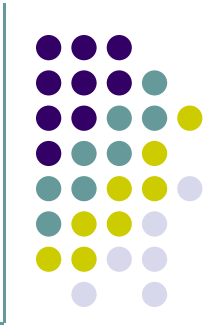
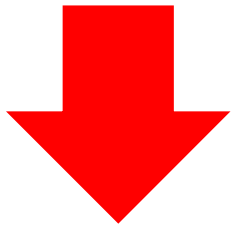


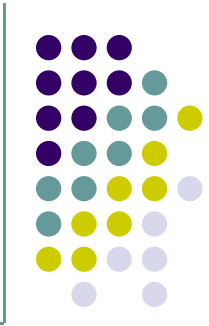
Welcome to SENG 480B / CSC 485A / CSC 586A Self-Adaptive and Self-Managing Systems

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



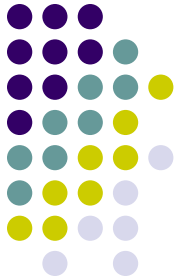
http://courses.seng.uvic.ca/courses/2015/summer/seng/480a
http://courses.seng.uvic.ca/courses/2015/summer/csc/485a
http://courses.seng.uvic.ca/courses/2015/summer/csc/586a

Announcements



- Wednesday, June 17
  - A2 tutorial in ECS 266 3:00-4:00 pm
- Friday, June 19
  - A2 due
- Grad project
  - Handed out June 15
  - Due July 25
- Midterm 1
  - Should be graded by June 18
  - Talk about answers on June 18

Assignment 2 Tutorial
Dr. Ron Desmarais



Wednesday, June 17

3:00-4:00 pm ECS 266

Autonomic Computing Vision

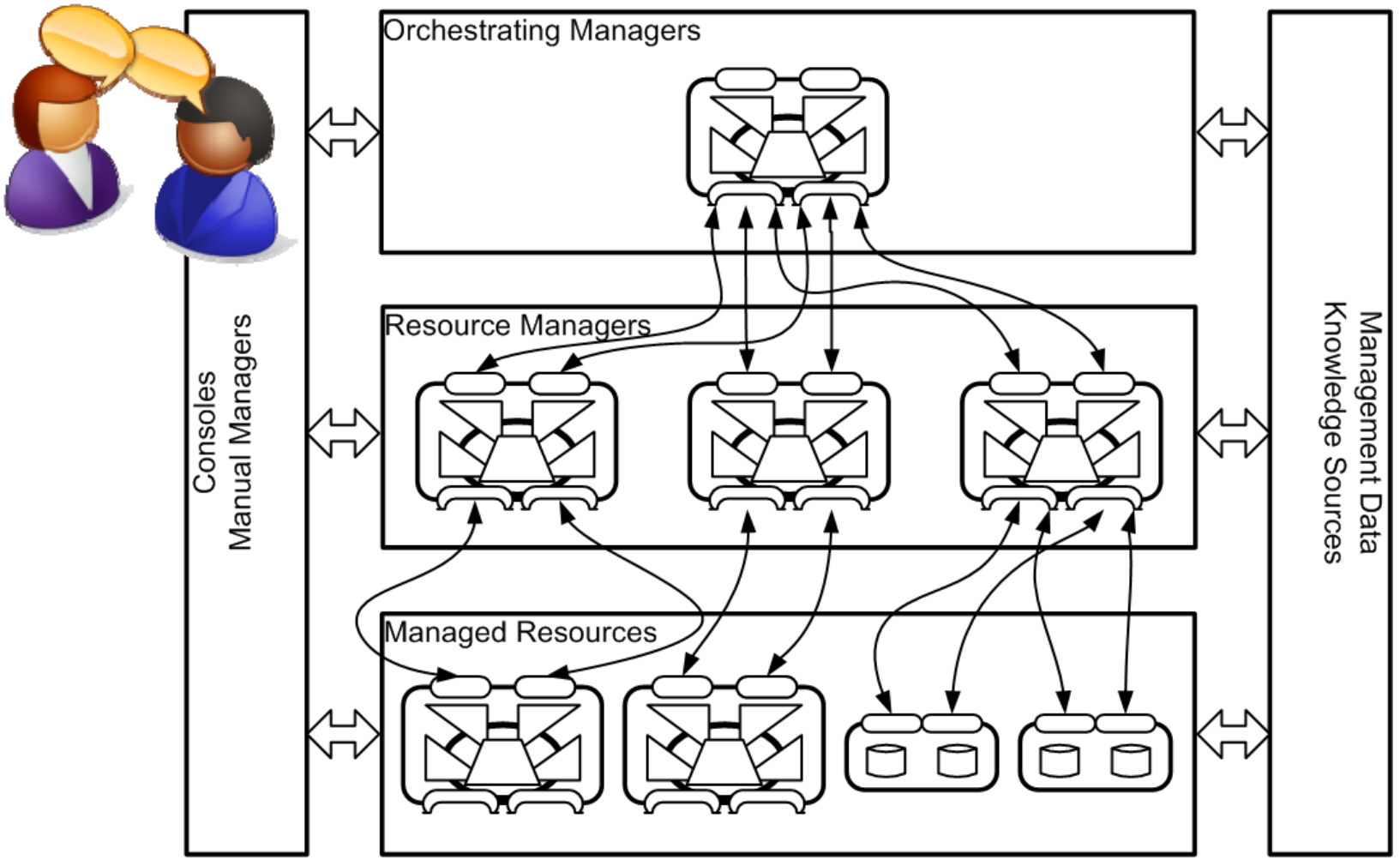
Autonomic Computing is really about making systems self-managing …

—Paul Horn, IBM Research, 2001

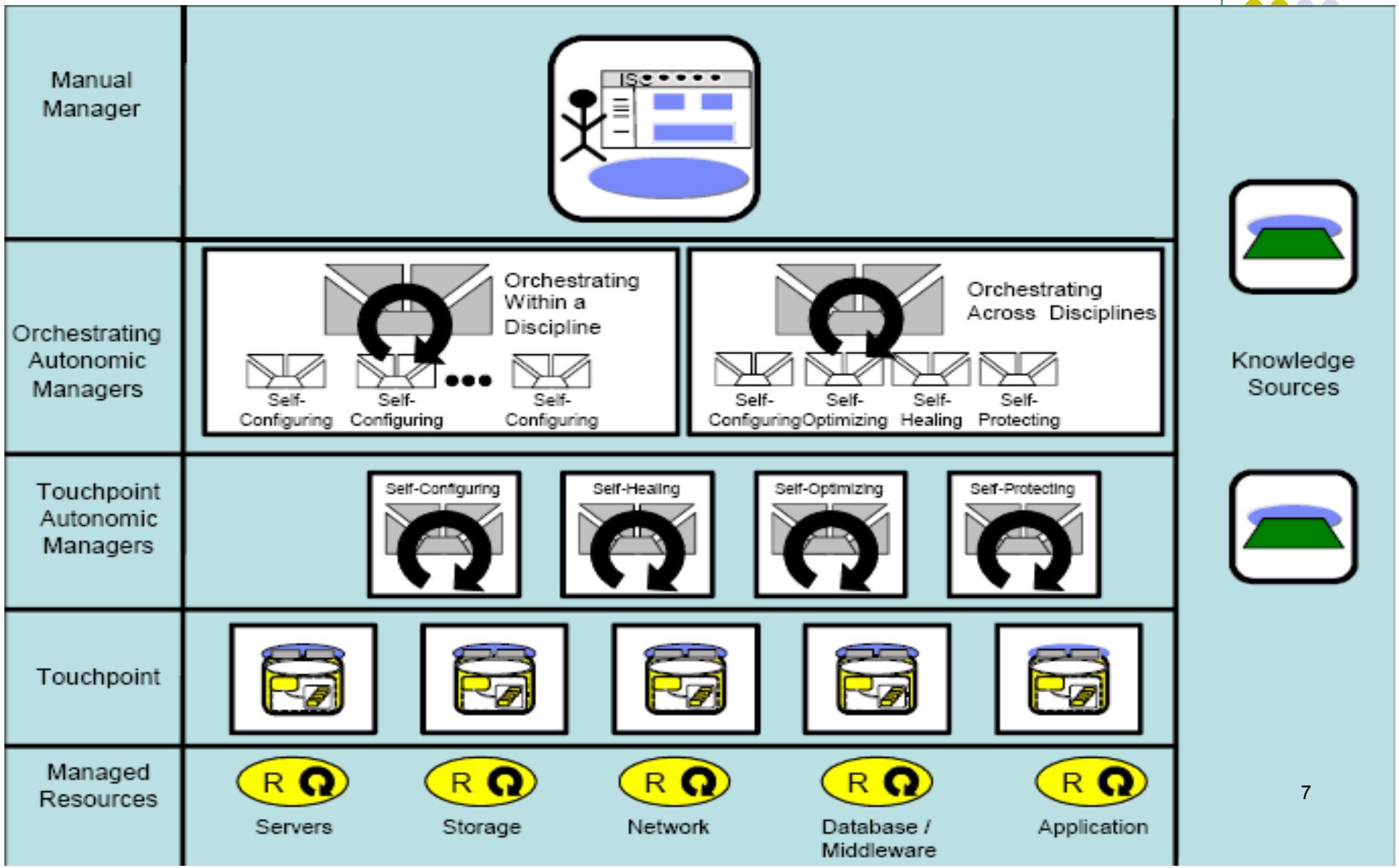
Reading Assignment

- Kephart, J.O., Chess, D.M.: The Vision of Autonomic Computing. IEEE Computer 36(1):41-50 (2003)
  ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1160055
- IBM: An Architectural Blueprint for Autonomic Computing, 4th Ed. (2006)
  citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.150.1011&rep=rep1&type=pdf

ACRA
AC Reference Architecture



ACRA
AC Reference Architecture



ACRA: Autonomic Computing Reference Architecture

Level 5—highest

- Manual manger who operates a common system management interface

Level 4

- Autonomic Managers to integrate and orchestrate several self-* capabilities for a particular domain (e.g., DB, weather station)
- Implements system-wide capabilities

Level 3

- Implements specific self-* using Autonomic Managers (AM)

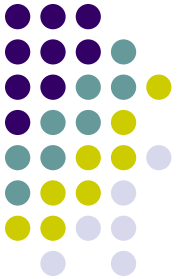
Level 2

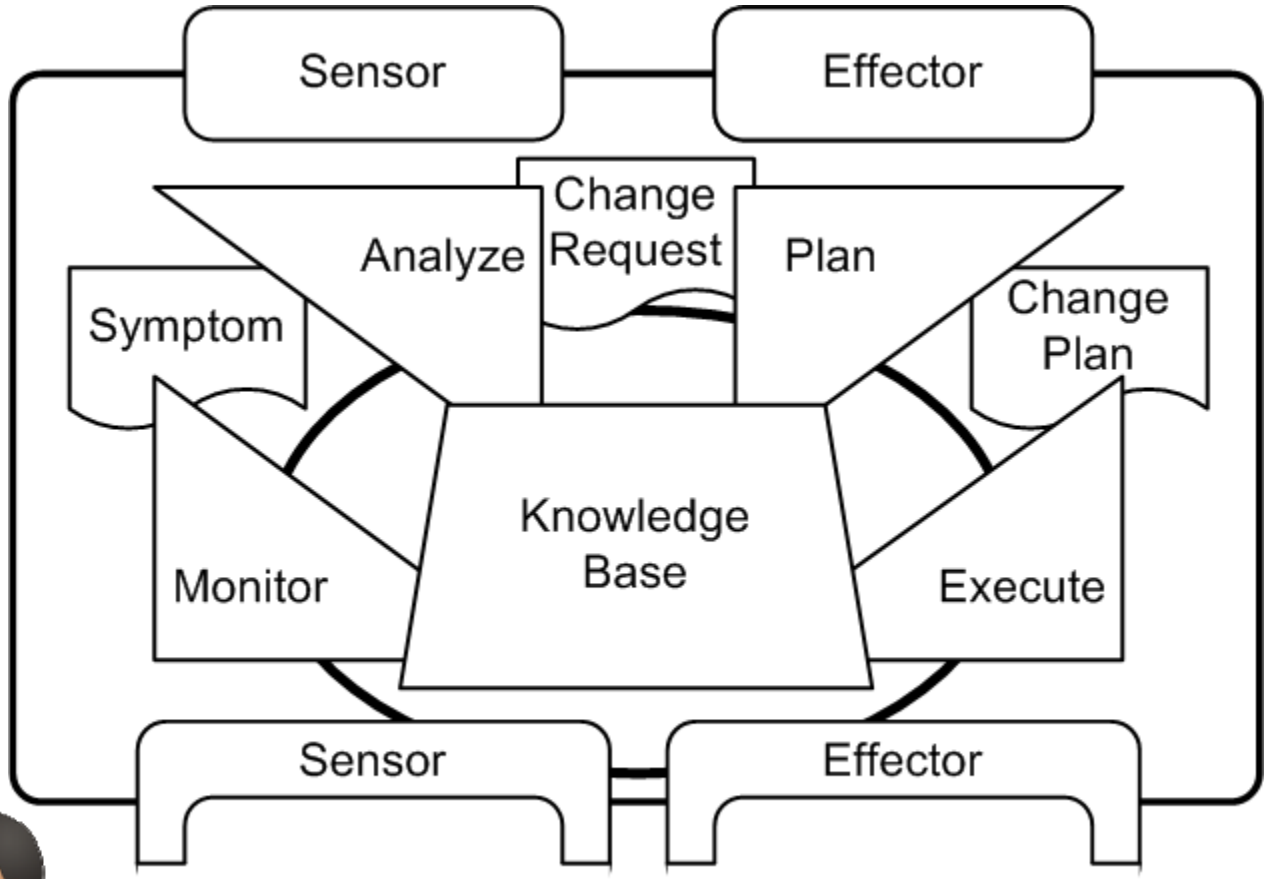
- Consistent, standard Manageability Interfaces (MI) for accessing and controlling the managed resources in a uniform manner
- The MIs are implemented with Manageability Endpoints (ME)

Level 1—lowest

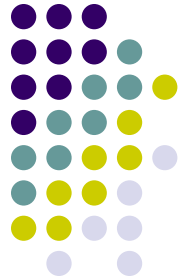
- System components or managed resources (hardware, software) possibly with embedded self-management

Autonomic Manager



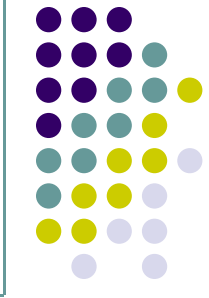


Autonomic Computing Manifesto Fundamental Objectives



- **Self-configuration**: the system sets and resets its internal parameters so as to conform to initial deployment conditions and to adapt to dynamic environmental changes, respectively.
- **Self-healing**: the system detects, isolates and repairs failed components so as to maximise its availability.
- **Self-optimisation**: the system proactively strives to optimise its operation so as to improve efficiency with respect to predefined goals.
- **Self-protection**: the system anticipates, identifies and prevents various types of threats in order to preserve its integrity and security.

Autonomic Managers Implement Self-* MAPE-K Loops



Increased Responsiveness

Adapt to dynamically changing environments

Operational Efficiency

Tune resources and balance workloads to maximize use of IT resources

Business Resiliency

Discover, diagnose, and act to prevent disruptions

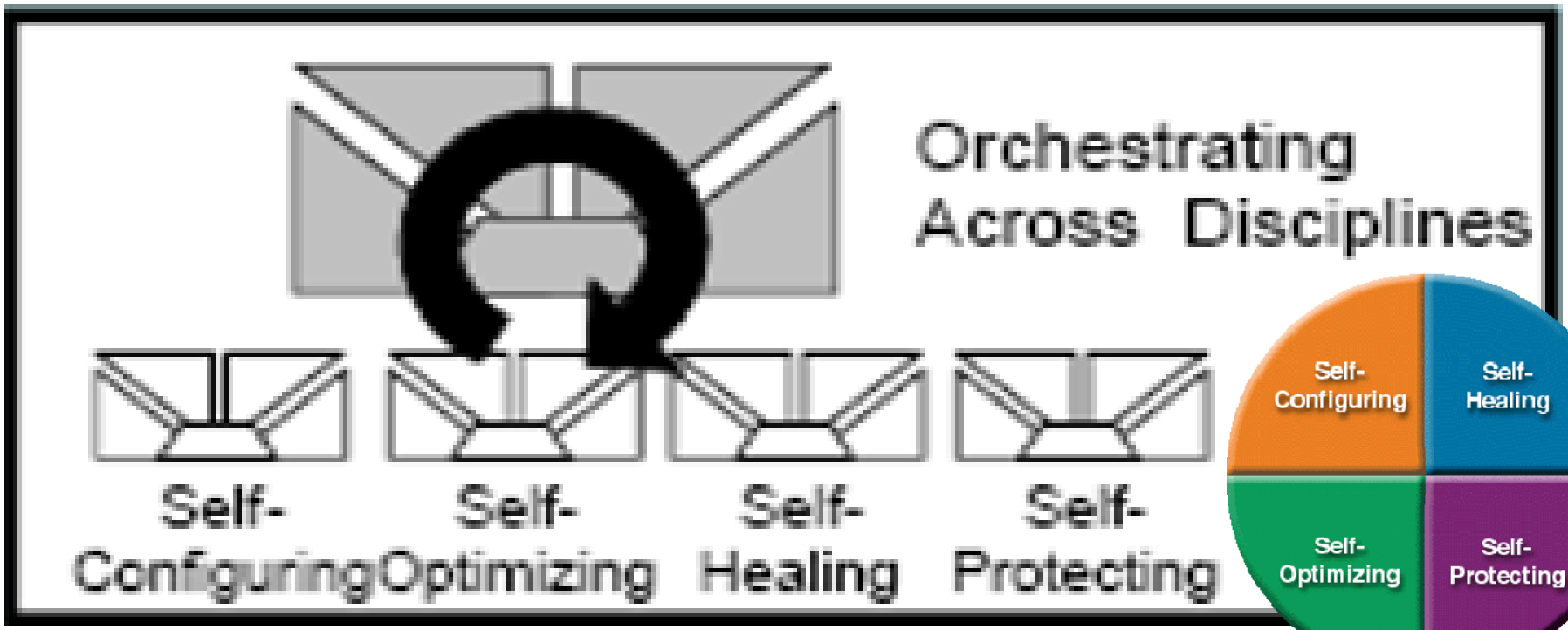
Secure Information and Resources

Anticipate, detect, identify, and protect against attacks

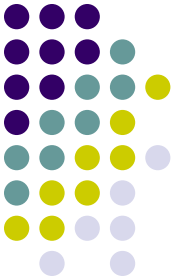
*Self – **

Class Exercise

- Identify an application requiring at least 3 of the 4 self* goals listed below
- Discuss the trade-off among the goals

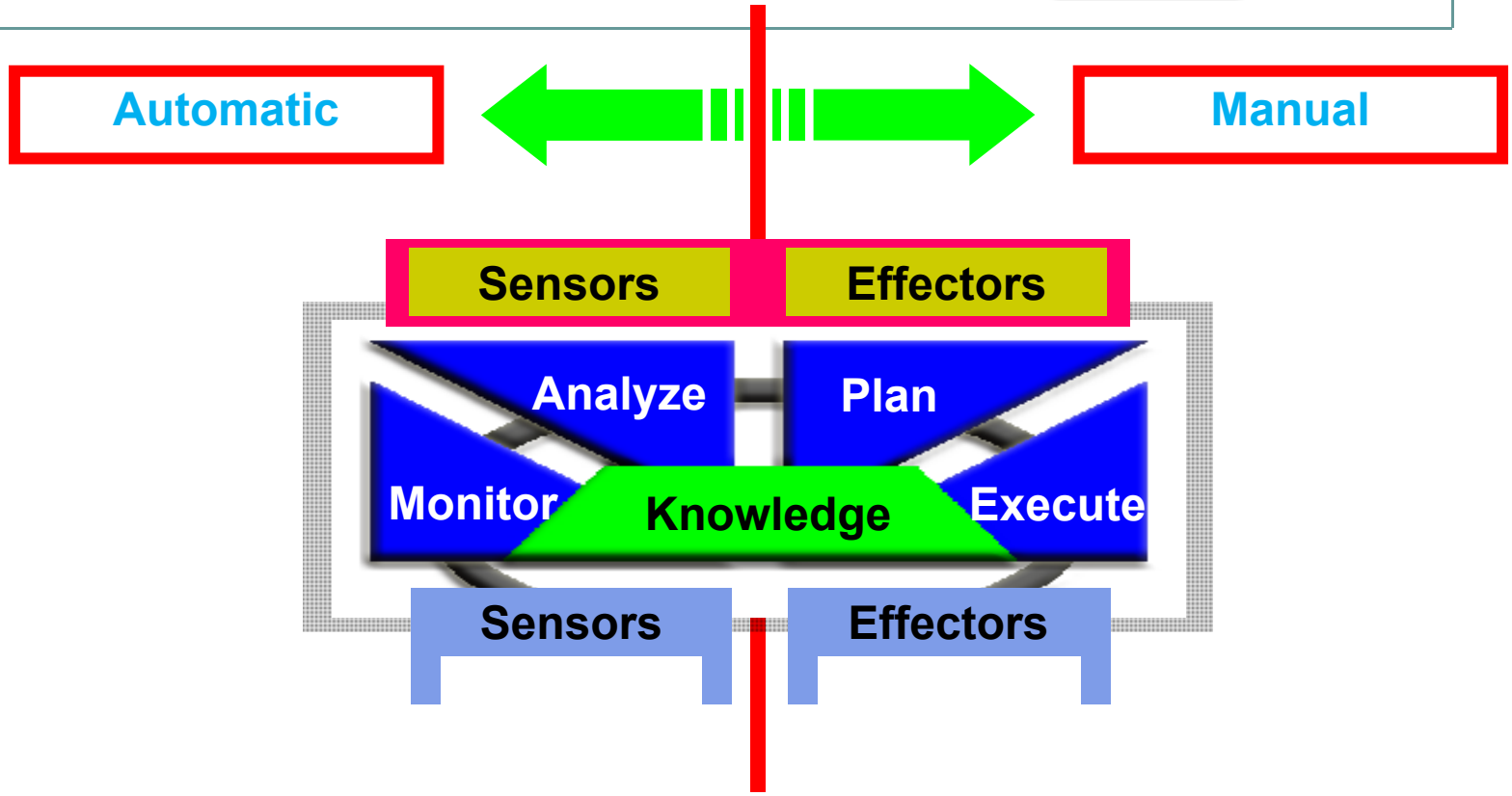


Manual Manager



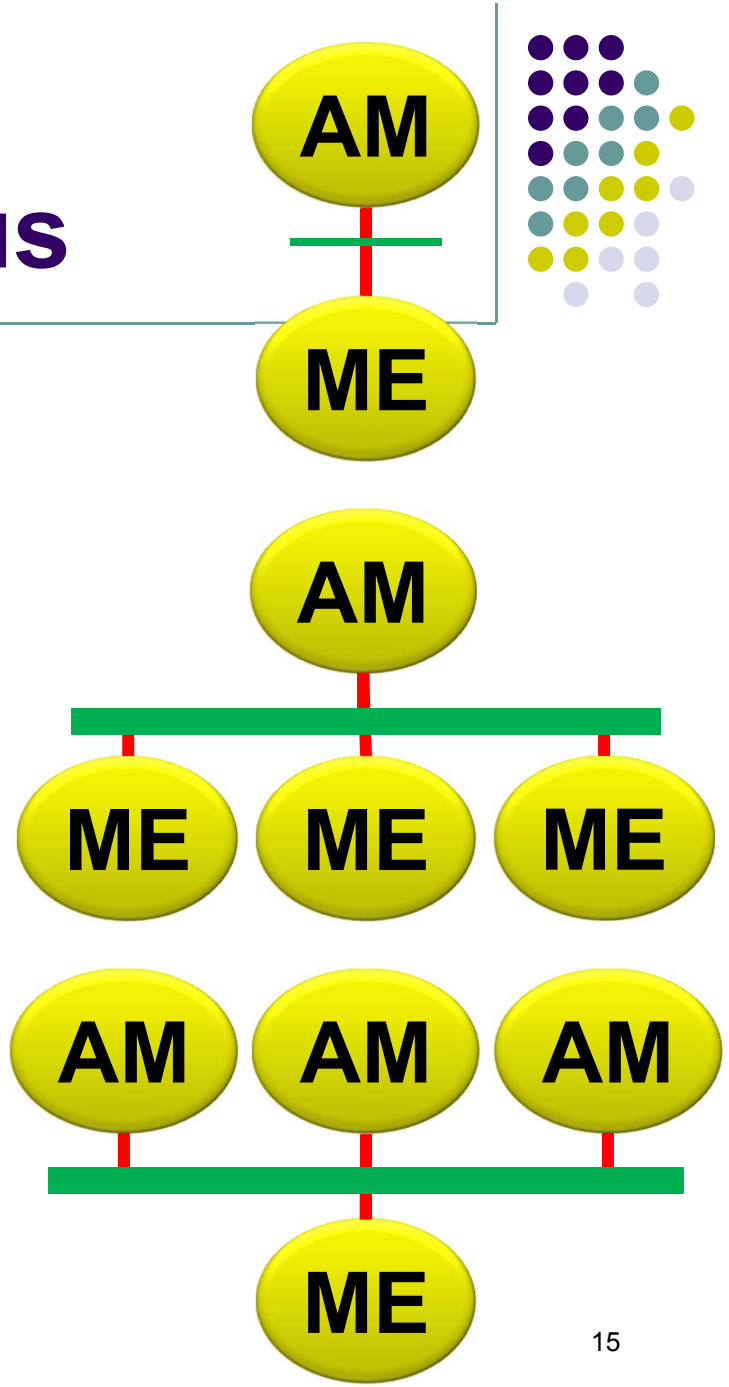
- Management or integrated solutions console
- Enables a human to perform and delegate management functions
- Collaborates with or orchestrates autonomic managers
- Set-up, configuration, run-time monitoring, control
- Manage trust—different levels of feedback
- Connecting knowledge source
- Specifying policies

Autonomic Manager



Enterprise Service Bus

- Connects and integrates various AC building blocks
  - Autonomic Managers (AMs)
  - Manageability Endpoints (MEs)
  - Knowledge repositories
  - Aggregating multiple manageability mechanisms for a single manageable resource
  - Facilitating one or more AMs to manage one or more MEs
- WSDM standard
  - Web Service Distributed Management



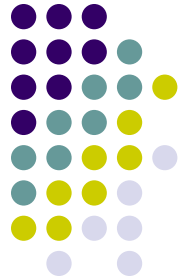
Useful Papers under Resources Course Web Site

- Ganek, A.G., Corbi, T.A.: The Dawning of the Autonomic Computing Era. IBM Systems Journal 42(1):5-18 (2003)
- Kephart, J.O., Chess, D.M.: The Vision of Autonomic Computing. IEEE Computer 36(1):41-50 (2003)
- Kluth, A.: Information Technology: Make It Simple. The Economist (2004)

- Huebscher, M.C., McCann, J.A.: A Survey of Autonomic Computing—Degrees, Models, and Applications. ACM Computing Surveys, 40 (3):7:1-28 (2008)
- Müller, H.A., Kienle, H.M., Stege, U.: Autonomic Computing: Now You See It, Now You Don't—Design and Evolution of Autonomic Software Systems. In: De Lucia, A.; Ferrucci, F. (eds.): Software Engineering International Summer School Lectures: University of Salerno. LNCS, Springer-Verlag, Heidelberg, pp. 32–54 (2009)

- Dobson, S., Denazis, S., Fernandez, A., Gaiti, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N., Zambonelli, F.: A Survey of Autonomic Communications. ACM Transactions on Autonomous and Adaptive Systems (TAAS) 1(2):223-259 (2006)

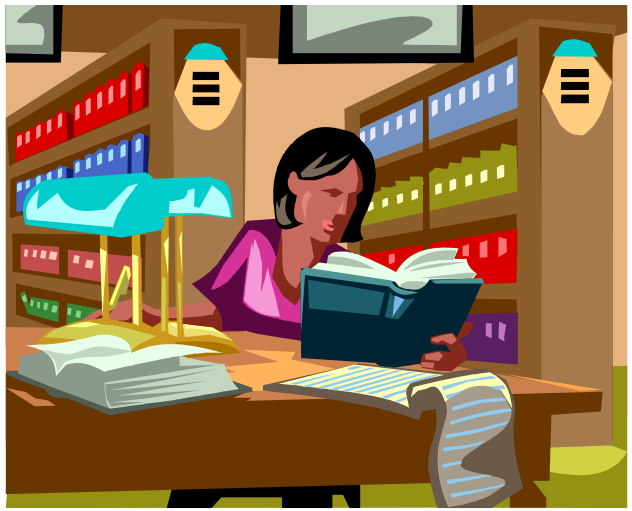
Useful Papers under Resources Course Web Site



- Diao, Y., Hellerstein, J.L., Parekh, S., Griffith, R., Kaiser, G.E., Phung, D.: A Control Theory Foundation for Self-Managing Computing Systems. IEEE Journal on Selected Areas in Communications 23(12):2213-2222 (2005)
- Müller, H.A., Pezzè, M., Shaw, M.: Visibility of Control in Adaptive System. In: 2nd ACM/IEEE International ICSE Workshop on Ultra-Large-Scale Software-Intensive Systems (ULSSIS 2008), pp. 23-26, ACM, New York, NY, USA (2008)
- Dawson, R., Desmarais, R., Kienle, H.M., Müller, H.A.: Monitoring in Adaptive Systems Using Reflection. In: 3rd ACM/IEEE International ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2008), pp. 81-88, ACM, New York, NY, USA (2008)
- OASIS: Web Services Distributed Management: Management of Web Services (WSDM-MOWS) 1.1 OASIS Standard (2006)
- OASIS: Web Services Distributed Management: Management Using Web Services (WSDM-MUWS) 1.1 OASIS Standard (2006)
- Kreger, H., Studwell, T.: Autonomic Computing and Web Services Distributed Management (2005)
- IBM: Symptoms Reference specification Version 2.0 (2006)

Useful Papers under Resources Course Web Site

- Study these papers
- Immerse yourself in the autonomic computing literature and technology
- Huge asset for your job application and future job



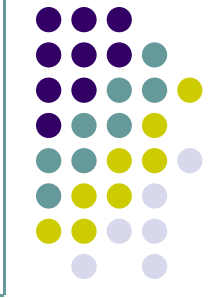
Implementing Autonomic Elements

The devil lies in the details ...

Standards, data and control integration, interfaces, endpoints, services, SOA ...

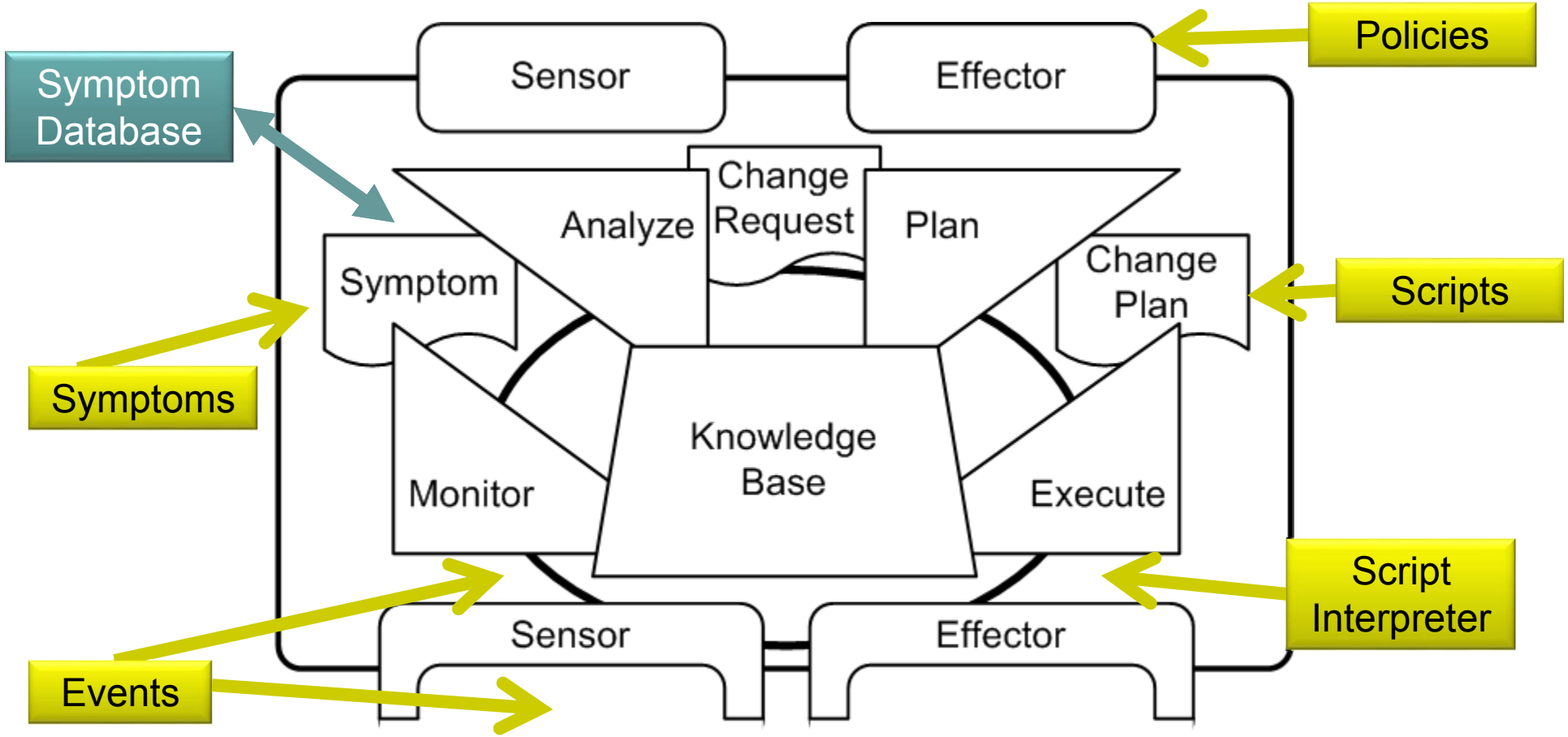
services, SOA ...
integration, interfaces, endpoints,

Information Interchange in the ACRA Architecture



- What information is passed between the components of an autonomic architecture adhering to the ACRA reference architecture?
- Information is exchanged in the form of events and knowledge in the knowledge bases
- Ideally the exchanged information is standardized
  - Formats
  - Schemas
- Information is exchanged between the manager and the managed element
  - Events
  - Set and get operations
- Policies are injected into autonomic elements through the effectors on top of the manager
- Information is passed around the MAPE-K loop

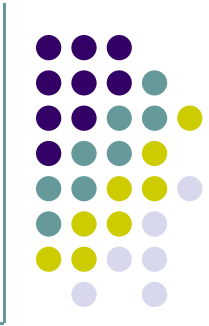
MAPE-K Loop Standards & Interfaces



Information Interchange in the ACRA Architecture

- What information is passed between the components of an autonomic architecture adhering to the ACRA reference architecture?
- Information is exchanged in the form of events and knowledge in the knowledge bases
- Ideally the exchanged information is standardized
  - Formats
  - Schemas
- Information is exchanged between the manager and the managed element
  - Events
  - Set and get operations
- Policies are injected into autonomic elements through the effectors on top of the manager
- Information is passed around the MAPE-K loop

Events



- An event is an asynchronous state transition in the managed element
- Events are generated by managed elements and are processed by autonomic managers
- Event processing is a discipline that aims to define and develop
  - Event abstractions
  - Event architectures
  - Event systems
  - Event languages
  - Event patterns
  - Event models
  - Event processing standards
  - Event exchange standards

Common Base Event Model CBE

- An *event*
  - An occurrence of a situation
  - Variety of forms: business, autonomic, management, tracing and logging events
  - Events encapsulate message data and constitute thus the foundation for complex distributed systems
  - Data elements of events need to be in a consistent format
    - to enable correlation
    - to facilitate effective intercommunication
- The CBE model is an event exchange standard for events exchanged in distributed applications
- The standard facilitates consistency in the elements themselves and in their format
- 3-tuple CBE element
  - Identification of the component that is *reporting* the situation
  - Identification of the component that is *affected* by the situation—may be the same as the component reporting the situation
  - The *situation* itself

Eclipse Log and Trace Analyzer

- The Eclipse Log and Trace Analyzer maps proprietary log formats into a common event model called *Common Base Event (CBE)*

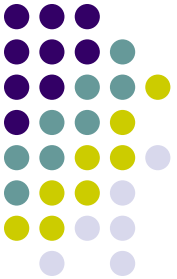
- Parsers provided with the Log and Trace Analyzer map the log records from their proprietary output format to the CBE model

Event Exchange Format Standard Common Base Events (CBE)

- CBEs communicate events in a structured way
  - De facto **standard** for reporting events
  - Logging, tracking, management, or business events can all be mapped to CBEs
- CBE is an XML structure consisting of three parts:
  - Identification of the component **reporting** the situation (**reporterComponentId**)—optional; can be source
  - Identification of the component that is **affected** by the situation (**sourceComponentId**)
  - The situation itself (**situation**)

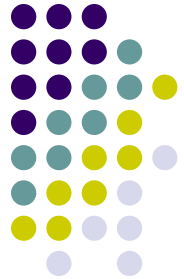
A Practical Guide to the IBM Autonomic Toolkit, IBM Redbooks, April 2004

Event Exchange Standard: CBE Example



- <CommonBaseEvent creationTime="2008-08-16T18:14:27Z" globalInstanceId="N1FB97200C5B11D88000AB0D1D704CDE" msg="[Fri Aug 15 18:14:27 IST 2008] ITSO001I SampleManagedResource starting..." severity="20" version="1.0.1">
  <sourceComponentId application="ITSOSimpleApp1“ component="ITSO Simple App#1" componentIdType="Name“ location="server1.itso.ibm.com" locationType="IPV4“ subComponent="ITSOSubComponent"/>
  <msgDataElement>
  <msgId>ITSO001I</msgId>
  </msgDataElement>
  <situation categoryName="StartSituation">
  <situationType xsi:type="StartSituation“ reasoningScope="INTERNAL“ successDisposition="SUCCESSFUL" situationQualifier="START INITIATED"/>
</situation>
</CommonBaseEvent>

Generating CBEs using Eclipse



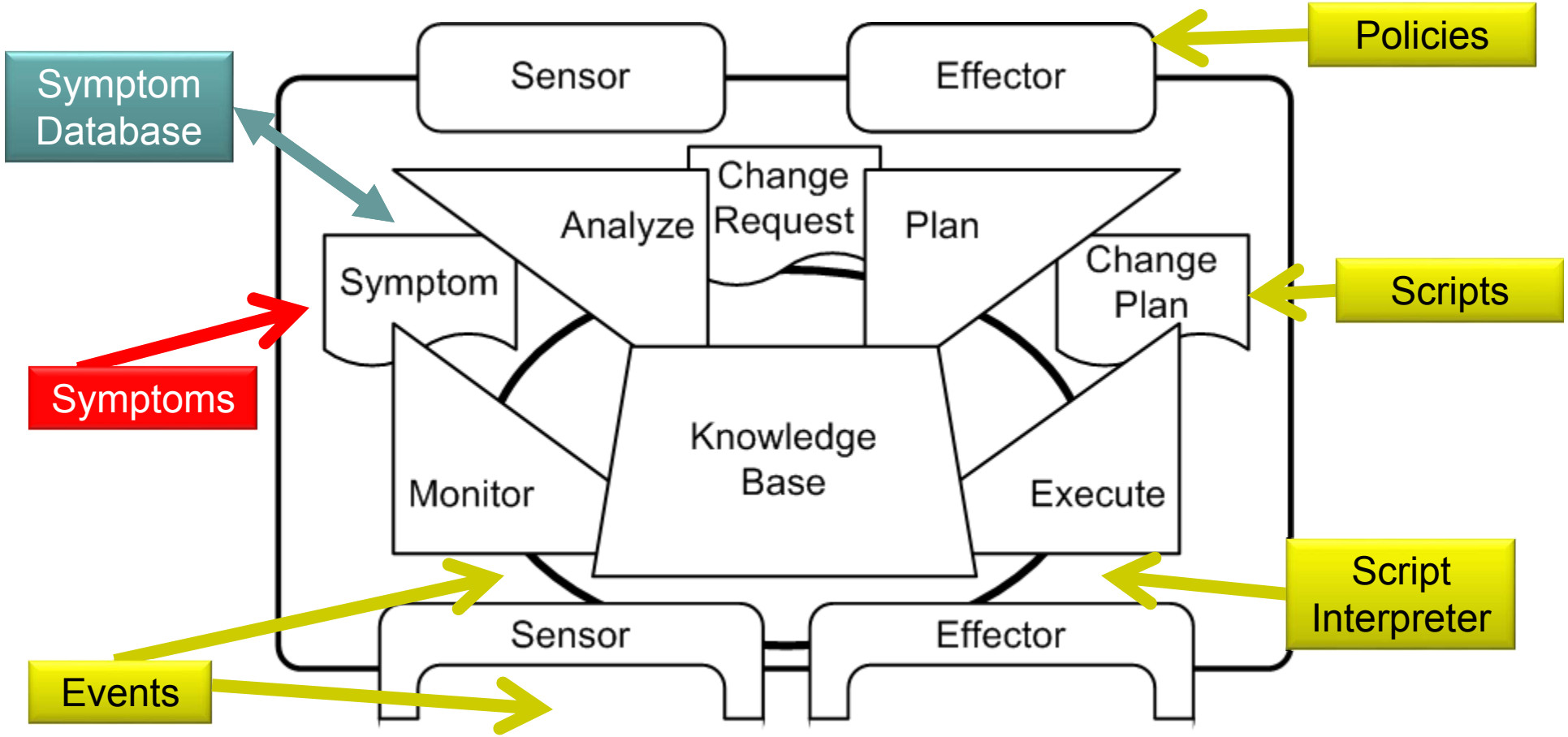
```
try {
  ISimpleEventFactory sefi = SimpleEventFactoryImpl.getInstance();
  ICommonBaseEvent cbe = sefi.createCommonBaseEvent();
  cbe.setCreationTime(System.currentTimeMillis());
  cbe.setPreferredVersion(ICommonBaseEvent.VERSION_1_0_1);

  // create a new instance of a Source Component and initialize it
  IComponentIdentification sourceComponentId =
      sefi.createComponentIdentification();
  sourceComponentId.setLocation("127.0.0.1");
  sourceComponentId.setLocationType("IPV4");
  sourceComponentId.setComponent("Ex App Server");
  sourceComponentId.setSubComponent("App Server DB");
  sourceComponentId.setComponentIdType("Application");
  sourceComponentId.setComponentType("Application Server");

  // now set source component in CBE
  cbe.setSourceComponentId(sourceComponentId);

  IConnectSituation connSituation = sefi.createConnectSituation();
  connSituation.setSuccessDisposition("UNSUCCESSFUL");
  connSituation.setSituationDisposition("AVAILABLE");
  ISituation situation = sefi.createSituation();
  situation.setCategoryName("ConnectSituation");
  situation.setSituationType(connSituation);
  cbe.setSituation(situation); // set the situation in CBE

  IMsgDataElement mde = sefi.createMsgDataElement();
  mde.setMsgId("AS005E");
  mde.setMsgIdType("AppServer");
  // add message data element to CBE
  cbe.setMsgDataElement(mde);

  // invoke manageability interface method
  // to send CBE to autonomic manager
  sendEventToManager(cbe);
}
catch (Throwable  th) {
  System.out.println("Could not create CBE: " + th); }
}
```

IBM: Autonomic Computing Toolkit Developer's Guide, Aug 2004

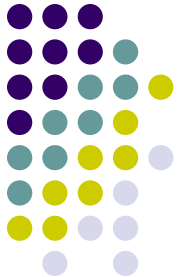
Advantages of Common Base Event Format

- Works for analysis tools from multiple sources and vendors provided CBE is used
- Enables cross-component and cross-vendor
  - Analysis
  - Generation
  - Parsing
  - Logging
  - Tracing
  - Diagnostics
  - ...

MAPE-K Loop Standards & Interfaces



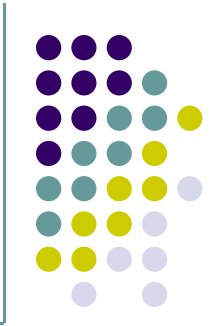
Symptoms



- A *symptom* is a form of knowledge that indicates a possible problem or situation in the managed environment.
  - For example, “high fever” might be defined as a temperature “greater than 39 degrees Celsius”
  - The symptom is defined by the expression “temperature greater than 39 degrees Celsius” and described as “high fever”
- Symptoms are
  - Recognized in the monitor component of the MAPE-K loop
  - Used as a basis for analysis of a problem or a goal
  - Based on predefined elements—for example, definitions and descriptions in a symptoms DB
- Symptom definition
  - Expresses conditions used by the monitor to recognize the existence of a symptom
  - Specifies the unique characteristics of a particular symptom that is recognized.
- Symptoms are not just for self-healing
  - Symptoms are connected to self-healing because their primary intent is to indicate a problem
  - Symptoms can also be used as triggers for other kinds of problems
  - Virtually all kinds of problems or predictions may start due to the occurrence of a symptom

IBM: Symptoms Reference Specification Version 2.0 2006

Symptom Artifacts



- Symptom element
  - Contains all information necessary to create a new symptom occurrence
- Symptom occurrence
  - Contains the run-time information associated with a specific instance of a symptom element
  - Each occurrence basically refers to the same symptom as it is defined in the symptom element, but the context to which it is applied may vary.
- Correlation engine
  - Contains the logic used to create symptom elements
  - As input the correlation engine receives external stimuli and checks if a symptom occurrence should be created as a response.

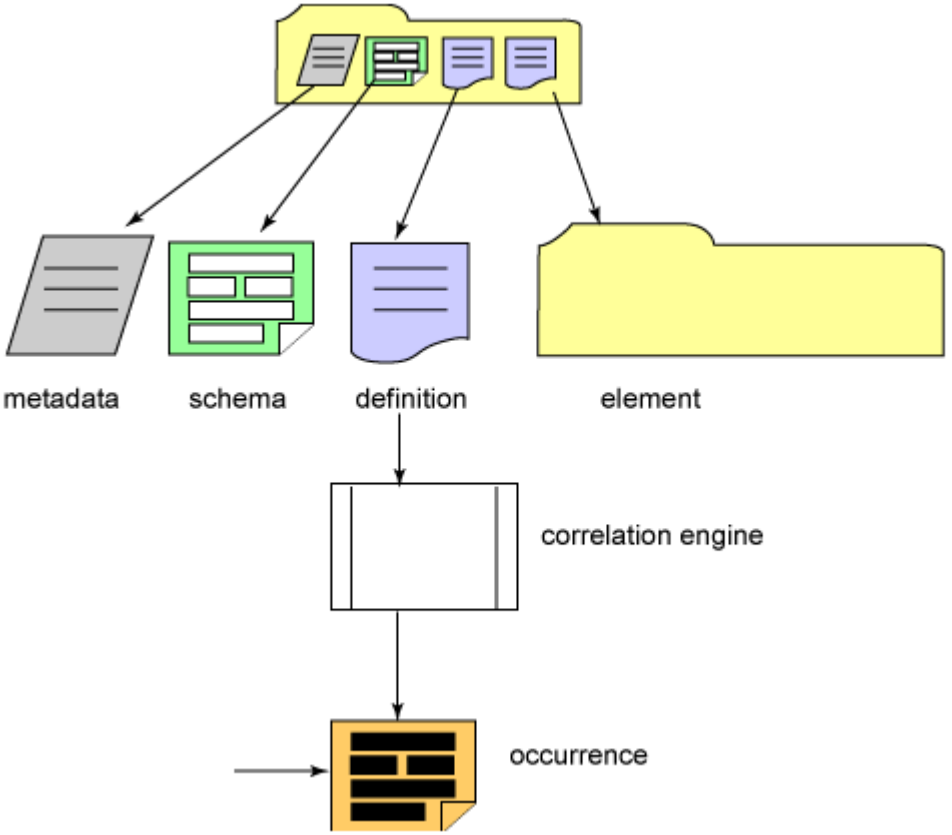
IBM: Symptoms Reference Specification Version 2.0 2006

Symptom Artifacts

- Symptom metadata
  - The generic part of the information that composes a symptom
  - It is present on all kinds of knowledge, and is used when knowledge must be treated generically, even though it is a symptom element
  - This is the “what” part of a symptom
- Symptom schema
  - The specific part of the information that composes a symptom
  - It is the template that is used when a symptom occurrence is created
  - The symptom schema contributes to the “what” part of a symptom
- Symptom definition
  - A generic piece of logic that can be used to recognize a symptom
  - As expected, this logic should be compatible with the respective correlation engine that will be used to process the symptom
  - This is the “how” part of a symptom

IBM: Symptoms Reference Specification, Version 2.0, 2006

Symptom Artifacts



M. Perazolo, IBM: Symptoms deep dive, Part 1
The autonomic computing symptoms format, Oct 2005

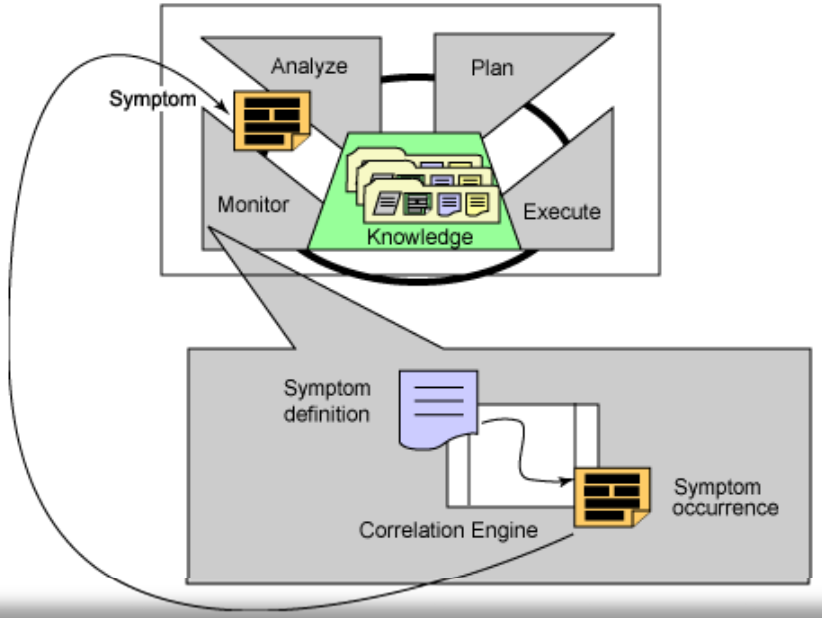
Symptom Effect Artifact

- In simple situations where no analysis or planning is performed, a symptom can be used to define the kind of reaction expected after it is recognized
- Symptom effects can also be used in an AM that implements an on-the-fly strategy for creating change requests
- Symptom effect artifact could be
  - An action to be performed in a manageable resource
  - A human readable recommendation
  - Something simple such as running a script or a piece of code
- The current symptom specification defines only two forms of effect
  - **Recommendation:** A textual representation of what an operator should do to fix the problem associated with a particular symptom
  - **Action:** A piece of code that defines tasks and procedures used to fix the problem associated with a particular symptom

M. Perazolo, IBM: Symptoms deep dive, Part 1
The autonomic computing symptoms format, Oct 2005

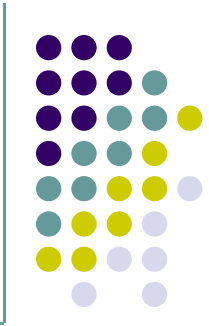
Symptoms in MAPE-K Loop

- Symptoms are recognized by a correlation engine in the monitor
- After recognition a new symptom occurrence is created
- The symptom occurrence is then passed from the monitor to the analysis part of the MAPE-K loop



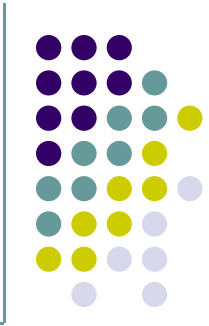
M. Perazolo, IBM: Symptoms deep dive, Part 1
The autonomic computing symptoms format, Oct 2005

Symptom Metadata



Property	Description
Identification	Identifies a symptom uniquely within the MAPE-K loop A unique alphanumeric id
Versioning	Contains the change history associated with the symptom
Annotation	Describes the symptom in a human readable form to explain different characteristics of the symptom
Location	Tells where the authoritative version of this symptom resides and points to the original K-source that contains the symptom
Scope	The manageable resource type a symptom can be applied to At run time, the scope property will also contain the context associated with the symptom occurrence (e.g., the instance of the manageable resource type that is the root cause of the problem or indication defined by the symptom).
Lifecycle	A run-time property containing the current state associated with a symptom occurrence In the case of symptoms, the states are: created, building, analyzed, planning, executing, scheduled, completed, expired, and fault.

Symptom Schema

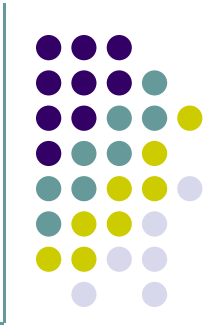


Attribute	Description
Description	Explains in a human readable form what the symptom is about. Describes the kinds of problems or situations associated with the symptom when a symptom occurrence is recognized by an AM.
Example	Shows in a human-readable form an example of a problem or situation where the symptom is likely to occur.
Solution	Shows in a human-readable form a possible solution for the problem or situation described by the example attribute.
Reference	Contains a URL associated with the symptom that lets a user get the latest information associated with that symptom from the Web.
Type	Contains the type associated with the symptom occurrence. It ultimately equates to a symptom category that enables you to organize multiple symptoms in a common taxonomy of symptoms.
Probability	Denotes the probability or certainty associated with the problem or situation indicated by a symptom occurrence.
Priority	Denotes the priority of a symptom occurrence in relation to other symptoms with the same scope.

Symptom Definition

- Symptom definitions
  - Is an artefact used to recognize a symptom occurrence
  - Must be compatible with their respective correlation engines
- A symptom definition can be anything
  - XPATH expression
  - Regular expression
  - Decision tree
  - Dependency graph
  - Prolog predicate
  - ACT pattern
  - TEC rule
  - Neural network

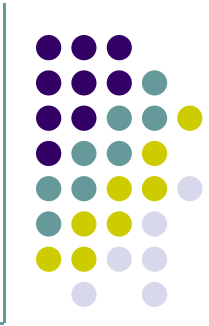
Symptom Examples



Symptom name	Symptom description	Symptom definition	Symptom recommendations
Authentication failure	Attempt to access resources associated with this symptom was made, but there was an authentication failure	Collection pattern: event(wrong_password) n=3 timeout=24h	Log for auditing purposes
Authorization failure	Unauthorized attempt to access resources associated with this symptom was made, and access was denied	Filter pattern: event(access_denied)	Log for auditing purposes
Prevention deployment failure	Failure occurred while deploying security prevention resources (virus update table, security patch, and so on)	Filter pattern: event(security_install_failed)	Analyze security prevention failure *and* alert security administrator

M. Perazolo, IBM: Symptoms deep dive
Part 2: Cool things you can do with symptoms, Dec 2005

Symptom Examples



Symptom name	Symptom description	Symptom definition	Symptom recommendations
Configuration unavailable	Some configuration information for the resources associated with this symptom was not found	Filter pattern: event(configuration_not_found)	Alert administrator and flag service provided by resource as "marginal"
Configuration invalid	Configuration information for the resources associated with this symptom was processed and determined to be invalid	Sequence pattern: event(configuration_found) event(configuration_invalid)	Alert administrator and flag service provided by resource as "marginal"
Dependency unavailable	One or more dependencies (resources) are non-existent and needed by other resources	Sequence pattern: event(dependency_request, resource) event(inventory, resource not within [inventory_list])	Install missing resource
Dependency mismatch	Release level of one or more resources associated with this symptom are not what was expected	Filter pattern: event(wrong_release)	Update resource to required release

M. Perazolo, IBM: Symptoms deep dive
Part 2: Cool things you can do with symptoms, Dec 2005