Linking Heterogeneous Models for Intelligent-Cyber-Physical Systems

Zhiming Liu

Centre for Research & Innovation in Software Engineering (RISE)

Southwest University, Chongqing, China

zhimingliu88@swu.edu.cn

http://computer.swu.edu.cn/zhimingliu/

@,Dagstuhl Seminar, Germany. November 12-15, 2017, Germany

Model, Thing & Theory of Models

- Modelling: The model, thing being modelled, modelling paradigm
- Modelling theory: support abstraction & refinement, rigorous analysis, verification and manipulation of models
- The thing are often modelled from different viewpoints
  - Orthogonal or but often interrelated
  - Different viewpoints maybe modelled different with different notations and paradigms
  - Models of different views of the thing need to be integrated to form a "whole" model of the thing
- Modelling Languages: describe models
- Specification Languages: describe properties of models
- **SL and ML can be the same, e.g. TLA**

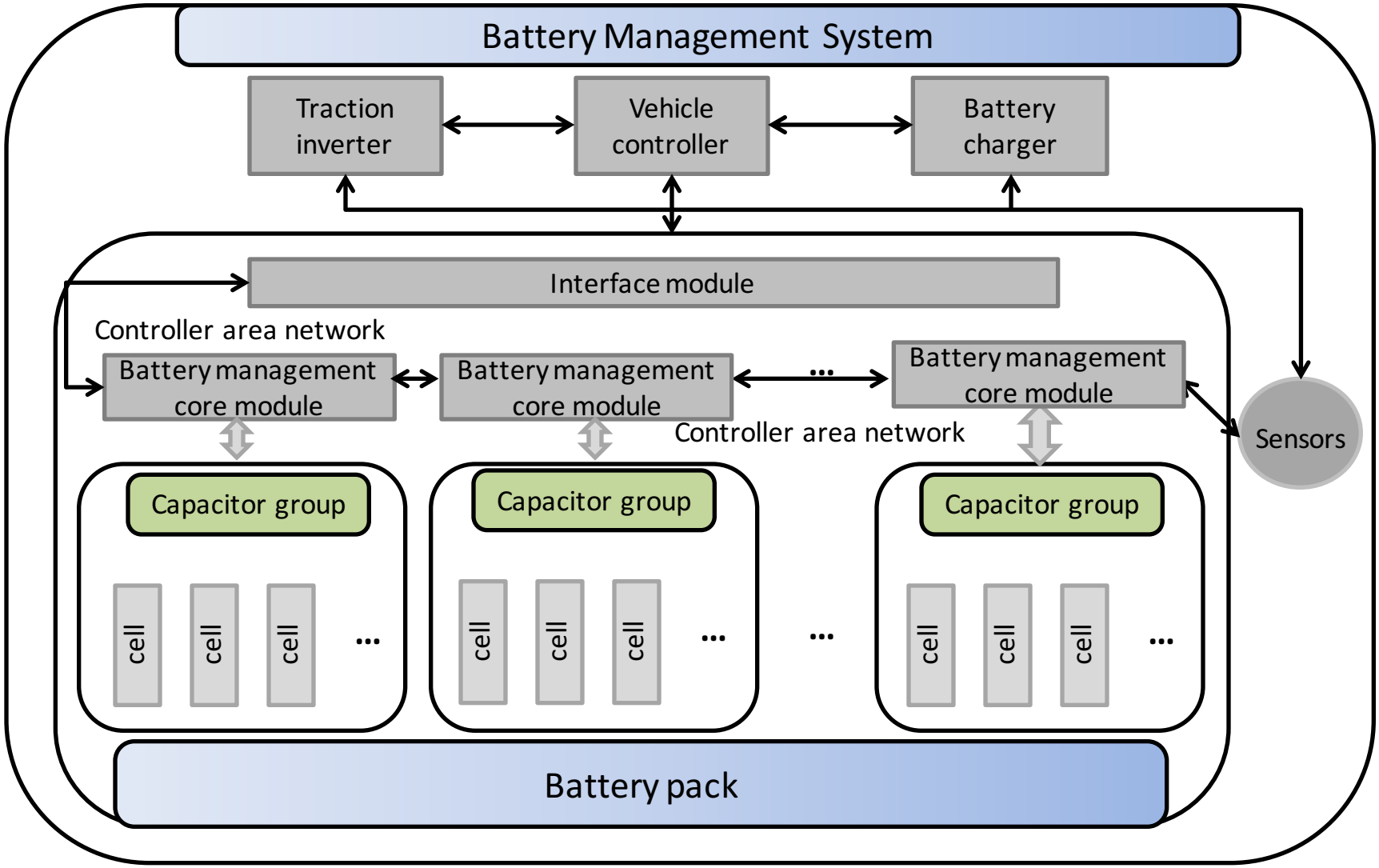
Models for Separation of Concerns

- Event-Based – Trace Models:
  - CCS and CPS-like languages abstract data away
  - Automata-based models (I/O automata)
- Data State-Based with an Operational semantic Model
  - TLA, Action Systems, B , Alloy - State Transition Systems
  - They are also declarative
- Declarative and State-Based: VDM, Z, JML (Hoare-Logic Based)
- Combination:
  - Value-passing CCS and CSP (limited)
  - Occam
  - Labelled Transition Systems
- Multi-view modelling: UML-like, Simulink, rCOS
- Timing model for scheduling

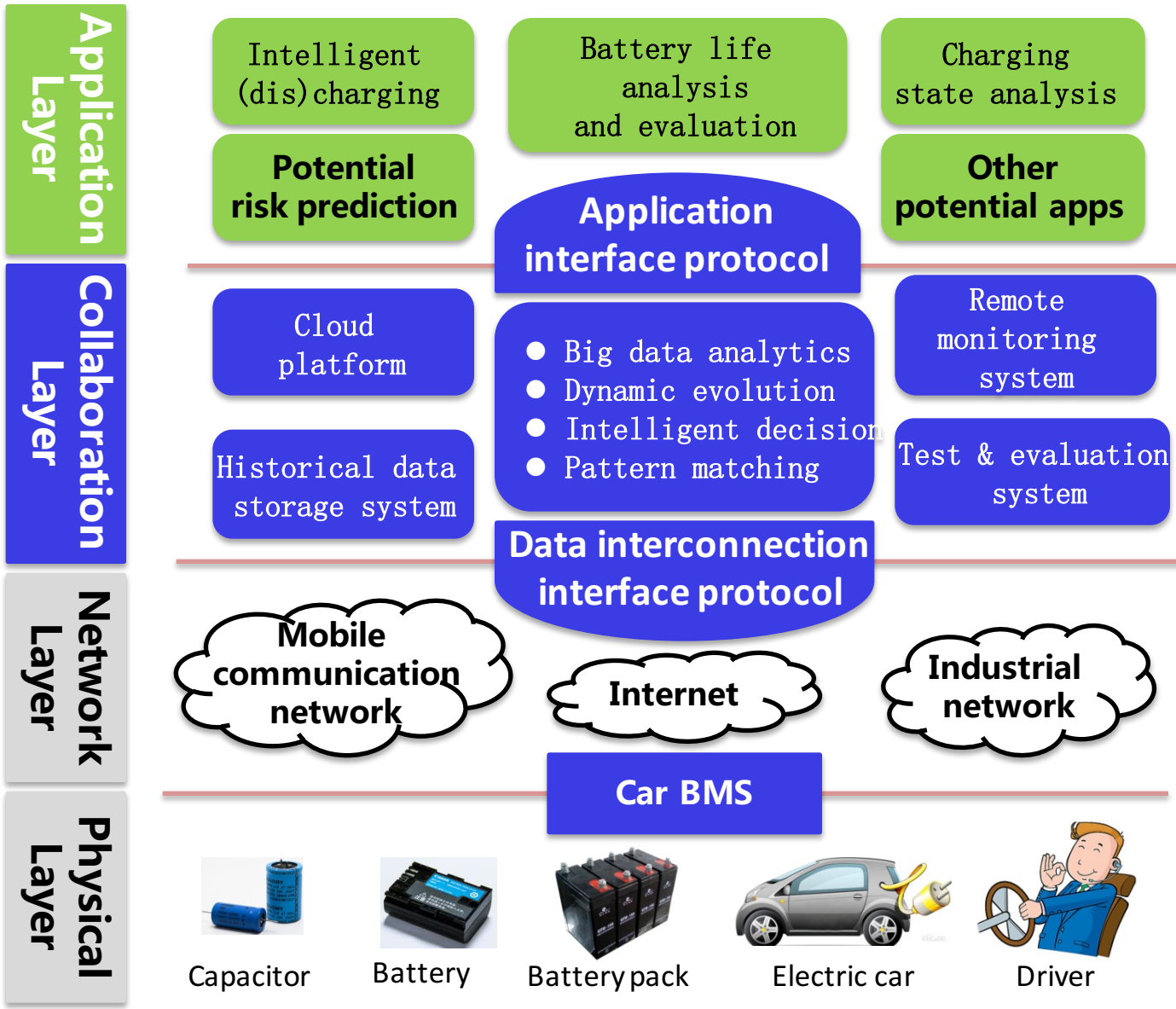
CPS

- A CPS combines a cyber side with a physical side.
- Cyber side – computing and networking
- Physical side – mechanical, electrical, and chemical processes
- The cyber components control the physical side using sensors and actuators, as well as providing services to users
- Sensors and actuators are interfaces that observe/sense the physical system and actuate the controls
- Human actors – involve human-machine interaction
- CPS include IoT, Data Centres and M2M

BMS as CPS


Battery Management System
Traction inverter
Vehicle controller
Battery charger
Interface module
Controller area network
Battery management core module
Battery management core module
...
Battery management core module
Controller area network
Sensors
Capacitor group
Capacitor group
Capacitor group
cell
cell
cell
...
cell
cell
cell
...
...
cell
cell
cell
...
Battery pack

CPS Based BMS



Solution:

- Refer to experience
- Use CPS architecture
- Unified standard management
- employ data from multidimensional

Architecture :

- Multilevel CPS
- Dynamic evolution architecture
- Adaptive components
- Interface based interconnection

BigData- Where from, What for & Why 4Vs

- Through sensors, "things" are virtualized as data in order to be identified and shared/transmitted
- Dynamically used in and drive the evolution of CPS, as well as processed as historic data
- The value of data as virtualisation of things is to create views from separate data sources
- Views implemented as APIs, and further linked to realise business processes and workflows
- BigData are an integrate part of CPS and used to develop value added services and drive system's evolving behavior

BigData are an Integrated Part of CPS

Software (Paradigms) in CPS

- IoT Layer - Embedded Software, Drivers and OS in sensors and devices
- Computation & Coordination Layer - bigdata processing and analytics, databases, control/monitoring/coordinating software
- Application Layer - Apps、web/cloud services、business & workflow managements
- Network layer - communication protocols、network infrastructure & resources management and scheduling (SDN)
- Involve different software architecture styles and technologies: OOA, SOA, CBA, MDA, AI

Need seamless integration of architecture styles and technologies

AI in CPS

- Knowledge reasoning, as well as data analytics, is involved in
  create views – smart service
  decision making in business processes and workflows
  intelligent control – monitor, analyse, plan and execute
  dynamic service discovery and binding – Evolving SOA.
- Computer vision and Natural Language Processing in human-machine interaction (HMI), and
  removing the barrier to software requirements elicitation, analysis, formalisation, and prototyping.
- Robots are obviously in CPS.

Evolving Architecture

CPSs cannot be built from scratch, but they are ever evolving

1. Develop new components and plug into the system
2. Dynamically find and connect components
3. Adding more interfaces and/or improving performance of interface, as to allow cyber components to
   1) sense more and better about its environment
   2) make more intelligent control decisions, and provide smarter services
   3) control and coordinate more and better physical components
4. With 1&2 to "connect" what were originally separated components to allows them to interact, collaborate and coordinate
5. With more and better connectors, coordinators, interfaces to improve trustworthiness

Challenges

- Requirements
  - Cope with changes and uncertainties: both during development and at runtime
  - The relation between performance and functionalities
- Handling evolving architecture with heterogeneous components of mixed criticality
  - Dynamic component plug in and play (composability)
  - dynamic discovery and binding services and components, (evolving SOA)
  - Predictability, safety, security, robust, self-organising , self-adaptive.
- **Verification of AI Components and Composition of non-learning and learning components**

A Proposed Position

- Taking an existing CPS ecosystem as the infrastructure, develop and integrate CPS components and Services
- A combined component-based and service oriented design and evolution
  - System architecture is horizontally component-based and vertically service-oriented
  - Develop new layer of services, monitoring and control (SOA)
  - Refine/evolve existing layer of coordination, monitoring and control
- Combining different modelling paradigms (for different concerns of design and verification), including AI paradigms

Objectives/Vision

- **A model-driven approach to Contract-Based CPS Component Design, integration and Evolution**
  1) Seamlessly and coherently combination of the various dimensions of the multi-scale design space - behavior, QoS, space and time
  2) Unifying semantic theories of different modelling paradigms
  3) Linked techniques and tool support for integration different development paradigms and heterogeneous artifacts.
- **Provide correct, secure and intelligence and healthy evolution by architectural design - a theory of generic refinement**
  - **in which integration verification and simulation techniques are driven by construction**

Contracts-Based Models/Specifications

- A contract is a very general notion in many disciplines.
- For CPS design, a contract C is given by a pair of properties

$$C = \mathcal{A} \text{ |-- } \mathcal{G}$$

$\mathcal{A}$: assumptions on the environment, and

$\mathcal{G}$: the promise of the component under these assumptions.

- The model of contracts is general for functions, interactions, and QoS, i.e. multi-concerns.

Contract-Based Model Supports CBD and SOA

- Use as much as possible elements from **available components and services**
- Component/service composition $C = \mathcal{A} \mid\!\!- \mathcal{G}$ of $C_1 = \mathcal{A}_1 \mid\!\!- \mathcal{G}_1$ and $C_2 = \mathcal{A}_2 \mid\!\!- \mathcal{G}_2$

$$\mathcal{A} = (\mathcal{A}_1 \wedge \mathcal{A}_2) \vee \neg(\mathcal{G}_1 \wedge \mathcal{G}_2), \qquad \mathcal{G} = \mathcal{G}_1 \wedge \mathcal{G}_2$$

- Key challenge in CPS is to develop a model of contracts to
  - model interactions between physical and components
  - mix different physical systems, control logic, and implementation architectures

Support MBD

- The main philosophy of MBD is integration virtualisation, and contract composition is model composition.
- Support correctness preservation model transformations, by **refinement**

$C_1 \sqsubseteq C_2$ if $\mathcal{A}_1 <== \mathcal{A}_2$ and $\mathcal{G}_1 ==> \mathcal{G}_2$

- Support **layered design**, correct realisation of contract or a component in higher layer by assembling components in a lower layer
- **Theorem:** $(Contracts, \sqsubseteq)$ forms a partial order, the lower and upper bounds are

$C_1 \sqcap C_2 = \mathcal{A}_1 \vee \mathcal{A}_2 \ |\!-\!- \ G_1 \wedge G_2 \quad C_1 [] C_2 = \mathcal{A}_1 \wedge \mathcal{A}_2 \ |\!-\!- \ \mathcal{G}_1 \vee \mathcal{G}_2$

- Separation of Multiple viewpoints, e.g. $C_f \sqcap C_t = A_f \vee A_t \ |\!-\!- \ G_f \wedge G_t$

Well-Known Contract-Based Models

- Imperative Programs
  - Hoare Logic $\{Pre\}\ P\ \{Post\}$
  - UTP $Pre \vdash Post$ [Hoare&He]
- OOP: Meyer's Design by Contracts, rCOS [He, Li &Liu]
- Reactive Systems
  - Jones' theory of Rely-Guarantee
  - Lamport's TLA: $E \Longrightarrow S$
- Component-Based Systems: rCOS [He, Li & Liu]

Model of Architectural Components

- Service component

```
Component M1
   Z d;
      provided interface M1IF {
        W(Z v) { d:=v };  R(;v) { v:=d };
   }
```

- Reactive component

```
Component M
   Z d, Bool w = true;
      provided interface MIF {
        W(Z v) { w&(d:=v,w:= not w };
        R(;Z v) {not w&(v:=d; w:=not w)};
   }
```

Equivalent Models

```
Component M
   Z d;
      provided interface MIF {
        W(Z v) { d:=v };  R(;v) { v:=d };
      protocol {(WR)*+(WR)*W}   // ** generally traces      }

 Component M requires M1  //** M is obtained through coordinating M1
   Bool w = true;
      provided interface MIF {
        W(Z v) { w&(M1.W(v);w:=not w )};
        R(;Z v) {not w&(M1.R(;v);w:=not w };
  }

Component M requires M1 //** M is obtained through coordinating M1
      provided interface MIF {
        W(Z v) {M1.W(v)};  R(;Z v) { M1.W(;v)};
      protocol {(M1.WM1.R)*+(M.1WM.1R)*W}
  }
```

More General Component

```
component fM {
   Z d;
      provided interface MIF {
      W(Z v) { d:=v };
       R(;v) { v:=d };
      protocol { (WR)*+(WR)*W}
      }
     actions { //fault modelling corruption of memory
           fault {true|- d'< > d }
      }
```

- Renaming as a built-in connector,

  fMi=fM[fMi.W/W,fMi.R/R], i=1,2,3,

- But can be built by composition

```
component fMi requires fM
   provided interface MIFi {
      fMi.W(Z v) {fM.W(v)};
      fMi.R(;Z v) {fM.R(;v)}
    }
```

Orchestration

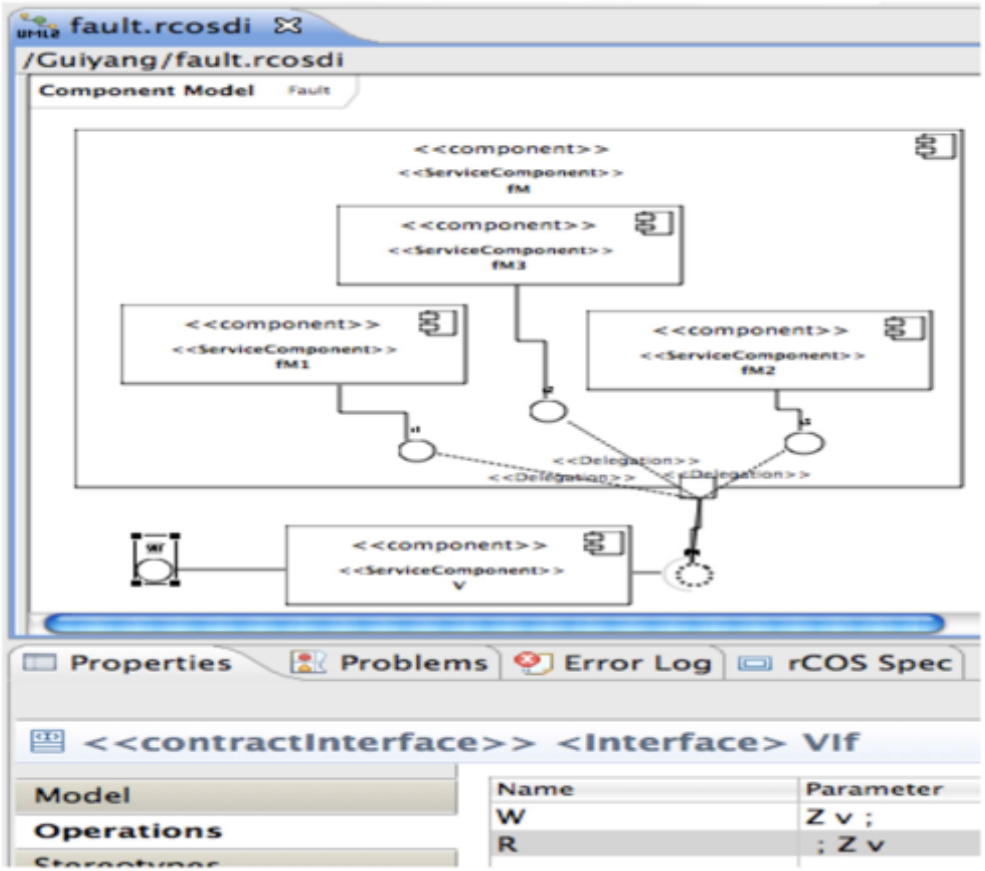
```
component V requires fM1, fM2, fM3
    Z v1,v2,v3,
    provided interface VIF {
        W(Z v) { fM1.W(v)||fM2.W(v)||fM3.W(v) };
        R(;v) { fM1.R(;v1)||fM2.R(;v2)||fM3.R(;v3);v=vote(v1,v2,v3))};
     protocol { W({W,R}) } //*notice one can specify different protocols
    }
   }
```

- **V⊑ M. i.e. V refines M,** provided at most one memory is corrupted

  Verification
- The proof need to introduce auxiliary variables

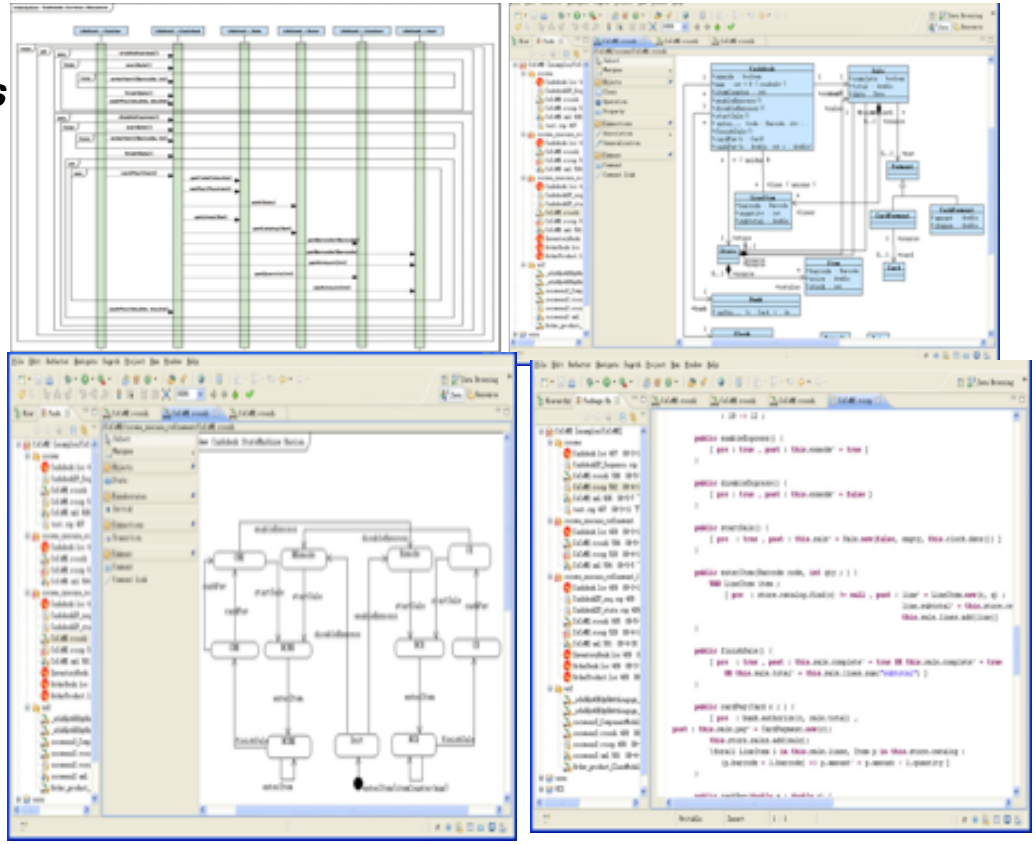
Note three different instances of fM are running in parallel in side these fMi's, this in V

Architecture Decision for Fault-Tolerance

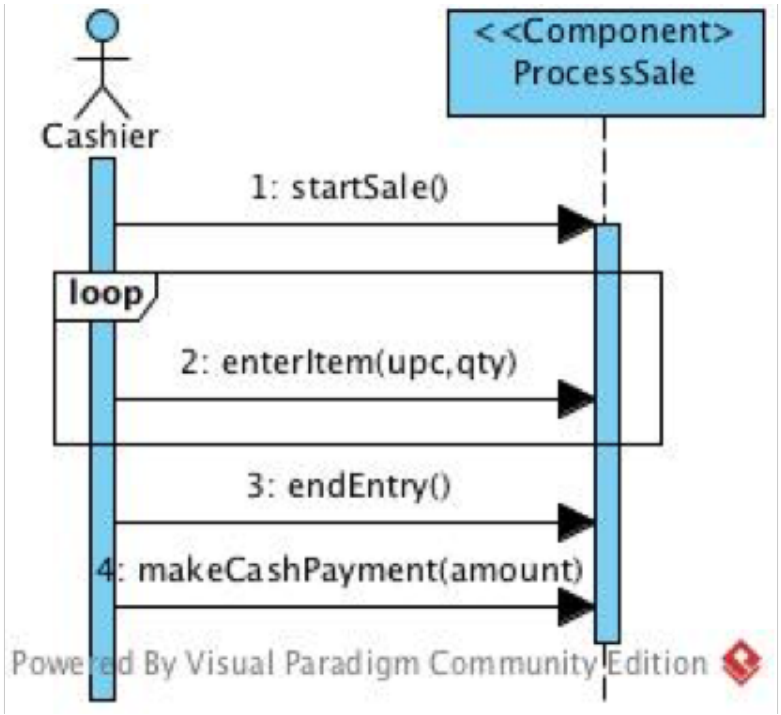


Model-Driven Development with rCOS

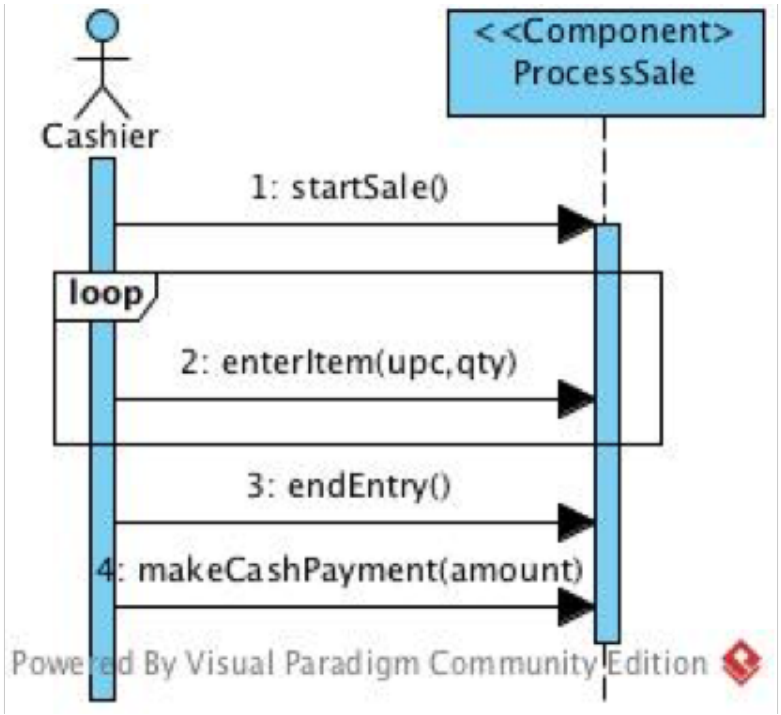
- Each phase is based on the construction of ***verifiable models***
- Models are analysed and verified
- Refined models are constructed by ***model transformations***
- ***Code*** is generated from design models
- ***Proof obligations*** are generated by model transformations
- ***rCOS modeler*** integrates UML model notation into rCOS



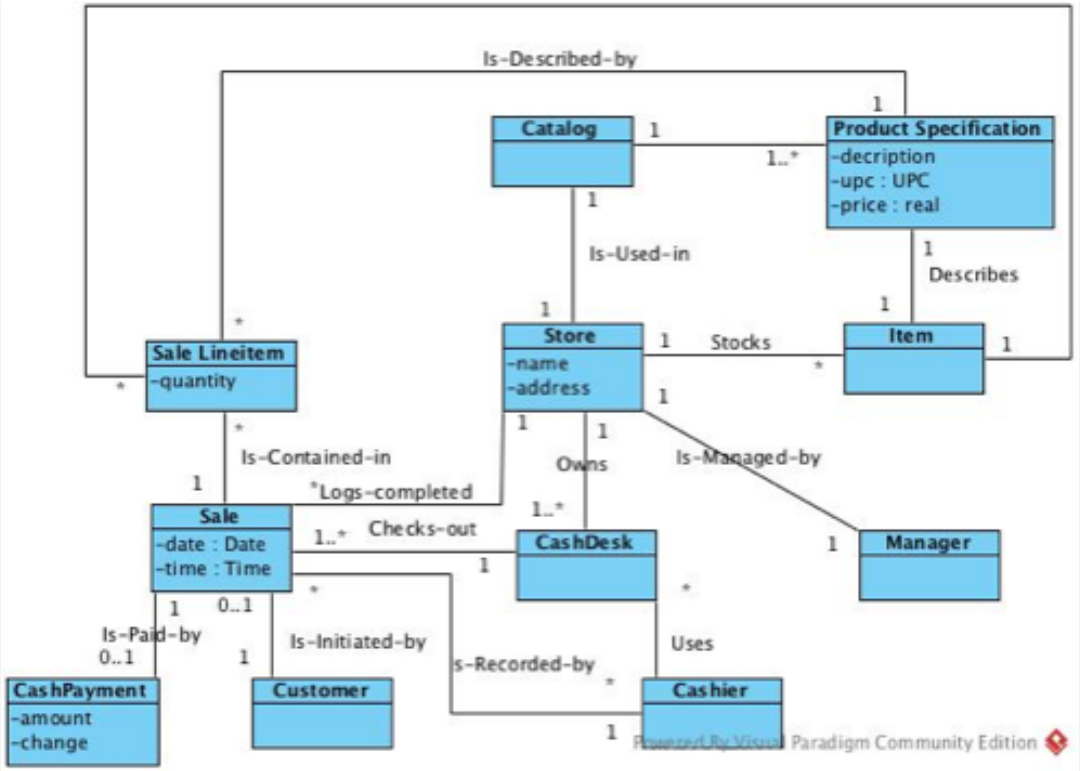
Component Interface Sequence Diagram



Component Interface Sequence Diagram



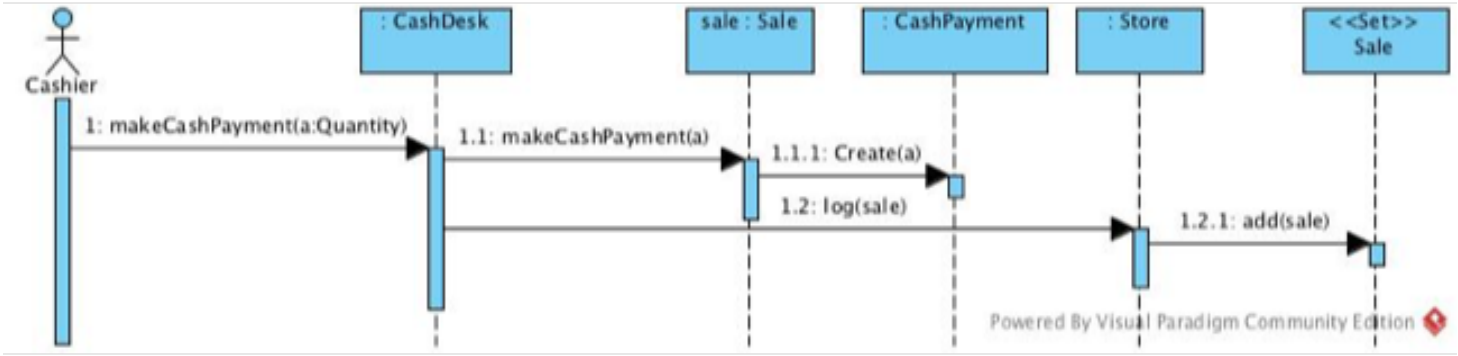
Component Class Diagram



Operation Contract makeCashPayment(a)

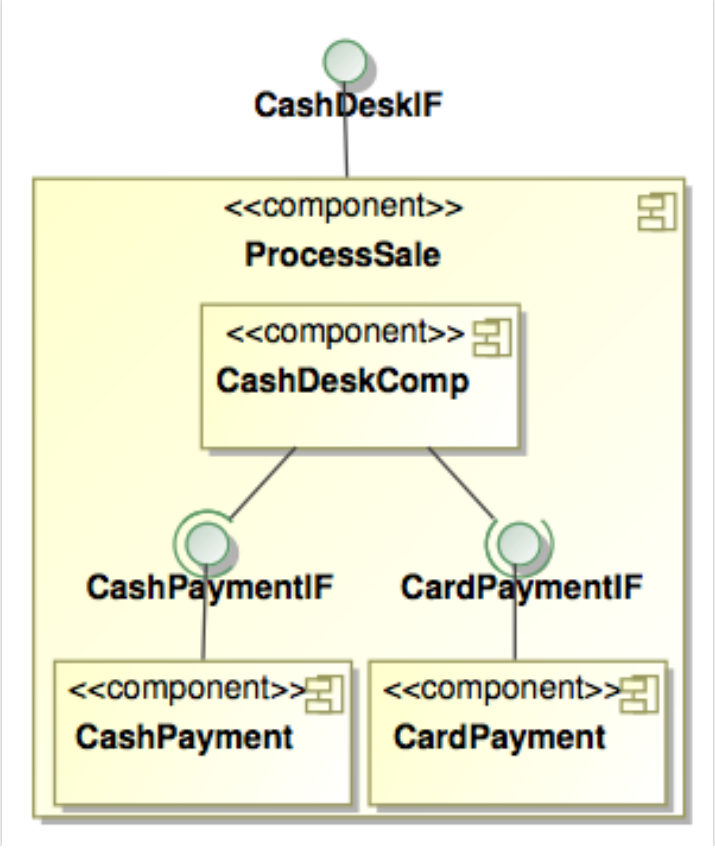
- Preconditions
  1. Store exits
  2. CashDesk exits
  3. Catalog exits
  4. Sale is complete
- Postconditions
  5. a new cashpayment is created
  6. amount of the new cashpayment is set to a
  7. the new cashpayment is linked to the sale

OO Design: makeCashPayment()



OO Design to Component-Based Design

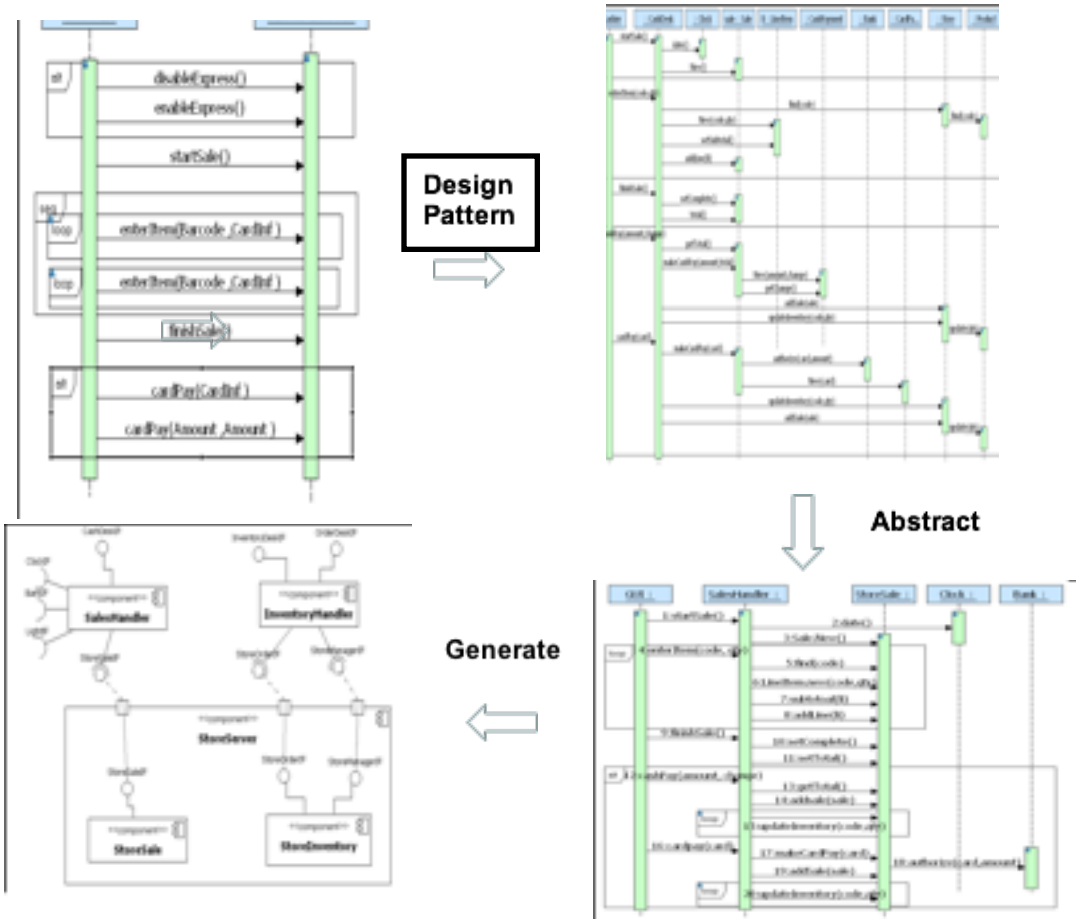
Automatically Generate Component Diagram



rCOS Design Process Summary

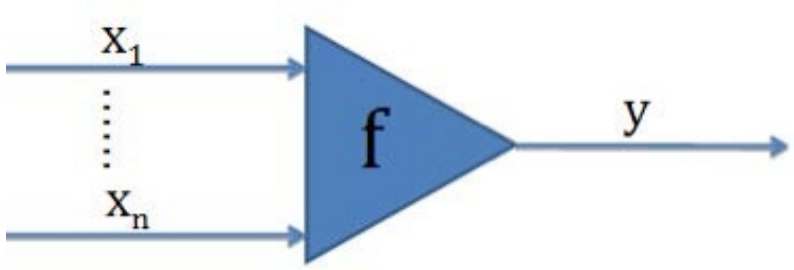
1. use case as component
2. refine use case operations by design patterns to generate an oo interaction model
3. generate design class model
4. transform the oo interaction model to a component interaction model
5. generate the component diagram
6. transform oo interfaces to specific middlewares, e.g. RMI, CORBA etc.
7. Integrates implemented use cases

Code generation and testing after 3 or 6

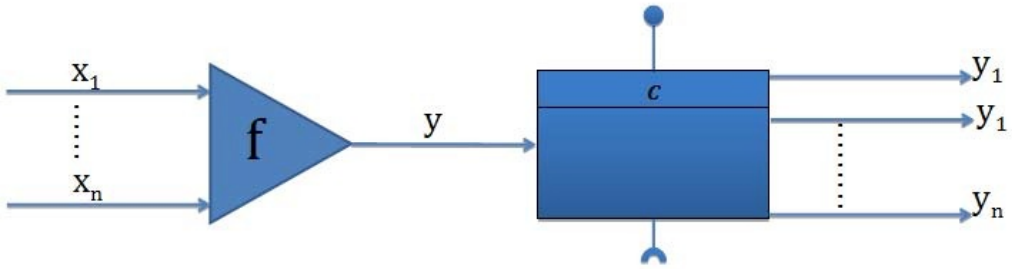


Cyber-Physical Components

- Physical Interface

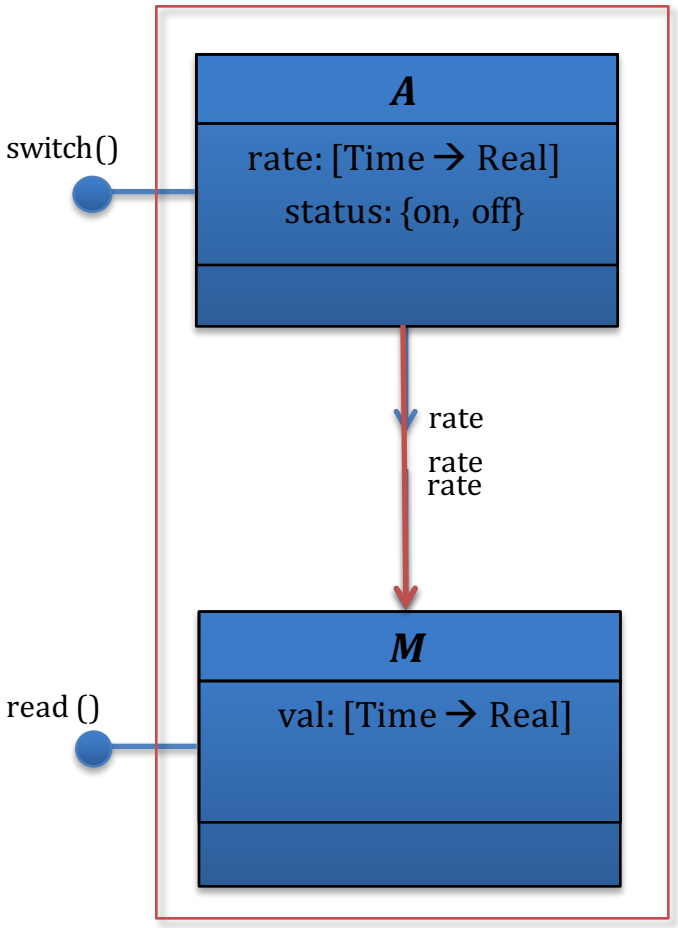


- Cyber-Physical Component



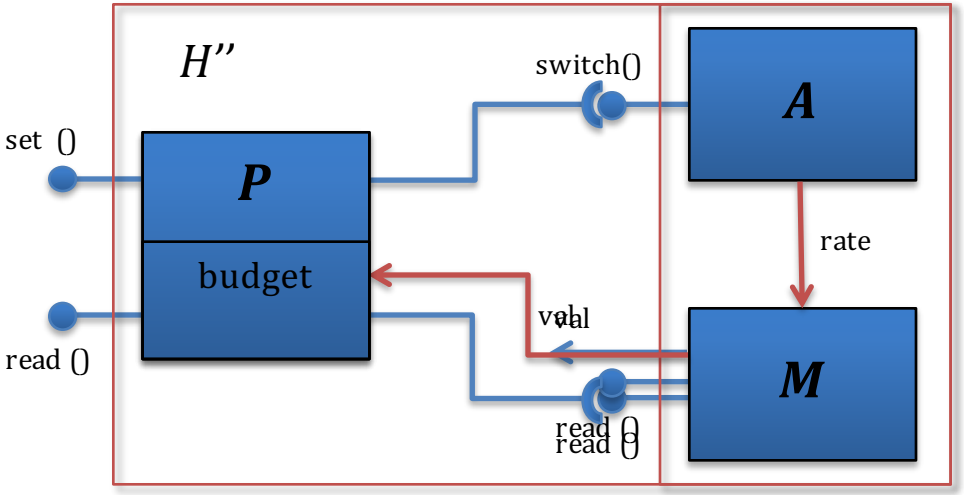
Physical Components and Interfaces

```
Component A{/*an appliance
      rate: [Time → Real];
      status: {on, off};
      provided interface {
        rate {/*signal: given by manufacture};
        switch() {/*operation: switch A on and off}
      }
 }
Component M {/*meter
   val: [time → Real];
   provided interface {
        read(;r){true |- r'=val};
   }
   required interface rate {/*signal
      val= energy(rate)
    }
}
```



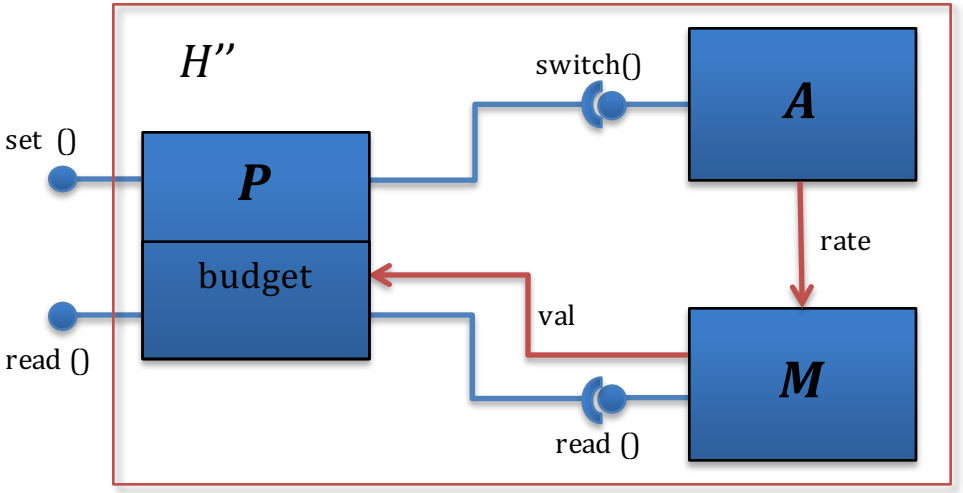
Composition: H = A||M

System Evolution for Home Automation (a)



- Add provided signal 'val' to M
- Add a control pad P that requires signal 'val', provides `set()', and calls A.switch, etc.
  M'=P||M, H'=M'||A
- Refine P with planning with daily budget, and schedule functionality
  M'' = P'||M, H''=M''||A

System Evolution for Home Automation (b)

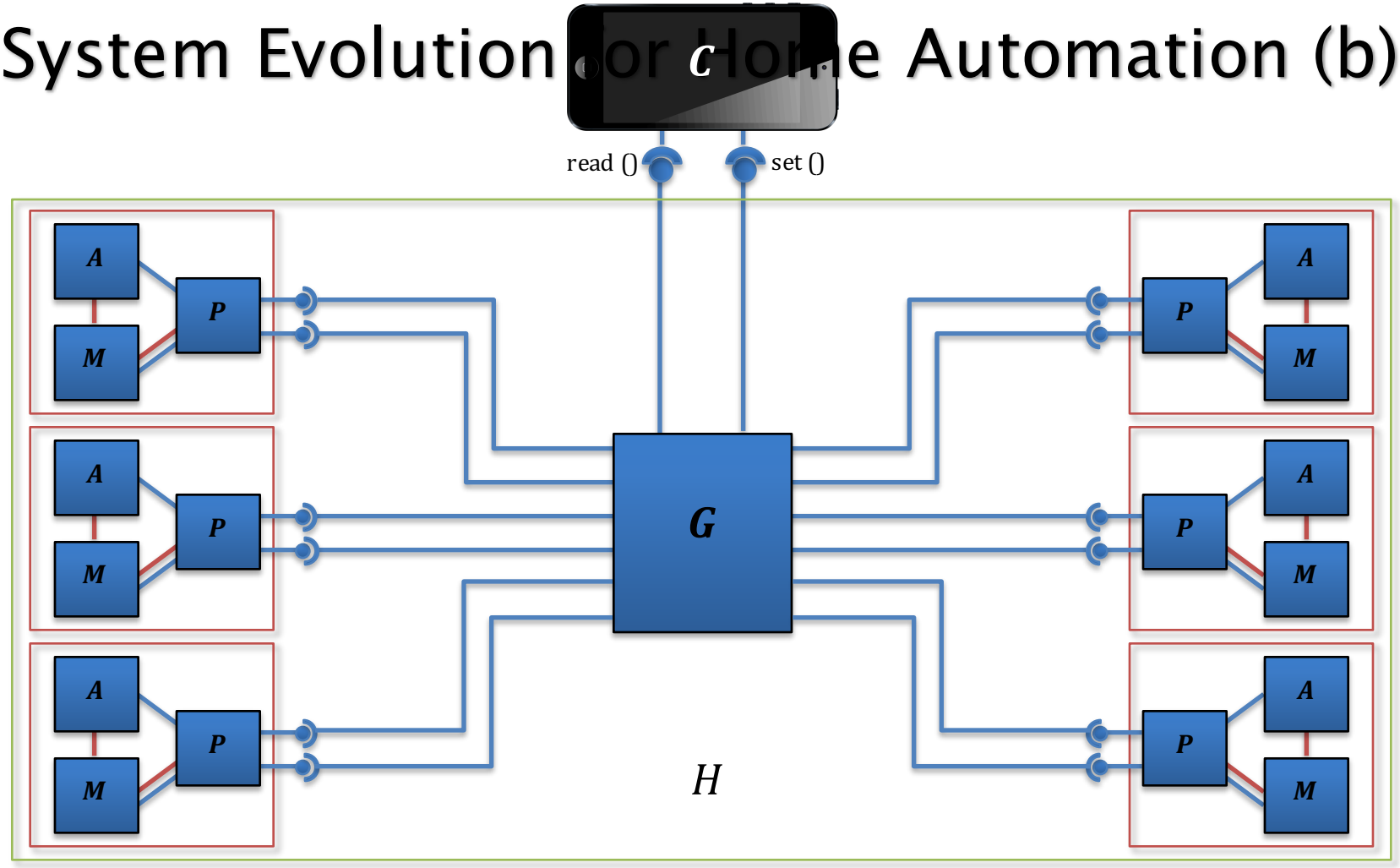


$Ai = A[switch_i/swich, rate_i/rate]$, $\underline{A} = A_1 || \ldots || A_m$

$Mi = M[read_i/read, val_i/val]$, $\underline{M} = M_1 || \ldots || M_m$

$P_i = P[\ldots,\ldots]$, $\underline{P} = P_1 || \ldots || P_m$

System Evolution for Home Automation (b)



- Add a global controller for planning and schedule
  H = G||P||M||A
- Control C with mobile phone from car or office

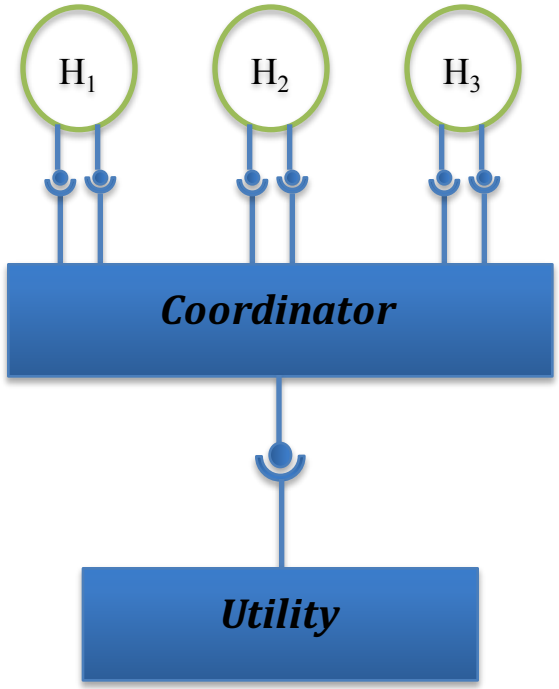
Network Evolution

- Consider k households Hj, each with its own budget

  $\underline{H} = H_1||H_2||....||H_k$
- Consider Coordinator, interacting the households to coordinate their budgets, based on the interaction with of a utility company Utility

  $N = \underline{H}||Coordinator||Utility$
- Evolve to a network $\underline{U}$ of utility companies

  $\underline{N} = \underline{H}||Coordinator||\underline{U}$
- Implement $\underline{H}||Coordinator$ by distributing the coordination activities among the households themselves



Note individual households evolve in parallel, and in parallel with the network with more households and companies

Conclusions

1. CPS as unified view of emerging systems
2. BigData, AI and Learning software, as well as non-AI systems, are all involved
3. Architecture models are essential for
   - correct and secure by design,
   - identification of safety vulnerabilities and security threats, and
   - making architecture decisions for different concerns (aspect-orientation)
4. There is a need of linking of different modelling paradigms for AI systems, Hybrid Systems, Component-Based and Service Based Software Systems.
5. Need to handle and support healthy/safe evolution from multi-dimensions

Indirections

"All Problems in Computer Science can be Solved by Another Layer of Indirection.

But that usually will create another problem."

--- David J. Wheeler