16th Workshop on Software Engineering Education and Reverse Engineering – Jahorina, 21-27.8.2016

Topic: Software Frameworks for Self-adaptive Systems

Vangel V. Ajanovski

Email: vangel.ajanovski@finki.ukim.mk
Web-site: https://ajanovski.info/

Faculty of Computer Science and Engineering
Saints Cyril and Methodius University
Skopje, Macedonia

Introduction and Concepts

- Origins of "self-adaptive"?
  - Earliest reference in IEEE databases in *Proceedings of the self adaptive flight control systems symposium*, 1959
- A more recent definition on Self-Adaptive (Software) Systems (abrev. SAS) will be more appropriate:
  - *"Self adaptive software is software that monitors its own operation, detects faults and opportunities, and repairs or improves itself in response to faults and changes. It effects the improvement by modifying or re-synthesizing its programs and subsystems, using a feedback control-system like behavior. "*
    - From: "Results of the Second International Workshop on Self-adaptive Software", *IWSAS 2001*, Lake Balaton, Hungary

Aim of the Presentation

- This presentation is intended as an advanced-level topic that introduces self-adaptivity concepts to students studying the following undergraduate courses, through discussions:
  - *Physical design and implementation of information systems*
  - *Software construction*
- Due to the nature of this presenation, the phrase *Software Frameworks* in the title is a generic term for:
  - Existing software products that ease the introduction of self-adaptivity
  - Design and implementation level patterns that are sufficiently developed and supported by existing software products

Change. Constant change.

- All software development methodologies discuss the eventuality of change and have some form of management process to deal with changes.
  - The idea is to be able to change. A lot.
  - In fact, the idea is that the only constant will be that everything can change.
- Why change?
  - No system is perfect.
    - Rules can be broken and exceptions accumulate.
    - Needs change. Requirements change.
    - Environment changes.

Change. Constant change...

- What kind of a change?
  - Change the Parameters (properties, attributes)
    - Not only to have parametrization, and certainly not only to be able to change parameter values (*adaptable*)
    - The system should be able to change its parameters, by itself, in order to improve some behaviour or solve an issue (*self-adaptive*)
  - Change the Structure (classes, components, associations)
    - Not only to have the possibility to change one implementation structure with another (*adaptable*)
    - The system should be able to change one implementation structure with another, by itself, in order to improve some behaviour or solve some issue (*self-adaptive*)

Change. Constant change...

- Who does in fact initiate and execute a change?
  - The system itself
- When can a change occur?
  - Whenever needed during run-time
  - When the system finds it necessary or that it can improve itself
- How could a system “know” that a change is needed or will be beneficial?
  - By monitoring its own operation
  - By computing or theorizing on alternatives
  - By testing alternatives in practice

Some implications

- Regarding Parametrization
  - Constants in the source should be prohibited
    - Allow all parameters to change value
  - Implement mechanisms to detect changes of values of the parameters
  - Implement mechanisms to define event-related behaviour as a reaction to parameter changes (event hooks)
  - Keep history of all changes (optional)

Some Other Implications …

- Regarding Structure
  - If the structure can change, than the behaviour of the system might change too.
  - What if it changes in the middle of the users' work process
    - imagine writing a document in Word 2003, scrolling the options in the Paragraph menu, with intent to put bullets on a list of pararaphs
    - suddenly the screen flashes and Word 2007 appears, and with it the traditional menu disappears altogether
  - So, users have to be informed of the change beforehand

Implications ...

- If the Users are informed beforehand, we give them choice:
  - Stop the sequence of actions and get back to the start of the use-case and start with the new implementation
    - We need mapping of use-cases and starting classes in each version of the source code and redirection mechanisms
  - Continue the sequence of actions within the older implementation
    - so we need to keep all versions of the structure of the software (and all versions of the structure of data)
    - new users will interact with the new structure
    - current users will be able to choose whether to interact with the old or new structure

Implications again …

- Seamless transition within the same use-case (optional). The user will continue working and experience migration from the old structure to the new structure without even knowing that it happened.
  - an implementation record of all use-case scenario steps, and the position of the user within the scenarios
  - mapping of each use-case scenario step to the history of changes of implementation details of the classes that implement the step,
  - so that the scenario will not be implemented as a path among the classes in one single source version,
  - but as a path among several source versions

Implications ...

- Users have to be able to learn to use the system as it changes, by themselves
  - It is not possible to organize training every couple of minutes
  - so, changes have to be
    - mostly small (atomic) and
    - happen in an orderly way for the user to be able to grasp them
  - which means that their should be restrictions on
    - how often is the allowed to change itself
    - to what extent is it sensible for the system to change itself at given moment

The Role of a Software Framework

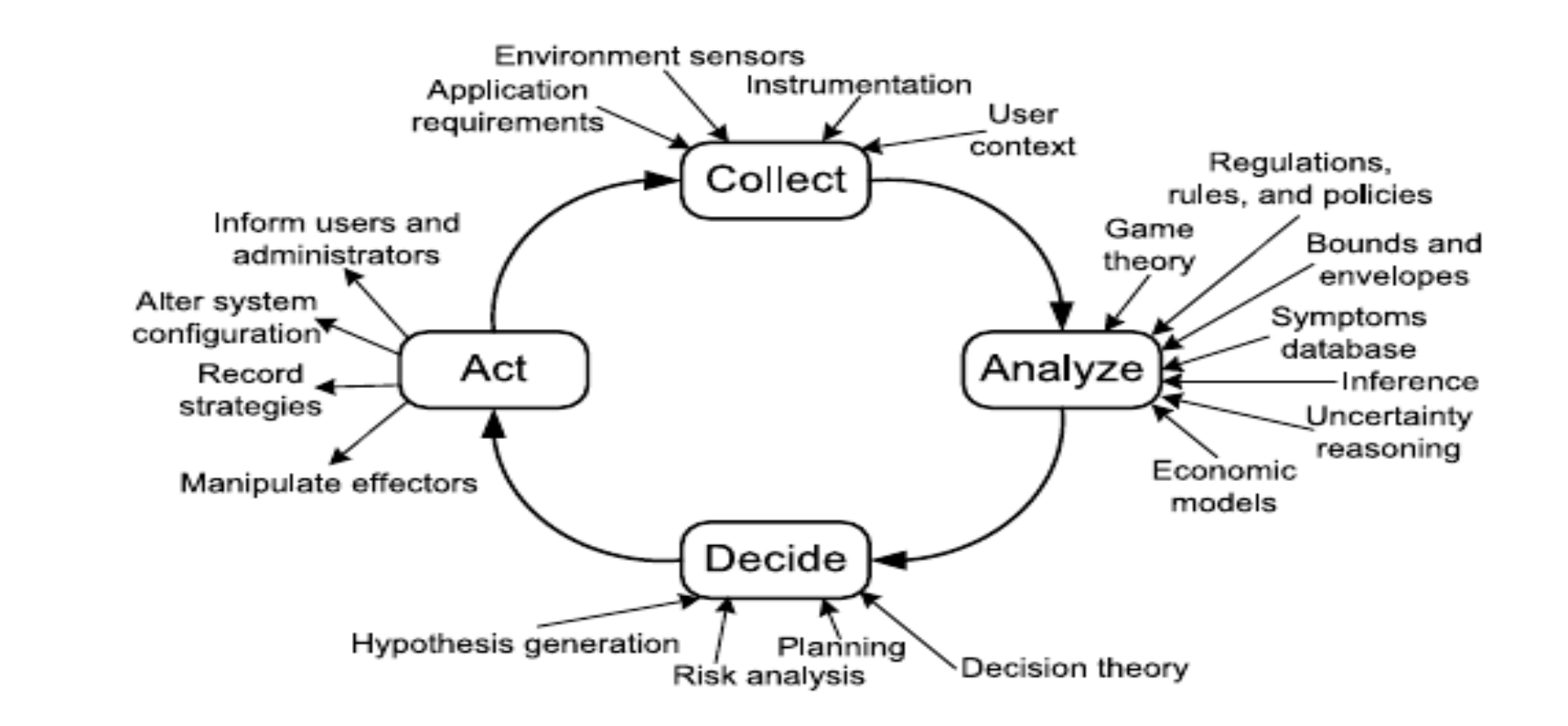
- Obviosly there are many aspects that have to be taken care of, where a good software framework would be beneficial
  - Regarding the objectives the system should achieve
    - Evolution, Flexibility, Duration, Multiplicity, Dependency
  - Regarding the causes of adaptations
    - Source, Type, Frequency, Anticipation
  - Regarding the reactions towards change
    - Type, Autonomy, Scope, Duration, Timeliness, Triggering
  - Regarding the impact of the adaptation
    - Criticality, Predictability, Overhead

Available Software Frameworks

- An investigation process is under way
  - Investigation started within SEAMS and IWSAS, SASO, ACM Digital Library and Web searches
    - Gathered references: 415
    - Short list round 1 – for further investigation: 163
    - Short list round 2 – based on abstract and summary: 125
    - Short list round 3 – relevant papers/presentations: 45
  - Software frameworks: **too many and too few at the same time***

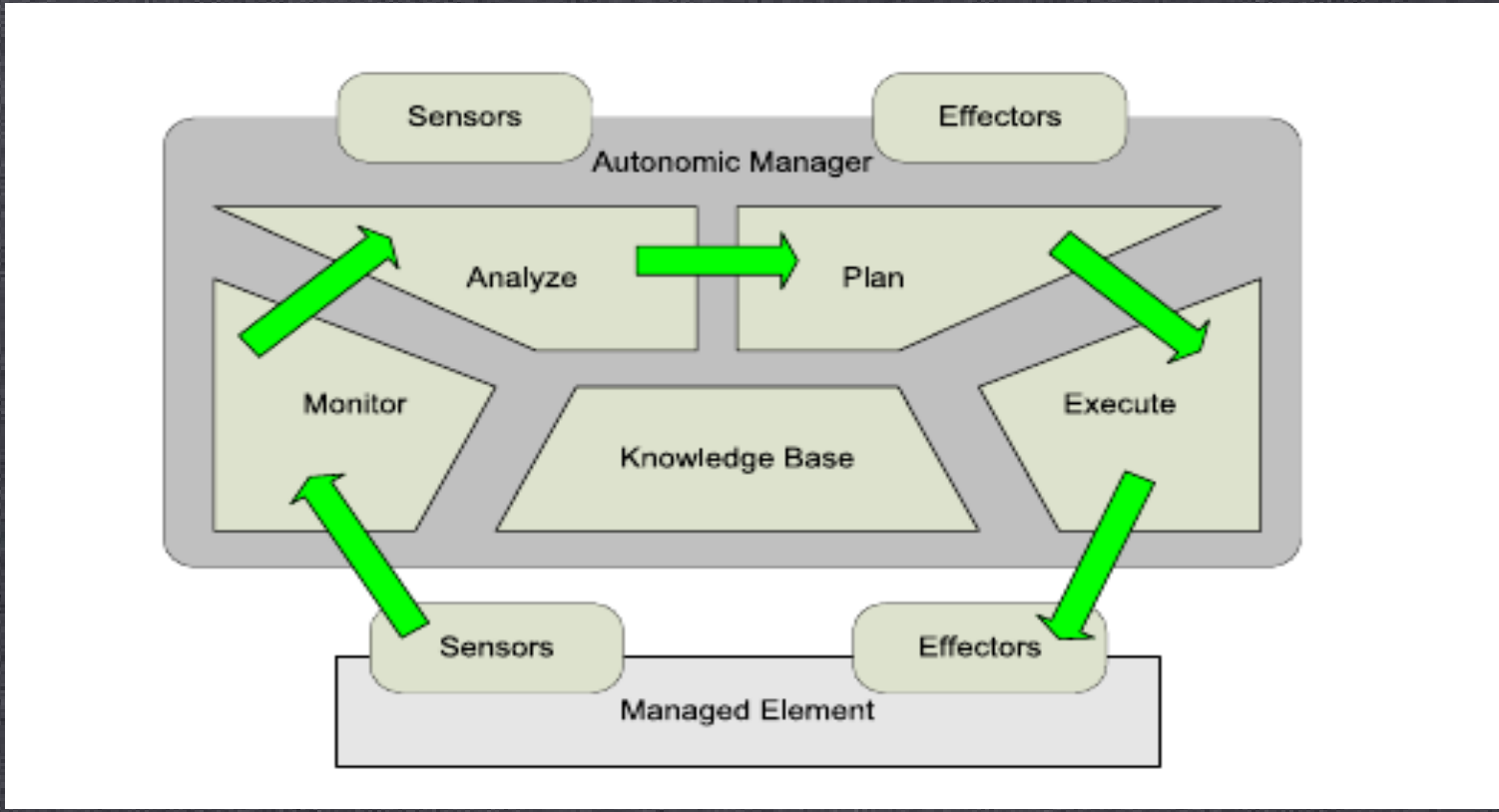
*having in mind the definition set at the beginning of the presentation

Autonomic Control Loop



- Dobson, S., Denazis, S., Fernández, A., Gaïti, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N., Zambonelli, F.: A survey of autonomic communications. ACM Transactions Autonomous Adaptive Systems (TAAS) 1(2), 223–259 (2006)

IBM's Autonomic Element



- IBM Corporation: An architectural blueprint for autonomic computing. White Paper, 4th edn., IBM Corporation, http://www-03.ibm.com/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf

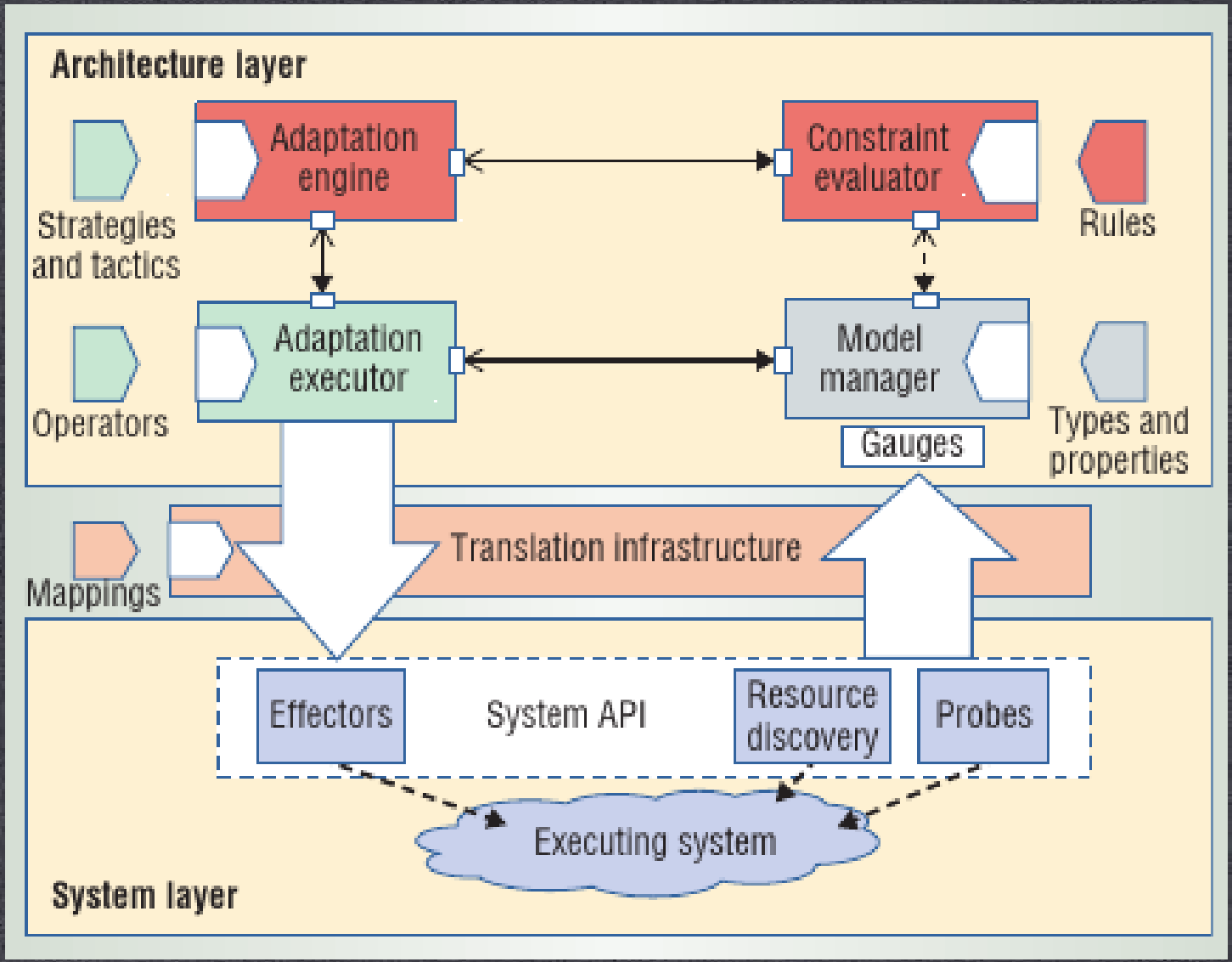
Available Software Frameworks...

- Many are conceptual level only
  - Purely theoretical, formal language frameworks, or
  - Frameworks conceptualized in order to provoke discussion
- Some are at implementation level and even have prototypes
  - Mix of various legacy tools and programming languages
  - Some use custom-built tools
- Key problem - repeatability
  - Code that was used is hard to find
    - Many web-sites are now forgotten unmaintained, or do not work
  - A database listing all such projects and status will be important

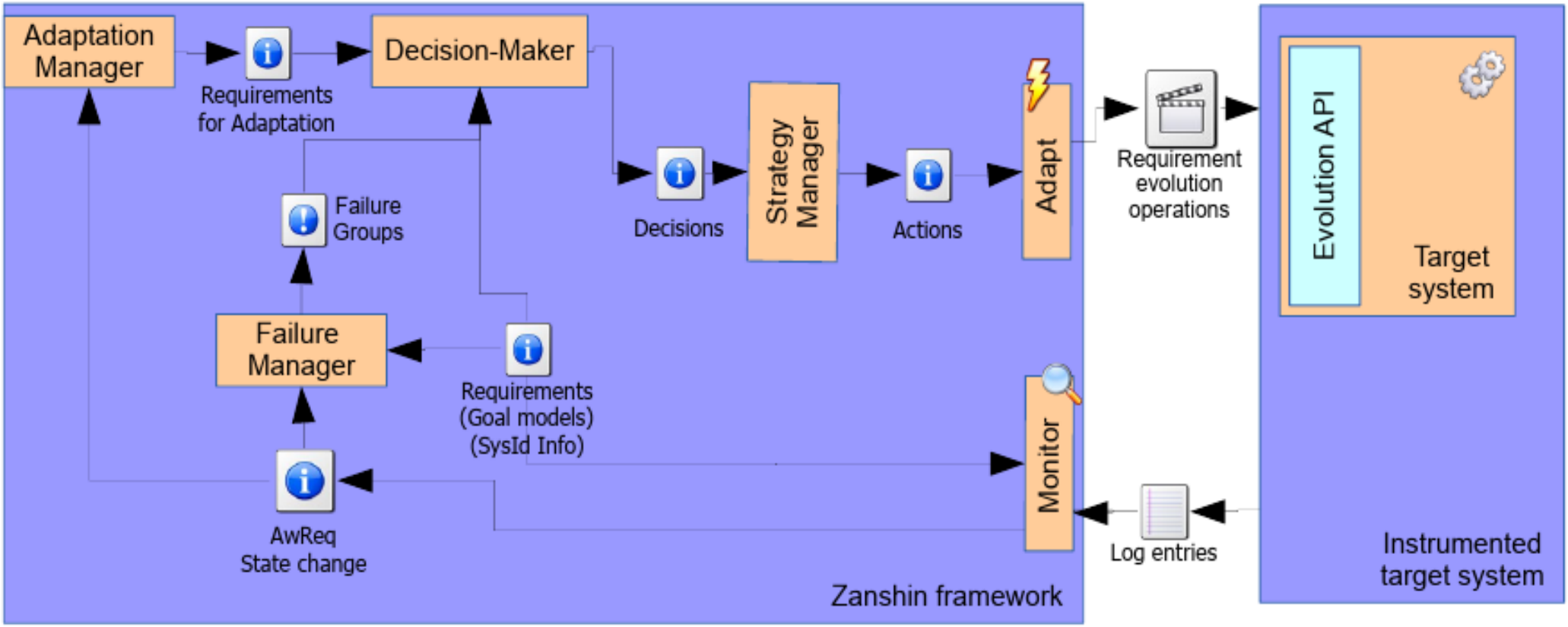
RAINBOW

- Framework that has most references
  - Base on top of which many have researched
- Summary: "*To reduce the cost and improve the reliability of making changes to complex systems, we are developing new technology supporting automated, dynamic system adaptation via architectural models, explicit representation of user tasks, and performance-oriented run-time gauges. This technology is based on innovations in three critical areas:*
  - ***1. Detection:*** *the ability to determine dynamic (run-time) properties of complex, distributed systems.*
  - ***2. Resolution:*** *the ability to determine when observed system properties violate critical design assumptions.*
  - ***3. Adaptation:*** *the ability to automate system adaptation in response to violations of design assumptions.*"
- Website: http://www.cs.cmu.edu/~able/research/rainbow/

RAINBOW Architecture



ZANSHIN



- Website: https://github.com/sefms-disi-unitn/Zanshin

Exemplars

- How do we know that a framework really achieves something
  - We need reference
  - The community has developed and provided several exemplars with typical problems, scenarios, test-cases that can be used as benchmarks
  - Website: https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/
    - ZNN – news site
    - Tele Assistance System (TAS)
    - Feed me, Feed me (FmFm) – conserning quality of food
    - Automated Traffic Routing Problem (ATRP)

Assignments for Students

- Self-adaptive systems are complex systems, and can be too advanced a topic for undergraduates,
  - but the topic is important and should be covered, since they will be involved with the topic in the near future
- Possibilities:
  - Discussion seminars (investigate, present and discuss one framework per lesson, to get the feeling of different types of solutions and ideas)
  - Hands-on projects (introduce some/all concepts from a fw into a small application that the students already have)
  - Work long-term under supervision (project spanning several yr)

Questions and Comments?