

SER001

Goal-Driven Adaptable Software Architecture for Autonomous Systems

(Completed Jan 2009)

William Heaven, Daniel Sykes, Jeff Magee, Jeff Kramer

SEAS DTC Annual Technical Conference 2009

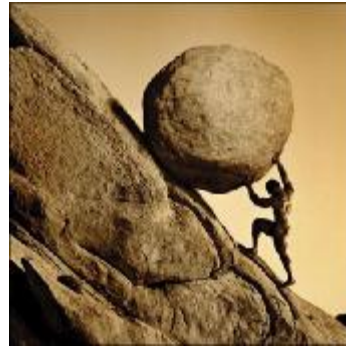
The Challenge

- Autonomous systems are deployed in environments in which contact with operators is infrequent or undesirable.
- To be reliable, autonomous systems should be able to adapt to new circumstances on their own.

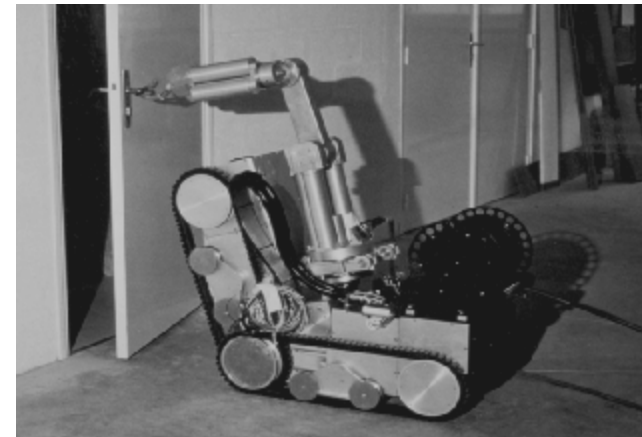
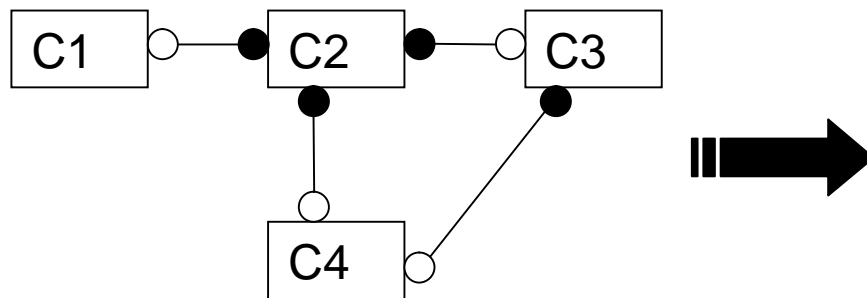


The Challenge

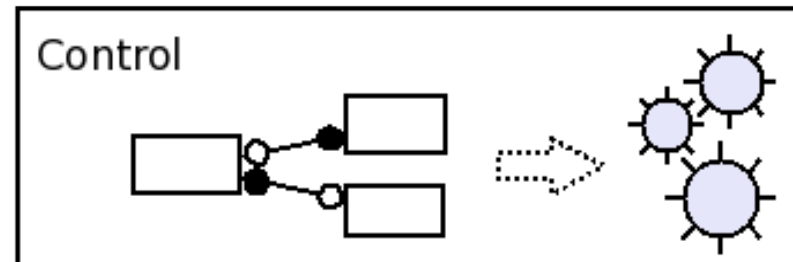
- The system should determine (without intervention)
 - **How** to achieve / maintain its goal and with **what** capabilities



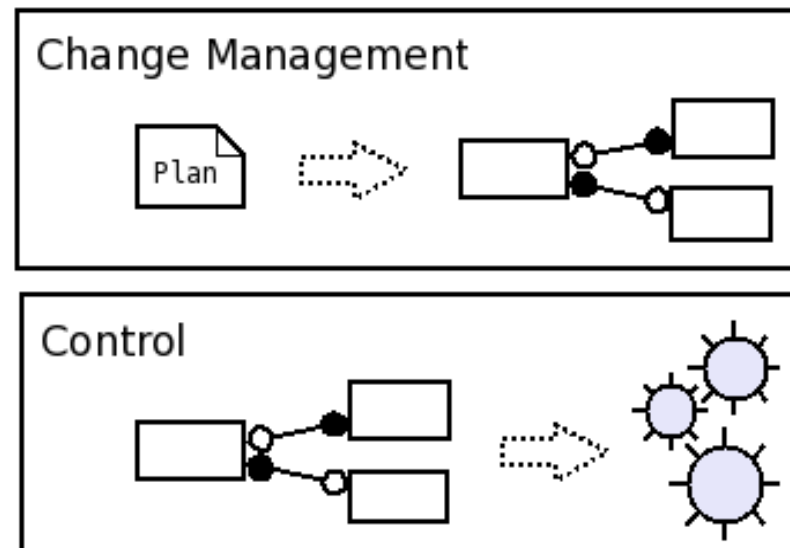
- The **software components** required to control these capabilities



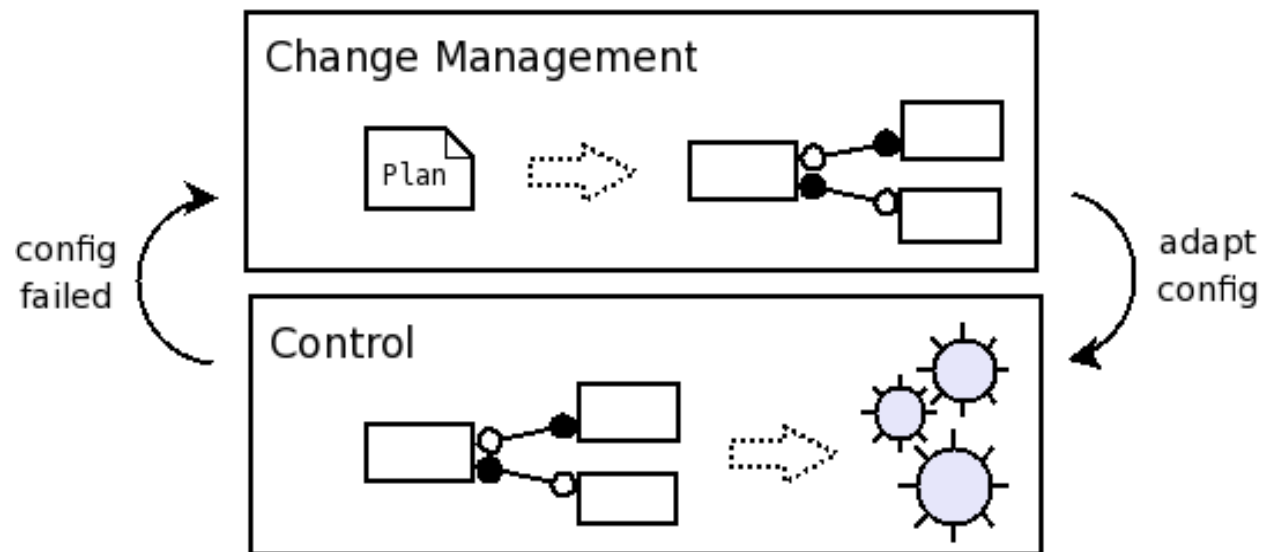
Summary of SER001 Results



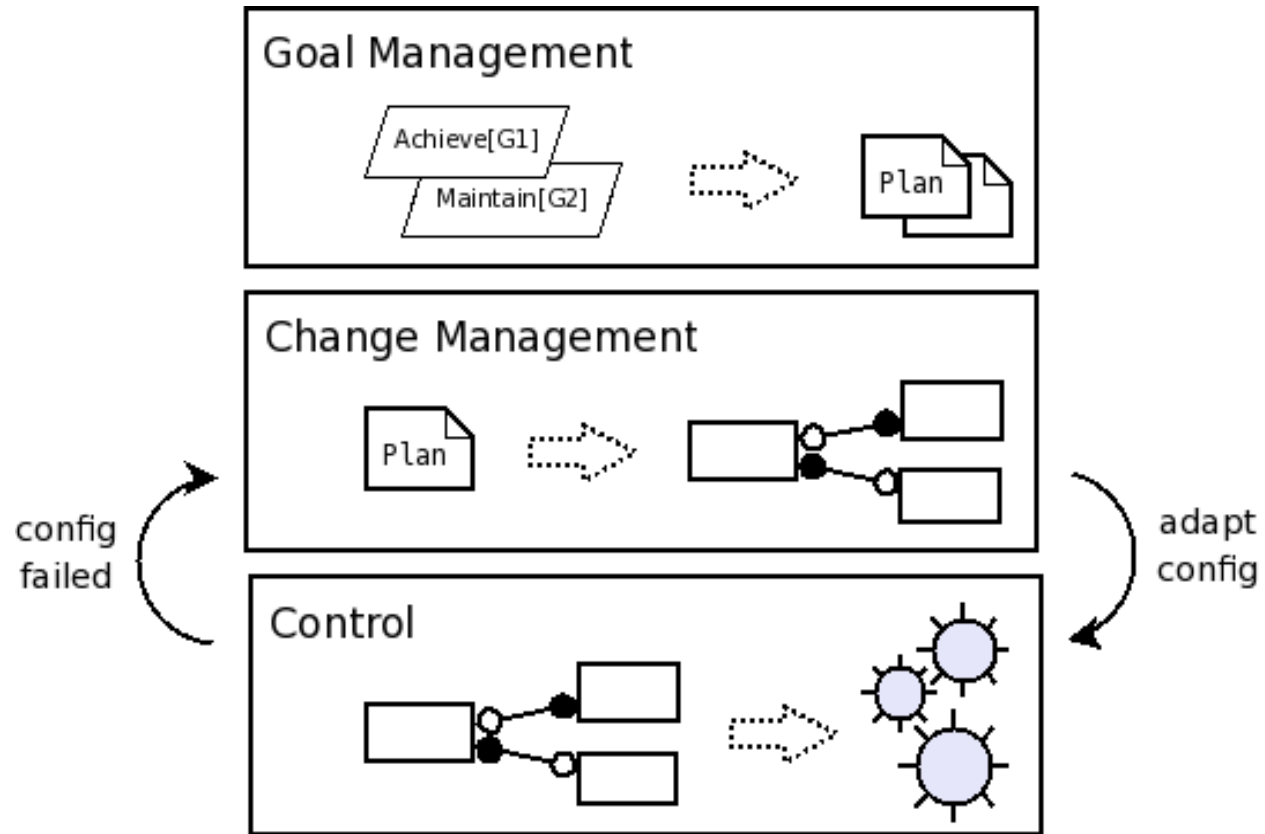
Summary of SER001 Results



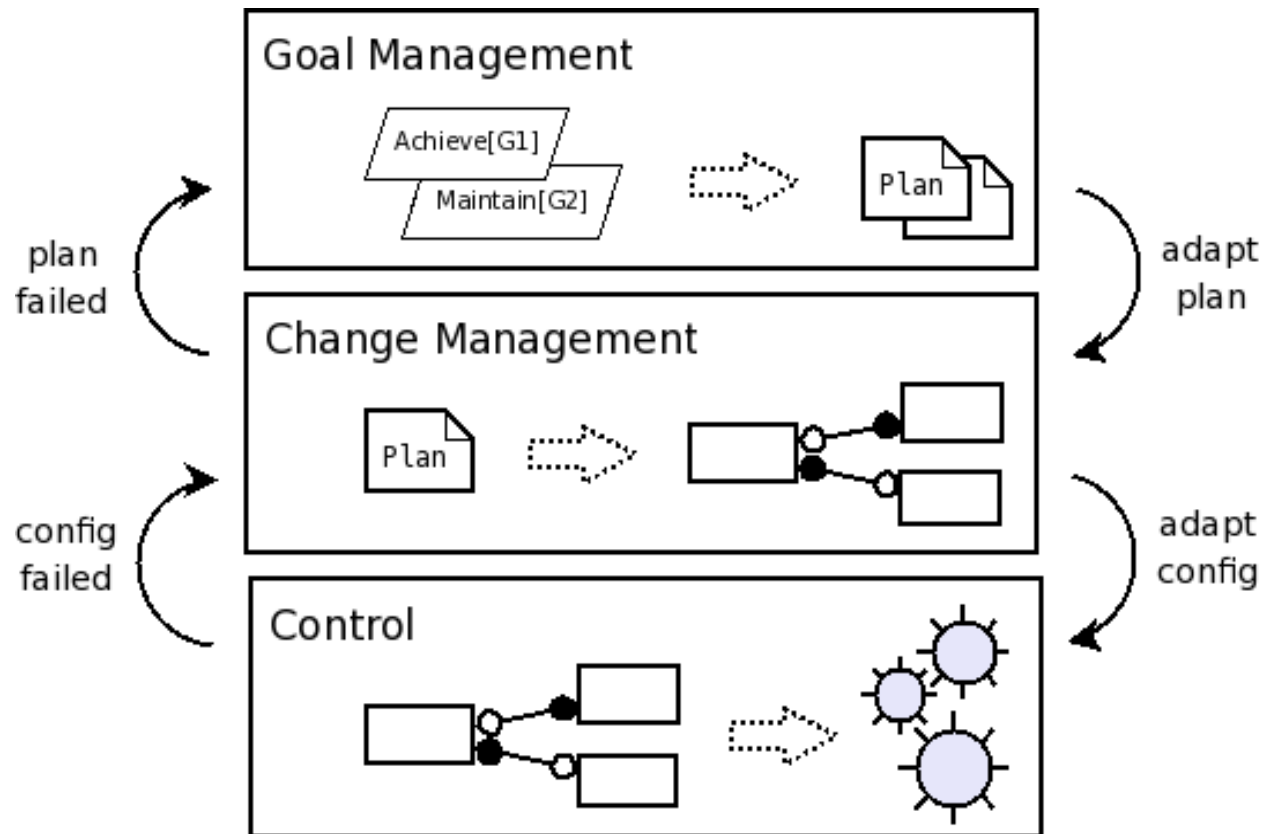
Summary of SER001 Results



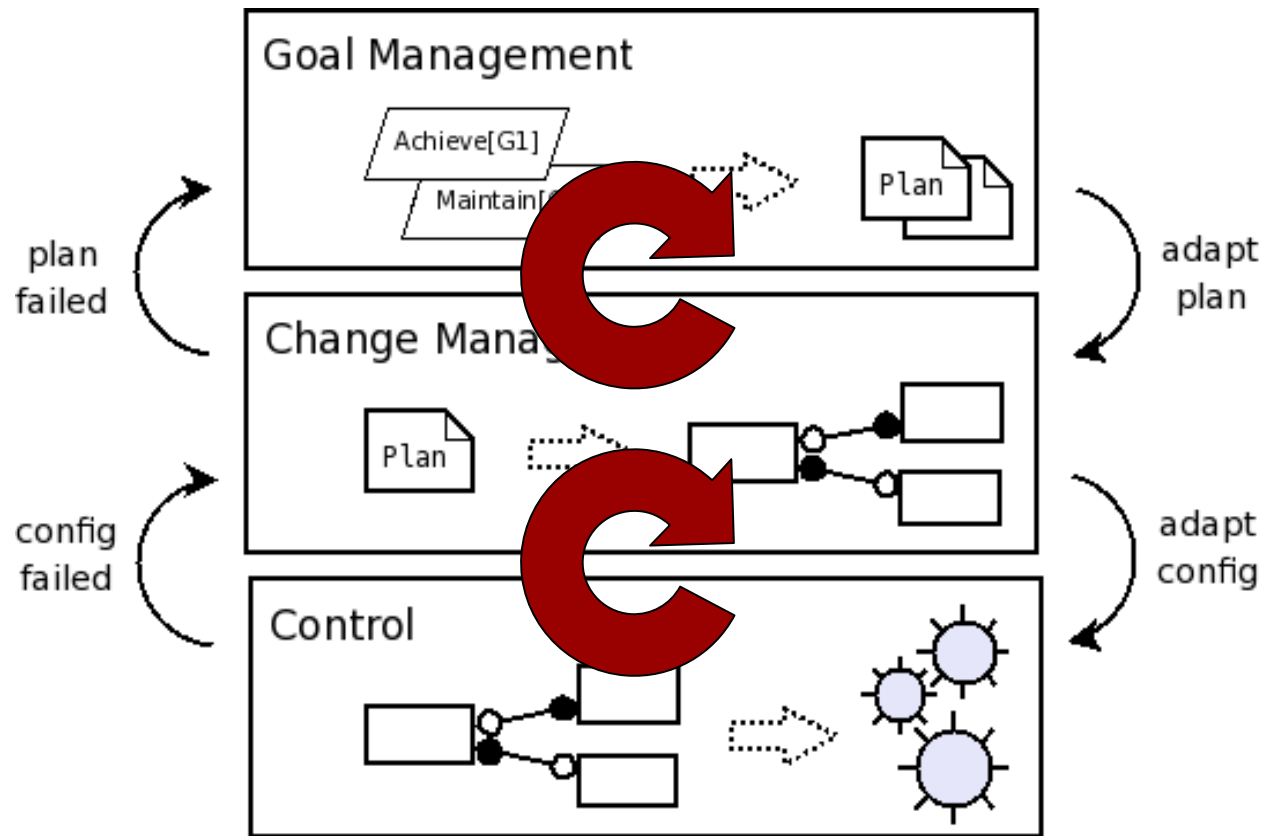
Summary of SER001 Results



Summary of SER001 Results



Summary of SER001 Results



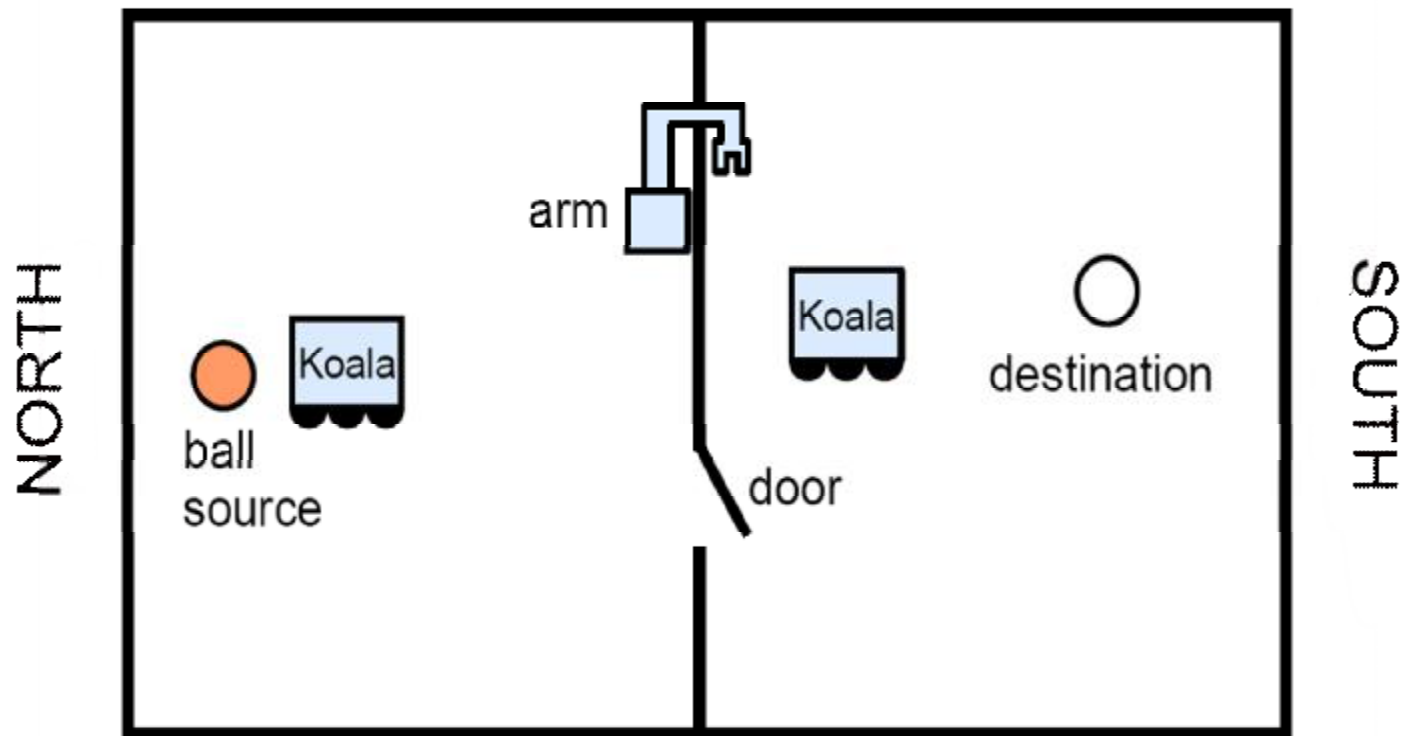
Ultimately, enables self-adaptation at two levels of abstraction.

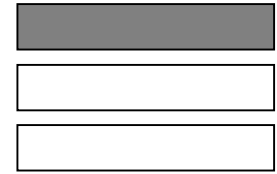
Example Scenario

- Delivery of aid package to person trapped in building.
- Must reach its destination in environment too dangerous for humans.
- Autonomous system must ensure reliability by adapting to changing circumstances.



Example Scenario





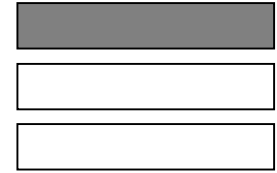
Modelling the Domain

- Autonomous system requires a **domain model**
- Defines salient **properties** of system and environment
e.g. *AtTarget*, *BatteryFull*, *DoorOpen* etc.
- Specifies key **behaviour** of system and environment
 - What system can do and how environment might respond



Behaviour modelled like a **game** between system and environment: a **system action** is followed an **environment response**. For example:

`startGotoTarget` → `_arrivedAtTarget`



Modelling the Domain

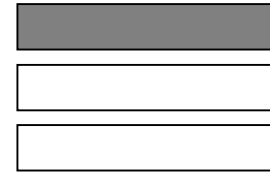
- Actions partitioned into those the system can and cannot control:

```
set Actions = {Sys, EnvDep, EnvInd}
```

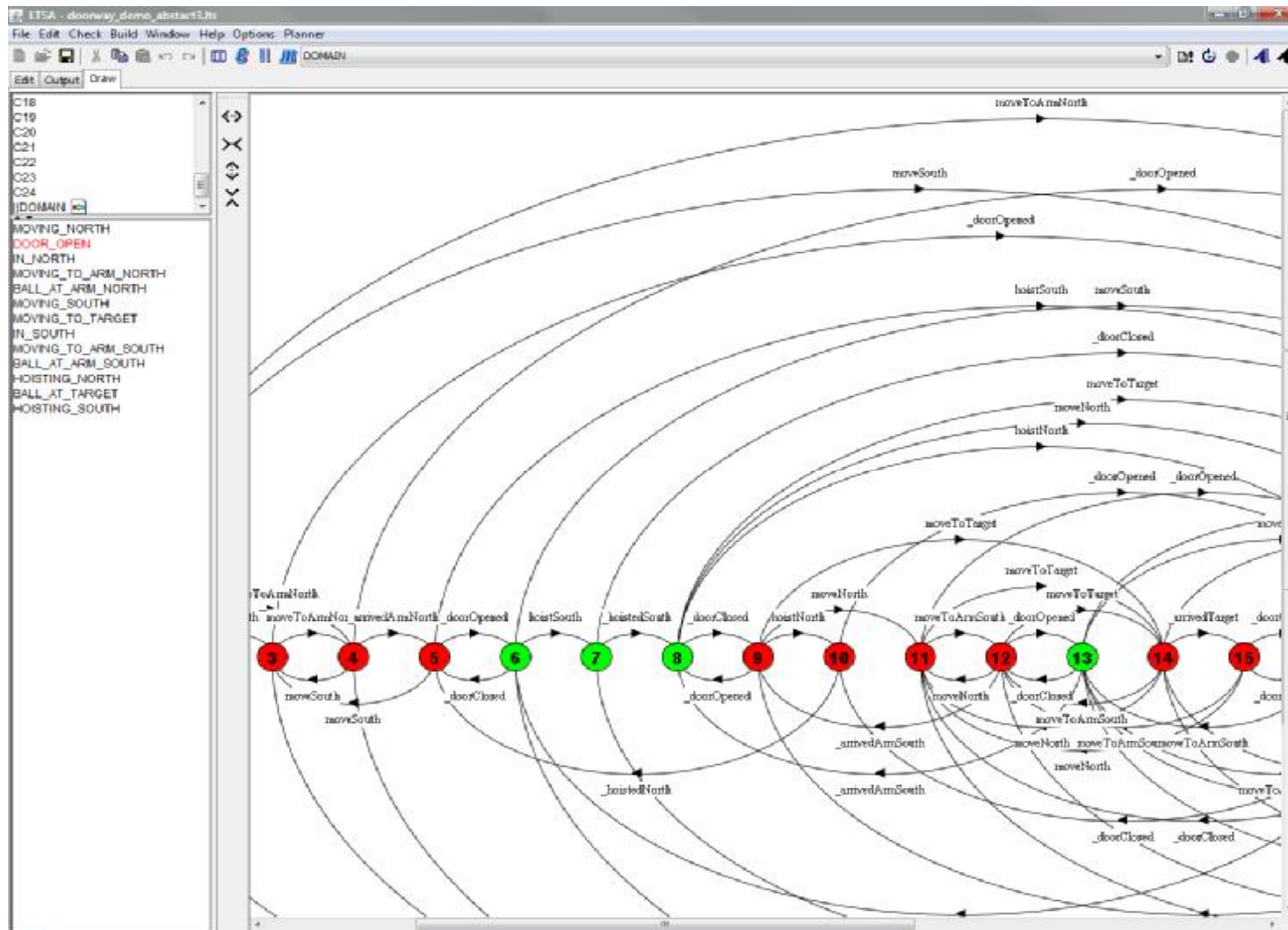
```
// system actions
set Sys = {
  moveToTarget, moveToArmNorth, moveToArmSouth,
  moveNorth, moveSouth, hoistSouth, hoistNorth
}

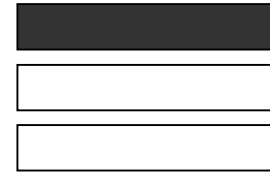
// dependent environment actions
set EnvDep = {
  arrivedTarget, arrivedArmNorth, arrivedArmSouth,
  arrivedSouth, arrivedNorth, hoistedNorth, hoistedSouth
}

// independent environment actions
set EnvInd = {doorOpened, doorClosed}
```

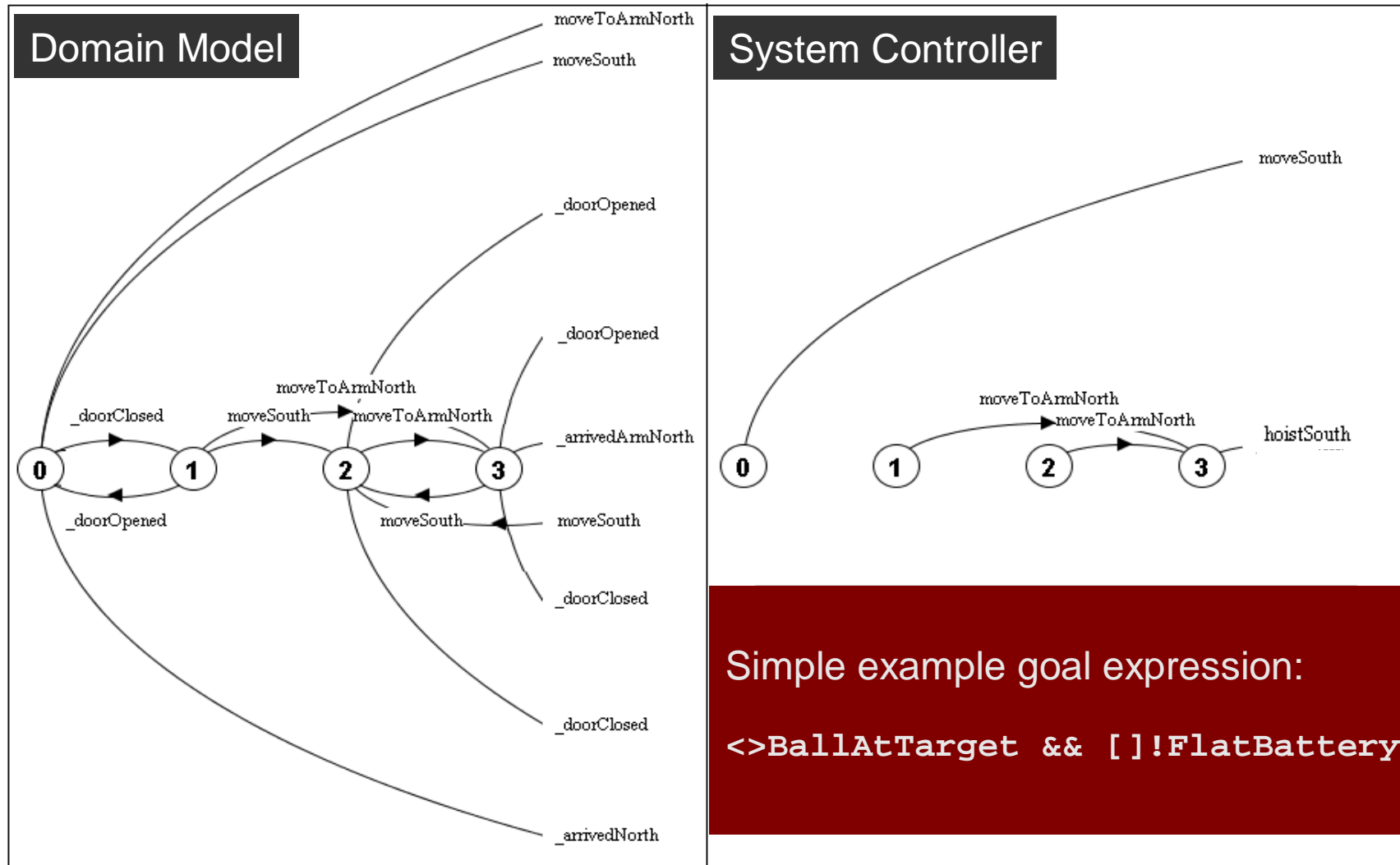


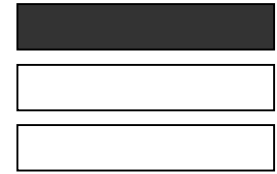
Modelling the Domain





Synthesising a Controller





Synthesising a Controller

```
(0) !MOVING_NORTH && DOOR_OPEN && IN_NORTH && !MOVING_TO_ARM_NORTH
    && !BALL_AT_ARM_NORTH && !MOVING_SOUTH && !MOVING_TO_TARGET
    && !IN_SOUTH && !MOVING_TO_ARM_SOUTH && !BALL_AT_ARM_SOUTH
    && !HOISTING_NORTH && !BALL_AT_TARGET && !HOISTING_SOUTH
    -> moveSouth

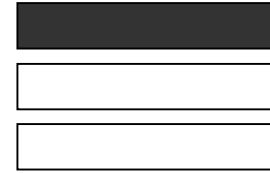
(1) !MOVING_NORTH && !DOOR_OPEN && IN_NORTH && !MOVING_TO_ARM_NORTH
    && !BALL_AT_ARM_NORTH && !MOVING_SOUTH && !MOVING_TO_TARGET
    && !IN_SOUTH && !MOVING_TO_ARM_SOUTH && !BALL_AT_ARM_SOUTH
    && !HOISTING_NORTH && !BALL_AT_TARGET && !HOISTING_SOUTH
    -> moveToArmNorth

(2) !MOVING_NORTH && !DOOR_OPEN && IN_NORTH && !MOVING_TO_ARM_NORTH
    && !BALL_AT_ARM_NORTH && MOVING_SOUTH && !MOVING_TO_TARGET
    && !IN_SOUTH && !MOVING_TO_ARM_SOUTH && !BALL_AT_ARM_SOUTH
    && !HOISTING_NORTH && !BALL_AT_TARGET && !HOISTING_SOUTH
    -> moveToArmNorth

...

(15) !MOVING_NORTH && DOOR_OPEN && !IN_NORTH && !MOVING_TO_ARM_NORTH
    && !BALL_AT_ARM_NORTH && !MOVING_SOUTH && !MOVING_TO_TARGET
    && IN_SOUTH && !MOVING_TO_ARM_SOUTH && !BALL_AT_ARM_SOUTH
    && !HOISTING_NORTH && BALL_AT_TARGET && !HOISTING_SOUTH
    -> _done

...
```



Executing a Controller

```
(0) !MOVING_NORTH && DOOR_OPEN && IN_NORTH && !MOVING_TO_ARM_NORTH  
&& !BALL_AT_ARM_NORTH &&  
&& !IN_SOUTH && !MOVING_TO_ARM_SOUTH && !BALL_AT_ARM_SOUTH  
&& !HOISTING_NORTH && !BALL_AT_TARGET && !HOISTING_SOUTH  
-> moveSouth
```

Condition (conjunction of domain properties)

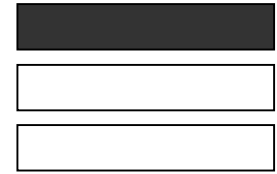
```
(1) !MOVING_NORTH && !DOOR_OPEN && IN_NORTH && !MOVING_TO_ARM_NORTH  
&& !BALL_AT_ARM_NORTH && !MOVING_SOUTH && !MOVING_TO_TARGET  
&& !IN_SOUTH && !MOVING_TO_ARM_SOUTH && !BALL_AT_ARM_SOUTH  
&& !HOISTING_NORTH && !BALL_AT_TARGET && !HOISTING_SOUTH  
-> moveToArmNorth
```

```
(2) !MOVING_NORTH && !DOOR_OPEN && IN_NORTH && !MOVING_TO_ARM_NORTH  
&& !BALL_AT_ARM_NORTH && MOVING_SOUTH && !MOVING_TO_TARGET  
&& !IN_SOUTH && !MOVING_TO_ARM_SOUTH && !BALL_AT_ARM_SOUTH  
&& !HOISTING_NORTH && !BALL_AT_TARGET && !HOISTING_SOUTH  
-> moveToArmNorth
```

...

```
(15) !MOVING_NORTH && DOOR_OPEN && !IN_NORTH && !MOVING_TO_ARM_NORTH  
&& !BALL_AT_ARM_NORTH && !MOVING_SOUTH && !MOVING_TO_TARGET  
&& IN_SOUTH && !MOVING_TO_ARM_SOUTH && !BALL_AT_ARM_SOUTH  
&& !HOISTING_NORTH && BALL_AT_TARGET && !HOISTING_SOUTH  
-> _done
```

...



Executing a Controller

```
(0) !MOVING_NORTH && DOOR_OPEN && IN_NORTH && !MOVING_TO_ARM_NORTH
&& !BALL_AT_ARM_NORTH && !MOVING_SOUTH && !MOVING_TO_TARGET
&& !IN_SOUTH && !MOVING_TO_ARM_SOUTH && !BALL_AT_ARM_SOUTH
&& !HOISTING_NORTH && !BALL_AT_TARGET && !HOISTING_SOUTH
-> moveSouth

(1) !MOVING_NORTH && !DOOR_OPEN && IN_NORTH && !MOVING_TO_ARM_NORTH
&& !BALL_AT_ARM_NORTH && !MOVING_SOUTH && !MOVING_TO_TARGET
&& !IN_SOUTH && !MOVING_TO_ARM_SOUTH && !BALL_AT_ARM_SOUTH
&& !HOISTING_NORTH && !BALL_AT_TARGET && !HOISTING_SOUTH
-> moveToArmNorth

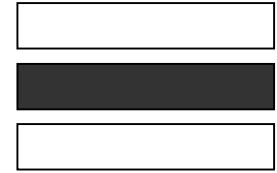
(2) !MOVING_NORTH && !DOOR_OPEN && IN_NORTH && !MOVING_TO_ARM_NORTH
&& !BALL_AT_ARM_NORTH && MOVING_SOUTH && !MOVING_TO_TARGET
&& !IN_SOUTH && !MOVING_TO_ARM_SOUTH && !BALL_AT_ARM_SOUTH
&& !HOISTING_NORTH && !BALL_AT_TARGET && !HOISTING_SOUTH
-> moveToArmNorth

...

(15) !MOVING_NORTH && DOOR_OPEN && !IN_NORTH && !MOVING_TO_ARM_NORTH
&& !BALL_AT_ARM_NORTH && !MOVING_SOUTH && !MOVING_TO_TARGET
&& IN_SOUTH && !MOVING_TO_ARM_SOUTH && !BALL_AT_ARM_SOUTH
&& !HOISTING_NORTH && BALL_AT_TARGET && !HOISTING_SOUTH
-> _done

...
```

Action

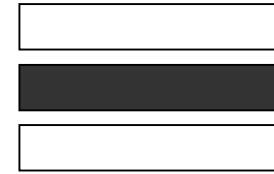


Component Selection

- Middle layer selects and instantiates the software components and configurations required to implement a generated plan

Component Selection

- Middle layer selects and instantiates the software components and configurations required to implement a generated plan
- Components implement actions

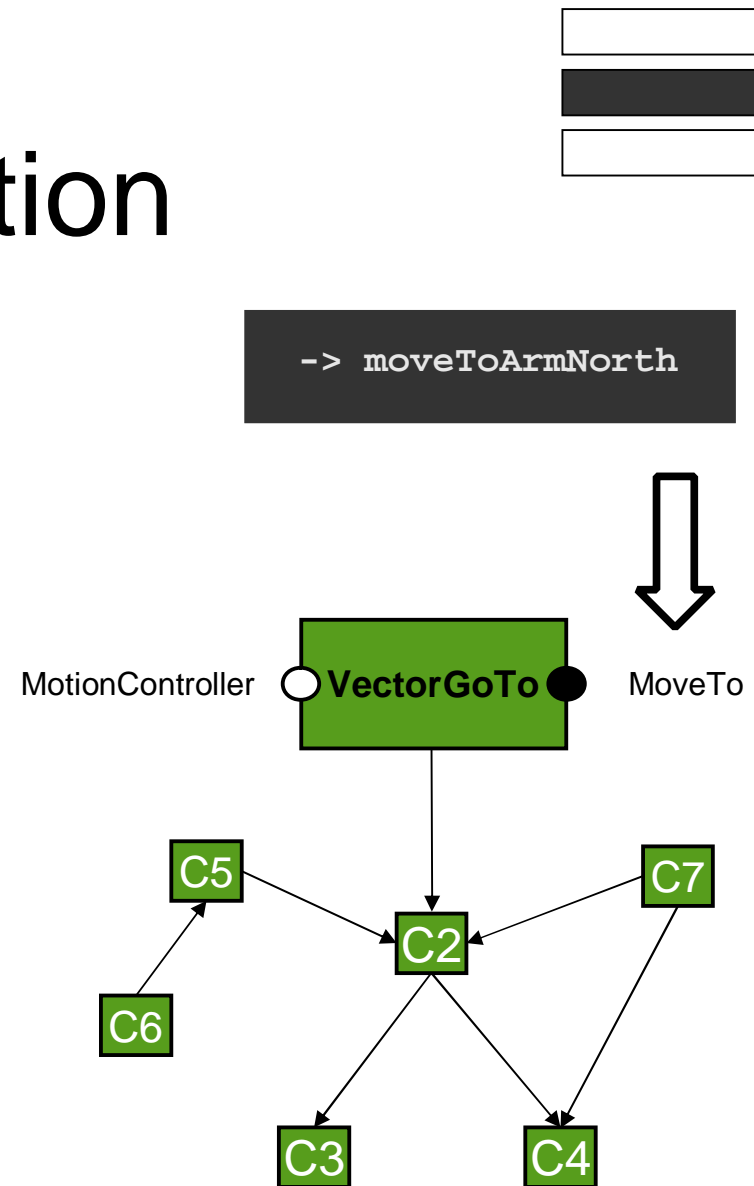


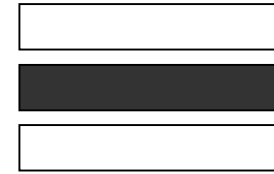
-> moveToArmNorth



Component Selection

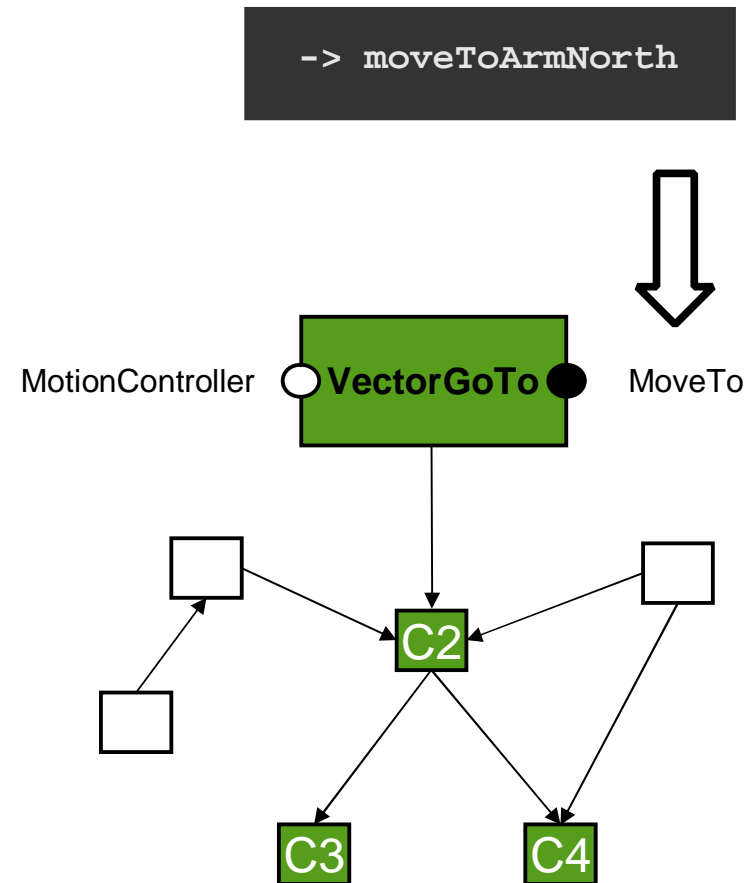
- Middle layer selects and instantiates the software components and configurations required to implement a generated plan
- Components implement actions
- Interfaces give a dependency graph between components

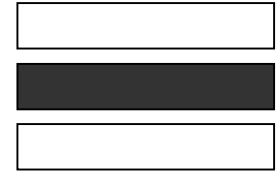




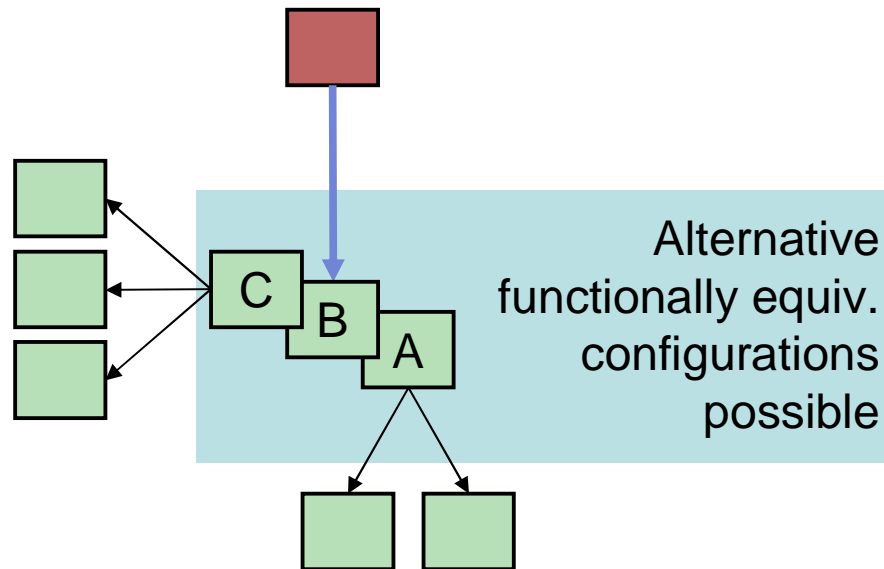
Component Selection

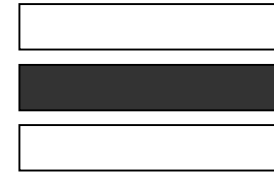
- Middle layer selects and instantiates the software components and configurations required to implement a generated plan
- Components implement actions
- Interfaces give a dependency graph between components
- Full configuration constructed by following dependency chains
- Selected configurations checked against structural constraints



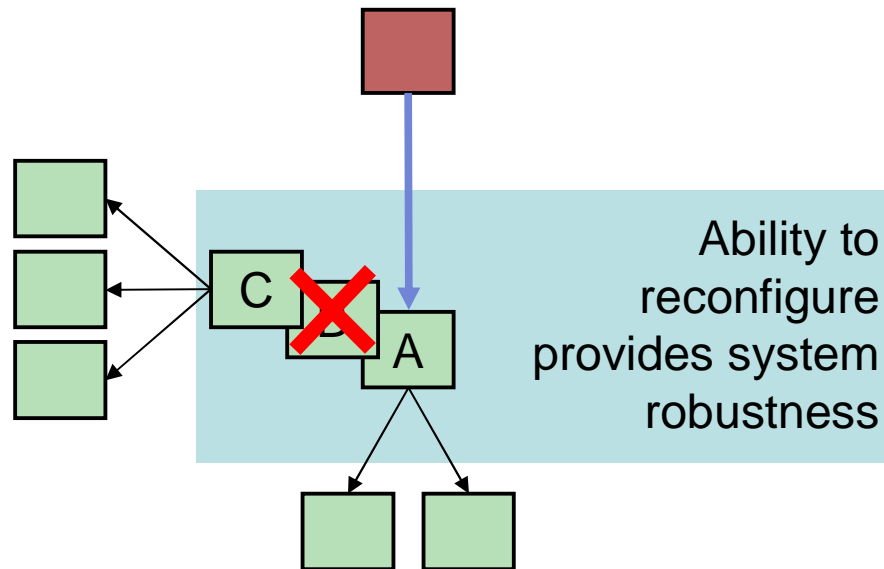


Selecting Alternatives

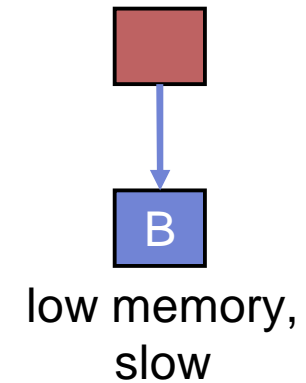
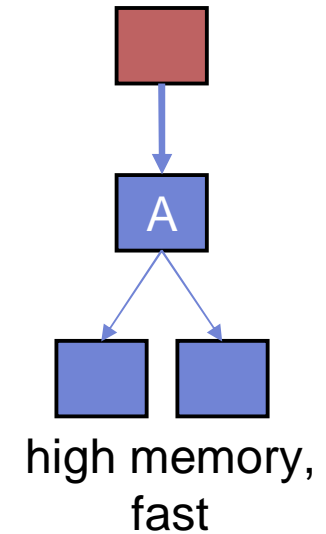
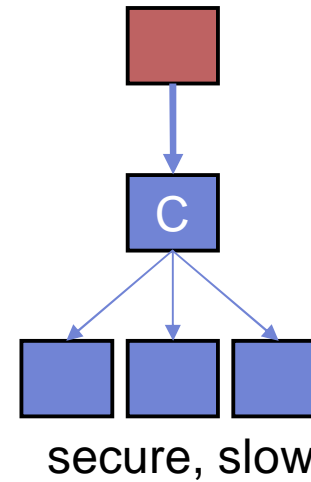
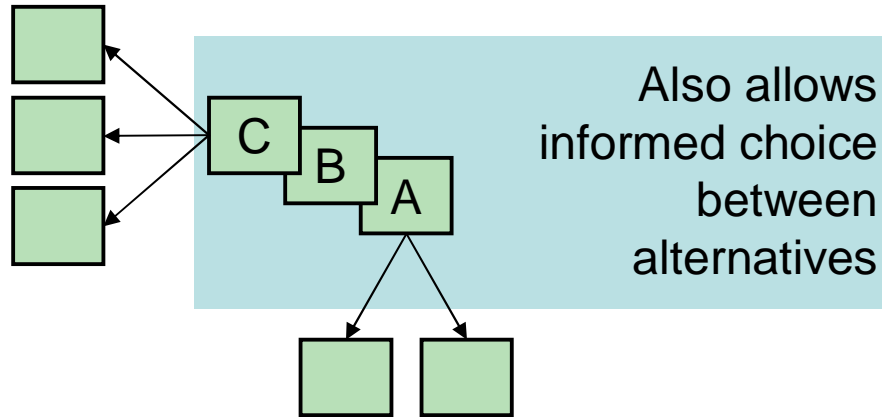
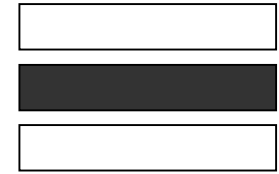




Selecting Alternatives

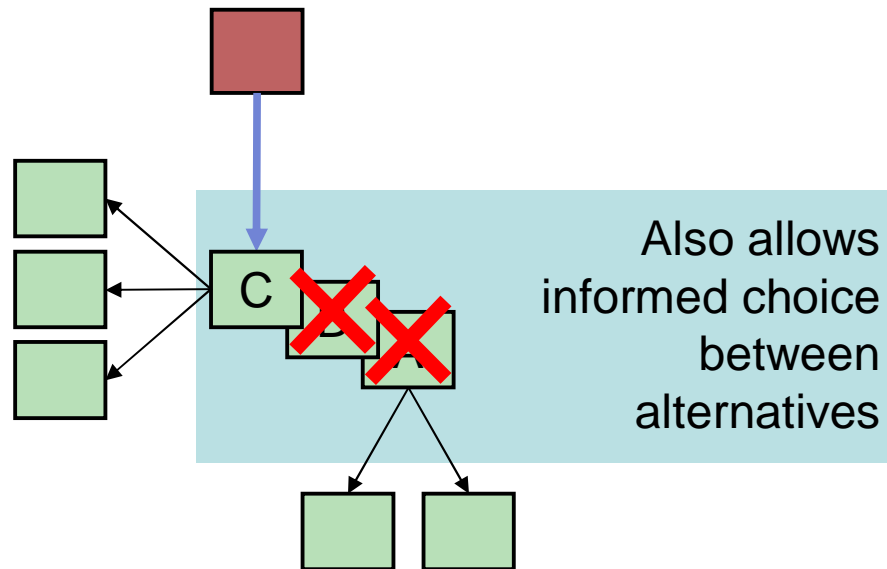


Exploiting NF Preferences



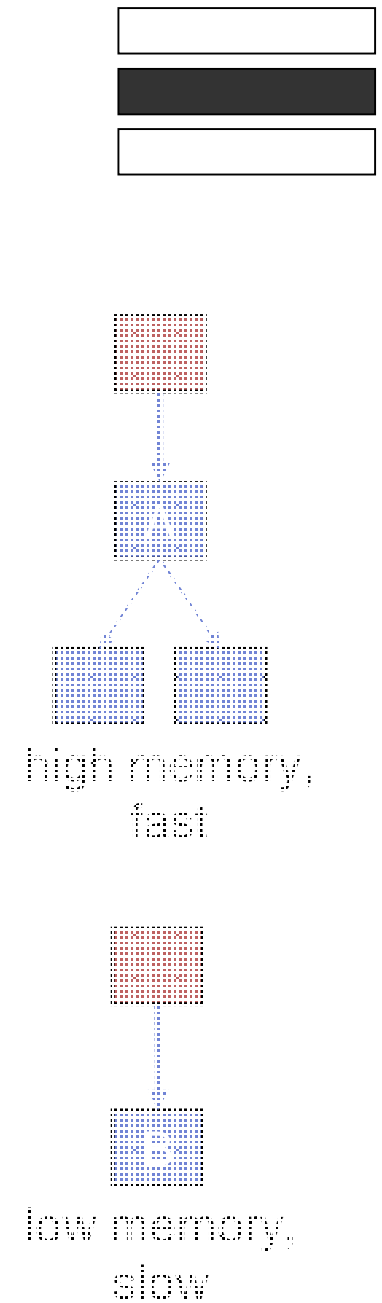
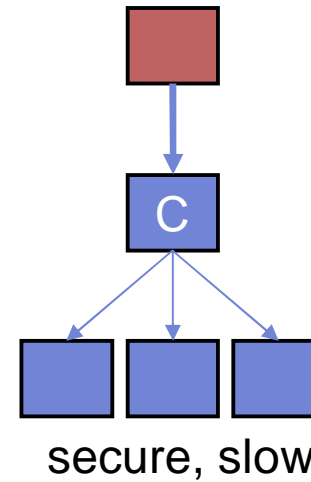
A, B and C provide same functionality.
Non-functional system requirement: **secure**

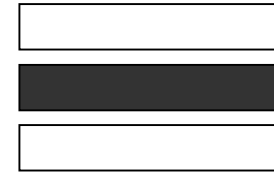
Exploiting NF Preferences



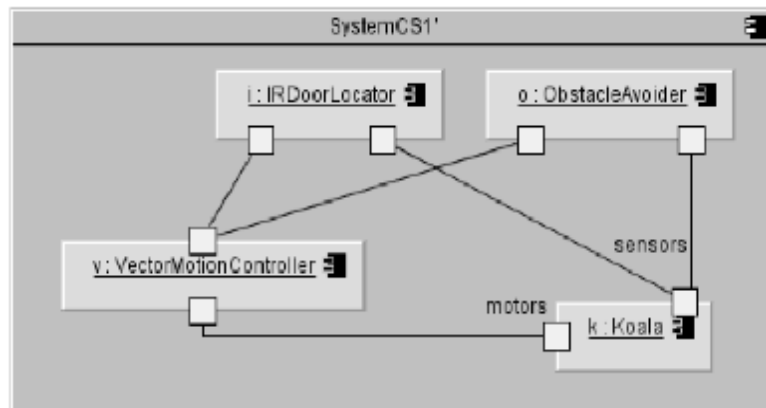
A, B and C provide same functionality.
Non-functional system requirement: **secure**

Configuration C selected



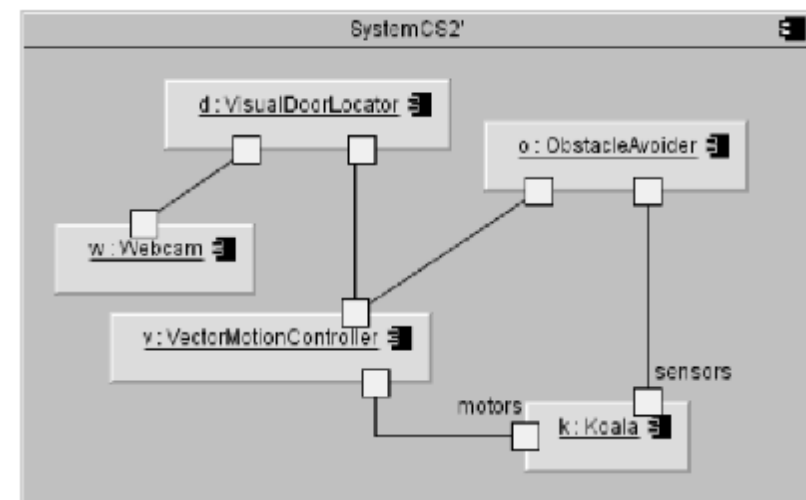


Scenario Example

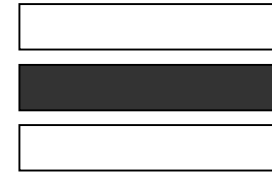


c_{s1}'

vs.



c_{s2}'

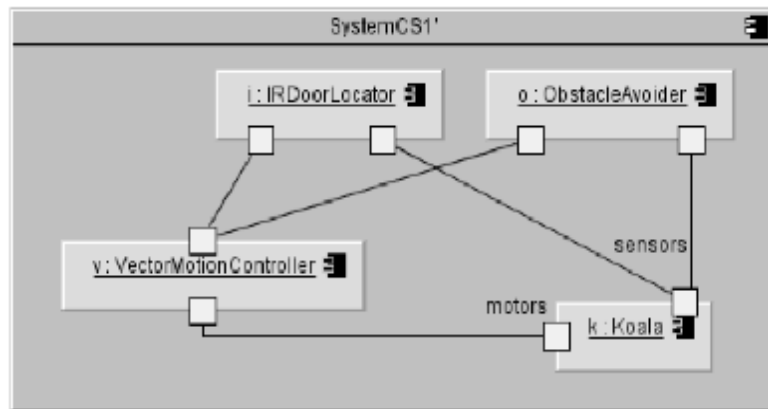


Scenario Example

- Functionally equivalent components may be distinguished by their respective **utility**:

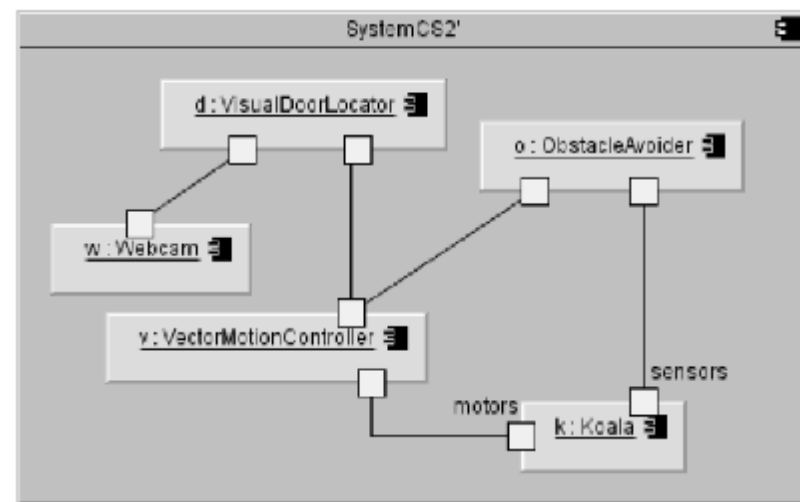
$$U(c) = \sum_{p \in NFPprop} w_p \times u_p(c.p)$$

- Assume we want low power consumption ...

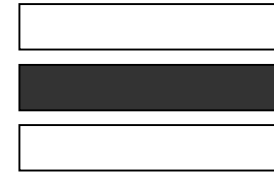


c_{s1}'

vs.



c_{s2}'



Scenario Example

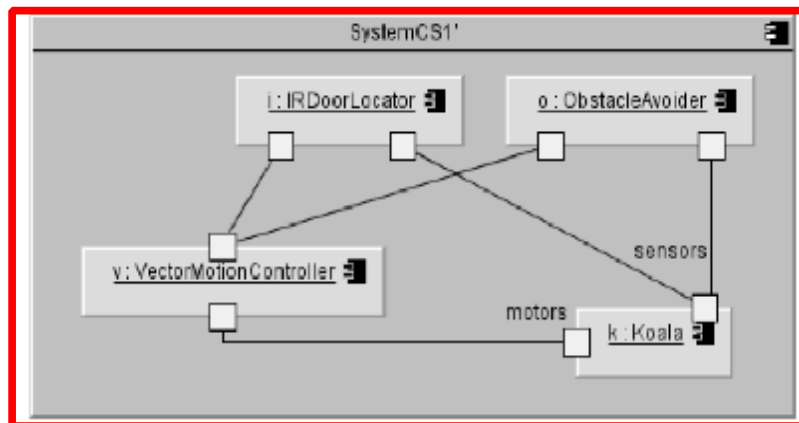
$$U(\text{Webcam}) = 0.8 \times u_{\text{power}}(800\text{mA}) + 0.2 \times 1 = 0.36$$

$$U(\text{VisualDoorLocator}) = 0.8 \times 1 + 0.2 \times u_{\text{reliability}}(\text{high}) = 0.98$$

$$U(c'_{s2}) = \frac{U(\text{Webcam}) + U(\text{VisualDoorLocator}) + 3}{5} = 0.868 \quad \text{X}$$

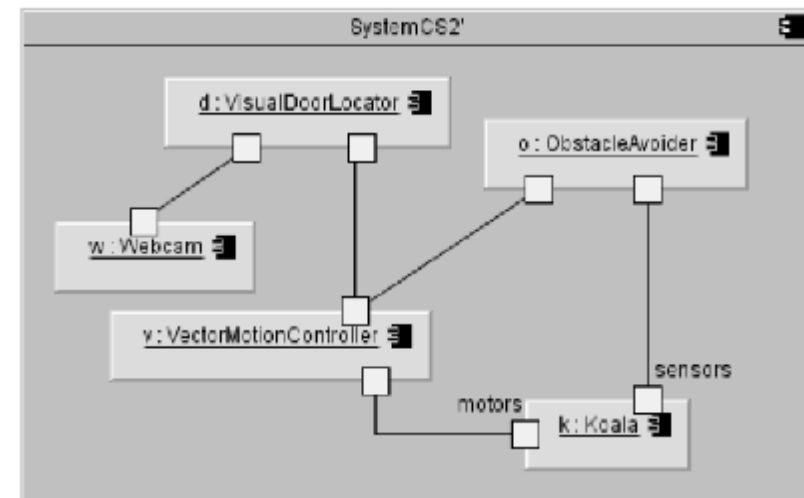
$$U(\text{IRDoorLocator}) = 0.8 \times 1 + 0.2 \times u_{\text{reliability}}(\text{low}) = 0.82$$

$$U(c'_{s1}) = \frac{U(\text{IRDoorLocator}) + 3}{4} = 0.995$$

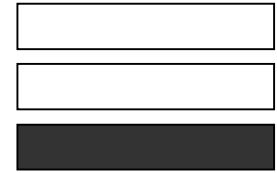


c'_{s1}

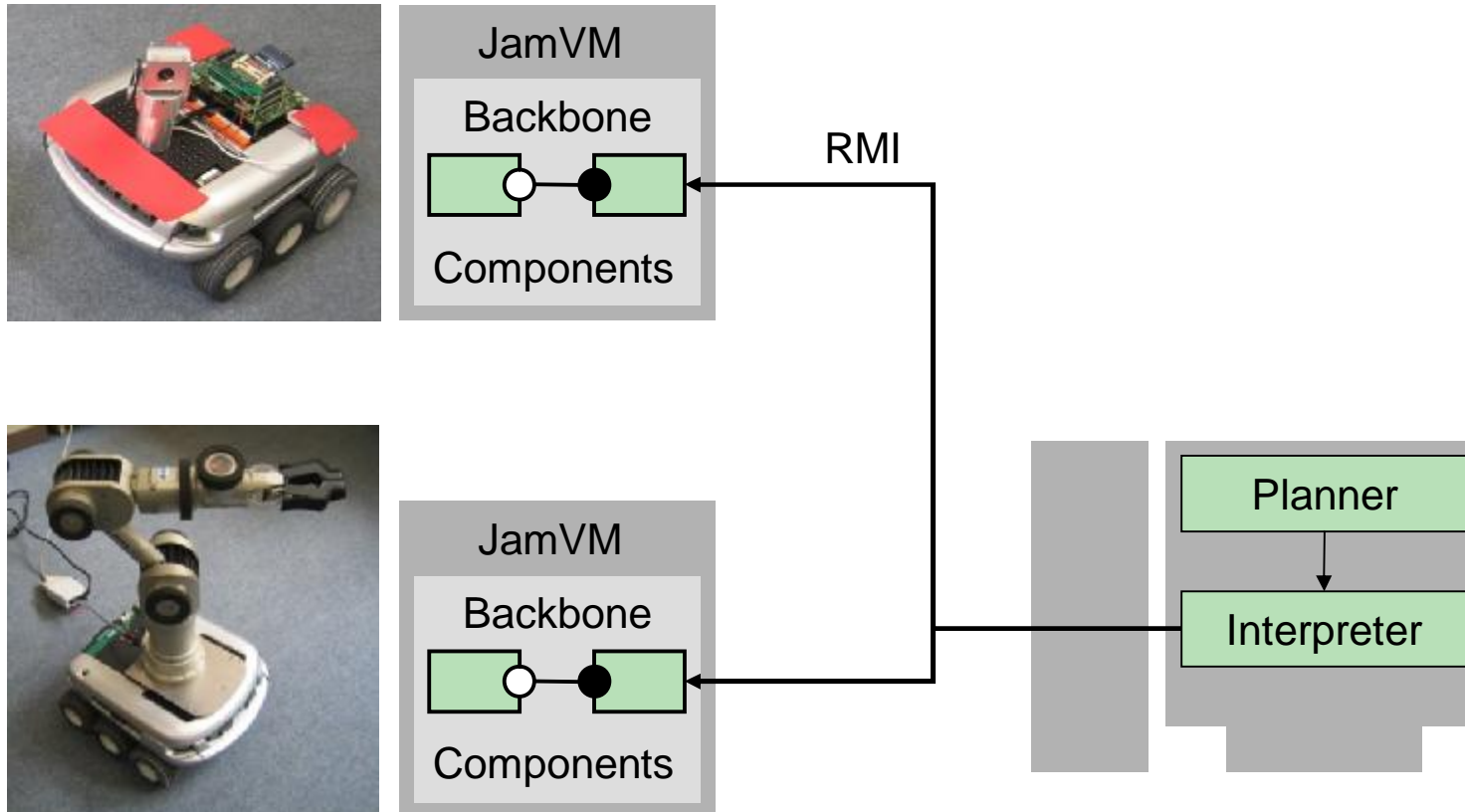
vs.



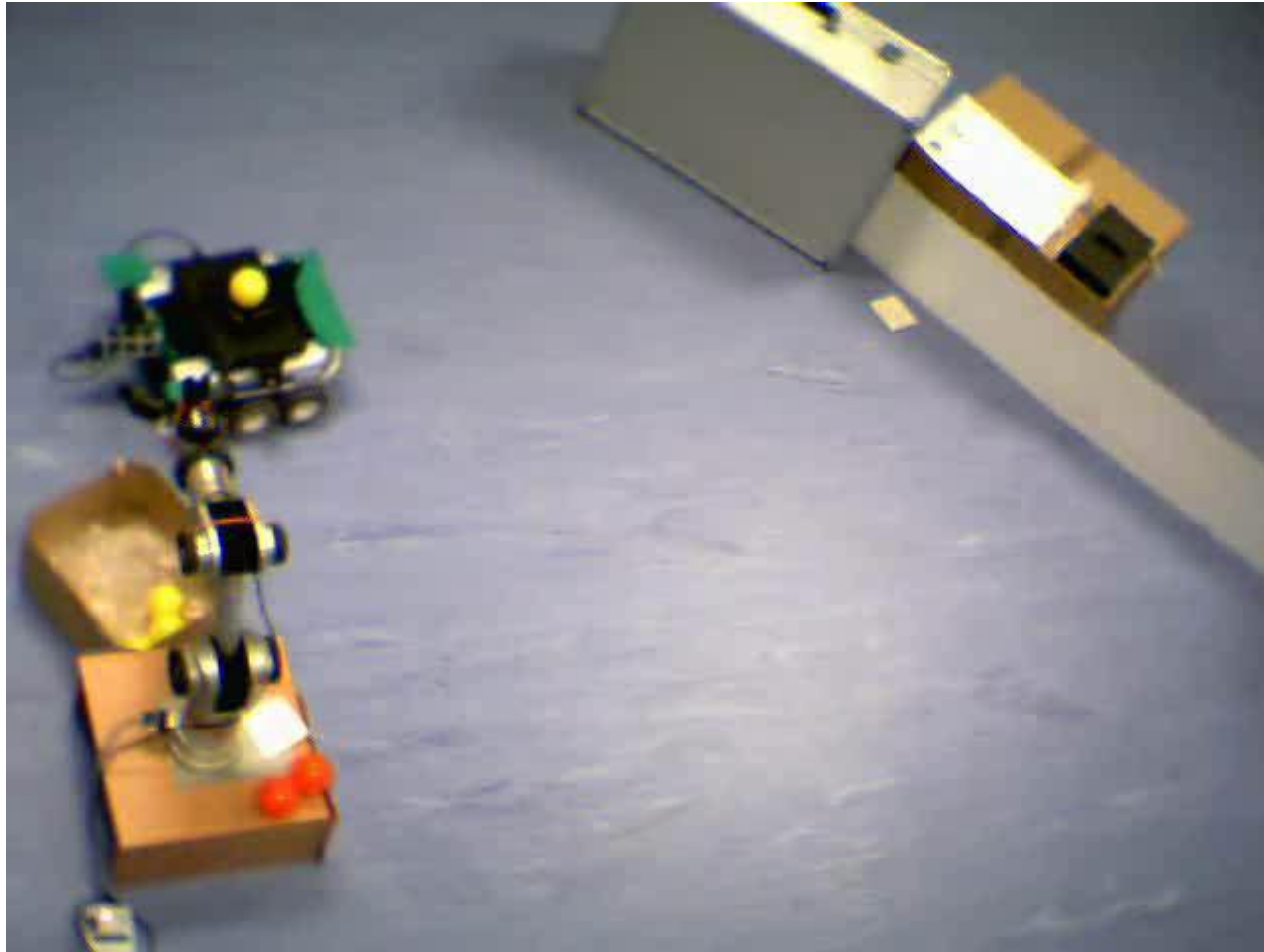
c'_{s2}



Our Testbed Control Layer



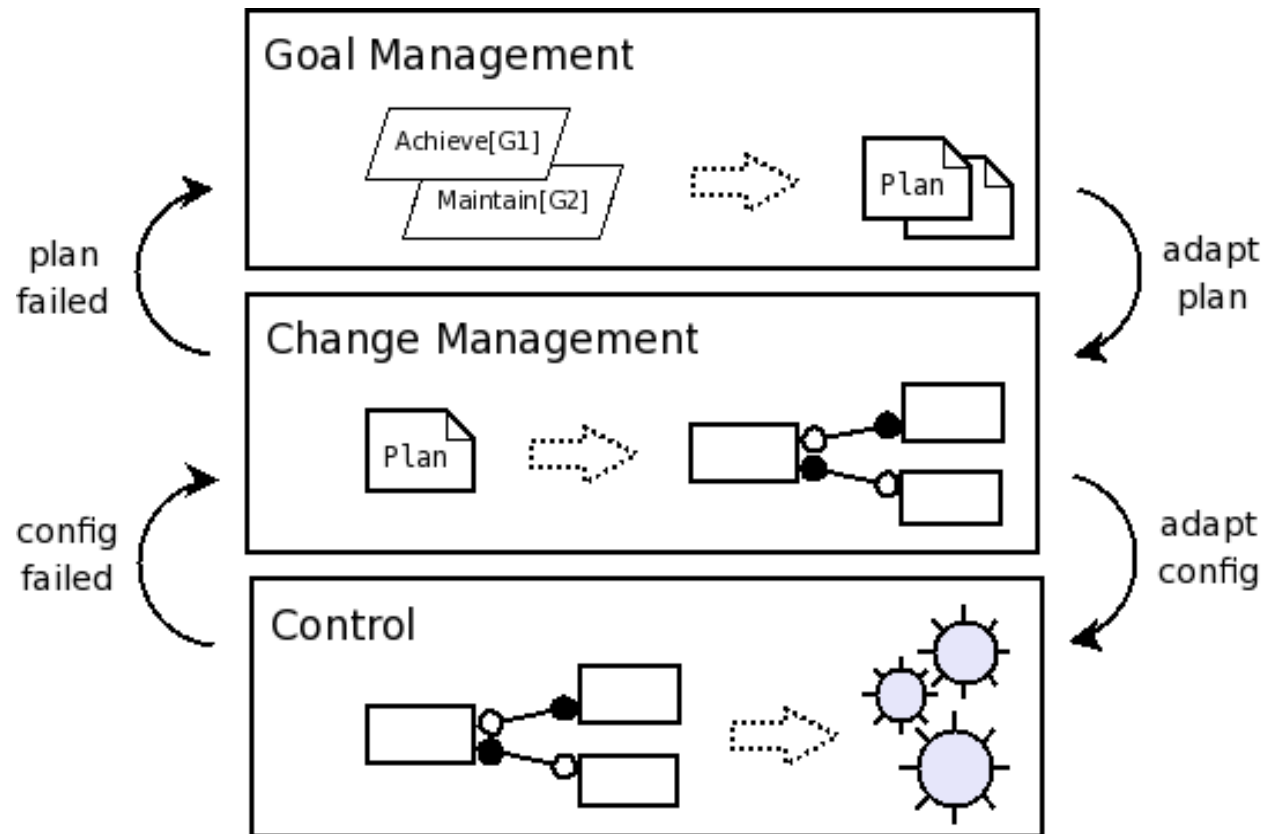
Demonstration



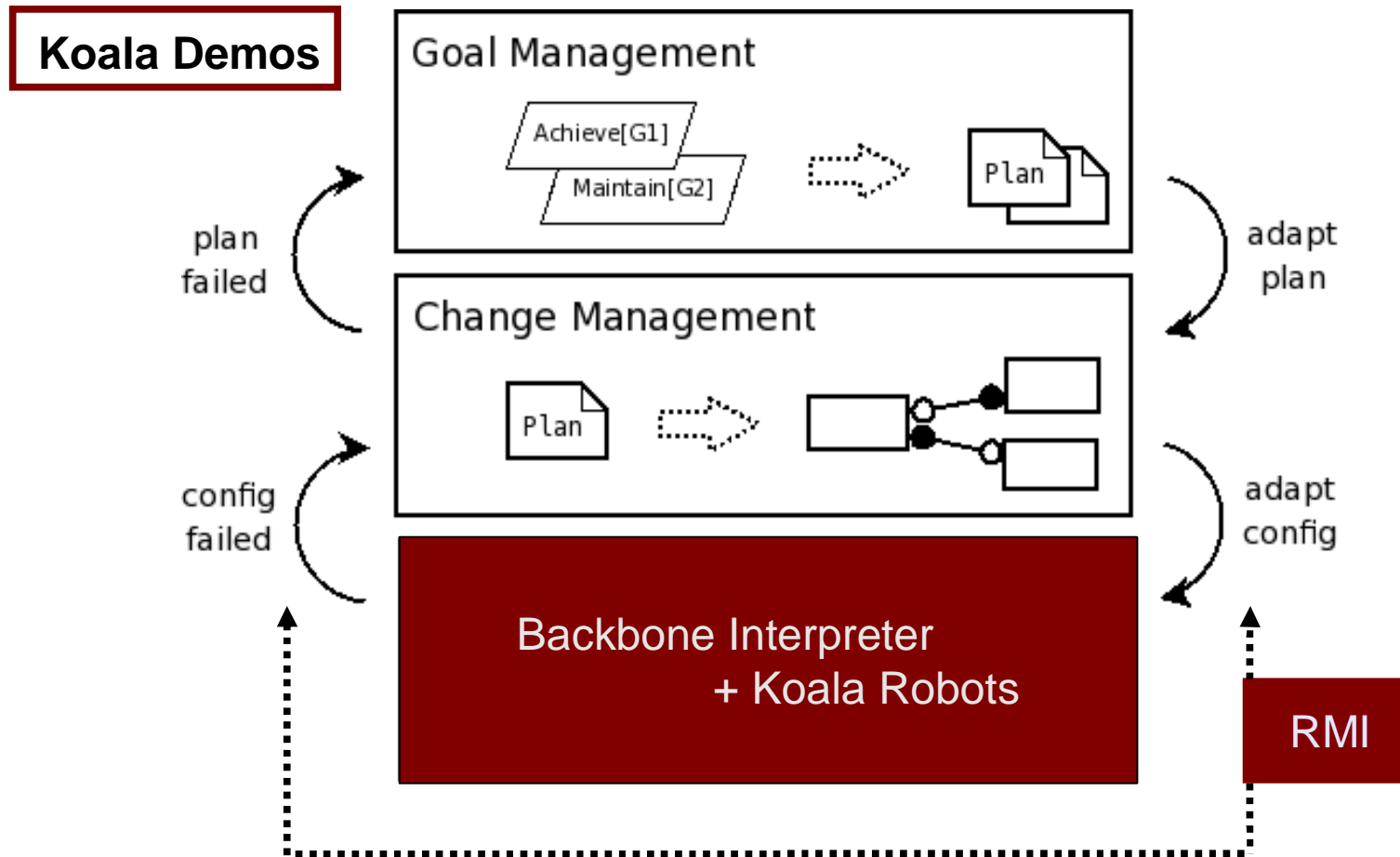
BAE Systems Feasibility Study



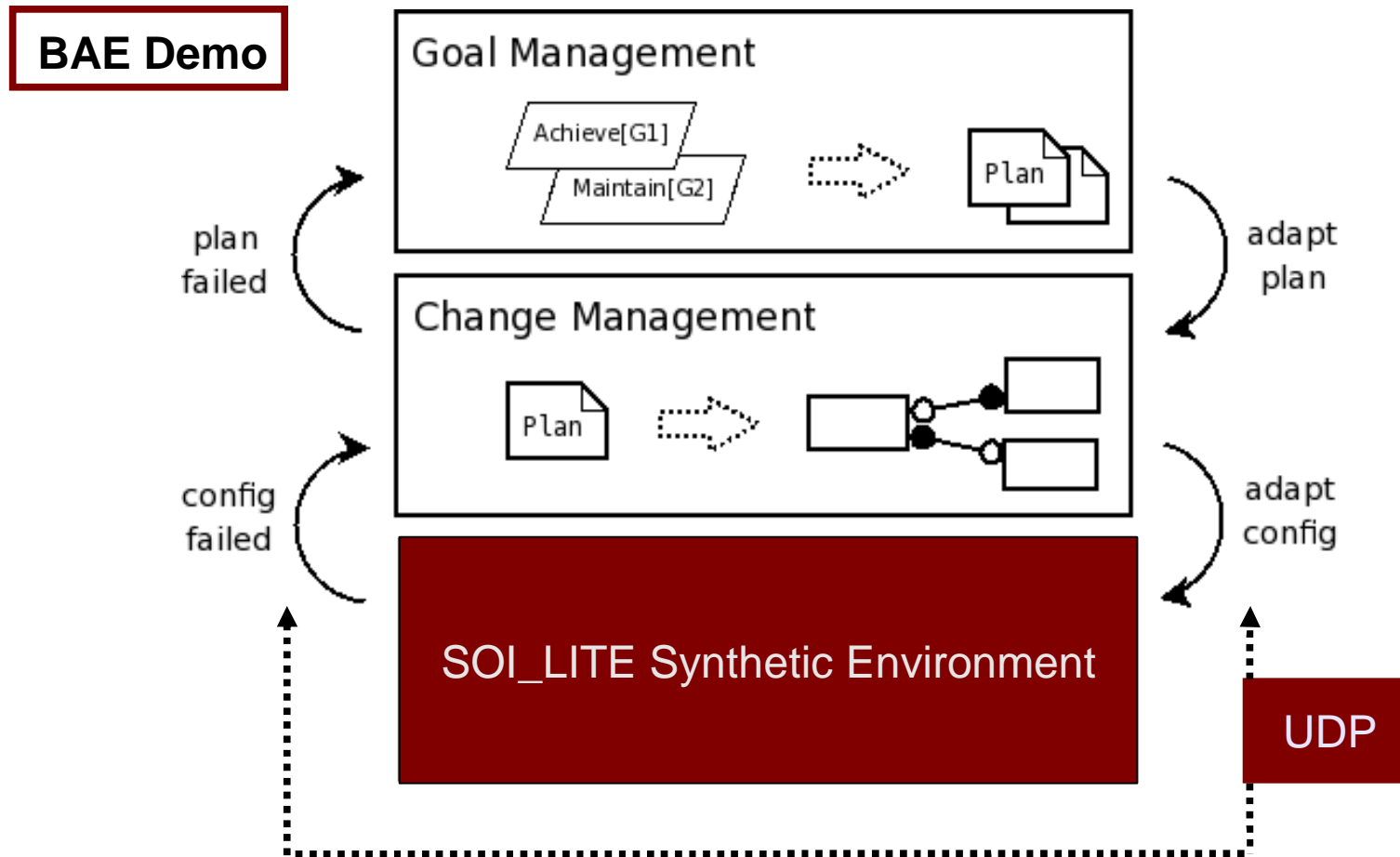
Reapplying the Architecture



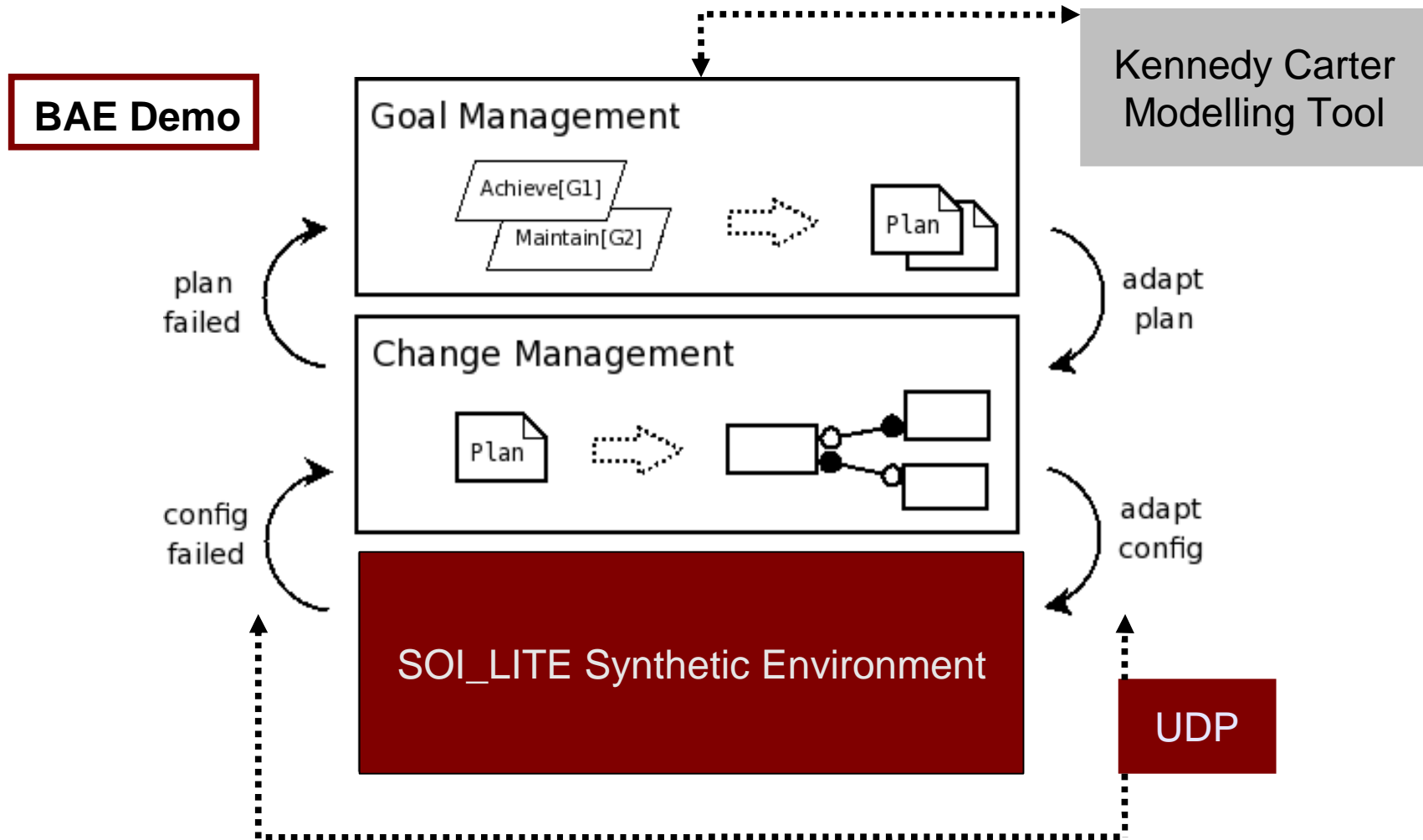
Reapplying the Architecture



Reapplying the Architecture



Reapplying the Architecture



Summary

- As the bedrock of a reliable autonomous system, our architecture supports
 - Automated generation of reactive plans, providing robustness in non-deterministic domains
 - Automated selection of alternative software configurations at runtime to cope with system failures, changes in the world, and changes to operator goals
 - Automated replanning when the current reactive plan and available software configurations are insufficient for achieving a goal

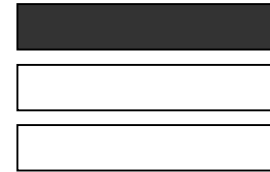
What's New?

- Improved controller synthesis algorithm handles both **safety** and **liveness** goals
- Exploitation of **non-functional** preferences for “smart” dynamic construction of software architectures
- Integration of SER001 with SEN001 through collaborative feasibility study with BAE and KC

Future Work

- Key areas still to work on include
 - Richer automated updating of the domain following an unexpected change in the world
 - Guaranteeing smooth and safe transitions between configurations, preserving state and avoiding inconsistency
 - Co-operation vs. decomposition for multi-agent, concurrent, distributed systems
 - Simulations in more diverse environments
 - Optimising adaptation using learning

Thank you.



Modelling the Domain

