Overview 2007-01-15

- IBM’s Initiative
- Self- {configuration — optimization — healing — protection }
- Reactive, adaptive, evolutionary
- Control loops
- Policies
IBM’s Vision, Manifesto in 2001

“Grand Challenge of building computing systems that regulate themselves.”

- Manual-user admin, maintenance, management cannot keep pace with rate of infrastructure change.
- Managing individual HW or SW modules does not guarantee overall performance.

In numbers:
1 dollar spent on computing infrastructure →
10 dollars spent on management.

IBM’s “Key Properties/Elements” for Autonomic

8 or 4, depending on where you look

- self-configuration
  - self-optimization
  - self-healing
  - self-protection

- plus: self-aware, self-learning, operate in heterogeneous computing environment, anticipate and adapt to user needs.

In the literature, from others: many more “self-star properties”: self-organizing, self-adaption, self-management, self-deployment, ...

© Tschudin/Jelger/Yamamoto, UBasel
Goal of this Session

- Define the most important self-terms
- Spectrum from reactive to evolutionary
- Spectrum from control to self-organization

Automatic vs Autonomic

- **Automatic**: pre-programmed task execution
  - system works fine until something goes wrong
  - at latest now, human intervention is needed

- **Autonomic**: self-regulation
  - system response is also automatic, but modulated
  - system can compensate or work around problems
  - no human intervention needed

Both need engineering effort, though.
Example: Install set of new SW upgrades

- Assumption:
  - means exist to *automatically* deploy upgrades
  - *automatic* regression tests exist
  - *automatic* problem determination

- Autonomic approach
  - deploy, run, test
  plus, in case of problems:
  - revert
  - identify problematic component, isolate it
  - re-start with the reduced set of updates

Update Example Contd

Potential application: building a new Linux/FreeBSD/etc release

- Programmers submit their patches, new applications

- System automatically makes release candidate:
  - does kernel still compile
  - can all library dependencies be resolved?
  - do unit tests still work?

- Predicting release dates:
  - analyzing the ratio of bug fixed to bugs introduced
Element 1: self-configuration

IBM’s concern: computing, hence the following wish list

- automatic SW
  - deploy
  - install
  - configure
  - re-configure
  - config documentation

- while adhering to administrative directives, and producing reports on compliance

Self-configuration (contd)

Self-Configuration in many other contexts:

- Networking:
  - addresses, routes, zero-config

- Parallel Computing:
  - assigning free processors

- (software) Multi-Agent Systems, MAS:
  - dynamic agent allocation and scheduling

- Complex systems:
  - self-organization (e.g., find new spacial configurations)
Definition for our usage:

- **system is self-configuring**
  - the system is able to (re-) configure itself according to high-level policies.

This is not necessarily autonomic by itself, but is required for an autonomic system.

Observation on current status (regarding configuration):

- Data centers have
  - multiple vendors, platforms, software systems
- Installation/configuration/integration of new elements (HW, SW, policies) is
  - time consuming (days, weeks)
  - error prone

Goal: Automated config of components according to high-level policies **and** then rest of system adjusts automatically.
Element 2: Self-Optimization

- Observation: Huge number of tuning parameters, for HW and SW
  - cache size
  - timeout values
  - cpu and bandwidth allocation
  - dimensioning of internal data structures, hash tables
  - several algorithms for same task, different profiles

IBM’s description, also our definition:

**system is self-optimizing** ⇔ the system and its components continually seek to improve their performance and efficiency.

Self-Optimization (contd)

How far can self-optimization go?

Many places to optimize:

- Design time:
  - Software architecture

- Implementation time:
  - choice of algorithms, language, optimizing compiler

- Run time (e.g. TCP adaption)

Current understanding: only parameter setting, at run-time.
But we could go further! (see later on)
Element 3: Self-Healing

Problem solving: an analytic activity for humans only?

Wish list:

- reactive:
  - recover from events causing failure or malfunction
- proactive:
  - anticipate/predict failures

Problem solving includes:
find cause, find cure, test cure, deploy

Self-Healing (contd)

IBM has a somehow restricted definition:

**system is self-healing ⇐
the system automatically detects, diagnoses, and repairs localized software and hardware problems.**

- Mostly centered around configuration errors, see regression test for SW upgrades
- Mostly based on log analysis
- But also requires infrastructure support
  (moving a set of apps and their data to a new server)
Self-Healing (contd 2): as an additive feature

- Characterizing IBM's approach: *adding* self-healing features
  Self-healing mostly applies to configuration parameters:
  - which server, which update etc

- Next level (beyond IBM): self-healing inside the apps
  - what algorithm to use
  - what data structure, what encoding etc
  Requires trade-offs and (internal) choice:
  - this algo is more robust, but slower

Still, this is “parameterization”.

Self-Healing (contd 3): Deep self-healing

Our definition:

**system is self-healing** ⇐⇒
the system asserts *goal integrity* over long times.

- Note: Code can change, changing the “performance envelope”!
- Self-healing at the code level:
  - app controls its own code, not only parameters
  - can walk algorithm space, not just config space
- First step:
  - assert code integrity over long times
  - example: application fights viruses itself
Self-Protection

According to IBM:

**system is self-protecting** ⇔
the system defends against malicious attacks or cascading failures, can prevent systemwide failures.

- Ambitious and difficult task:
  attacks are not know in advance

- Focus on automatic reaction, tiered security
  (breach on one level does not endanger whole system)

---

IBM’s Autonomic Elements, practically: Self-Conf

Self-Configuration:

- Examples mentioned before:
  software updates, regression tests
IBM’s Autonomic Elements, practically.: Self-Opt

Self-Optimization:

- Adding more (computing) resources on demand, when some apps become too slow
- Automatic outsourcing of tasks (Grid)
- Picking optimal connectivity with sub-second response time (WLAN, ethernet, GPRS ...), or cheapest etc

Optimization includes resolution of conflicting goals.

IBM’s Autonomic Elements, practically: Self-Heal

Self-Healing:

- “self-correcting job control language”: job failures are analyzed, job is automatically restarted
- Database index file is corrupted: automatic re-indexing, testing, going productive
- Hardware failure, server goes down: “[websphere app server v6] can transfer those transactions to another server [automatically]”
IBM’s Autonomic Elements, practically: Self-Prot

Self-Protection:

- Confirm ability to backup and recover data resources
- Network monitoring and IDS, automatic disconnection of suspicious computers
- Verify that all client machines have latest patches
- Track security advisories

Assessment of IBM’s definitions/efforts

- No magic – it’s the mind set!
  Design for self-* in mind
- Additive approach:
  – add self-* “features”
- Autonomics with a methodology:
  – structure template for autonomic elements
  – loop model: plan, execute, monitor, analyze
  – loop sits “on” the managed element
  – controller hierarchy
Assessment of self-* properties in general

- Properties are not orthogonal
  - self-opt results in re-config
  - self-healing as special case of self-opt?
    (minimize number of open problems)
- Properties are not specific about their target
  - design/compile/deploy/run-time
- Properties are not specific about system:
  - closed (static) set of elements
  - open system, including evolution?

Overview 2007-01-15

- IBM’s Initiative
- Self- \{ configuration — optimization — healing — protection \}
- Reactive, adaptive, evolutionary
- Control loops
- Policies
Extending the Landscape: What is Adaptive?

Evolutionary programming
(algorithm generation, genetic algorithms, AI–based learning)

Algorithm selection

Generic or parametrized algorithms

Online algorithms (deterministic, randomized, probabilistic)

Conditional expressions

(Oreizy et al, 1999)

Adaptive (contd)

- “Conditional expressions”:
  - reactive
  - predetermined fixed response
  - examples: network protocols

- “Online algorithms”:
  - assumption that future events are uncertain
  - keeping options open, anticipate problems
  - has to measure system state, reacts accordingly
  - example: swapping, opportunistic execution inside CPU
Adaptive: Revisiting “Reliable Transport”

In the network lecture, we went through stages: first handle bit errors, then message loss, then congestion, etc.

Apply “adaptivity spectrum” to protocol variations:

– conditional expr: selective retransmission
– online algo: compute timeout value
– generic: choosing TCP buffer and window sizes
– algo selection: TCP vs STCP vs UDP vs ...
– evolution: transport protocol “breeding”
Overview 2007-01-15

- IBM’s Initiative
- Self- {configuration — optimization — healing — protection }
- Reactive, adaptive, evolutionary
- Control loops
- Policies

System Decomposition

Global self-* behavior cannot be achieved “as a whole”

- Autonomic system is broken into smaller but autonomic entities, each being dedicated to fulfill a specific task.
- Reduces complexity of designing a large system
- Potentially increases the complexity of managing the resulting multiple interactions.

The self-* behavior is an “external” characteristics, i.e. the smaller entities that compose the system do not necessarily exhibit self-* properties.
“Autonomic Element”

Autonomic Element = “manager” for some resource or task

- Has defined behavior, constrained by policies and goals.
- Decisions based on observations of their environment, policies and goals, and “configuration-range”.
- Negotiation between autonomic elements

Examples:
- Install or upgrade software.
- Restart a system or component after a failure.
- Isolate systems after intrusion detection.
- Re-configure task to cope with changing conditions (e.g. load)

Input/Output Model, Control Loop

“Steering” of autonomic elements is achieved via control loops

Input can vary, self-adjustment between actual and desired output.
Autonomic Ctrl Loop

“Intelligent” control inside an autonomic element, based on four phases/activities:

1. Monitor: collect and filter data.
2. Analyze: compare system operation wrt policies and goals.
3. Execute: flexible and re-configurable exec of tasks
4. Plan: (re-)configure operation with respect to policies and goals.

Needs Sensors: to collect data, states of the managed element.
Needs Effectors: to apply changes (i.e. re-configure)
Nested Ctrl Loops: Managed Managers
IBM’s Initiative

Self-configuration — optimization — healing — protection

Reactive, adaptive, evolutionary

Control loops

Policies

Policies

System cannot guess “what is good behavior”:

- Autonomic element attempts to fulfill its goals according to policies
- Wrong goals + wrong policies = wrong outcome
- Challenge: deriving consistent rules for multiple autonomic elements such that global behavior is fulfilled.

Examples of policies:
- Minimize memory and disk usage for database.
- Maximize lookup speed inside database.
- What about “minimize mem usage and maximize speed”? 
Plan Component

Plan component re-configures system behavior based on analysis of current behavior.

- Adaption range might be pre-defined
  - e.g. Adaptability of TCP is large but limited by design.
- Might consider only current state (TCP has little memory)
- For better adaptivity: Knowledge of the past and "learning from experience" are required
- System learns the (good or bad) consequences of actions taken (TCP could "invent" new window management or congestion control).

Policy Management Tools

- Policy Management for Autonomic Computing (PMAC): PMAC policies are typically expressed with an XML-derived language called ACPL (Autonomic Computing Policy Language). ACPL requires good XML knowledge and is highly error-prone.
- IBM's Simplified Policy Language (SPL): Simplified interface to PMAC (Policy Management for Autonomic Computing): IBM's management framework. SPL is automatically converted into ACPL.

Conflicting rules are not detected, unless obvious collision. Clearly needs more elaborated ways to express policies and goals.
Analyze Component

Infers system behavior based on current and past observations, and according to current policies and goals of the system.

- May have to operate on incomplete data sets or partial view of the running elements.
- Can potentially request additional data from the monitoring element

Critical part of the system: even the best car-driving algorithm cannot get round a cliff if cameras don’t look in the right direction at the right time.

Knowledge = information repository
Knowledge Component

“Database” storing data, events, states, collected by sensors, and decisions made by the autonomic element(s).

- Potentially maintains “a history” of past events and changes (e.g. to cope with long-term changes)
- Information should be stored in standard format, potentially in a distributed manner.
- Information sampling is dynamically adjusted according to load (CPU, bandwidth), storage space, system health
- Information might be compressed: averages, linear regression

Policy specifications

- Traditional view: Policies are rules governing the choices in behavior of a system ([Sloman 1994])
- Policies are persistent: a one-off command to perform an action is not a policy.
- Obligation policies: event triggered condition-action rules (also called action policies):
  – IF (condition) THEN (action).
  Note: not a state machines: i.e., no “target” state
- Authorization policies: define what services or resources an entity can access.
Goal policies (Kephart 2004)
  – A single or rather a set of desired states.
  – The system “overall target is to fulfill the goal policies.
  – Static and possibly conflicting policies.

Utility function policies (Kephart 2004)
  – Generalization of goal policies.
  – curve instead of binary value
  – “On-the-fly” computation of most desired states.
  Permits unambiguous and dynamic decision making.

Policy specification languages

  • Logic based
  • Event-driven
  • Scoped policies
Logic Based Policy Spec Languages

- first order logic, stratified logic etc, well understood formalisms:
  easy to analyse (e.g. conflict resolution), but difficult to use and implement efficiently

- Example (for authorization):
  
  \[
  \text{canDo(file1,s,+read) } \leftarrow \text{in(s,Employees)} \land \\
  \quad \neg \text{in(s,Soft-Developers)}
  \]

  “all subjects belonging to group Employees but not to Soft-Developers are authorized to read file1”

Event Based Policy Spec Language

- Form:  IF (condition) THEN (action)

- Easy to translate into implementation code. Relatively easy to identify enforcement entities: for a given policy, those that can monitor the elements specified by the (condition).

- Example: IETF/DMTF Policy Core Information Model

  \[
  \text{if } ((\text{srcIPAddress} = 192.168.12.17 \text{ AND } \text{dstIPAddress} = 192.168.24.8} \text{ OR } (\text{srcIPAddress} = 192.168.24.8 \text{ AND } \\
  \text{dstIPAddress} = 192.168.12.17)) \text{ then set Rate := 400Kbps;}
  \]

  \[
  \text{if } ((\text{sourceIPSubnet} = 224.0.0.0/240.0.0.0) \text{ AND } (\text{timeOfDay} = 1800-2300) \text{ AND } (\text{dayofweek} = \text{Monday}))
  \]

  \[
  \text{then set Priority := 5}
  \]
Scoped Policies: Authorization and Obligations

- Policies are restricted to certain objects or domains
- Rule-based policies: easy to implement.
- Example (an “obligation” in Ponder):

```c
inst oblig loginFailure {
    on 3*loginfail(userid) ;
    subject s = /NRegion/SecAdmin ;
    target <userT> t = /NRegion/users ^ {userid} ;
    do t.disable() -> s.log(userid) ;
}
```

“After 3 failed login attempts, user is disabled by security admin and action is logged”

Policy Specifications: Open Challenges

- Goals specification: Nice to have **simple goal expressions**, but can complex behaviors be achieved in this way?
- **Dynamic computation of policy:**
  Fulfilling certain policies require to fulfill other policies.
  System should not waste resources on unreachable goals.
- **Dynamic policy conflict resolution:**
  System should not reach ambiguous states.
- **Trust** that policies are accurately implemented, urge to steer (rather than to police) an autonomic system.