*WETSoDA '17, December 4, Nanjing, China*

# Decentralization of Control Loop for Self-Adaptive Software through Reinforcement Learning

**Kishan Kumar Ganguly, Kazi Sakib**
**Institute of Information Technology**
**University of Dhaka, Dhaka, Bangladesh**

**Presented By**

Kishan Kumar Ganguly
Institute of Information Technology
University of Dhaka

# Presentation Outline

- Introduction
- Background
- Related Work
- Problem and Motivation
- Running Example
- Proposed Approach
- Experimental Evaluation
- Results
- Discussion
- Conclusion and Future Work

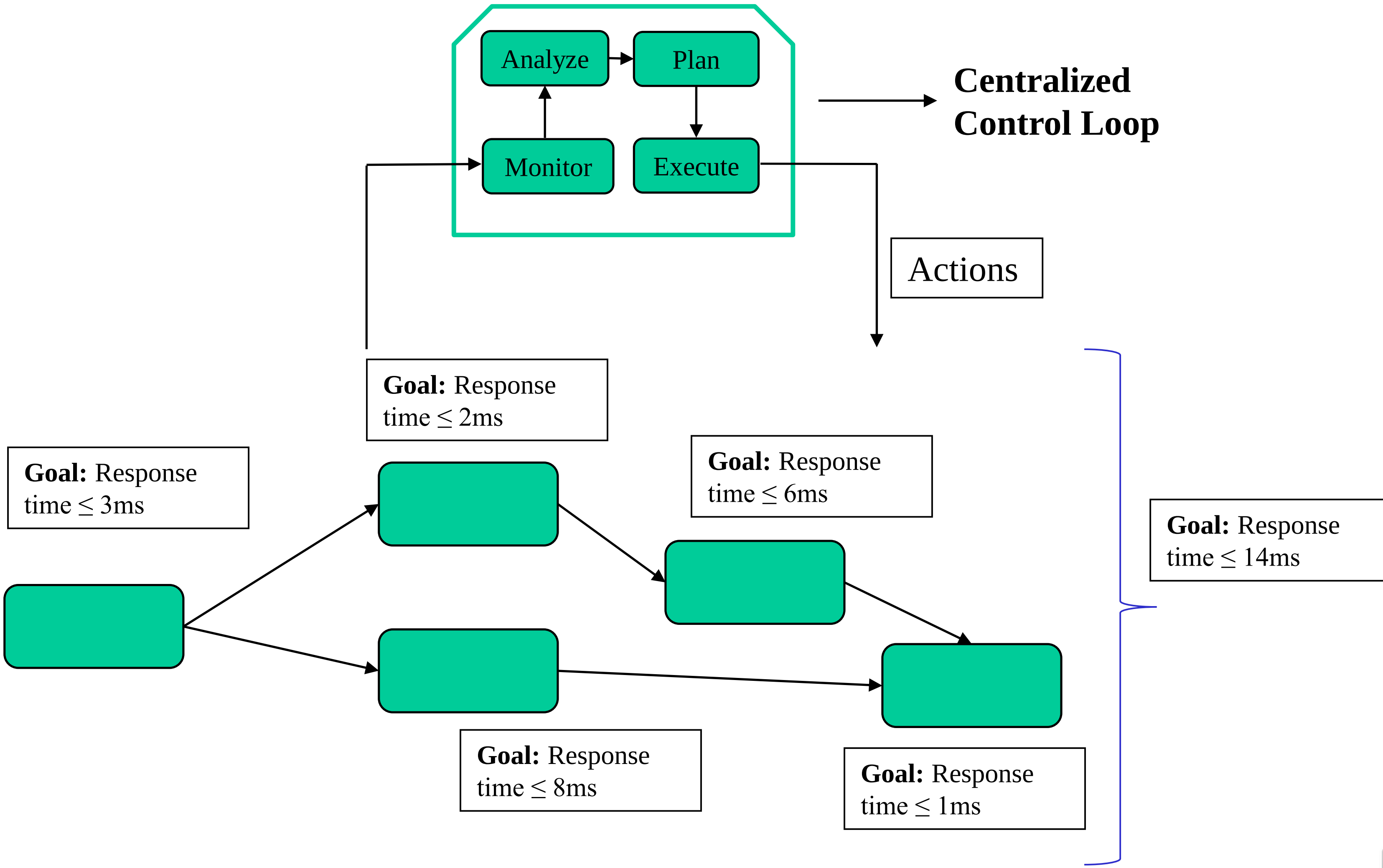
# Introduction

- Self-adaptive systems change their behavior at runtime to conform to their goals
- Goals are generally non-functional requirements
  - Response time
  - Throughput etc.
- Decentralization has become a widespread concept [1]
- Designing self-adaptive software with decentralized control loop is still a research challenge [1]–
  - A control loop helps to respond to the goal violation at runtime without interrupting its service

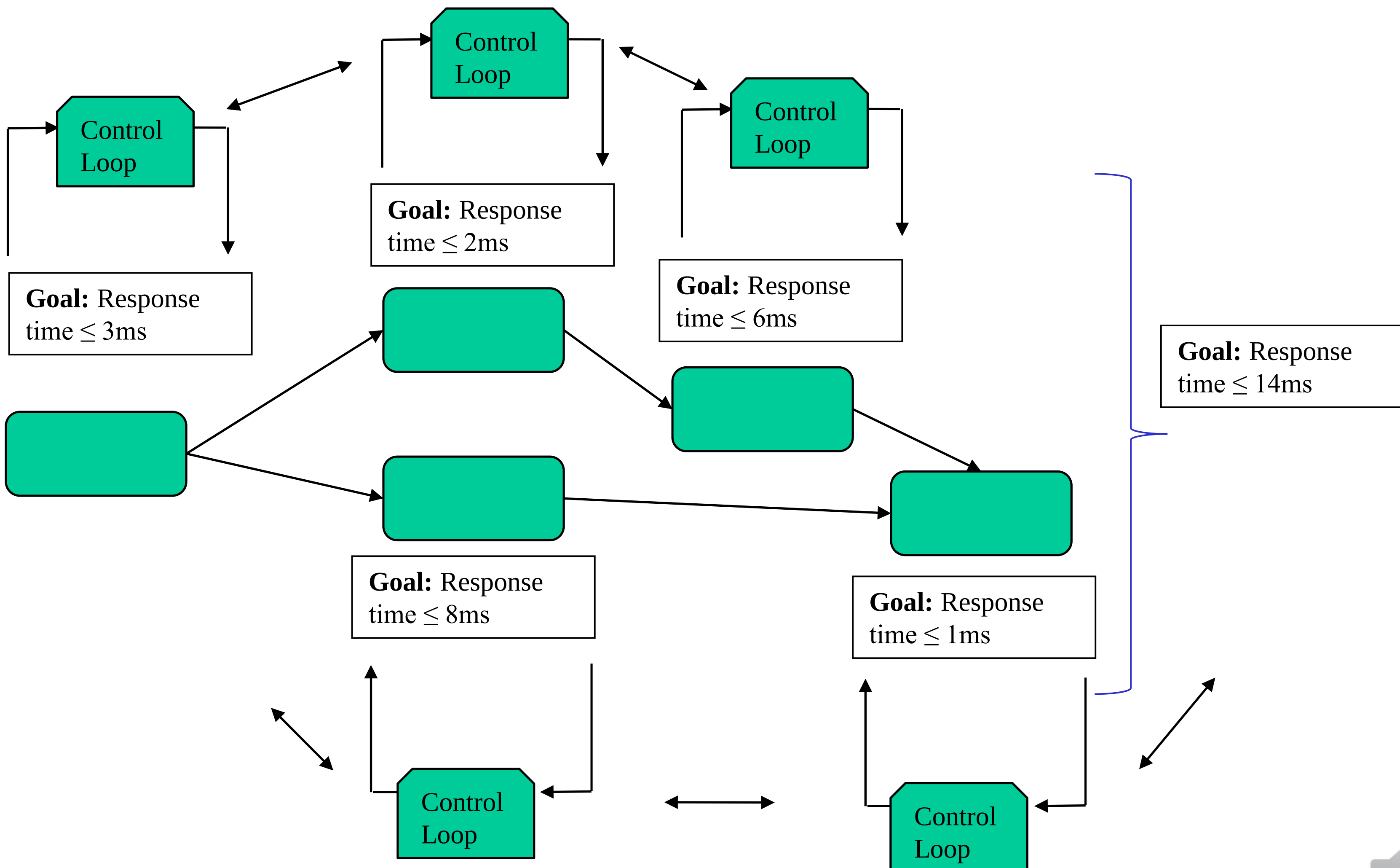
# Background

- In self-adaptive system with decentralized control loops, multiple control loops coordinate to satisfy some goals
- These goals can be divided as –
  - Local goal: component-level goal [2]
  - Global goal: system-level goal [2]
- A goal violation leads to –
  - Coordination of multiple control loop (information sharing) to take adaptation decisions
  - These adaptation decisions / actions are selection of variants (software variants with different configurations)
  - Adaptation decisions must satisfy local and global goals

# Background: Centralized Self-Adaptive Systems



# Background: Decentralized Self-Adaptive Systems



# Related Work

### Weyns et al. [4]

- Proposed a generic model for decentralized self-adaptive system
- It was highly abstract and did not explicitly address coordination of multiple control loops

### Sykes et al. [5]

- Proposed a distributed self-assembly approach
- Used aggregated gossip for coordination
- Lacked dynamism, used static strategies

### Schmerl et al. [13]

- Proposed five patterns for decentralized self-adaptation
- Facilitated further research into this domain

2010 — 2011 — 2013 — 2014 — 2016

### Grassi et al. [2]

- For self-adaptive service assembly
- Used gossip protocol for service specification dissemination
- Selected best service based on predefined utility
- Problem: static utility value

### Wang et al. [3]

- Dynamic service composition based on reinforcement learning
- Opponent model-based coordination
- Can be further improved using opponent model for both learning and action selection and updating weight of violated goal for more importance

### Caporuscio et al. [6]

- Performed TD learning on global quality functions of services to estimate their long term quality
- Service matching was performed by choosing the service that has maximum quality
- Local goals and coordination was not explicitly considered

# Problem and Motivation

**Problem**

- Writing static strategies or action selection rules, similar to some centralized control loops is not practical due to large state space
- In a specific state, action selection rules of a local control loop depend on the strategies followed by other control loops
- Reward functions (that calculate goal conformance [3]) need to be defined in such a way that these successfully capture both local and global goal violations

**Motivation**

- Reinforcement learning provides a great opportunity to introduce dynamism into the self-adaptive decisions
- Each local control loop needs to estimate the strategies used by other ones which can be done through an opponent model along with multiagent reinforcement learning
- The reward functions can be aggregated with a dynamic weight to provide more importance to the violated goals

# Running Example

- The Tele Assistance System (TAS) is a service-based system that provides medical service to patients [9]
- Three services are used –
  – MedicalAnalysisService – Checks the vital parameters of the patient and takes actions
  – DrugService – Change drug or dosage of the drug
  – AlarmService – Provides alarm in case of emergency
- MedicalAnalysisService variants with different configurations
  – MedicalAnalysisService1, MedicalAnalysisService2 and MedicalAnalysisService3
- AlarmService similarly has three variants

# Running Example

| Goal | Attribute | Goal Type | Goal Threshold |
|---|---|---|---|
| MedicalAnalysisService must have response time less than or equals 5.6 ms | Response Time | Local & Threshold | 5.6 |
| AlarmService must have response time less than or equals 5.2 ms | Response Time | Local & Threshold | 5.2 |
| MedicalAnalysisService must have failure rate less than or equals 0.12 | Failure Rate | Local & Threshold | 0.12 |
| AlarmService must have failure rate less than or equals 0. | Failure Rate | Local & Threshold | 0.1 |
| At least one service must have failure rate less than 0.08 | Failure Rate | Global & Threshold | 0.08 |
| Average cost per service must be minimized | Average Cost Per Service | Global & Minimization | - |

# Running Example

| Goal Attribute | Value Range | Corresponding Category |
|---|---|---|
| Failure Rate | [0, 0.002) | Low |
| | [0.002, 0.08) | Medium |
| | [0.08, 0.1) | High |
| | [0.1, ∞) | Extreme |
| Average Cost Per Service | [0, 2) | Low |
| | [2, 5) | Medium |
| | [5, ∞) | High |
| Individual Service Response Time | [0, 2.5) | Low |
| | [2.5, 5.2) | Medium |
| | [5.2, ∞) | High |

# Proposed Approach

- A state is represented by a specific combination of the different goal attribute values in different agents
  - *{{low,extreme},{high,high},{low}}* is a state that expresses that MedicalAnalysisService has low response time and extreme failure rate, AlarmService has high response time and failure rate and globally average cost per service is low
- An action is considered as choosing a specific variant
- The action set of a specific agent consists of all of its variants
- The joint action set is the action selection of all the agents
  - {{MedicalAnalysisService1, MedicalAnalysisService2, MedicalAnalysisService3},{AlarmService1, AlarmService2, AlarmService3}}

# Proposed Approach

- Reward functions measure goal conformance
- For minimization and maximization goal, if $G$ is the set of the values of a goal attribute, a reward function is defined as $R : G \rightarrow [0, 1]$
- For threshold goals, it is defined as $R : G \times T \rightarrow [0, 1]$
- The reward function value is restricted between 0 and 1

# Proposed Approach

- Reward functions for TAS –

Global
Failure Rate
Reward

$$r_{glf} = \begin{cases} \dfrac{(2 \times th_{glf} - min(f_r^1, f_r^2, \cdots, f_r^n))}{2 \times th_{glf}} & \text{if } (th_{glf} - min(f_r^1, f_r^2, \cdots, f_r^n)) > 0 \\ \dfrac{1}{2} - \dfrac{min(f_r^1, f_r^2 - 2 \times th_{glf}}{2 \times (1 - th_{glf})} & \text{if } (th_{glf} - min(f_r^1, f_r^2, \cdots, f_r^n)) \leq 0 \end{cases}$$

Local
Response
Time Reward

$$r_t = \begin{cases} \dfrac{2 \times th_t - t^i}{2 \times (th_{glf} - t^i_{max})} & \text{if } th_t - t^i > 0 \\ \dfrac{1}{2} - \dfrac{th_t - t^i + 1}{2 \times (th_t - t^i_{max} + 1)} & \text{if } th_t - t^i < 0 \\ \dfrac{1}{2} & \text{if } th_t - t^i = 0 \end{cases}$$

# Proposed Approach

- Reward functions for TAS –

Local Failure Rate Reward

$$r_f = \begin{cases} \dfrac{2 \times th_f - f^i}{2 \times (th_f - f^i_{max})} & \text{if } th_f - f^i > 0 \\ \dfrac{1}{2} - \dfrac{th_f - f^i + 1}{2 \times (th_f - f^i_{max} + 1)} & \text{if } th_f - f^i < 0 \\ \dfrac{1}{2} & \text{if } th_f - f^i = 0 \end{cases}$$

Average Cost Per Service Reward

$$r_{\overline{c}} = \frac{1}{\frac{1}{n} \sum_{i=1}^{n} \overline{c^i} + 1}$$

Total Reward

$$tot_r = 0.25 \times r_{glf} + 0.25 \times r_t + 0.25 \times r_f + 0.25 \times r_{\overline{c}}$$

Threshold-based reward functions provide [0, 0.5) values for goal violation and [0.5, 1] values for goal conformance

# Proposed Approach

- Weights need to be updated at runtime for providing more importance on reward function values indicating goal violation
- For example, consider four reward functions $r1$, $r2$, $r3$, $r4$
  - For reward values 0.7, 0.8, 0.3, 0.6, the total reward value is 0.6
  - For reward values 0.6, 0.8, 0.5, 0.5, the total reward value is also 0.6
  - The first reward value should be less than the second
- Solution –
  - The weight for 0.3 can be updated to 0.3542 from 0.25
  - The remaining weight 0.6458 can be equally distributed among the other three agents
  - The total reward value becomes 0.558

# Proposed Approach
# Action Selection through Q-Learning

**Algorithm 1** Algorithm for Multiagent Q-learning

$Q_i(s, a) \leftarrow 0, \forall s, a, i$
$decay \leftarrow d_f$
**while** $TerminationCondition \neq true$ **do**
  **for** $i = 1$ to $n$ **do**
    $a_i \leftarrow selectAction()$
    $a^{-i} \leftarrow receiveOtherAgentActions()$
    $a \leftarrow a_i \cup a^{-i}$
    *observe transitioned state* $s'$ *and reward* $r_i$
    **for** $i = 1$ to $n$ **do**
      $Q_i(s, a) \leftarrow Q_i(s, a) + \alpha \times [r_i(s, a) + \gamma \max_{a_i} Q_i(s', a_i) - Q_i(s, a)]$
      $\alpha \leftarrow \alpha - decay \times \alpha$
    **end for**
  **end for**
**end while**

# Proposed Approach
# Action Selection through Q-Learning

- An opponent model chooses the maximum next state strategy based on other agents' strategies [8]

$$Q_i(s, a_i) = \sum_{a^{-i} \in A_c^{-i}} P(s, a^{-i}) \times Q_i(s, a_i, a^{-i})$$

Opponent Model

$$P(s, a^{-i}) = \frac{F(s, a^{-i})}{\sum_{a^{-i'} \in A_c^{-i}} F(s, a^{-i'})}$$

# Proposed Approach
# Action Selection through Q-Learning

- The *ε-greedy* strategy is used for action selection
- This provides a balance between exploration and exploitation in Q-learning [11]
- *ε-greedy* strategy –

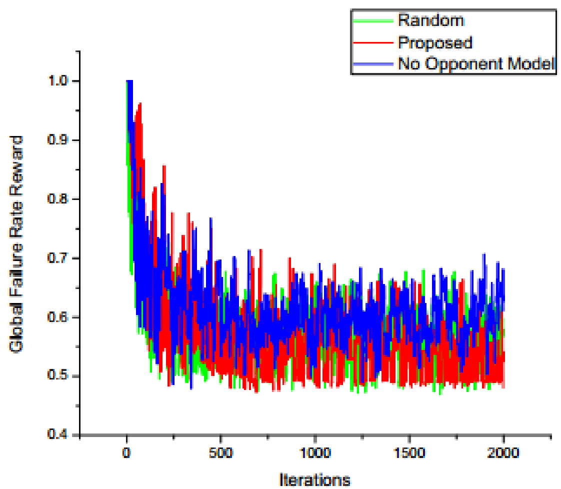
An ε value is prespecified where $0 \leq \varepsilon < 1$ → A value *p* between 0 and 1 is randomly selected → If $p < \varepsilon$, the agent chooses a random action → Otherwise, it selects the action with maximum Q-value in the current state

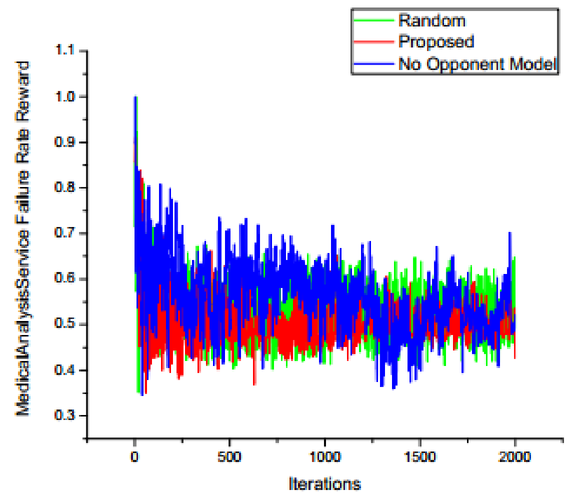
# Experimental Evaluation

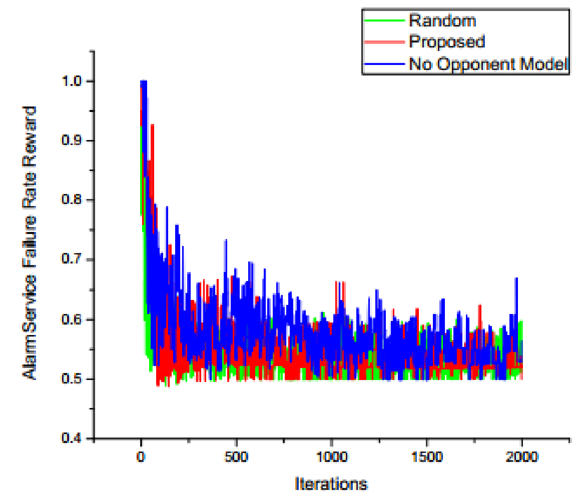
- The Tele Assistance System is used to evaluate the proposed approach
- It is extended to support decentralized adaptation by adding a control loop to each of the services
- The approach is compared to two techniques –
  – The first one chooses actions randomly
  – The second one learns and chooses actions based on maximum Q-values without considering opponent models
- Parameters –
  – $\alpha$, $\gamma$ and $\varepsilon$ were chosen to be 0.1, 0.9 and 0.8 respectively
  – A small number 0.001 was chosen as the decay factor
  – 1.25 was chosen as the value of *fr*

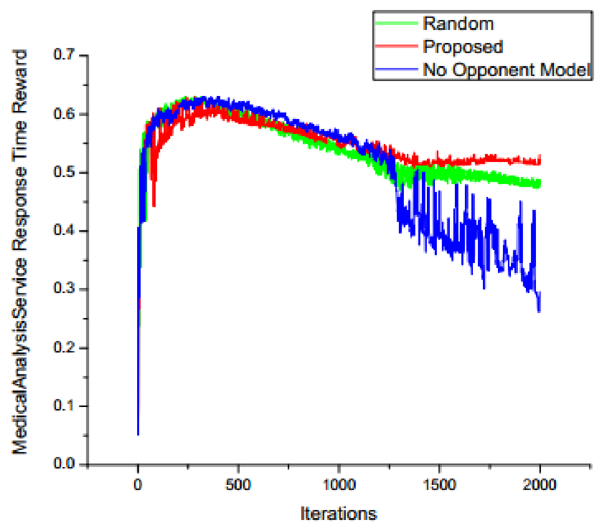
Chosen through empirical experimentation

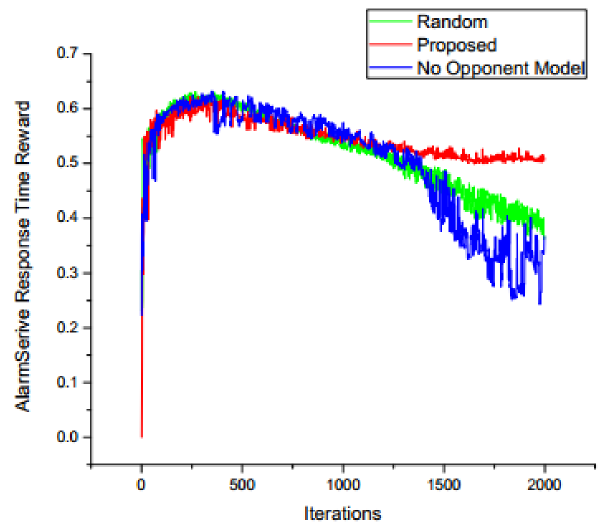
# Results: Comparison of Reward



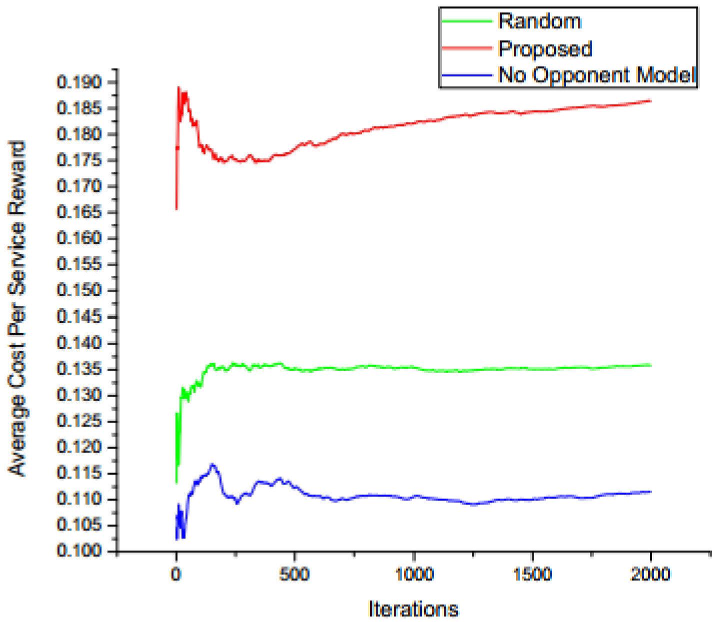



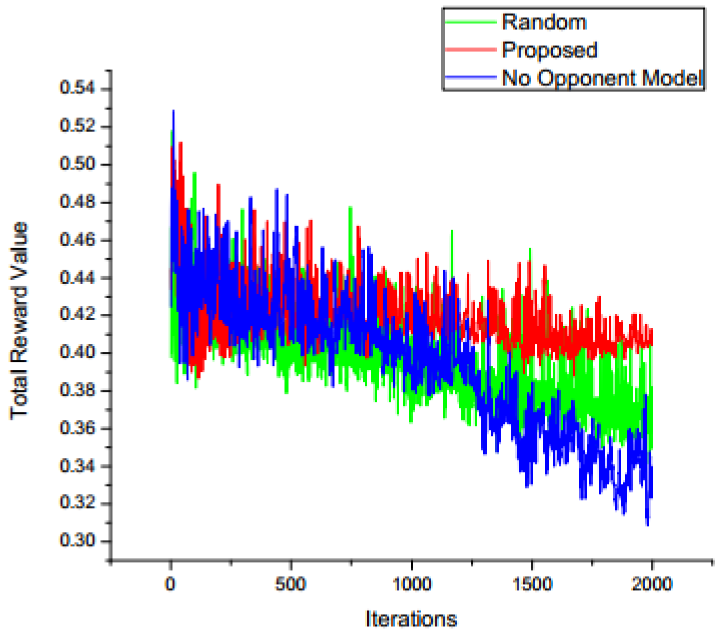



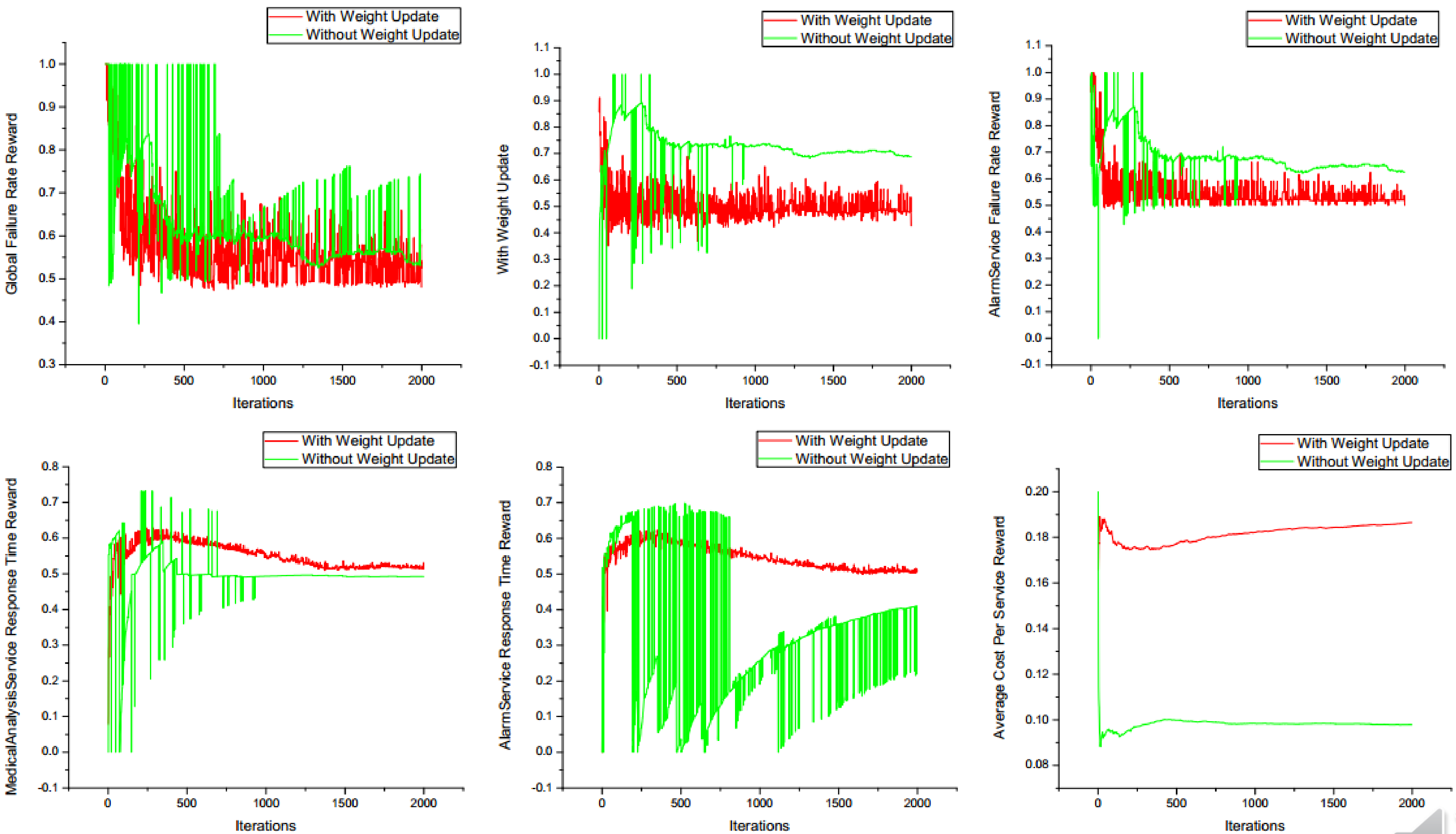



# Results: Comparison of Reward



# Results: Effectiveness of the Dynamic Weight Update Technique



# Discussion

- The reward values are over 0.5 in most cases which indicates adaptation
- Reward values are more stable and total reward values are higher when opponent model is considered
- The weight update mechanism supports adaptation of multiple goals as the reward values remain stable over time
- Q-learning over joint actions become computationally challenging in large scale systems
- A promising direction towards solving this problem can be the use of sparse cooperation Q-learning (future work) [12]

# Conclusion and Future Work

- A decentralized control loop for self-adaptive software has been proposed –
  - Considering other agent strategies
  - A better total reward calculation mechanism
- It was evaluated on a Tele Assistance System where it was observed that –
  - Reward stays over threshold
  - Reward values are stable over time
  - Both indicates successful self-adaptation
- Future Work –
  - Applying to large scale systems
  - Self-tuning the required parameters to achieve the highest reward
  - Tool support

# References

[1] Rogerio De Lemos, Holger Giese, Hausi A Muller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M Villegas, Thomas Vogel, et al. Software engineering for self-adaptive systems: A second research roadmap. In Software Engineering for SelfAdaptive Systems II, pages 1–32. Springer, 2013.

[2] Vincenzo Grassi, Moreno Marzolla, and Raffaela Mirandola. Qosaware fully decentralized service assembly. In Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2013 ICSE Workshop on, pages 53–62. IEEE, 2013.

[3] Hongbing Wang, Qin Wu, Xin Chen, Qi Yu, Zibin Zheng, and Athman Bouguettaya. Adaptive and dynamic service composition via multiagent reinforcement learning. In Web Services (ICWS), 2014 IEEE International Conference on, pages 447–454. IEEE, 2014.

[4] Danny Weyns, Sam Malek, and Jesper Andersson. On decentralized self-adaptation: lessons from the trenches and challenges for the future. In Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, pages 84–93. ACM, 2010.

[5] Daniel Sykes, Jeff Magee, and Jeff Kramer. Flashmob: distributed adaptive self-assembly. In Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, pages 100–109. ACM, 2011.

[6] Mauro Caporuscio, Mirko D'Angelo, Vincenzo Grassi, and Raffaela Mirandola. Reinforcement learning techniques for decentralized selfadaptive service assembly. In European Conference on Service-Oriented and Cloud Computing, pages 53–68. Springer, 2016.

[7] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction, volume 1. MIT press Cambridge, 1998.

[8] Ann Nowe, Peter Vrancx, and Yann-Michael De Hauwere. Game theory and multi-agent reinforcement learning. In Reinforcement Learning, pages 441–470. Springer, 2012

# References


[9] Danny Weyns and Radu Calinescu. Tele assistance: A self-adaptive service-based system examplar. In Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, pages 88–92. IEEE Press, 2015.

[10] Gwo-Hshiung Tzeng and Jih-Jeng Huang. Multiple attribute decision making: methods and applications. CRC press, 2011.

[11] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. AAAI/IAAI, 1998:746–752, 1998.

[12] Jelle R Kok and Nikos Vlassis. Sparse cooperative q-learning. In Proceedings of the twenty-first international conference on Machine learning, page 61. ACM, 2004.

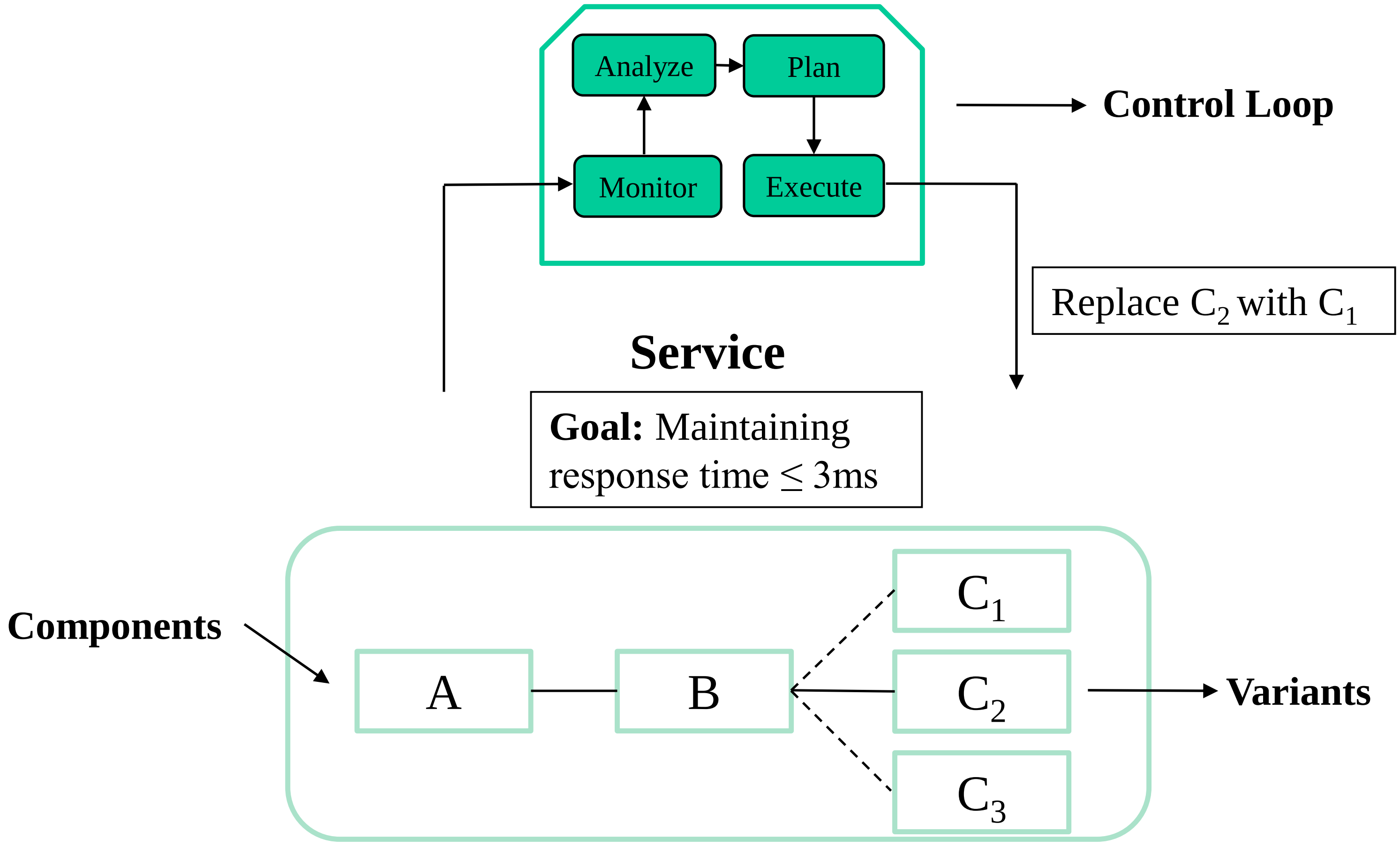
[13] Danny Weyns, Bradley Schmerl, Vincenzo Grassi, Sam Malek, Raffaela Mirandola, Christian Prehofer, Jochen Wuttke, Jesper Andersson, Holger Giese, and Karl M Goschka. On patterns for decentralized control in self-adaptive systems. In Software Engineering for Self-Adaptive Systems II, pages 76–107. Springer, 2013.

[14] Vivek Nallur and Rami Bahsoon. A decentralized self-adaptation mechanism for service-based applications in the cloud. IEEE Transactions on Software Engineering, 39(5):591–612, 2013.

[15] Ivana Dusparic and Vinny Cahill. Distributed w-learning: Multi-policy optimization in self-organizing systems. In Self-adaptive and Selforganizing Systems, 2009. SASO’09. Third IEEE International Conference On, pages 20–29. IEEE, 2009.

# Thank You

# Background: Action Selection



# Proposed Approach

**Lemma 1.** *Let $n_r, n_a$ be the total number of reward functions and the number of reward functions requiring weight update respectively. Let $r_i, th$ and $fr$ be the $i-th$ reward function value requiring weight update, reward function threshold and the fraction of the initial reward used to update the weight respectively. The weight of the $i-th$ reward function is updated with $\Delta w_i = k \times (th - r_i)$, where $k = \frac{1}{fr \times n_r \times (n_a \times th - \sum_{i=1}^{n_a} r_i))}$.*

- For the previous example –
  - $r_i = 0.3$
  - $th = 0.5$
  - $fr = 2.4$