TECHNISCHE UNIVERSITÄT DRESDEN

Detecting Emergent Interference in Integration of Multiple Self-Adaptive Systems

Somayeh Malakuti
Software Technology Group
Technical University of Dresden
Germany
26.08.2014

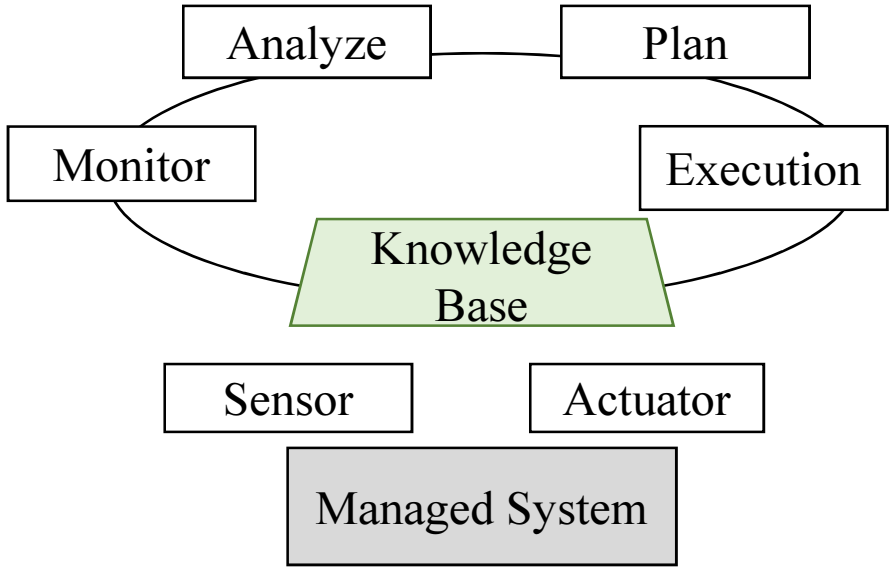
DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Outline

- Background
- Detecting Emergent Interference via Formal Verification
  - Modeling execution environment
  - Modeling self-adaptive systems
  - Verification of self-adaptive systems
  - Modeling and verifying SoS
- Conclusion and Future Work

Background

- Self-adaptive software systems (SAS's) are characterized by their ability to adapt according to changes in their environment.
- MAPE-K model is widely adopted to design and implement SAS's as a feedback control loop.

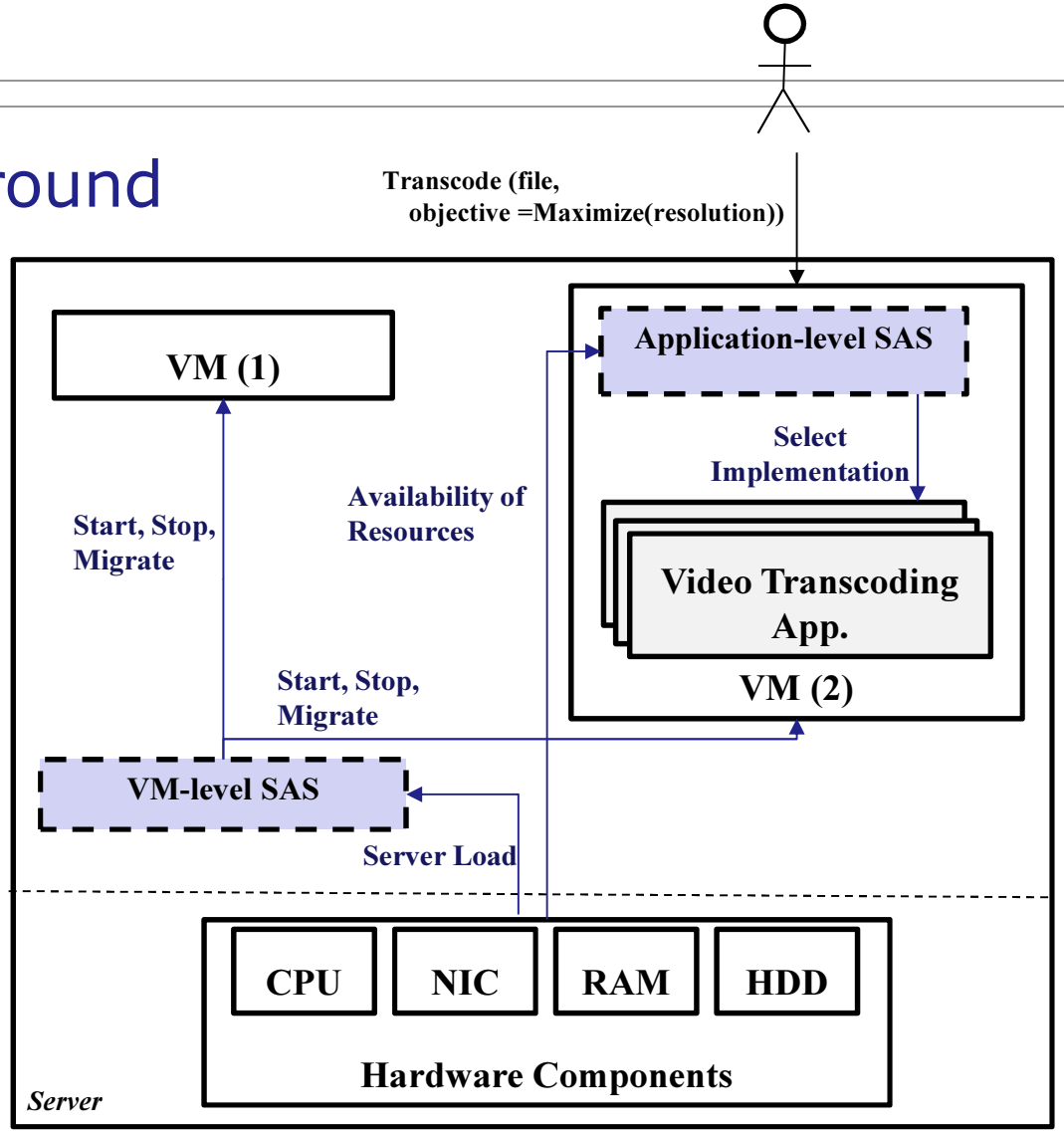


Background

- Within the context of the HAEC (Highly Adaptive Energy-efficient Computing) project, different research teams are investigating various kinds of SAS at different layers:
  - operating systems, database management systems, application software and virtual machines.
- The common goal of these SAS's is to improve the energy-utility of the system:
  - The system must serve the requests of users in an energy efficient way while fulfilling the desired utility objectives of the users.

TECHNISCHE UNIVERSITÄT DRESDEN

Background



Background

- Integrating multiple SAS's, which are developed and managed independently, is a novel example of SoS.
  - ***Is there any emergent interference as the result of such integration, which may negatively influence the overall energy-utility of the system***?

- Emergent behavior is an effect that is caused from the interactions of multiple constituent entities.
  - The term "emergent" implies that it is very hard or even impossible to reduce the behavior to the behavior of individual constituent entities.
- Various types and categories of emergent behavior are defined in literature. Examples are:
  - Expected and desirable emergent behavior
  - Expected and undesirable emergent behavior
  - Unexpected and desirable behavior
  - Unexpected and undesirable emergent behavior

TECHNISCHE UNIVERSITÄT DRESDEN

Detecting Emergent Interference via Formal Verification

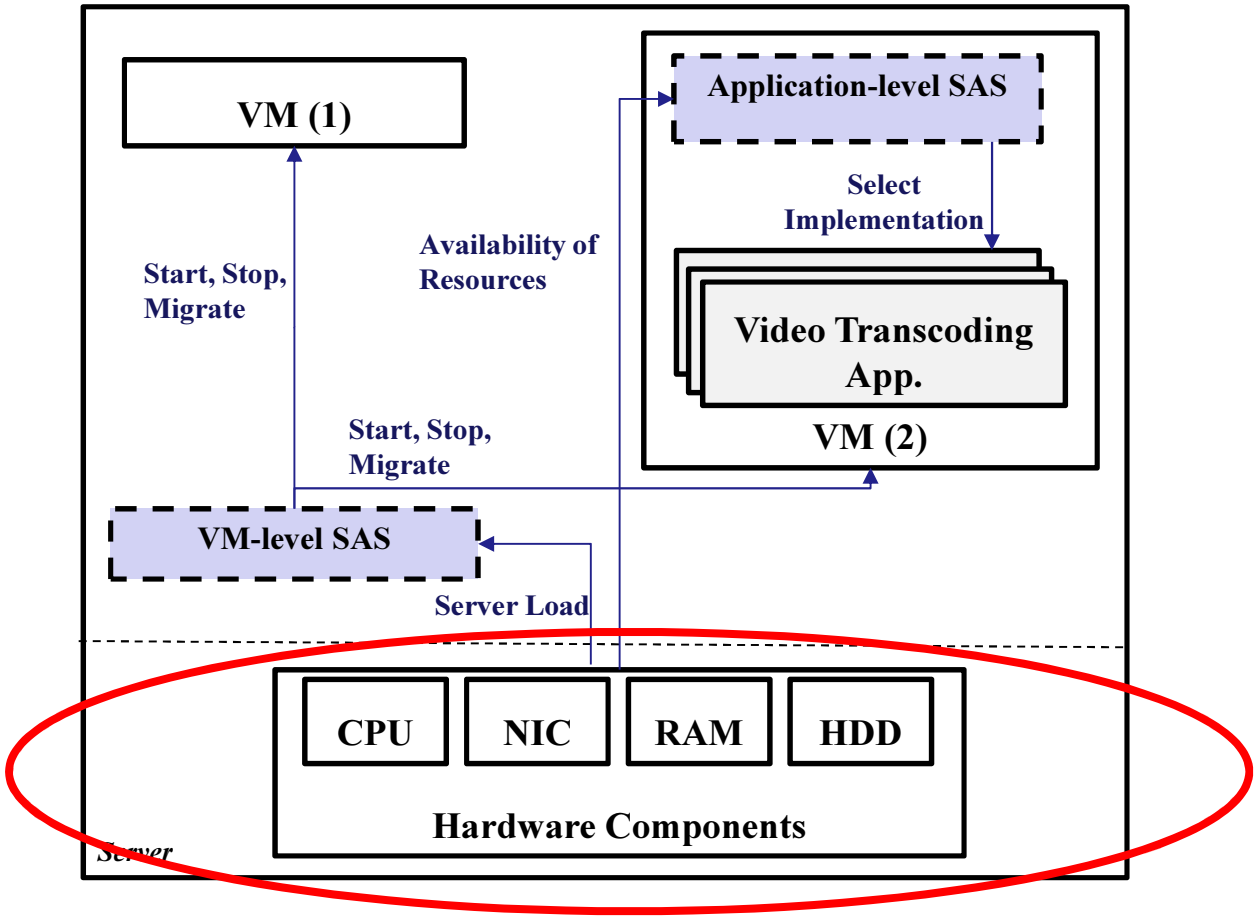
- Due to the complexity of SAS's, there is an inevitable need to provide methodological and tool support to detect emergent interference.
- There are various methods to detect emergent behavior:
  - Variable-based
  - Event-based

  ➢ These require ***a priori definition*** of emergent behavior

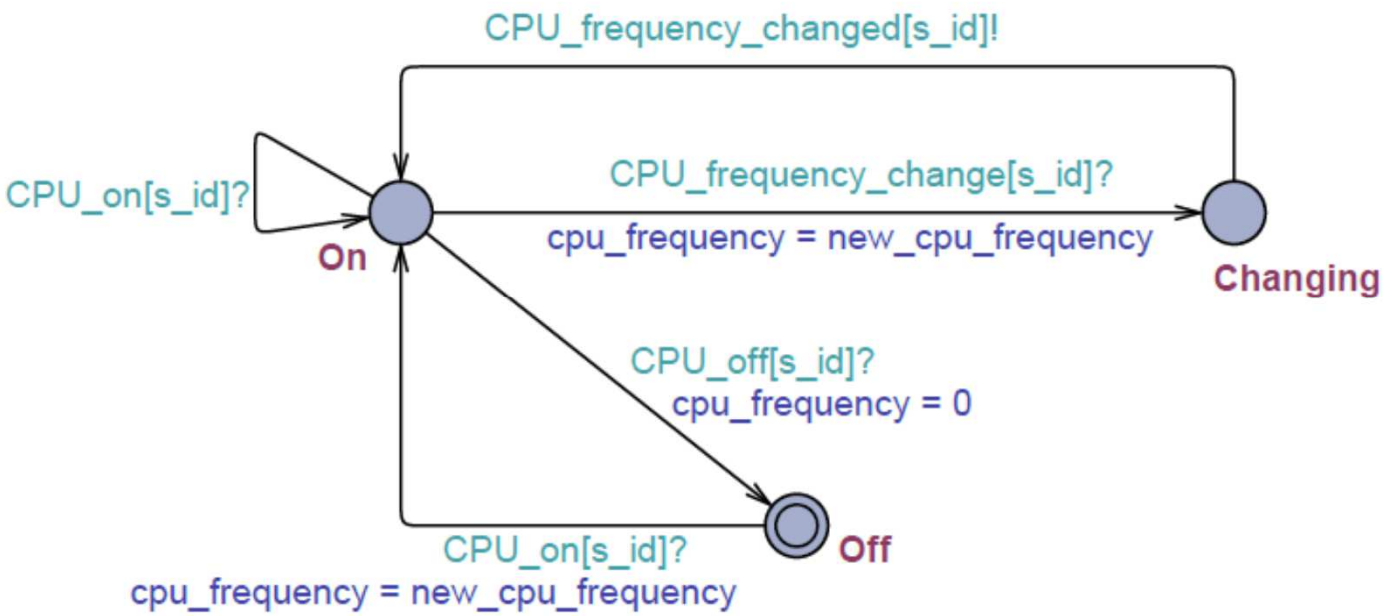
- Adopting formal modeling and model checking to detect emergent behavior can be regarded as a combination of the variable-based and event-based methods.
  - The behavior of each constituent system is modeled as an event-based automaton, and the states of each automaton is represented via a set of variables.
  - A specific global state of the system, which is automatically detected by a model checker, is regarded as emergent behavior.

TECHNISCHE UNIVERSITÄT DRESDEN

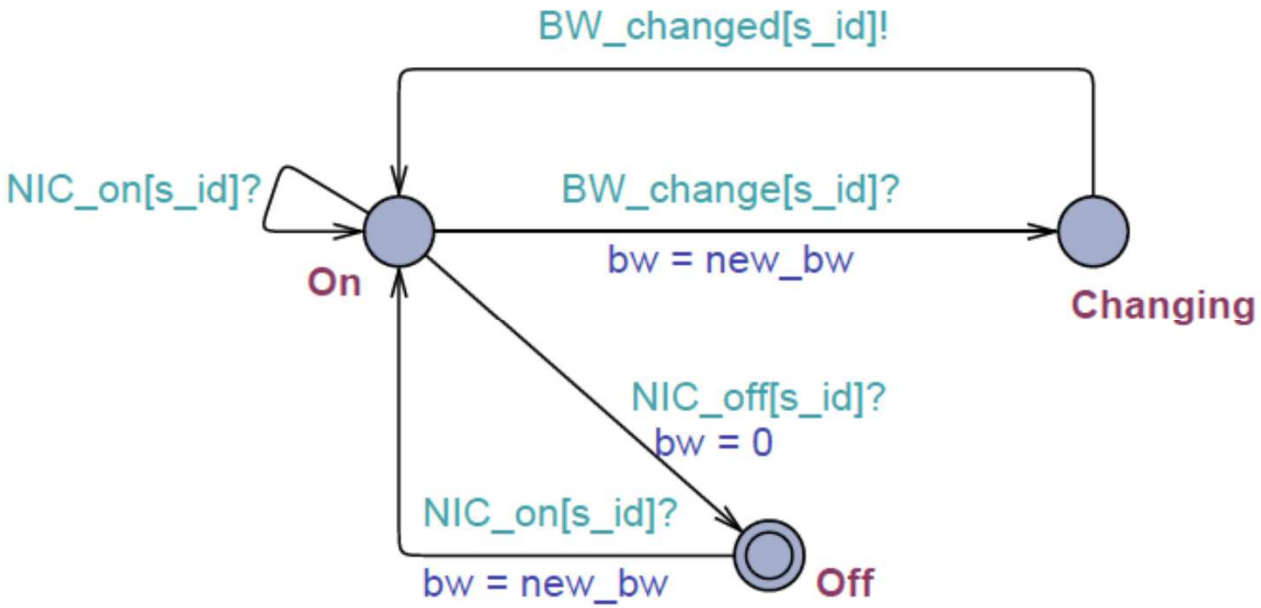
Step 1: Modeling Execution Environment



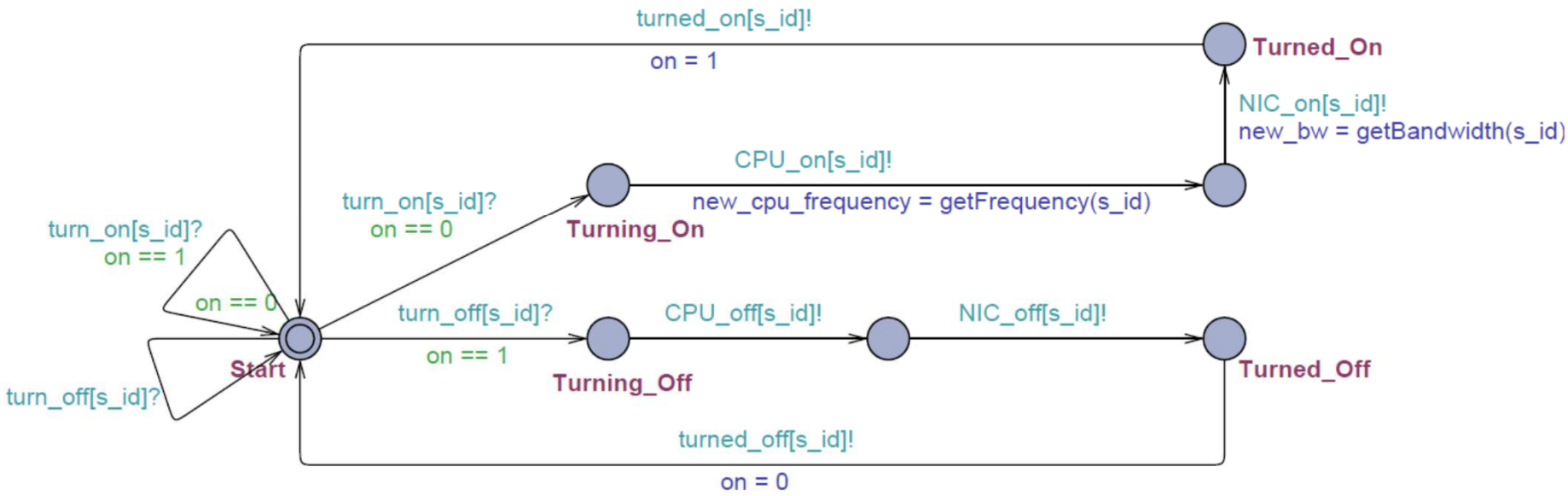
Step 1: Modeling Execution Environment (CPU)



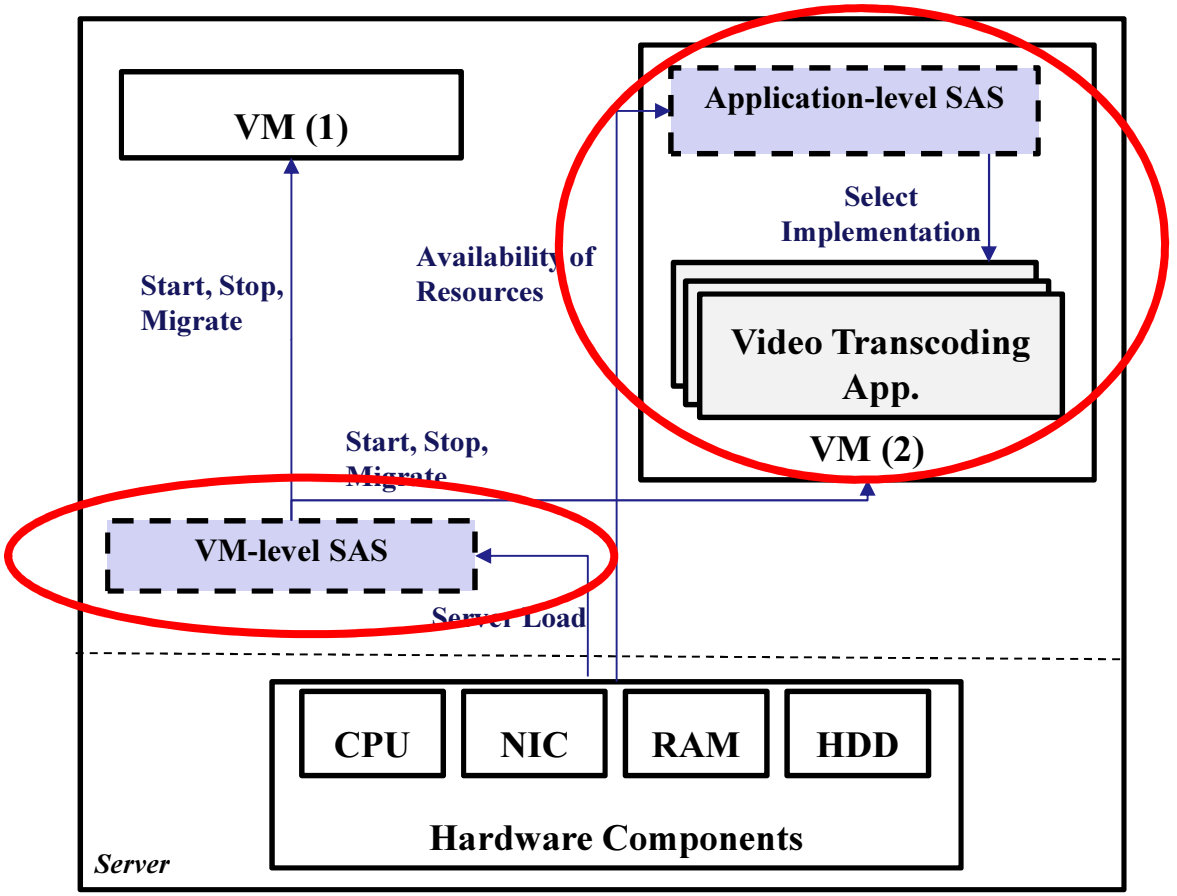
Step 1: Modeling Execution Environment (NIC)



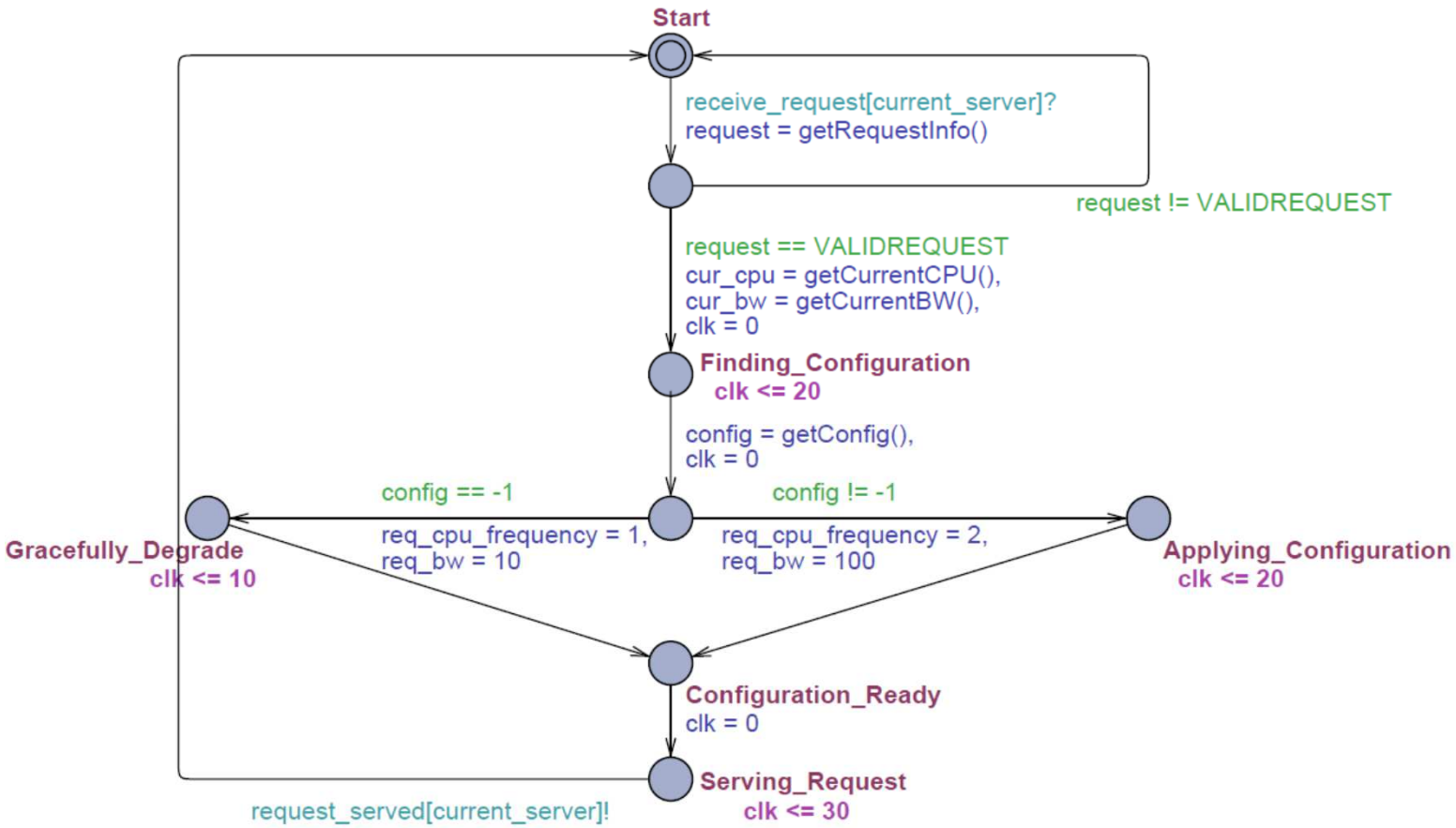
Step 1: Modeling Execution Environment (Server)



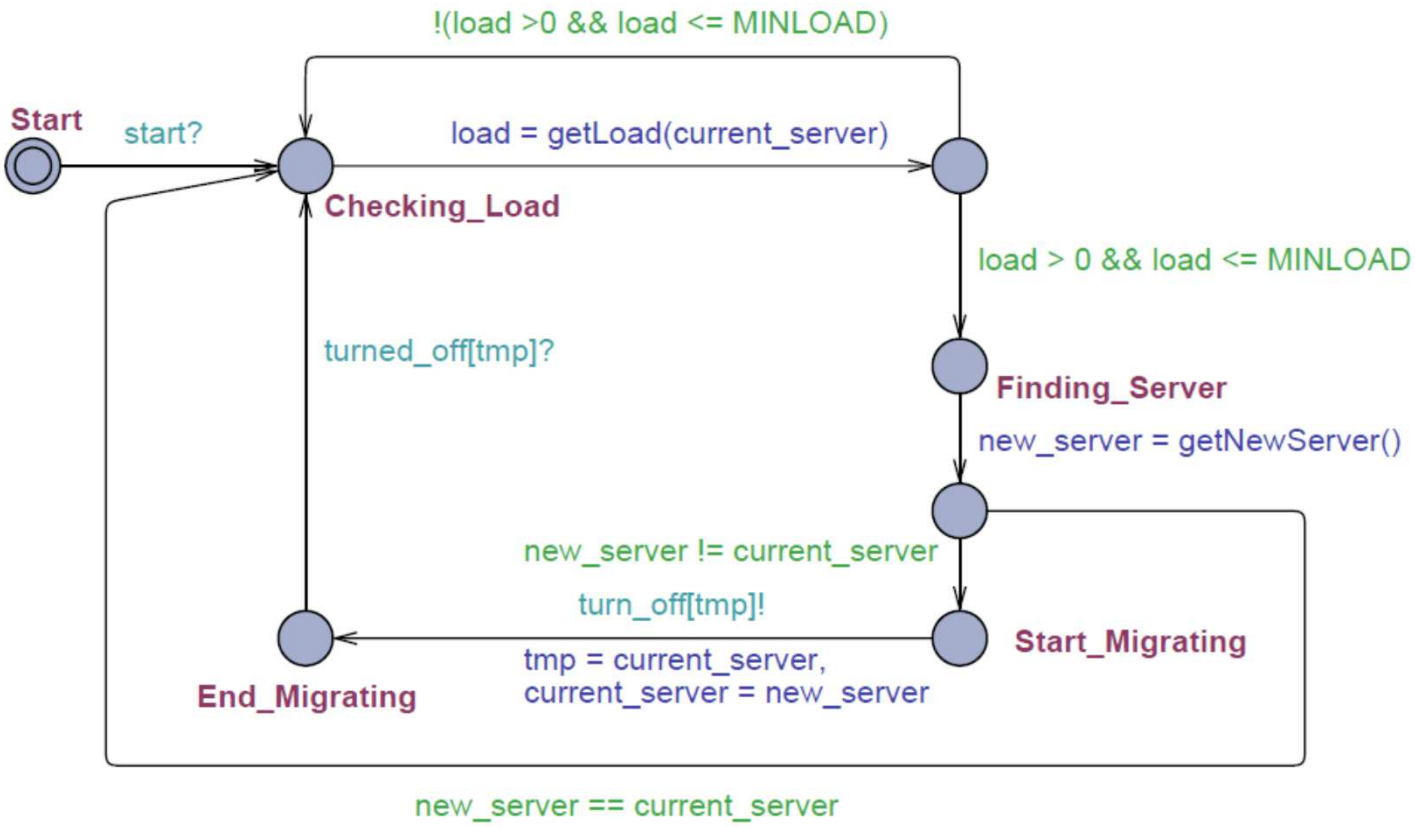
Step 2: Modeling Individual Self-adaptive Systems



Step 2: Modeling Individual Self-adaptive Systems



Step 2: Modeling Individual Self-adaptive Systems



Step 3: Verifying Individual Self-adaptive Systems

- If there is a valid request from user, the request will eventually be served by the application.

$$\text{E<>}\ (\text{ApplicationLevelSAS.request} == \text{VALID\_REQUEST} \\ \text{and ApplicationLevelSAS.Serving\_Request})$$

- It will never be the case that while configuring the application with a new implementation, the CPU frequency and network bandwidth goes below the amount being used for selecting the implementation.

$$\text{A[] not (ApplicationLevelSAS.Applying\_Configuration and} \\ (\text{CPU(current\_server).cpu\_frequency} < \text{cur\_cpu} \\ \text{or NIC(current\_server).bw} < \text{cur\_bw}))$$

- It will never be the case that while serving a request, the CPU frequency and network bandwidth goes below the amount required for serving the request.

$$\text{A[] not (ApplicationLevelSAS.Serving\_Request and} \\ (\text{CPU(current\_server).cpu\_frequency} < \text{req\_cpu\_frequency} \\ \text{or NIC(current\_server).bw} < \text{req\_bw}))$$

Step 3: Verifying Individual Self-adaptive Systems

- If a server is underutilized, and if a new server is found to migrate the VMs to, the migration will take place.

$$
\begin{aligned}
&\mathrm{E}\langle\rangle\ (\mathrm{load} >= 0 \text{ and } \mathrm{load} <= \mathrm{MINLOAD} \text{ and} \\
&\mathrm{new\_server} \mathrel{!}= \mathrm{current\_server} \text{ and} \\
&(\mathrm{VMLevelSAS.Start\_Migrating} \text{ imply} \\
&\mathrm{VMLevelSAS.End\_Migrating}))
\end{aligned}
$$

Step 4: Detecting Emergent Interference

- Defining an SoS in UPPAAL:

$$\textit{system}$$
$$\textit{CPU, NIC, Server, ApplicationLevelSAS, VMLevelSAS, RunTimeScenario;}$$

- Detecting emergent interference:

$$\mathrm{A} \models \rho \text{ and } (\mathrm{A} \otimes \mathrm{B} \otimes \dots) \not\models \rho$$

- If a property is violated, model checkers give a counter-example trace. This trace can be adopted as a means to understand the potential interference among constituent systems.

Step 4: Detecting Emergent Interference

- Emergent interference (1):
  - If the VM-level SAS migrates a VM while application-level SAS is selecting an implementation of the application, there may be a case that the available resources in the new server is below the amount of resources that the application-level SAS is considering for selecting the best implementation.
- Emergent interference (2):
  - If the VM-level SAS migrates a VM while a selected implementation is serving a request, there may be a case that the available resources in the new server is below the amount of resources that is required by the implementation.
- The execution of the application takes more time and energy on the new server.

Conclusion and Future work

- Suitable methodological and tool support are needed to detect and represent emergent behavior.
  - Like any other kind of behavior, we require to provide both design-time and runtime support to detect and represent emergent behavior.
  - Formal modeling can be regarded a suitable mechanism to exhaustively comprehend the behavior of SoS.
  - However, it may not scale up properly.

- To cope with emergent interference, it is necessary to break the autonomic behavior of each constituent SAS, and provide means to coordinate their interactions.
  - Modularity of each SAS must be preserved.
  - Loose coupling must be achieved in the integrations.
  - Coordination protocols must also be modularized.

- We plan to adopt the concept of ***event-based modularization*** to modularly integrate multiple constituent systems with each other.
  - An event-based language is being implemented based on this concept.