HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Introduction to mRUBis

Slides based on Sona Ghahremani

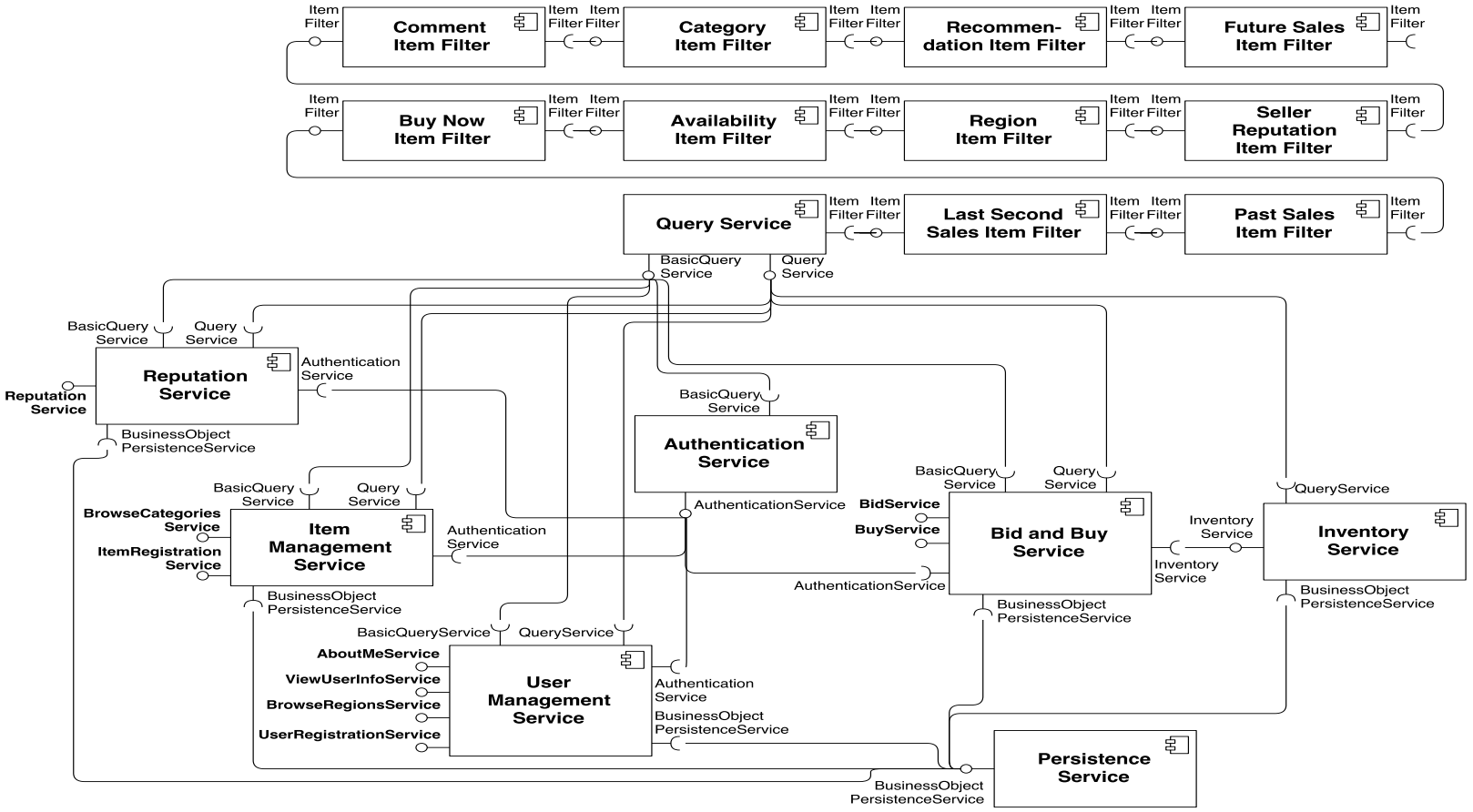
mRUBiS

- mRUBiS is a marketplace modelled after eBay on which users sell or auction items
- Derived from RUBiS (Rice University Bidding System), a popular case study to evaluate control theoretic algorithms
- modularized RUBiS
- A marketplace that hosts arbitrarily many shops

https://www.hpi.uni-potsdam.de/giese/public/mdelab/mdelab-projects/case-studies/mrubis/

Thomas Vogel. 2018. MRUBiS: an exemplar for model-based architectural self-healing and self-optimization. In Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '18). Association for Computing Machinery, New York, NY, USA, 101–107. DOI:https://doi.org/10.1145/3194133.3194161

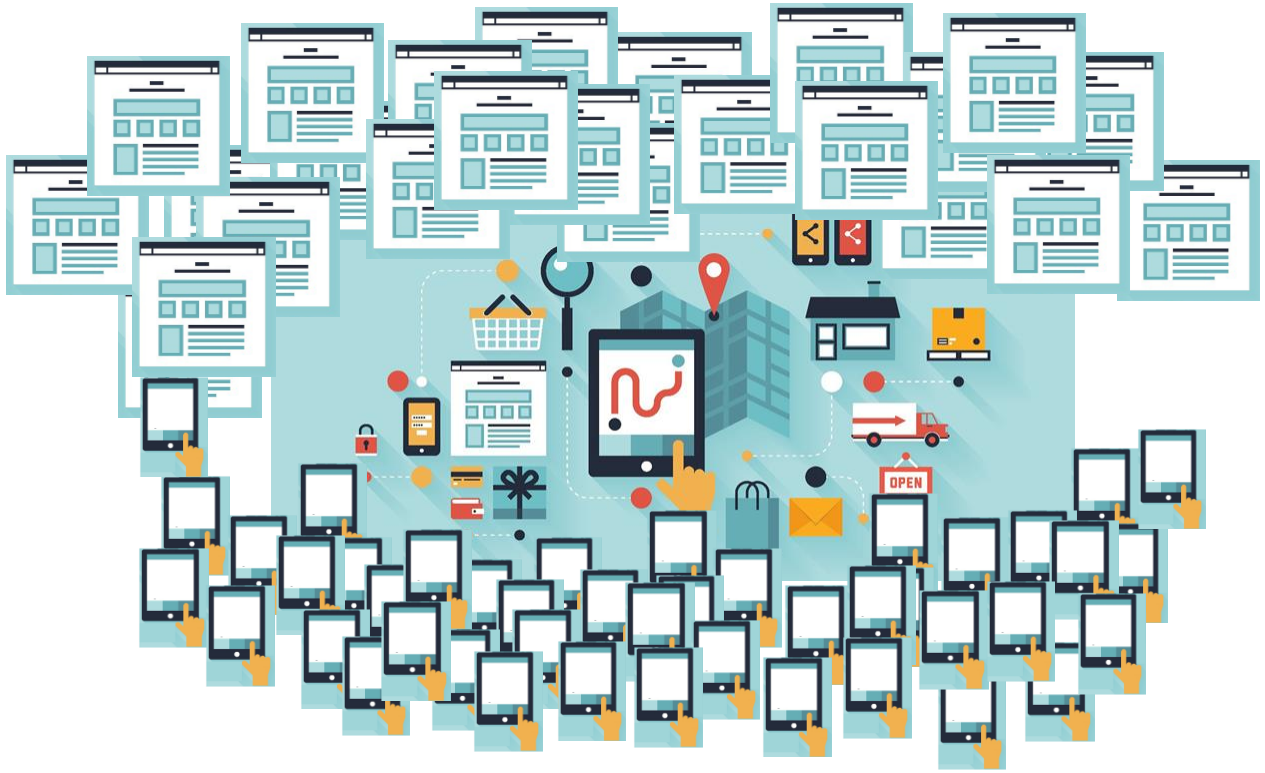
Architecture of a single shop in mRUBiS



- Each shop consists of 18 components and belongs to a tenant
- All shops share the same *component types* but each shop has its own, individually configured components

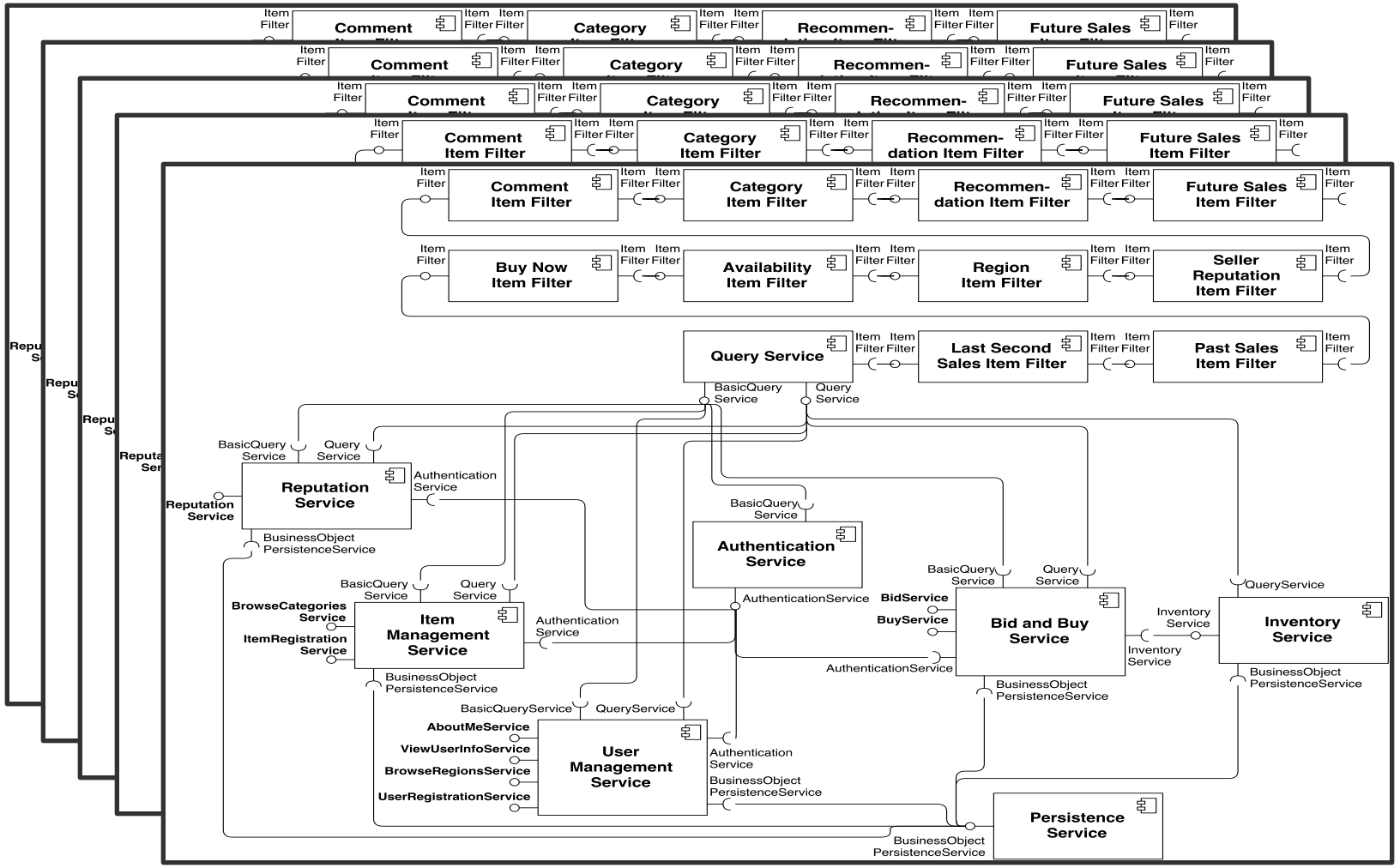
- Architecture of each shop is isolated from the architectures of the other shops (cf. multi-tenancy)
- Multi-tenancy setting enables scaling up the system

mRUBiS as a Large Market Place

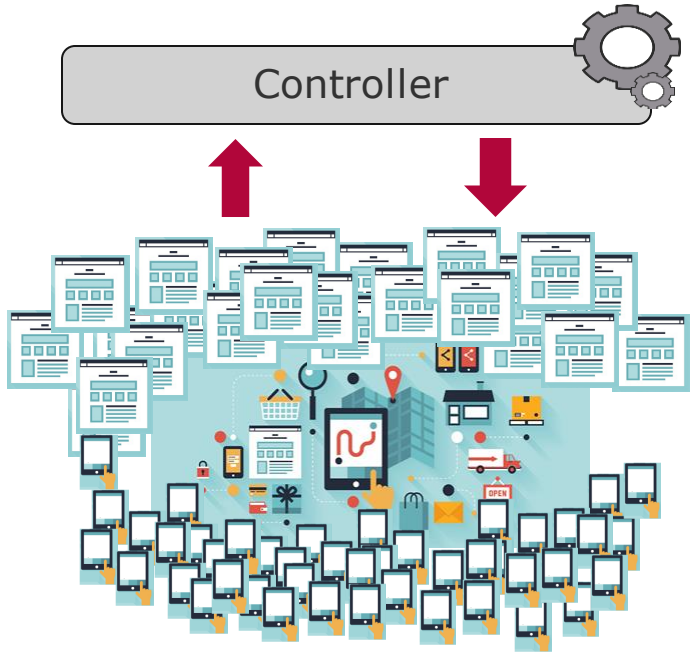


Architecture of the Market Place

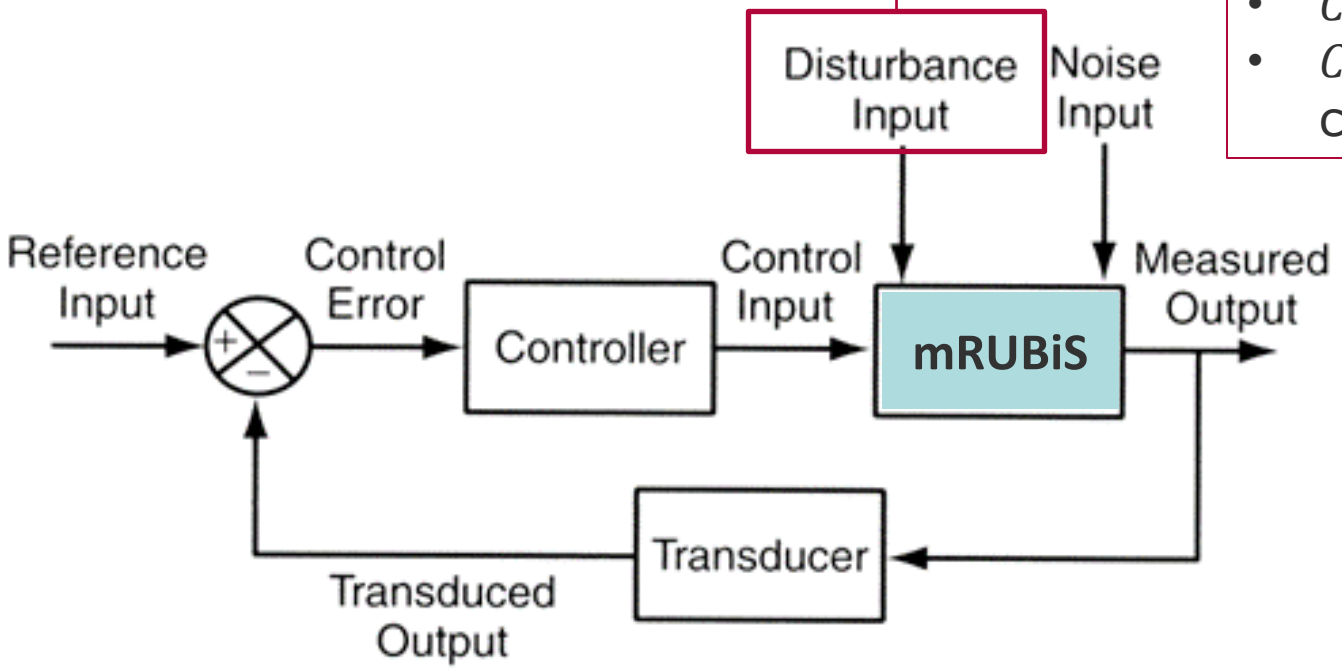
HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam



- A single controller controls all mRUBiS instances.

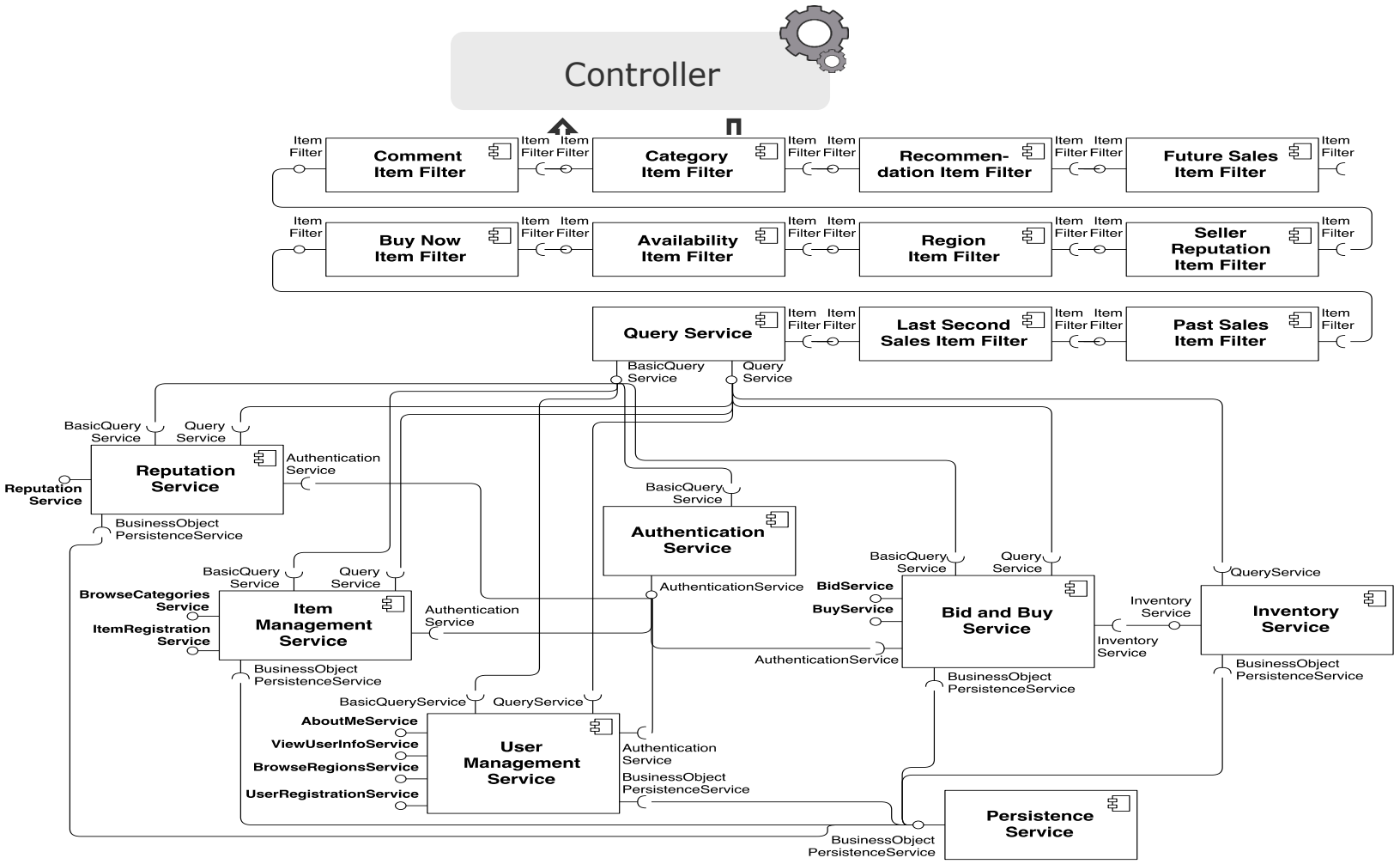


Disturabance in mRUBiS



- CF_1 = Component crashed
- CF_2 = Component throws exceptions
- CF_3 = Component is undeployed
- CF_5 = Change in the system load causing sub-optimal performance

Failures and Affected Components



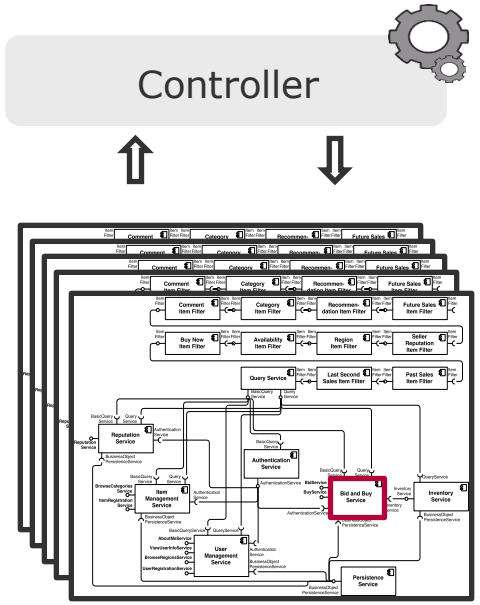
Affected Component

Affected Component Uid:
(_SEu7n-cdEeet0YmmfbMwkw)

Affected Component :
(Shop#,Component_type)

Input Failure Traces in mRUBiS

Critical Failures



Affected Component

Affected Component Uid:
(_SEu7n-cdEeet0YmmfbMwkw)

Affected Component :
(Shop#,Component_type)

```
injections.add(new Injection<Component>(IssueType.CF5,
        this.getComponent(17, 10)));
injections.add(new Injection<Component>(IssueType.CF3,
        this.getComponent(8,2)));
injections.add(new Injection<Component>(IssueType.CF1,
        this.getComponent(12, 10)));
injections.add(new Injection<ProvidedInterface>(IssueType.CF2,
        this.getComponent(1, 5).getProvidedInterfaces().get(0)));
```

Shop index

Affected Component index

Possible Observations (Outputs) in mRUBiS

- Output 1: list of all components and their corresponding CF status (CF0 to CF5): <Shop_id, Component_type, CF_x>
- Output 2: for each component, the current values of their parameters and the corresponding utility <Param_1,…,Param_N, Component_utility>
- Output 3: the list of <Compoment_type, CF_x, Action, Cost>
- Output 4: system current *Overall Utility, Shop Utility or, Component Utility*

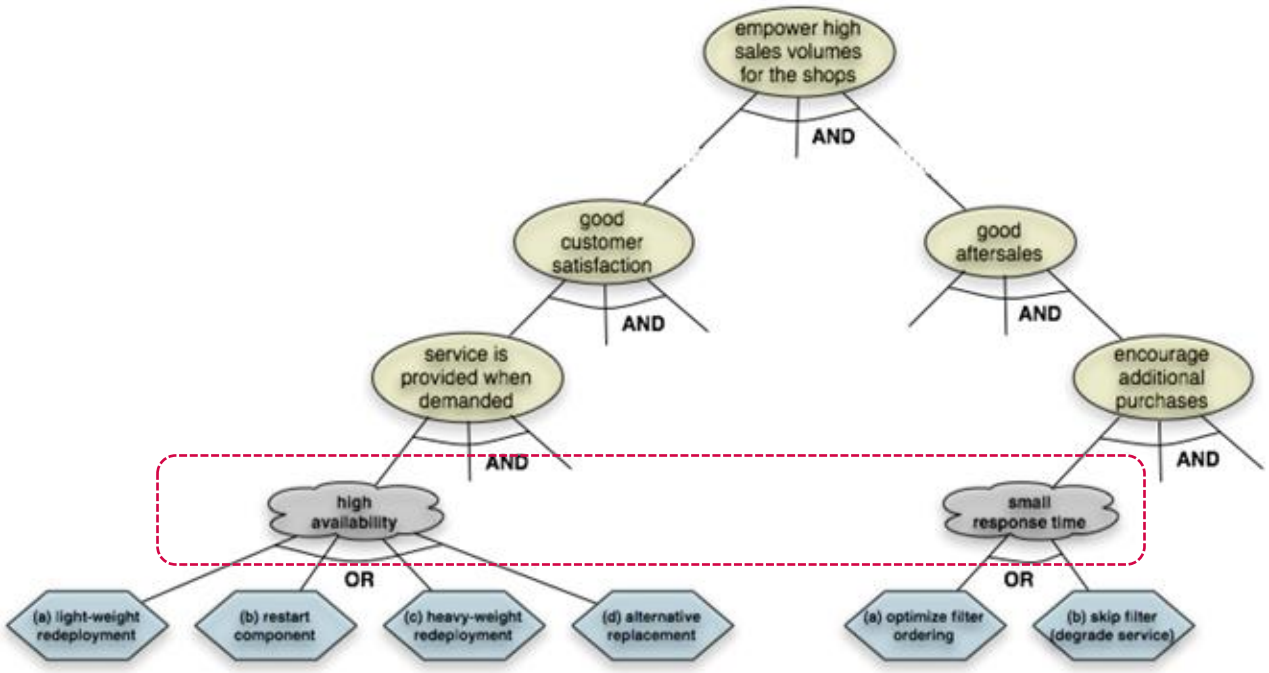
Instance of mRUBiS for Sampling

```
public class Observations {
    public static void makeObservation(Architecture mRUBiS){


        ArchitectureUtilCal.computeOverallUtility(mRUBiS);

        for (Tenant shop : mRUBiS.getTenants())
        {ArchitectureUtilCal.computeShopUtility(shop);

            shop.getName();
            shop.getCriticality();
            shop.getPerformance();
            shop.getRequest();


        for ( Component component : shop.getComponents())
        {    ArchitectureUtilCal.computeComponentUtility(component);
            component.getInUseReplica();
            component.getIssues();
            component.getType();
            component.getCriticality();
        }

        }


        List<Issue> allIssues = new LinkedList<>();
        allIssues.addAll(mRUBiS.getAnnotations().getIssues());
        for (Issue issue : allIssues)
        {issue.getAffectedComponent();
        issue.getUtilityDrop();
        issue.getHandledBy();
        issue.getHandledBy().get(0).getCosts();}

}
}
```

- Output 1: list of all components and their corresponding CF status (CF0 to CF5): <Shop_id, Component_type, CF_x>
- Output 2: for each component, the current values of their parameters and the corresponding utility <Param_1,…,Param_N, Component_utility>
- Output 3: the list of <Compoment_type, CF_x, Action, cost>
- Output 4: system current *Overall Utility*

mRUBiS Goal Model

- The controller follows a goal-based approach focused on fulfilling the soft goals of *High availability* and *Low response time* for the shops.
- Disturbances are addressed via two sets of actions:
  - Repair Action
  - Optimization Actions



Possible Actions in mRUBiS

- Repair Action
  - Restart Component
  - Heavy Weight Redeployment
  - Light weight Redeployment
  - Replace Component (if an alternative component is available)

- Optimization Actions
  - Add Replica
  - Remove Replica

Mapping Actions to CFs

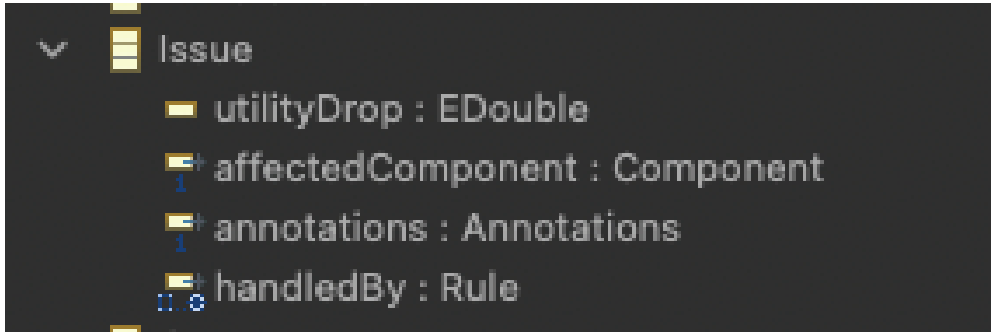
- CF1,CF2,CF3 can be addressed by:
  - Restart Component
  - Heavy Weight Redeployment
  - Light Weight Redeployment
  - Replace Component (if an alternative component is available)

- CF5 can be addressed by:
  - Add Replica
  - Remove Replica

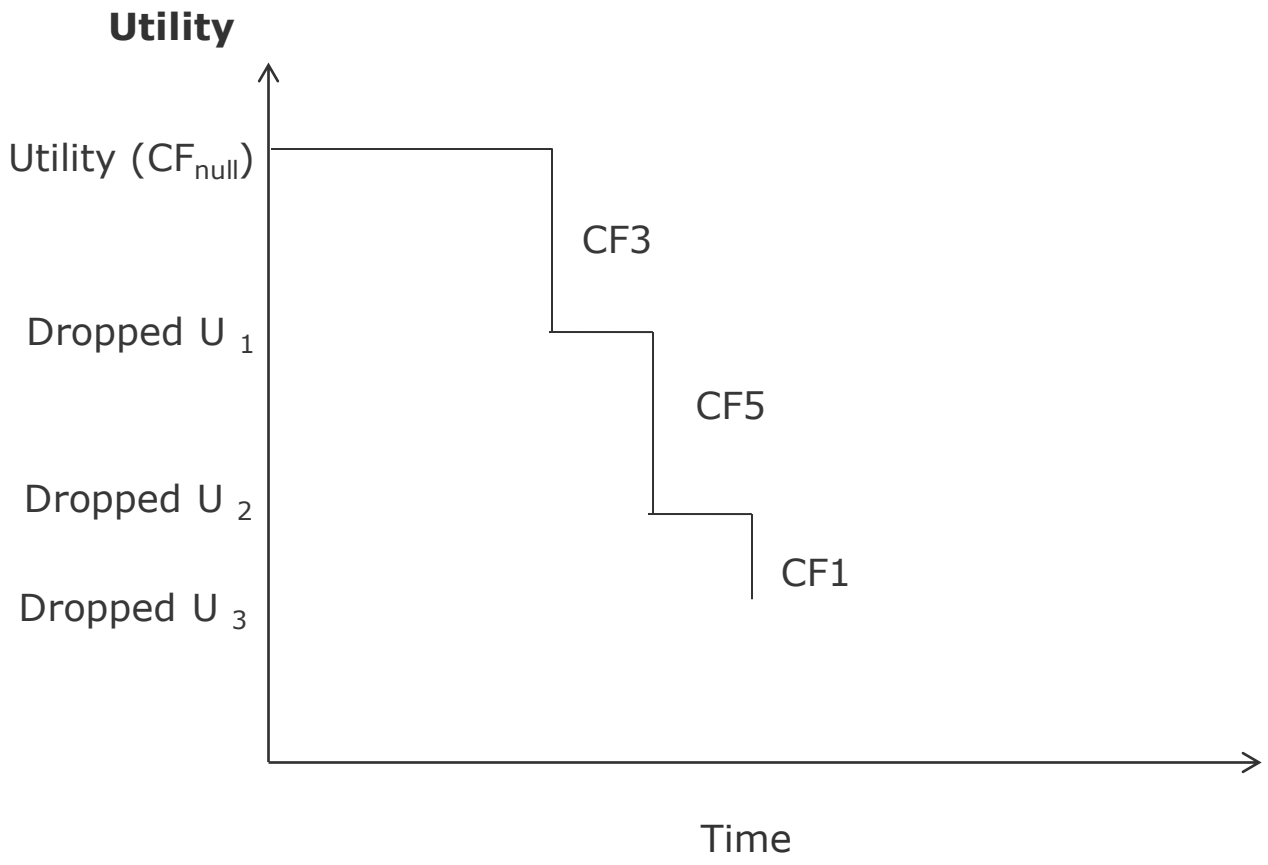
Action: <Reward, Cost>
Reward (Affected_Component , CF_type)

CFs (issues) in mRUBiS

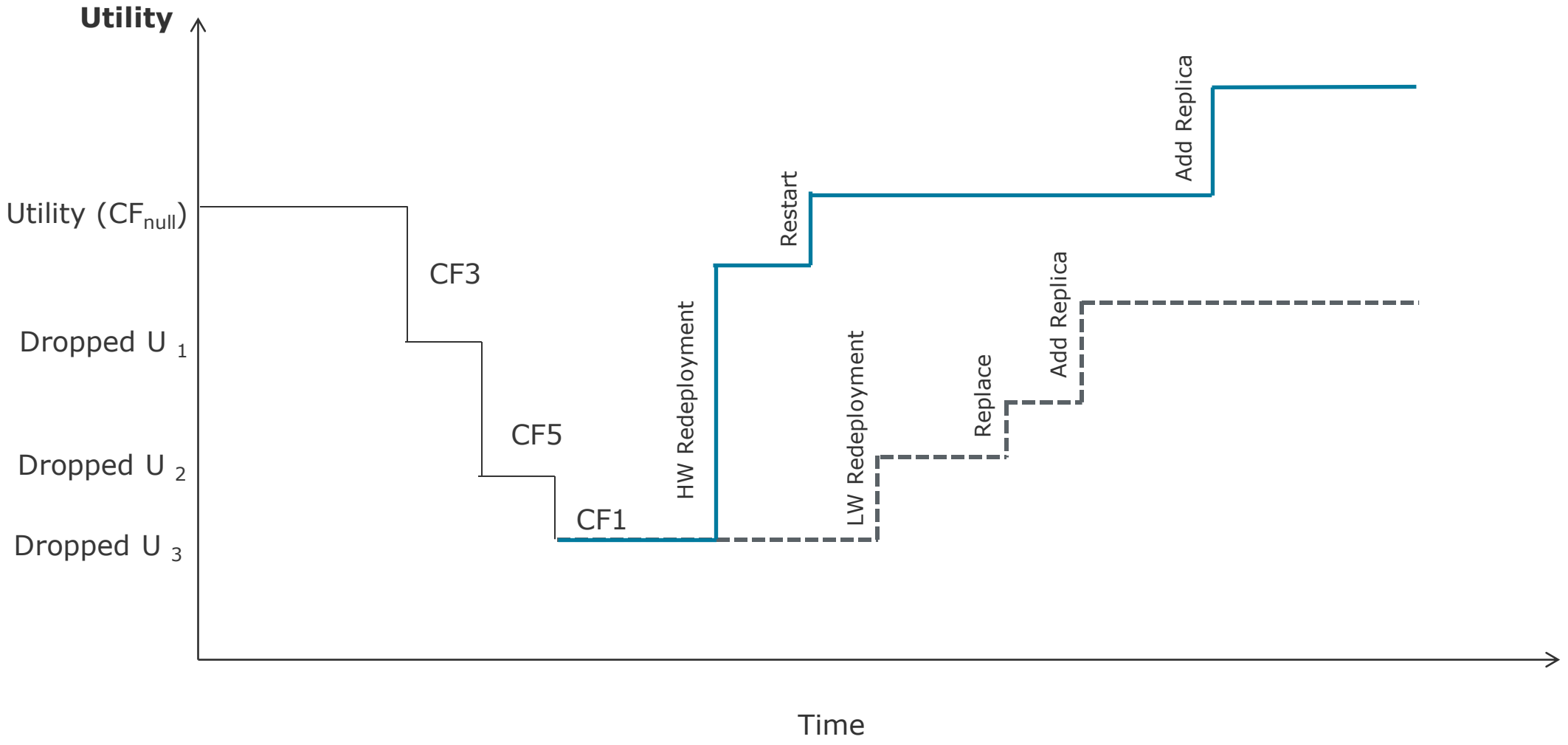
- For each CF_x
  - Affected Component
    - Component parameters can be observed!
  - Utility Drop
  - Possible Actions to Resolve
    - Action: <Reward, Cost>
    - Reward is the Utility Increase or Delta



Utility Drop



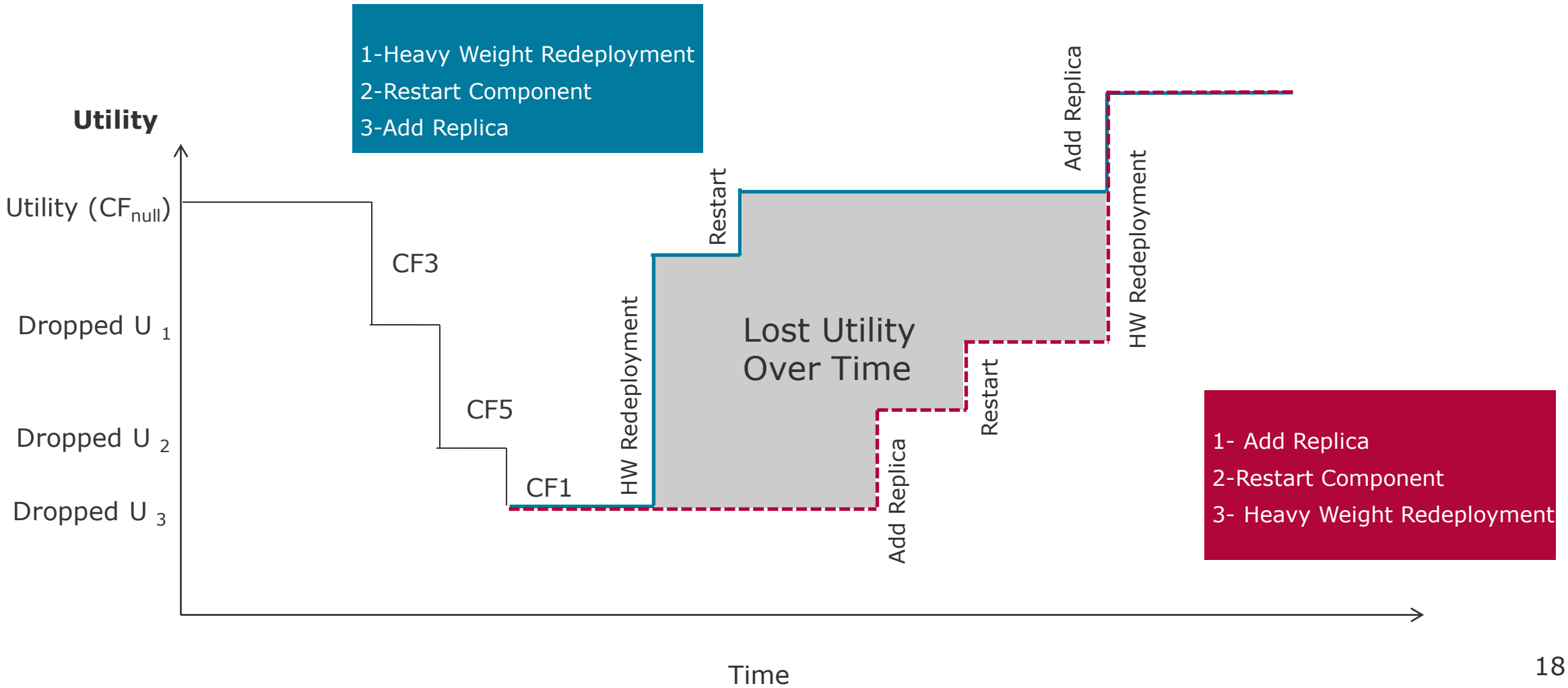
Which Actions to Take? -→ You tell mRUBiS



Which Actions to Take? -→ You tell mRUBiS

```java
package mRUBiS_Tasks;

import java.util.LinkedList;
import java.util.List;

import de.mdelab.morisia.comparch.Issue;
import de.mdelab.morisia.comparch.Rule;

public class Input {
public static void selectAction(Issue issue) {

        issue.getAffectedComponent();
        issue.getAffectedComponent().getType();
        issue.getAffectedComponent().getTenant().getName();


        issue.eClass().getName();

        //this should be read from input
        String actionToExecute="HWRedeployComponent";

        //Remove all the other possible actions
        List<Rule> actionsToRemove= new LinkedList<Rule>();
        List<Rule> actionToKeep = new LinkedList<Rule>();
        for (Rule rule : issue.getHandledBy()) {
            if(rule.eClass().getName().equals(actionToExecute))
                {actionToKeep.add(rule);
                }
            else actionsToRemove.add(rule);

        }


        for (Rule rule : actionsToRemove) {
            rule.setAnnotations(null);
            rule.setHandles(null);
        }



    }

}
```

And in Which Order?



And in Which Order?

```java
package mRUBiS_Tasks;
import java.util.Comparator;
import de.mdelab.morisia.comparch.Issue;

public class RLIssueComparator implements Comparator<Issue> {

	@Override
	public int compare(Issue issue_1, Issue issue_2) {
		issue_1.getAffectedComponent();
		 issue_1.getAffectedComponent().getType();
		 issue_1.getAffectedComponent().getTenant().getName();

		 issue_2.getAffectedComponent();
		 issue_2.getAffectedComponent().getType();
		 issue_2.getAffectedComponent().getTenant().getName();

	    String issue_1_name = issue_1.eClass().getName();
	    String issue_2_name = issue_2.eClass().getName();


	    if (issue_1_name.equals("CF3")) {
	        return -1;
	    } else if (issue_2_name.equals("CF3")&&issue_1.getAffectedComponent().getTenant().getName().equals("BidAndBuyService")) {
	        return 1;
	    } else if (issue_1_name.equals("CF1") & issue_2_name.equals("CF2")) {
	        return -1;
	    }  else if (issue_2_name.equals("CF1") & issue_1_name.equals("CF2")) {
	        return 1;
	    } else {
	        return 0;
	    }
	}
}
```

- Input: ordered list of <component , action>

Or Input: ordered list of <component , CF_x>

Back-end

Thomas Vogel. 2018. mRUBiS: an exemplar for model-based architectural self-healing and self-optimization.
In Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '18).
Association for Computing Machinery, New York, NY, USA, 101–107. DOI:https://doi.org/10.1145/3194133.3194161

Eclipse Modeling Framework

- The mRUBiS Simulator maintains a model of mRUBiS that simulates the real system.
- This model is an instance of a metamodel developed with the Eclipse Modeling Framework (EMF).
  - Framework and code-generation tool for building Java applications based on models (*model-driven development*).
- To use the simulator a *Java installation* and the *Eclipse Modelling Tools* edition of Eclipse Download here:

http://www.eclipse.org/downloads/packages/release/2021-03/r/eclipse-modeling-tools

SDM

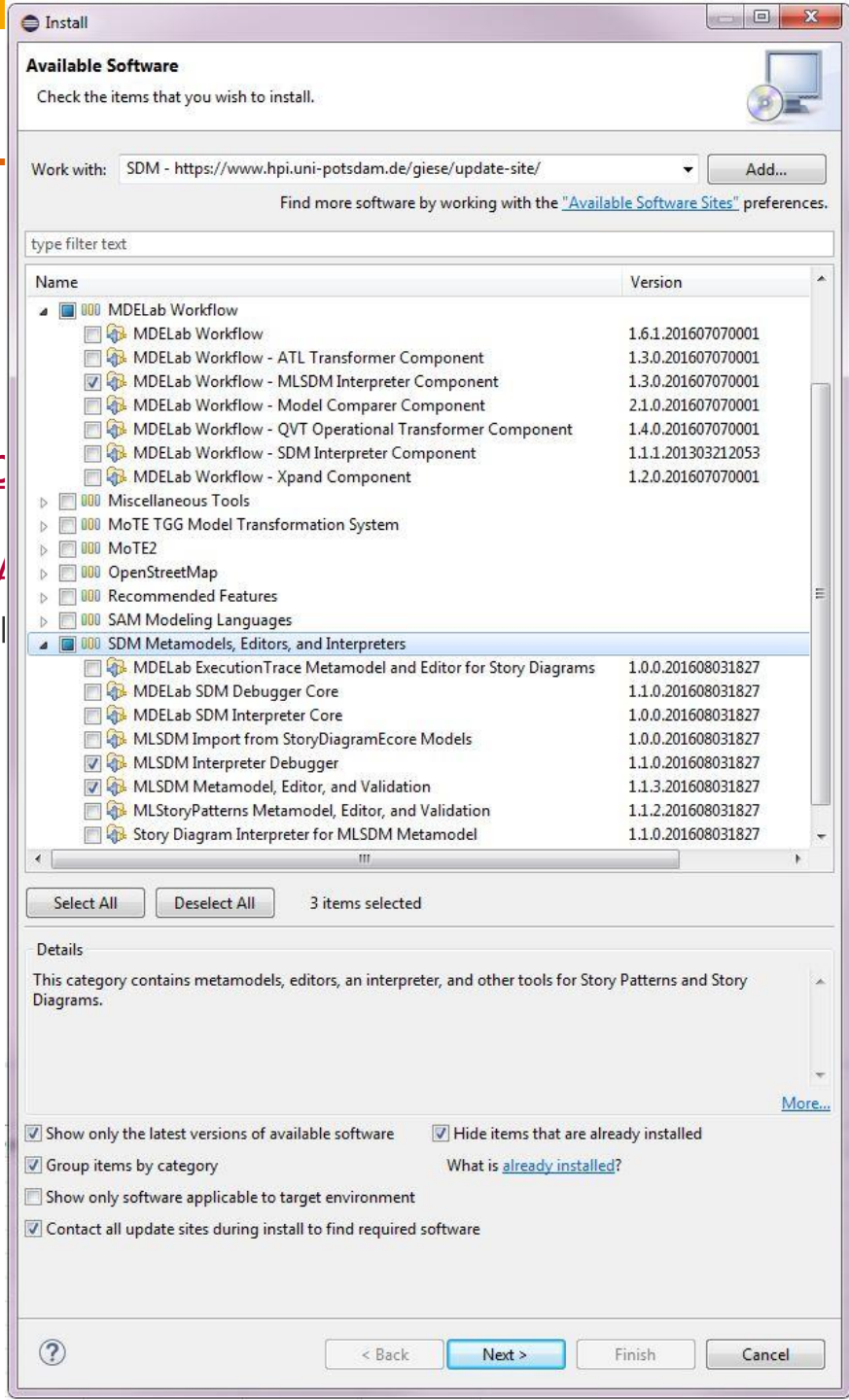
Download the Eclipse Modeling Tools package

Add the following update sites to the Eclipse update manager:

MDELab update site: https://www.hpi.uni-potsdam.de/giese/up

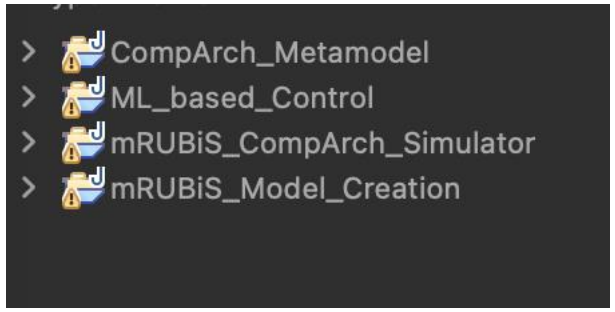
Eclipse Neon update site: http://download.eclipse.org/releases/ latest Eclipse releases no longer include GMF (Graphical Modeli by the SDM graphical editors).

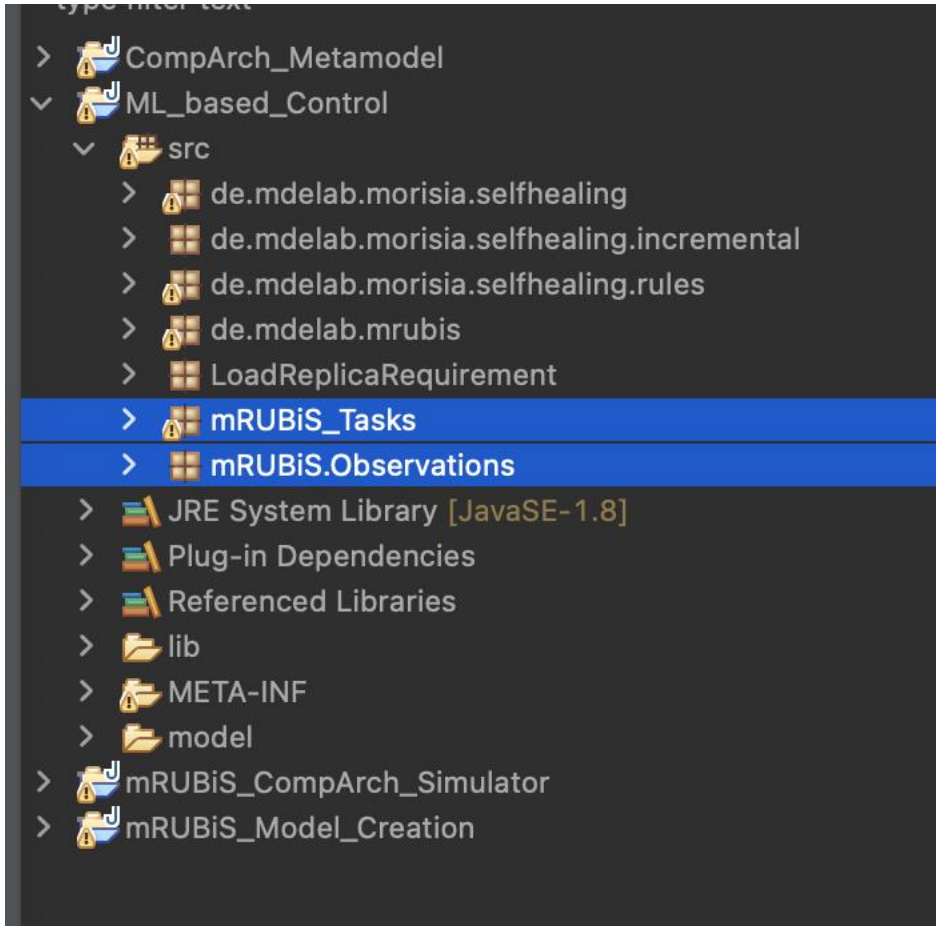
Select the following features:

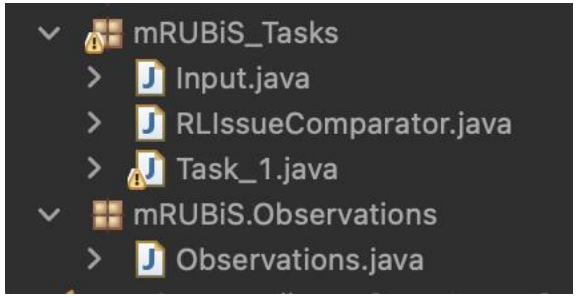


Project Files

- Will be uploaded on GitHub







End