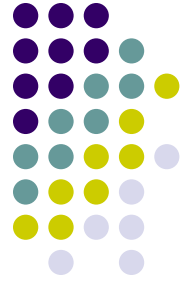
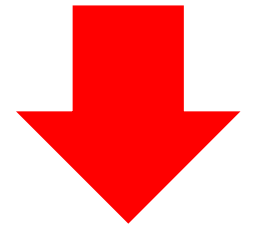


Welcome to **SENG 480A / CSC 485A / CSC 586A** **Self-Adaptive and** **Self-Managing Systems**



Dr. Hausi A. Müller
Department of Computer Science
University of Victoria



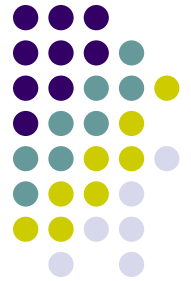
<http://courses.seng.uvic.ca/courses/2015/summer/seng/480a>
<http://courses.seng.uvic.ca/courses/2015/summer/csc/485a>
<http://courses.seng.uvic.ca/courses/2015/summer/csc/586a>

Announcements



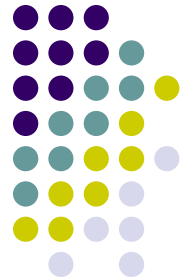
- A4
 - Due Friday, July 31
 - Adaptive control
- Marks
 - Midterm 2 will be posted soon
- Grad project
 - Slides due Friday, July 24
 - Presentations Mon, July 27 and Thu, July 30
 - All students are expected to assess the presentations as part of their course participation mark
- Teaching evaluations
 - Complete CES at <http://ces.uvic.ca>

Guidelines for Grad Student Presentations



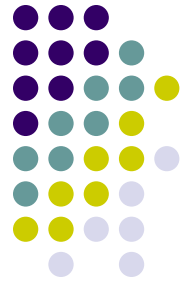
- Format of presentation
 - Presentation 10 mins
 - Q&A 5 mins
 - Practice talk (!!)
 - Practice of the best of all instructors
- Slides
 - High quality and polished
 - Submit slides by July 24 to instructor for approval
 - Submit final slides 1 day after presentation for posting on website
- Talk outline
 - Motivation
 - Problem
 - Approach
 - Contributions of the paper
 - Relation to what we learned in the course so far
- Assessment
 - All students have to fill out an evaluation form
 - Counts towards class participation

Presentation Assessment



Evaluator's name:		
Graduate students:		
Quality of presentation		
Did I learn something? Did the presentation stimulate my interest?	5	
Do I know now what the paper is all about?	5	
Does the presenter know the subject well?	5	
Presentation style: main points reiterated; positive attitude; excited about the subject.	5	
How did the presenter perform in the Q&A session?	5	
Subtotal	25	
Other comments		
<div style="background-color: yellow; padding: 10px; text-align: center;"> <h3>July 27 and July 30 CSC 586A Presentations</h3> </div>		

Graduate Student Research Paper Presentations



- [Brun, Y., Di Marzo Serugendo, G., Gacek, C. Giese, H. Kienle, H.M., Litoiu, M., Müller, H.M., Pezzè, M., Shaw, M.: Engineering Self-Adaptive Systems through Feedback Loops. Software Engineering for Self-Adaptive Systems, pp. 48–70 \(2009\)](#) — **Presentation by Simar Arora Khushboo Gandhi: July 27**
- [Garlan, D., Cheng, S.-W., Huang, A.-C., Schmerl, B., Steenkiste, P.: Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. IEEE Computer 37\(10\):46–54 \(2004\)](#) — **Presentation by Stephan Heinemann and Waseem Ullah: July 27**
- [Oreizy, P., Medvidovic, N., Taylor, R.N.: Runtime Software Adaptation: Framework, Approaches, and Styles. In: ACM/IEEE International Conference on Software Engineering \(ICSE 2008\), pp. 899–910 \(2008\)](#) — **Presentation by Sumit Kadyan and Adithya Rathakrishnan: July 27**
- [Kramer, J., Magee, J.: Self-Managed Systems: An Architectural Challenge. In: ACM /IEEE International Conference on Software Engineering 2007 Future of Software Engineering \(ICSE\), pp. 259–268 \(2007\)](#) — **Presentation by Ernest Aaron and Harshit Jain : July 27**

ENGINEERING SELF- ADAPTIVE SYSTEMS THROUGH FEEDBACK LOOPS

PRESENTED BY:

SIMAR ARORA (V00824821)

KHUSHBOO GANDHI (V00794157)

FOR: SAS CSC 586, SUMMER 2015

AUTHORS

- **Yuriy Brun:** University of Southern California, Los Angeles, CA, USA
- **Giovanna Di Marzo Serugendo:** Birkbeck, University of London, London, UK
- **Cristina Gacek:** University of Newcastle upon Tyne, Newcastle upon Tyne, UK
- **Holger Giese:** Hasso Plattner Institute at the University of Potsdam, Germany
- **Holger Kienle, Hausi Muller:** University of Victoria, British Columbia, Canada
- **Marin Litoiu:** York University and IBM Canada Ltd., Canada
- **Mauro Pezz`e:** University of Milano Bicocca, Italy and University of Lugano, Switzerland
- **Mary Shaw:** Carnegie Mellon University, Pittsburgh, PA, USA
mary.shaw@cs.cmu.edu

Introduction to Self Adaptive Systems

- “Self” prefix indicates that the systems decide and adapt autonomously (i.e., without or with minimal interference)
- Evolution of software engineering techniques require to keep up with ever-changing landscapes
- Characterization of Self-adaptive systems:
 - Centralized, top-down (self-managing: explicit adaptation mechanisms, central control)
 - Decentralized, bottom-up (self-organizing: emergent self-adaptation, local information based decision control)

For Example: The Web

Refer: Pages 49-50

Core of SAS: Feedback Loops

- Inspiration derived from control theory and nature
- Control Engineering elevates FEEDBACK LOOPS as first class entities
- Focus of this paper: Relevance of feedback loops towards engineering of SAS
- Importance of dynamic architecture emphasized by: Magee and Kramer
- Self-adaptive systems : Design decisions are handled at runtime to control dynamic behavior
- Lehman's work on software evolution points towards the importance of multi-loop and multi-level feedback system

Refer: Pages 50-51

Generic Feedback Loop

- A feedback loop typically involves four key activities: collect, analyze, decide, and act.
- Derived from 1980's AI community's sense-plan-act approach to control autonomous mobile robots
- Generic feedback loops are unidirectional single control loop, in contrast: multiple separate loops are typically involved in a practical system.

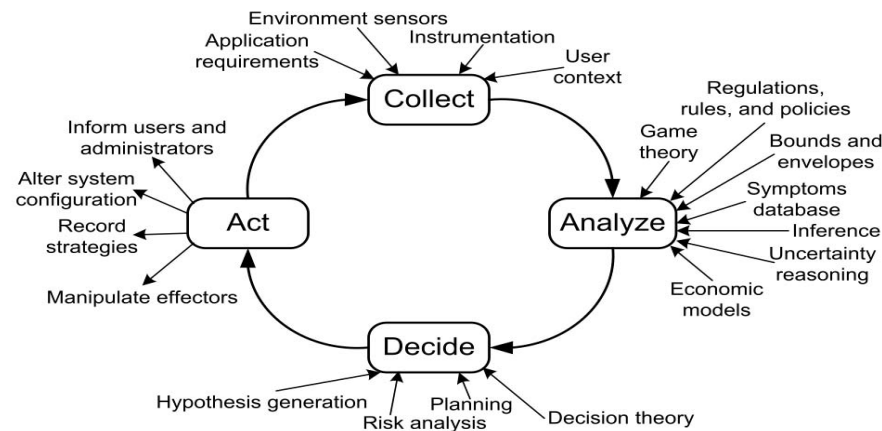


Fig. 1. Autonomic control loop [21]

Refer: Pages 52-53

Feedback loops in Control Engineering

- Key reason for using feedback : counter-measure disturbances or noise in variables or imperfections in the models of the environment
- Control theory provides well-established mathematical models, tools and techniques for analysis of system parameters that can be applied for SAS
- Example: feedback systems are used to manage QoS in web server farms.



Fig. 2. Feedback control loop

Adaptive Con

- Adaptive control in control theory: modifying the model or the parameters of the controller, Second control loop is installed on top of the main controller
- The MRAC strategy relies on a predefined reference model (e.g., equations or simulation model) which includes reference inputs.
- The MIAC strategy builds a dynamical reference model by simply observing the process without taking reference inputs into account.

Refer: Pages 54-55

Feedback Loops in Natural Systems

- Numerous examples available in nature: social insect behaviors (e.g., ants), immune systems, etc.
- Highly complex and decentralized
- Resilient with built-in error correction, fault tolerance, and scalability
- Countering attacks with reduced performance instead of system wide failure
- Two types of feedback in nature-
 1. Positive: creates amplified disorder
 2. Negative: counters the amplification
- Both combine to enable system stability: positive feedback pushes system boundaries and negative feedback stabilizes it

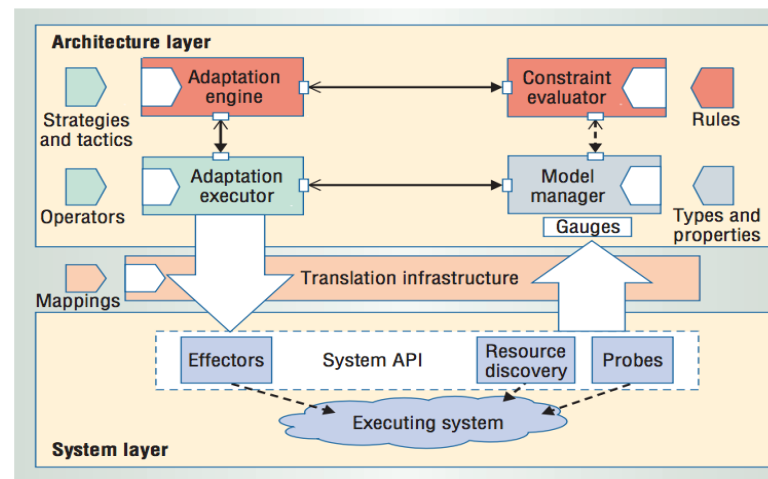
Feedback Loops In Software Engineering

- Feedback loops: often hidden, abstracted, dispersed, or internalized.
- Lack of a notation leads to the absence of explicit usage and control (e.g.: UML)
- Both control loops and their properties should be explicit as advocated by Garlan
- Visibility of feedback loops is essential :
 - Understanding SAS
 - Building SAS with crucial properties for the guaranteed adaptive properties
- IBM's Autonomic Computing: Major breakthrough in making feedback loops explicit.

Refer: Pages 57-58

Solutions Inspired by Expert Control

1. **MAPE-K**: Autonomic element : building block for realizing the four self-* properties.
2. **Garlan's Rainbow architecture**: reusable architecture to support self-adaption
(Reference: <http://repository.cmu.edu/cgi/viewcontent.cgi?article=1663&context=compsci>)



2. **Shaw's work summarizes:** "When the execution of a software system is affected by uncontrollable external disturbances, it indicates that a control paradigm should be considered for the software architecture" (Reference: <http://www.cs.cmu.edu/~Compose/ByndObj.pdf>)

Refer: Pages 58 to 61

Solutions Inspired by Natural Systems

- Challenge: To utilize biological systems knowledge to design and build architectures and programming tools
- Motivated Solutions:
 1. Tile software architectural style: inspired by feedback mechanism of crystal growth to allow fault and adversary tolerance
 2. Process schedulers, Network routing protocols inspired by mechanisms used for direct communication by schools of fish/flock of birds
 3. Stigmergy used by ants, wasps for indirect communication :
 - Research in swarm robotics to solve static and dynamic optimization problems
 - Coordinating unmanned vehicles

Refer: Pages 62-63

Challenges

- Modeling: Explicit control loop modeling and exposure of self-adaptive properties
- Control Loops: Reference library of control loop types, interactions and mechanisms
- Architecture and Design: Decoupling of intertwined control loops, Hierarchical organization, Requirement of reference architectures
- Unintended-Interaction Detection: Bifurcation and Integration of independent subsystems
- Maintenance: Increased complexity of dynamically variable systems
- Middleware Support: Need for foundation of standardized interfaces and middleware
- Verification and Validation: authenticate and substantiate effects of feedback to ensure stability
- Reengineering: Reconfiguring legacy systems into self-adaptive systems by medium of control mechanism is costly
- Human-Computer Interaction: Parallel control abilities to the user for explicit and legal commandment

Refer: Pages 63 to 66

Conclusion

- Feedback loops are the lynchpin in software engineering for SAS.
- Inspiration derived from natural systems and control theory regarding feedback loops has been helpful
- Recognition of such key concepts and addressing the discussed challenges is relevant to developing complex self-adaptive systems, models, architectures, etc.



Thank you!



**KEEP
CALM
AND
ASK
QUESTIONS**

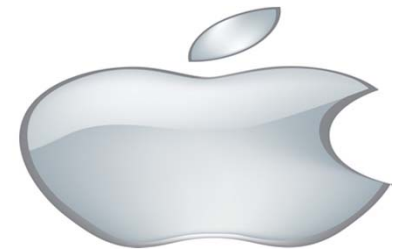


Runtime Software Adaptation Framework, Approaches and Styles

Presentation by Adithya Rathakrishnan, Sumit Kadyan
Department of Computer Science

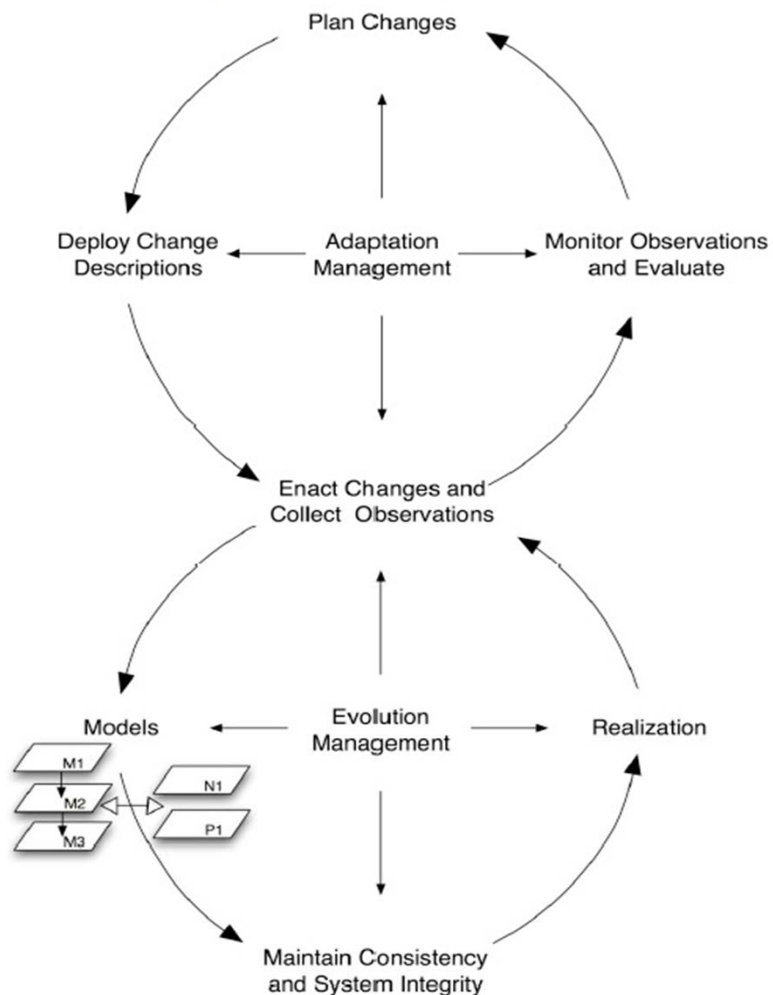
Introduction/Motivation

- ❖ Change: Intensive software use breeds/needs change.
- ❖ Ex: Business communication by email services, security patches and OS, Online banking system.
- ❖ Need of runtime evolution at low costs and without incurring downtime.
- ❖ Our first instinctive approach towards RE would be fault tolerant hardware, hot pluggable devices, etc.
- ❖ Runtime Evolution in Software Systems is complex.
- ❖ Software architecture plays a valuable role in the runtime evolution.



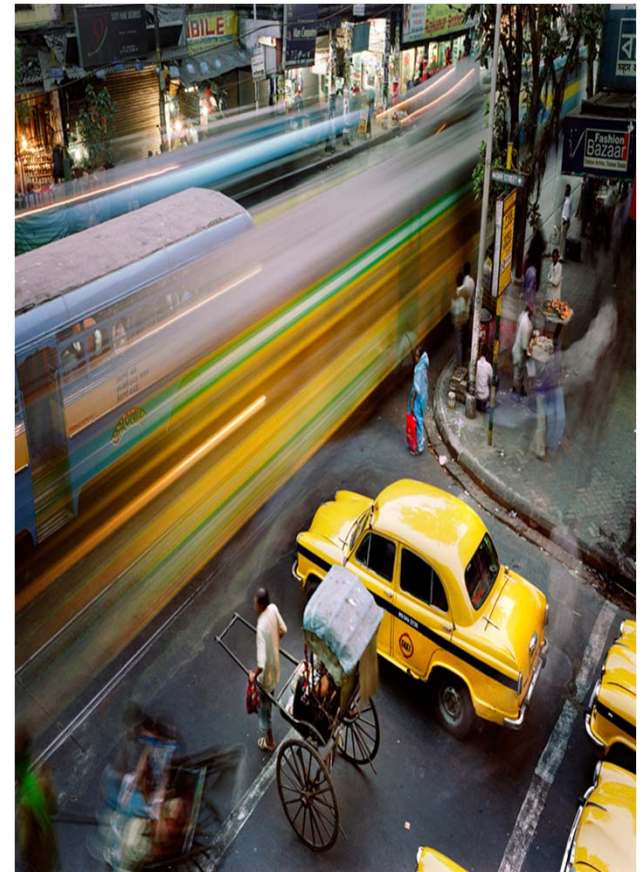
Unifying Framework (MAPE-K LOOP)

- Changes to the model's behavior.
- Change's to the program's state.
- Changes to the execution context of the machine running our program.
- Asynchrony of change.



Dynamic Adaption Model - 1

- ❖ Dynamic software evolution not only required software architecture but required software engineering.
- ❖ Architectural designs are CHAM and GRAPH Grammars, ADL are Rapide, Darwin and Dynamic Wright.
- ❖ WHY these models failed?
 - Models were not accompanied by actual system level facilities for dynamic evolution
 - Dynamism supported was overly constrained



Dynamic Adaption Model - 2

- ❖ Figure 8 model proposed an approach for above mentioned limitation
- ❖ It argued that dynamic system evolution must be properly planned and carefully executed.
- ❖ Then came 'Rainbow' which suggested: Maintaining and implementing by performing on the fly analysis
- ❖ Layered reference architecture for autonomous or self managed system proposed by Kramer and Magee

Examples Research Projects/Commercial Solution

Aura	Architectural style and supports middleware platform	Focuses on context awareness and context switching ,supports software component mobility	Provides an infrastructure with self -monitoring for detecting changes in requirement
Mobipads	A class of mobile middleware platform	Dynamically reconfigures usage of resources to meet QOS	Provides active deployment of middleware but doesn't provide application level dynamic capabilities
Skype	It is build on peer to peer architecture	Dynamically designating a status of supernode to a regular node or demoting a supernode to a regular node resulting in modification of current topology on the fly	Inherits from architectural style, p2p, the ability to support runtime addition, removal, and even physical movement of hosts.
MapReduce	Google's infrastructure for processing and generating large data sets.	MapReduce users specify a map function which essentially divides a large data set into a number of subsets that can be processed in parallel.	Supports dynamic adaptation targeted at handling node failures by automatically re-routing data from failed node to a live node

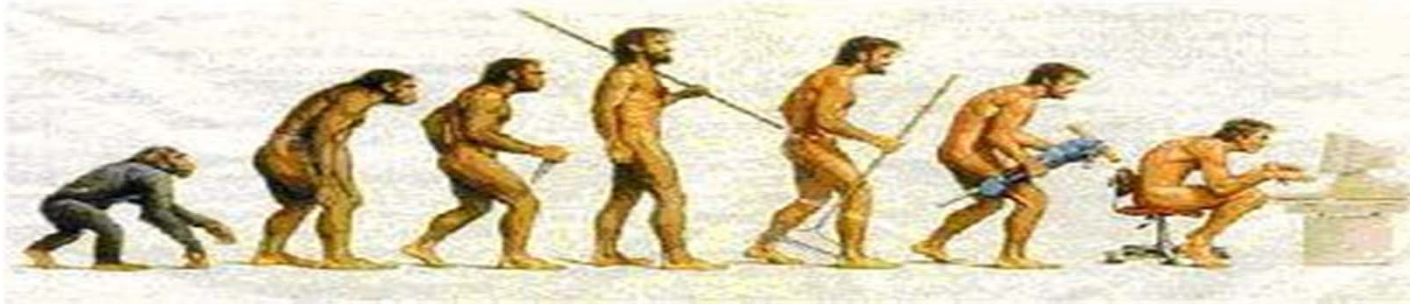
Conference, Symposia, Workshops

- Dynamism as Primary Focus
 - ◆ International Conference on Autonomic Computing (ICAC).
 - ◆ Software Engineering for Adaptive and Self-Managing Systems (SEAMS). (WS)
 - ◆ Dagstuhl Seminar on Software Engineering for Self-Adaptive Systems.
- Dynamism as Means or By-Product.
 - ◆ Percom- pervasive computing and communication.
 - ◆ Working Conference on Component Deployment - software deployment (Pervasive)
 - ◆ Middleware
- Dynamism in our Flagship Conferences

Making Adaptation Easier

- ❖ *Focus on building system that makes adaptation easier than otherwise .*
- ❖ *3 key points for making Adaptation*
 1. *Making the parts subject to change identifiable and manipulable.*
 2. *Controlling interaction with parts subject to change.*
 3. *Managing state.*





Necessities of adaptation

- ❖ Identifying the element to be changed is necessary ,so is supporting change which is achieved by encapsulation.
- ❖ **Controlling Interaction** i.e. interaction between elements.
- ❖ 2 general strategies that are apparent for adaptation are Delay Bindings and Explicit events/messages in communication

Managing state

- ❖ Addressing the state of computational element or communication element when they are changed.
- ❖ A strategy suggested is :
 1. Components present an interface that forces it to checkpoint its state externally
 2. Another interface that causes the component to initialize itself from external store.
- ❖ OR a better strategy is to require components to maintain their state externally .
- ❖ For example REST style.



Adaptation styles: Past, Present and Future

- ❖ Proto-runtime evolution: Pipe and Filter.
- ❖ Dynamic pipe and Filter: Weaves
- ❖ Events and notifications: Field and Publish-Subscribe
- ❖ Event- based components and connectors: C2
- ❖ Dynamism through replication: Tile Style.
- ❖ Externalization of state: Representational State Transfer (REST).
- ❖ Future: CREST.



REST and Computational REST or CREST

- ❖ Most successful architectural style in supporting runtime evolution of large -scale application.
- ❖ Key abstraction of information is a resource,named by an URL.
- ❖ Interaction are context free and intermediaries are promoted.
- ❖ Computational REST or CREST is the future.
- ❖ CREST extends URLs to locate active computations and their execution environments.
- ❖ Ex: A smart energy microgrid could intelligently balance supply and adapt in the face of brownouts, environmental disasters,etc.

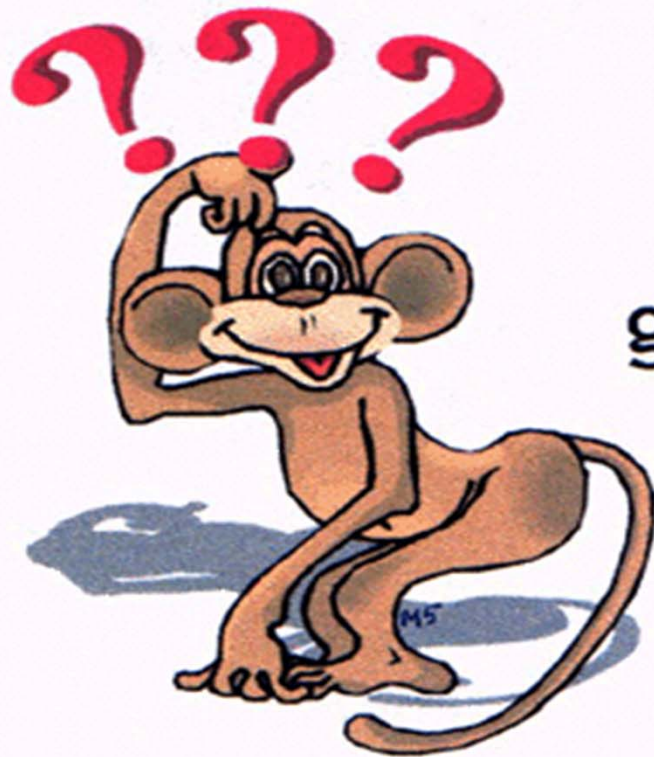
Conclusion

- ❖ Think five years in future/past on a technology perspective.
- ❖ Sciences of Software Synthesis
 - Designing- a Science of design.
 - Implementation- a Science of realization.
 - Adaptability-a Science of dynamic adaptation.
 - Domain Characteristics- a Science of domain-specific software engineering.
- ❖ Dynamic adaptation will be an obligation in the next few years specifically for software engineering.

References

- ◆ Oreizy, Peyman, Nenad Medvidovic, and Richard N. Taylor. "Runtime software adaptation: framework, approaches, and styles." Companion of the 30th international conference on Software engineering. ACM, 2008.
- ◆ Erenkrantz, Justin Ryan. Computational REST: A New Model for Decentralized, Internet-Scale Applications DISSERTATION. Diss. University of California, Irvine, 2009.
- ◆ Oreizy, Peyman, et al. "An architecture-based approach to self-adaptive software." IEEE Intelligent systems 3 (1999): 54-62.

Questions?



Questions
are
guaranteed in
life;
Answers
aren't.

Comparison

	Update behaviour	Update state	Update execution context	Asynchrony of change	Implementation Probes
REST	stateless http server can be restarted for updates; database servers updated using vendor specific techniques.	state is externalized: all messages carry state and is inspected;http server is stateless and application state stored in DB servers.	before update drain in process requests and refuse new req.	various techniques.e.g: shift load to ½ nodes,update,shift load,etc	server logs; query state in database
CREST	stateless servers may offer URL-specific interpreters;nominal behavior encapsulated in computations that are transmitted.	all aspects of a computations state made explicit and externalized.	fully included within the computations exchanged between peers	same a REST	server logs; computations are explicit and transmitted, may be examined by intermediaries



University
of Victoria
Computer
Science

Self-Managed Systems: an Architectural Challenge

Jeff Kramer and Jeff Magee

By: Ernest Aaron and Harshit Jain

Outline

- Dilemma
- Self Managed System
- Motivation
- Why architectural approach?
- Related work
- Architecture model
- Research issues
- Summary
- Course Reflection

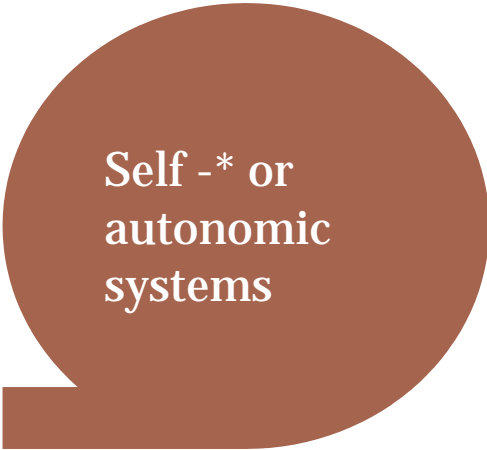
Dilemma

“As the size, complexity and adaptability required by applications increases, so does the need for software systems which are scalable, support dynamic composition and rigorous analysis, and are flexible and robust in the presence of change.”

-We Need Self Management System

Self Managed Systems

- Self- configuration
- Self- adaptation and Self-healing
- Self- monitoring
- Self- tuning



Self -* or
autonomic
systems

Motivation

“Their focus is on an architectural approach to self management, not because the language-level or network-level approaches are uninteresting or less promising, but because they believe that the architectural level seems to provide the required level of abstraction and generality to deal with the challenges posed with self management system.”

-Objective is to minimise the degree of explicit management

Why an architecture based approach?

- Generality.
- Level of abstraction.
- Potential for scalability.
- Builds on existing work.
- Potential for an integrated approach.

Self management in Robotic Systems

Sense-Plan-Act (SPA)

Garlan's self-healing system-

- Monitoring
- Analysis/resolution
- Adaptation

Gats architecture-

- Control (reactive feedback control)
- Sequencing (reactive plan execution)
- Deliberation (Planning)

Gats Architecture



Control layer consists of sensors, actuators, control loops.

- Self tuning, event and status reporting to higher levels.

Sequencing layer reacts to changes in state reported from lower levels.

- Execute plans based on existing control behavior.

Goal management (Deliberation) is planning based on current state to achieve the specification of high level goal.

- Introduction of new goals, produces change management plans according to requests from layers.

Architectural Model

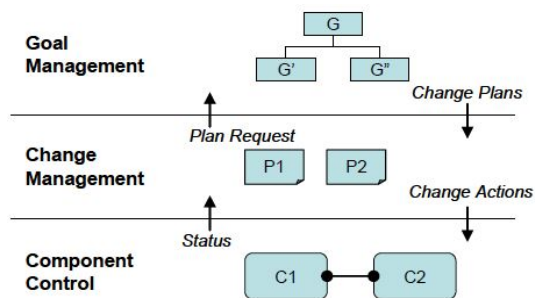


Figure 1 – Three Layer Architecture Model for Self-Management.

Architecture model of **3** layers contains:

- Component Layer
- Change Management
- Goal Management

Component Layer

Change Management Layer

Goal Management Layer

1--Component Layer

Component Layer

- Implements the set of services that it provides.
- Mode: It determines abstracted view of internal status of components.

Challenges

- Preserving safe application operation during change.
- No state loss during configuration for transactions.

2-- Change Management Layer

Change
Managem
ent Layer

- Responsible for execution changes in response to change in state after reported from lower layer or in response to goal changes.
- Precompiled sets of plans that can respond to predicted class of state change.

Challenges

- Distribution and decentralization.
- Distribution raises issues like latency, concurrency, partial failures.

3-- Goal Management Layer

Goal
Managem
ent Layer

- Top layer is responsible for the plan required by below layers
- Refinement from high- level goals to specified goals(processable by machines) with human assistance

Challenges

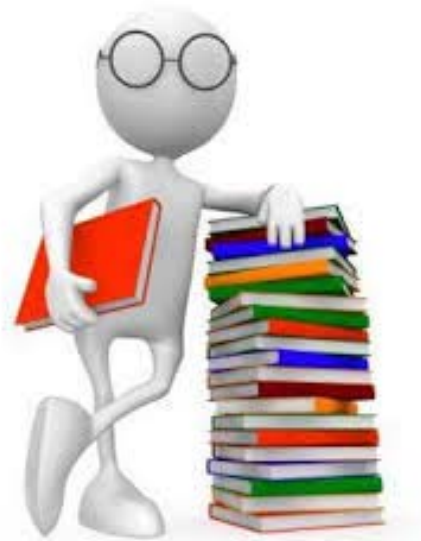
- Goal specification that it is both comprehensive by human users and machine readable.
- Producing a change plan based on system goals and current state of the system

Summary

- Self management at architectural level.
- Three layers defines self managed system which are
 - Component layer
 - Change Management layer
 - Goal Management layer
- To achieve the goals of the system components automatically configure their interaction in a way that is compatible with an overall architecture specification.
- Research challenges posed by individual layer need to be addressed and comprehensive integrated solution is needed.

Course Reflections

- Self Adaptive System
- Ultra Large System(ULS)
 - Continuous Evolution
- Software Architecture
- Self Managing System



Thank You!

Self-Managed Systems: an Architectural Challenge

Jeff Kramer and Jeff Magee

By: Ernest Aaron and Harshit Jain

Graduate Student Research Paper Presentations



- [Aksanli, J. Venkatesh, L.Z., Tajana R.: Utilizing Green Energy Prediction to Schedule Mixed Batch and Service Jobs in Data Centers. In: Proceedings 4th Workshop on Power-Aware Computing and System \(HotPower 2011\), Article 5 \(2011\)](#) — **Presentation by Junnan Lu and Francis Harrison: July 30**
- [Ebrahimi, S., Villegas, N.M., Müller, H.A., Thomo, A.: SmarterDeals: a context-aware deal recommendation system based on the SmarterContext engine. CASCON 2012: 116-130 \(2012\)](#) — **Presentation by Carlene Lebeuf and Maria Ferman: July 30**
- [Villegas, N.M., Müller, H.A., Tamura, G., Duchien, L., Casallas, R.: A framework for evaluating quality-driven self-adaptive software systems. In: Proc. 6th Int. Symposium on Software Engineering for Adaptive and Self-Managing Systems \(SEAMS 2011\), pp. 80-89 \(2011\)](#) — **Presentation by Parminder Kaur and Navpreet Kaur: July 30**
- [Villegas, N.M., G. Tamura, H.A. Müller, L. Duchien, and R. Casallas, DYNAMICO: A reference model for governing control objectives and context relevance in self-adaptive software systems, in: R. de Lemos, H. Giese, H.A. Müller, and M. Shaw \(Eds.\), Software Engineering for Self-Adaptive Systems, LNCS 7475, Dagstuhl Seminar 10431. Springer, pp. 265-293 \(2013\)](#) — **Presentation by Arturo Reyes Lopez and Babak Tootoonchi: July 30**