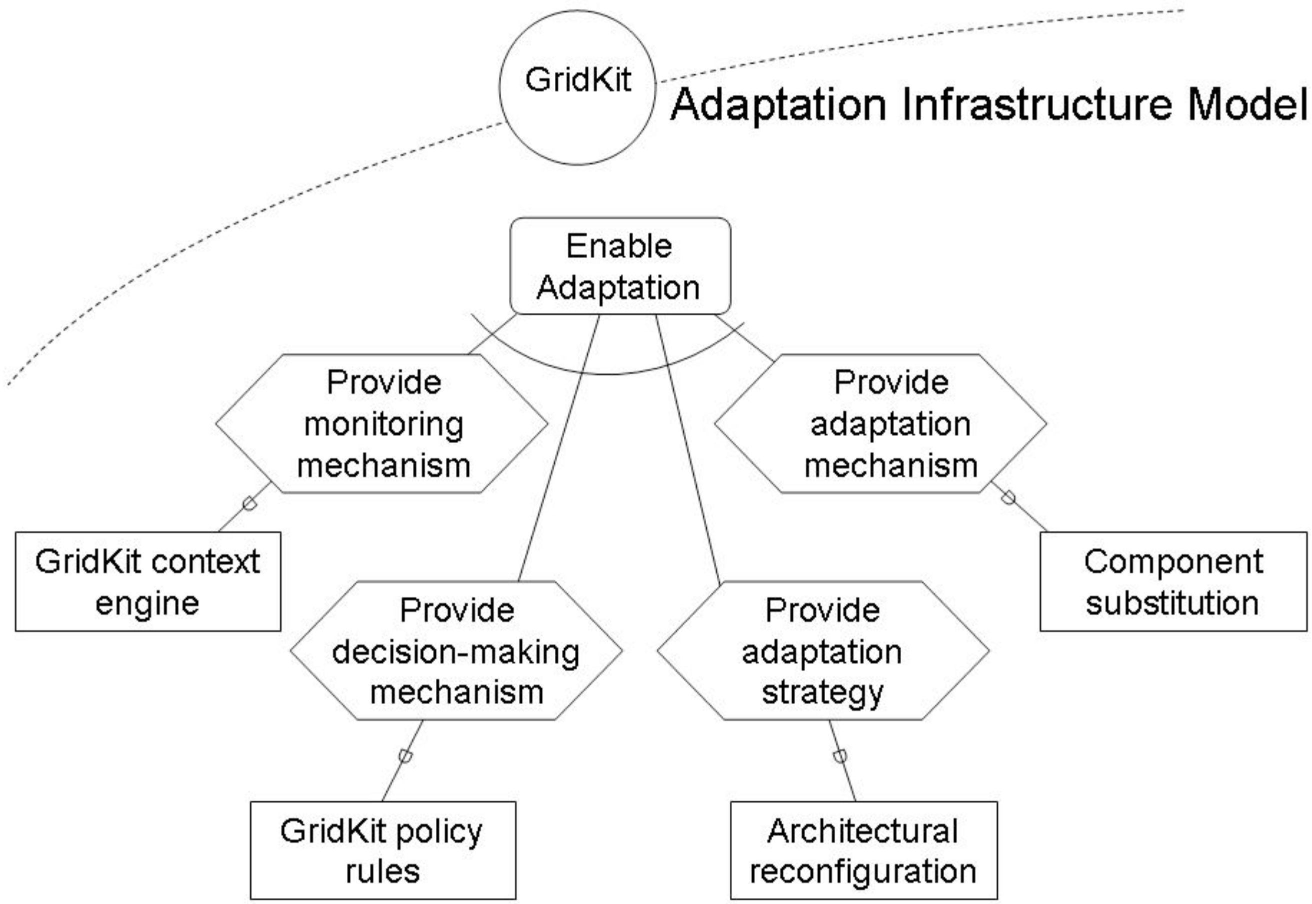
Show and tell (5):

- **Level 2: S1 to S2 adaptation model**



Show and tell (6):

- **Level 3: adaptation infrastructure model**

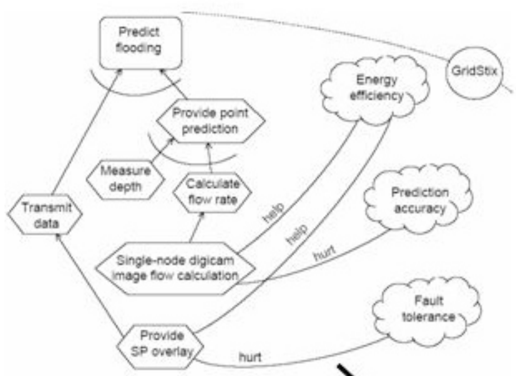


Show and tell (7):

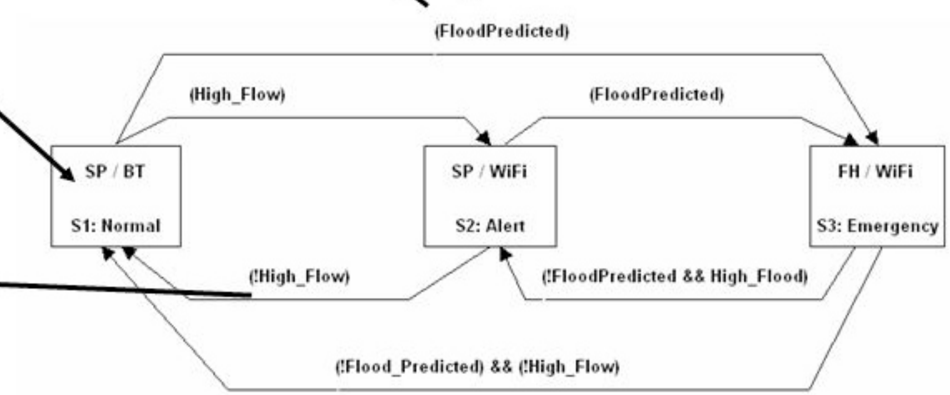
- From Requirements to Design

```
<ReconfigurationRule>
  <Framework>Spanning Tree</Framework>
  <Events>
      <Event>
        <Type>FLOODPREDICTED</Type>
        <Value>true</Value>
      </Event>
</Event>
  <Events>
  <Reconfiguration>
    <FileType>Java</FileType>
    <Name>Reconfigurations.FewestHop</Name>
  </Reconfiguration>
</ReconfigurationRule>
```

```
<ReconfigurationRule>
  <Framework>Overlay</Framework>
  <Events>
      <Event>
       <Type>FLOODPREDICTED</Type>
       <Value>true</Value>
      </Event>
    </Event>
  <Events>
  <Reconfiguration>
    <FileType>Java</FileType>
    <Name>Reconfigurations.Wifi</Name>
  </Reconfiguration>
</ReconfigurationRule>
```




```
<ReconfigurationRule>
  <Framework>Overlay</Framework>
  <Event>
 <Type>! HIGH_FLOW </Type>
    <Value>false</Value>
  <Event>
  <Reconfiguration>
    <FileType>Java</FileType>
    <Name>Reconfigurations.Bluetooth</Name>
  </Reconfiguration>
</ReconfigurationRule>
```



Pros, cons, open issues

- Pros:
  - Pulls out the separate issues of domain behaviour from adaptive behaviour
  - Good match with large subset of DAS applications that have technology constraints
- Cons:
  - Addresses a subset of DASs
- Open Issues:
  - How scalable is it?

Next steps

- Develop a process model for applying the approach
- Formalize the *i** models
- Map onto existing work on model-driven engineering for adaptive infrastructures